Neural Frame Interpolation for Rendered Content (Supplementary Document)

KARLIS MARTINS BRIEDIS, DisneyResearch|Studios, Switzerland and ETH Zürich, Switzerland
ABDELAZIZ DJELOUAH, DisneyResearch|Studios, Switzerland
MARK MEYER, Pixar Animation Studios, USA
IAN MCGONIGAL, Industrial Light & Magic, United Kingdom
MARKUS GROSS, DisneyResearch|Studios, Switzerland and ETH Zürich, Switzerland
CHRISTOPHER SCHROERS, DisneyResearch|Studios, Switzerland

ACM Reference Format:
Karlis Martins Briedis, Abdelaziz Djelouah, Mark Meyer, Ian McGonigal, Markus Gross, and Christopher Schroers. 2021. Neural Frame Interpolation for Rendered Content (Supplementary Document). *ACM Trans. Graph.* 40, 6, Article 239 (December 2021), 8 pages. https://doi.org/10.1145/3478513.3480553

In this document we provide more details on the optical flow data generation, describe a training data generation process that is based on publicly available data and report its performance. We then provide more visual comparisons and outputs of our method.

1 TRAINING DATA SAMPLES

Fig. 1. Frame samples from the training dataset. © 2021 Disney

The training dataset was sampled from 2 full-length feature animation films (*Moana*, *Ralph Breaks the Internet*). Some random sample frames that were used as part of the training are shown in Figure 1. Note that the crops shown here are 2x larger than the ones used in training, *i.e.* 896 × 512, for more visually pleasing output.

2 FLOW DATA GENERATION DETAILS

In this section, we provide more details on our *dynamically* generated flow training dataset. High-level psuedocode for the generation of the flow data is provided in Algorithm 1, where $\mathbf{f}_{a\to b}$ is the ground truth flow from temporal position a to b, and I_a - color values, A_a - set of auxiliary feature buffers, mask$_a$ - outline alpha mask for the frame at position a. The `SampleRandomFlow()` returns a random smooth flow field of size 512×384. While different approaches could be used with similar results, in our implementation it is generated from a uniform flow with magnitude ~ Gamma(1.2, 25) summed with flow fields from the following transformations:

0730-0301/2021/12-ART239 $15.00
https://doi.org/10.1145/3478513.3480553

Fig. 2. Frame samples from the alternative TUNGSTEN training dataset

ALGORITHM 1: Optical flow dataset generation

```
Result: $\mathbf{f}_{0\to1}, \mathbf{f}_{1\to0}, I_0, A_0, I_1, A_1$
Function GetLayer():
    $\mathbf{f}_{t\to1} \leftarrow$ SampleRandomFlow()
    $\mathbf{f}_{0\to t} \leftarrow$ ProjectFlow($-\mathbf{f}_{t\to1}$)
    $\mathbf{f}_{1\to t} \leftarrow$ ProjectFlow($\mathbf{f}_{t\to1}$)
    $I_t, A_t \leftarrow$ SampleStaticCGImage()
    $I_0, A_0 \leftarrow \mathcal{W}_{\mathbf{f}_{0\to t}}(I_t), \mathcal{W}_{\mathbf{f}_{0\to t}}(A_t)$
    $I_1, A_1 \leftarrow \mathcal{W}_{\mathbf{f}_{1\to t}}(I_t), \mathcal{W}_{\mathbf{f}_{1\to t}}(A_t)$
    return $\mathbf{f}_{0\to t}, \mathbf{f}_{1\to t}, I_0, A_0, I_t, A_t, I_1, A_1$

$\mathbf{f}_{0\to t}, \mathbf{f}_{1\to t}, I_0, A_0, I_t, A_t, I_1, A_1 \leftarrow$ GetLayer()
for iteration $\leftarrow$ 1 to randint(5, 15) do
    $\mathbf{f}'_{0\to t}, \mathbf{f}'_{1\to t}, I'_0, A'_0, I'_t, A'_t, I'_1, A'_1 \leftarrow$ GetLayer()
    mask$_t \leftarrow$ SampleCOCOOutline()
    mask$_0 \leftarrow \mathcal{W}_{\mathbf{f}_{0\to t}}$(mask$_t$)
    mask$_1 \leftarrow \mathcal{W}_{\mathbf{f}_{1\to t}}$(mask$_t$)
    $\mathbf{f}_{0\to t}, I_0, A_0 \leftarrow$ Merge($\{\mathbf{f}_{0\to t}, I_0, A_0\}, \{\mathbf{f}'_{0\to t}, I'_0, A'_0\}$, mask$_0$)
    $A_t \leftarrow$ Merge($\{A_t\}, \{A'_t\}$, mask$_t$)
    $\mathbf{f}_{1\to t}, I_1, A_1 \leftarrow$ Merge($\{\mathbf{f}_{1\to t}, I_1, A_1\}, \{\mathbf{f}'_{1\to t}, I'_1, A'_1\}$, mask$_1$)
end
return $\mathbf{f}_{0\to t}, \mathbf{f}_{1\to t}, I_0, A_0, I_t, A_t, I_1, A_1$
```

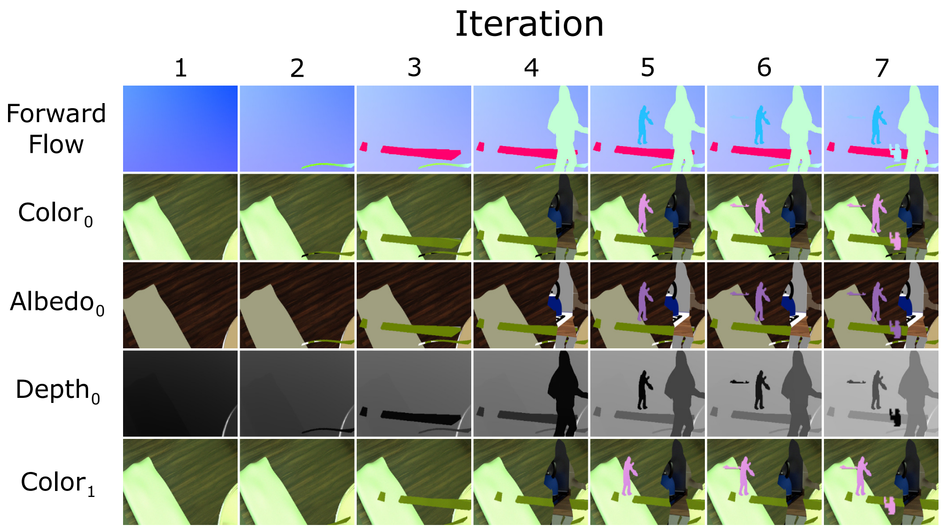

Fig. 3. Progression of an optical flow training sample. On each iteration a sample is built in a bottom-up manner by adding a new masked layer with known flow on the top of the flow, color, albedo, and depth buffers (not showing albedo and depth buffers at positions t and 1). Outlines © COCO Consortium. Textures from the TUNGSTEN dataset.

- with $p = 1/4$, zooming at a uniformly random position with factor $\sim \mathcal{N}(0, 0.2^2)$;
- with $p = 1/8$, rotating around a uniformly random position by angle $\sim \mathcal{N}(0, 30)$;
- with $p = 1/2$, generating random flow vectors of 2×2 to 5×5 (chosen uniformly) pixels with magnitude $\sim \text{Gamma}(1.1, 15)$ and bilinearly upscaling it to the original resolution to obtain smooth localised deformations.

The `ProjectFlow()` function performs normalized forward warp and fills the warping holes by repeatedly applying Gaussian smoothing with kernel size 11 and $\sigma = 11$, affecting only holes until all image pixels have valid values. `SampleStaticCGImage()` returns a randomly sampled static image from our frame interpolation training dataset with the same resolution as the flow field.

A random object outline from the MSCOCO [Lin et al. 2014] is returned by `SampleCOCOOutline()`. For performance reasons, we practically sample only up to 3 different texture images and COCO sources and reuse multiple masks from those.

The `Merge(destination, source, mask)` simply puts `source` pixels on top of the `destination` w.r.t. to the alpha of `mask`. For depth, all `destination` values are shifted by the maximum value of `source` depth plus offset $\sim \mathcal{U}\{0, 20\}$.

The progression of a training sample is visualized in Figure 3.

3 SYNTHETIC TRAINING DATASET

While almost all state-of-the-art frame interpolation methods use the publicly available Vimeo90K [Xue et al. 2019] dataset for the training of their models, unfortunately no such large scale dataset yet exists for rendered content. As we train our models on a proprietary production dataset, in order to facilitate further research we train the model on alternative publicly available scenes.

To build a large-scale publicly available training dataset, we follow the dataset generation procedure by [Bako et al. 2017] and adapt it for generation of sequences with motion. We use the scenes from [Bitterli 2016] and the TUNGSTEN renderer.

For the training set we choose scenes "Contemporary Bathroom", "Country Kitchen", "Bedroom, "The Breakfast Room", "The Wooden Staircase", "Japanese Classroom", and "The White Room", while other scenes can be used for building an evaluation set.

First, we generate 200 static variants of each scene by randomly sampling camera position and *look-at* object, field of view, object materials, and environmental maps. We randomly sample the emission for the existing light sources and optionally add new ones. To make scenes even more diverse, we place new objects taken from the available scenes[1].

In order to add motion to the static scenes, we apply camera transformations and random displacements for the newly added objects. For the camera path we choose one of the following options:

- moving the camera while fixing *look-at* coordinates;
- changing only *look-at* coordinates;
- moving the camera and *look-at* equally such that the camera direction remains constant.

The final sequence of 5 frames is obtained by linearly interpolating these position between the original and transformed scene. Assembled frames are rendered at 1280×780 with 1024 samples-per-pixel and denoised with [Vogels et al. 2018]. As random camera paths can result in crossing of object boundaries, such samples were filtered out in a post-processing step resulting in total of 1325 quintuplets.

Samples of the final frames can be seen in Figure 2. As no volumes, motion blur, and depth-of-field was used, it is less complex than the

[1]sphere; table and chair from "The Wooden Staircase"; pillow, plate, table, and vase from "The Modern Living Room"; chair, vase, and teapot from "The Breakfast Room"

Table 1. Quantitative comparisons with prior methods and the alternative dataset

	training dataset	Production PSNR ↑	Production SSIM ↑	Production LPIPS ↓	Production SMAPE ↓	Production VMAF ↑	Blender PSNR ↑	Blender SSIM ↑	Blender LPIPS ↓	Blender SMAPE ↓	Blender VMAF ↑
BMBC	Vimeo90K	30.08	0.904	0.1071	4.868	59.26	26.65	0.872	0.1693	7.653	65.12
CAIN	Vimeo90K	30.68	0.909	0.1254	4.661	60.43	27.25	0.876	0.1974	7.773	66.37
DAIN	Vimeo90K	31.21	0.915	0.0754	4.189	70.28	28.00	0.885	0.1120	6.313	67.76
DAIN w/ rendered depth	Vimeo90K	31.28	0.916	0.0749	4.181	70.31	28.11	0.885	0.1114	6.325	67.89
AdaCoF	Vimeo90K	30.75	0.907	0.1000	4.652	56.38	27.14	0.875	0.1583	7.502	65.31
SoftSplat*	Vimeo90K	31.28	0.917	0.0650	4.049	67.33	28.06	0.886	0.0974	6.246	68.55
Ours	Tungsten	37.54	0.963	0.0527	2.708	90.95	**36.89**	**0.972**	**0.0261**	**2.666**	**89.13**
Ours	proprietary	**38.49**	**0.967**	**0.0460**	**2.380**	**92.24**	36.85	0.971	0.0268	2.710	87.96

Table 2. Time comparison for obtaining a single intermediate frame between traditional rendering (excluding denoising), image-based interpolation with no additional inputs required, and our interpolation method that requires to render auxiliary feature buffers for the intermediate frame. Output quality is measured on the Production dataset. (see text for more details)

	Intermediate frame CPU rendering time	Interpolation time CPU	Interpolation time GPU	Output quality PSNR	Output quality SSIM
Color render	$2h58m$	-	-	-	-
Color-only interpolation	-	$35s$	$0.40s$	31.27	0.918
Interpolation w/ auxiliary features	$40m$	$53s$	$0.65s$	38.49	0.967

primary dataset that was used, but performs almost equally well as described in the supplementary document Section 4.

4 ADDITIONAL RESULTS

In this section we provide more results of our method. In Table 1 we additionally report VMAF[2] score and report results when using the Tungsten training dataset.

The VMAF score is obtained by running VMAF v2.2.0 on frame triplets, where for one of the sequences the middle frame has been interpolated, and by taking only the middle frame score. Due to many outliers, where subjectively the VMAF score did properly measure the relative quality, we report the median score over the sequences.

Surprisingly, the model trained on Tungsten dataset marginally outperforms the one trained with our primary proprietary dataset on the Blender evaluation set, while being slightly worse on the Production evaluation set, which could be explained by Blender scenes having fewer complex effects.

In Table 2 we report the average time it took to render a single Tungsten dataset frame in CPU core time compared to the interpolation approaches. We implement the option to generate only auxiliary feature buffers by stopping the integrator once these values are recorded at each of the original 1024 sample per pixel, thus only reducing the path length that would be needed to estimate the global illumination. Such naive implementation already showed significant speedup by requiring only 22% of the full render time ($40m$ compared to $178m$). Note that in practical applications, the sample count for feature buffers can be significantly reduced as most regions have very little sample variance. While in theory faithful auxiliary buffers could be estimated by rasterization on the GPU, it would require significant implementation effort for the existing renderers and production scenes often have memory requirements much higher than currently available graphics cards can offer.

More visual comparisons between our method and DAIN [Bao et al. 2019], AdaCoF [Lee et al. 2020], and our re-implementation of SoftSplat [Niklaus and Liu 2020] are provided in Figures 6-5. More interpolation results of our method are shown in Figures 10-18.

[2]https://github.com/Netflix/vmaf

ACKNOWLEDGMENTS

We thank the following Blendswap artists for creating scenes used in the Tungsten dataset: Mareck, Wig42, SlykDrako, Jay-Artist, nacimus, and NovaZeeke

REFERENCES

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 Transactions on Graphics (Proceedings of SIGGRAPH 2017)* 36, 4, Article 97 (2017), 97:1–97:14 pages. https://doi.org/10.1145/3072959.3073708

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

Benedikt Bitterli. 2016. Rendering resources. https://benedikt-bitterli.me/resources/.

Hyeongmin Lee, Taeoh Kim, Tae young Chung, Daehyun Pak, Yuseok Ban, and Sangyoun Lee. 2020. AdaCoF: Adaptive Collaboration of Flows for Video Frame Interpolation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR).*

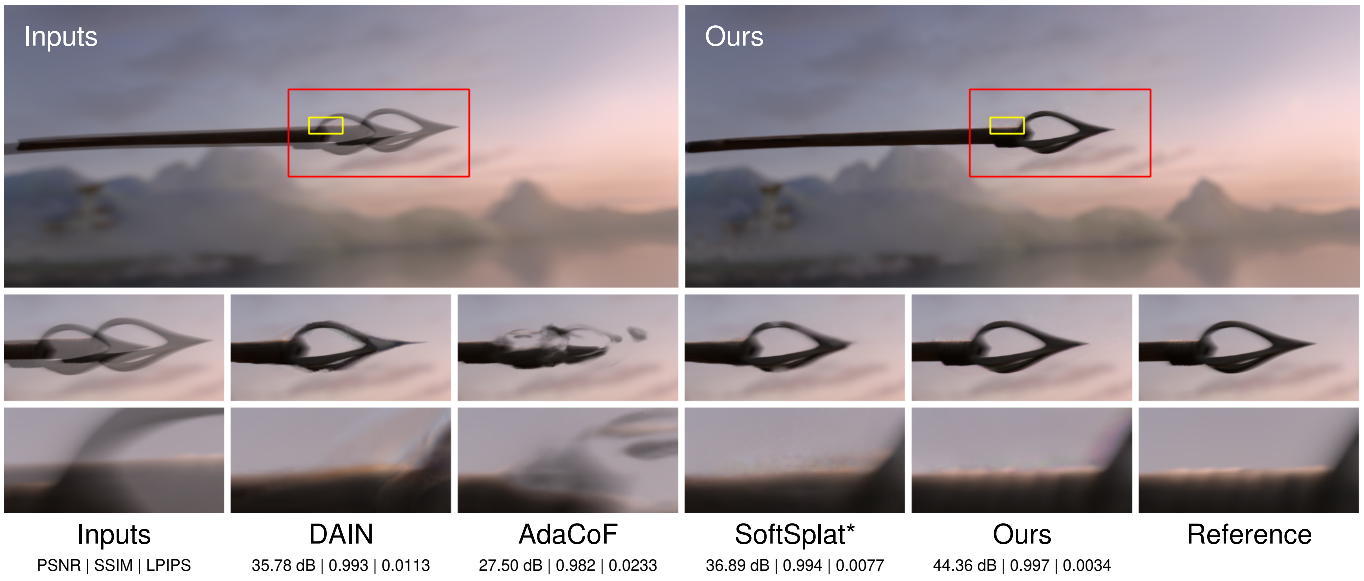
Inputs
Ours
Inputs
PSNR | SSIM | LPIPS
DAIN
35.78 dB | 0.993 | 0.0113
AdaCoF
27.50 dB | 0.982 | 0.0233
SoftSplat*
36.89 dB | 0.994 | 0.0077
Ours
44.36 dB | 0.997 | 0.0034
Reference

Fig. 4. Visual results on a production sequence. © 2021 Disney

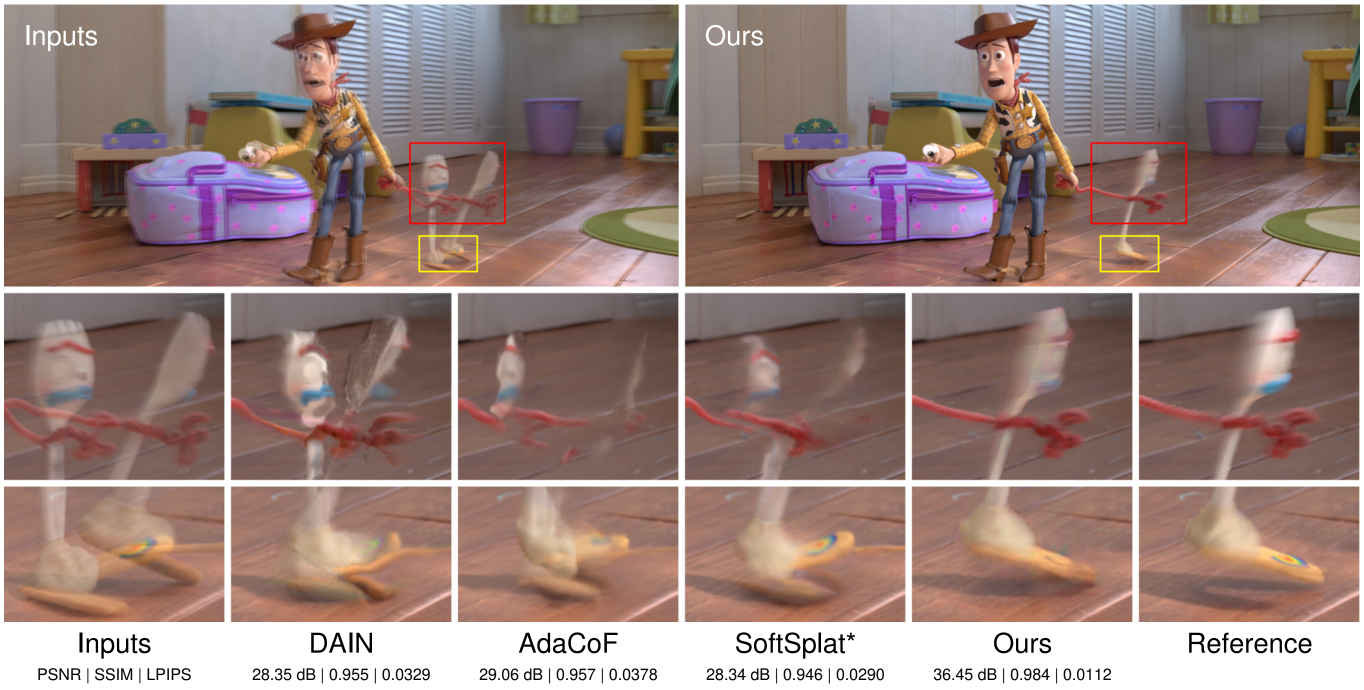
Inputs
Ours
Inputs
PSNR | SSIM | LPIPS
DAIN
28.35 dB | 0.955 | 0.0329
AdaCoF
29.06 dB | 0.957 | 0.0378
SoftSplat*
28.34 dB | 0.946 | 0.0290
Ours
36.45 dB | 0.984 | 0.0112
Reference

Fig. 5. Visual results on a production sequence. © 2021 Disney / Pixar

Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. 2014. Microsoft COCO: Common Objects in Context. In *Computer Vision – ECCV 2014*, David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars (Eds.). Springer International Publishing, Cham, 740–755.

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

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 (Proceedings of SIGGRAPH 2018)* 37, 4, Article 124 (2018), 124:1–124:15 pages. https://doi.org/10.1145/3197517.3201388

Tianfan Xue, Baian Chen, Jiajun Wu, Donglai Wei, and William T Freeman. 2019. Video enhancement with task-oriented flow. *International Journal of Computer Vision* 127, 8 (2019), 1106–1125.

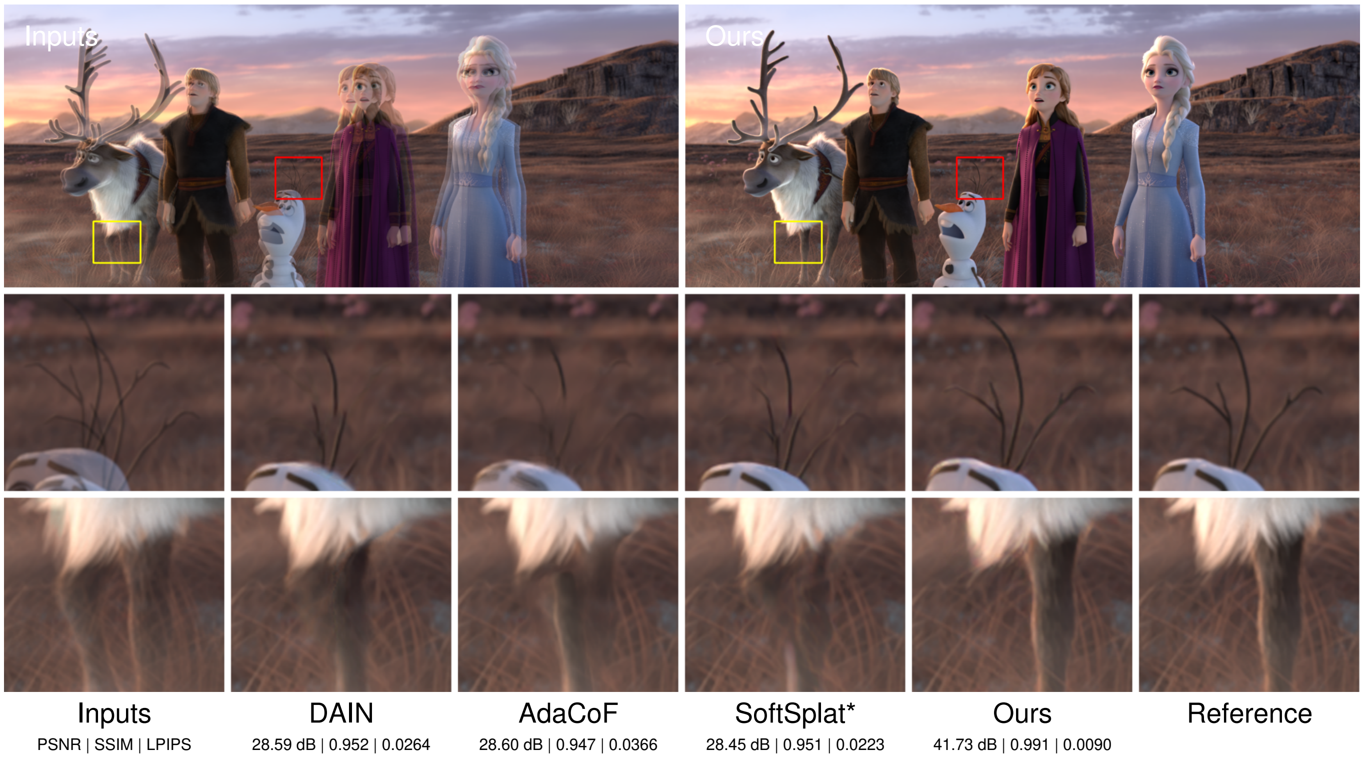
Inputs
Ours
Inputs
PSNR | SSIM | LPIPS
DAIN
28.59 dB | 0.952 | 0.0264
AdaCoF
28.60 dB | 0.947 | 0.0366
SoftSplat*
28.45 dB | 0.951 | 0.0223
Ours
41.73 dB | 0.991 | 0.0090
Reference

Fig. 6. Visual results on a production sequence. © 2021 Disney

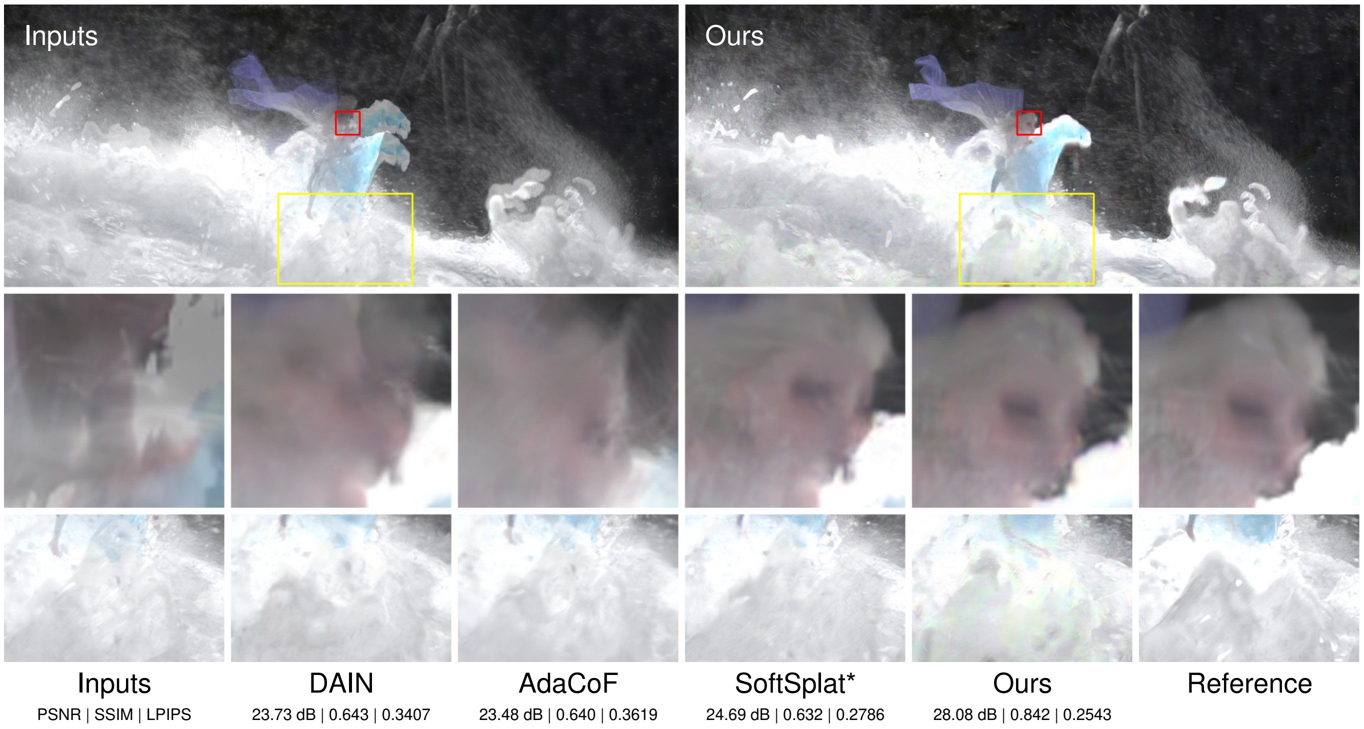
Inputs
Ours
Inputs
PSNR | SSIM | LPIPS
DAIN
23.73 dB | 0.643 | 0.3407
AdaCoF
23.48 dB | 0.640 | 0.3619
SoftSplat*
24.69 dB | 0.632 | 0.2786
Ours
28.08 dB | 0.842 | 0.2543
Reference

Fig. 7. Visual results on a production sequence. © 2021 Disney

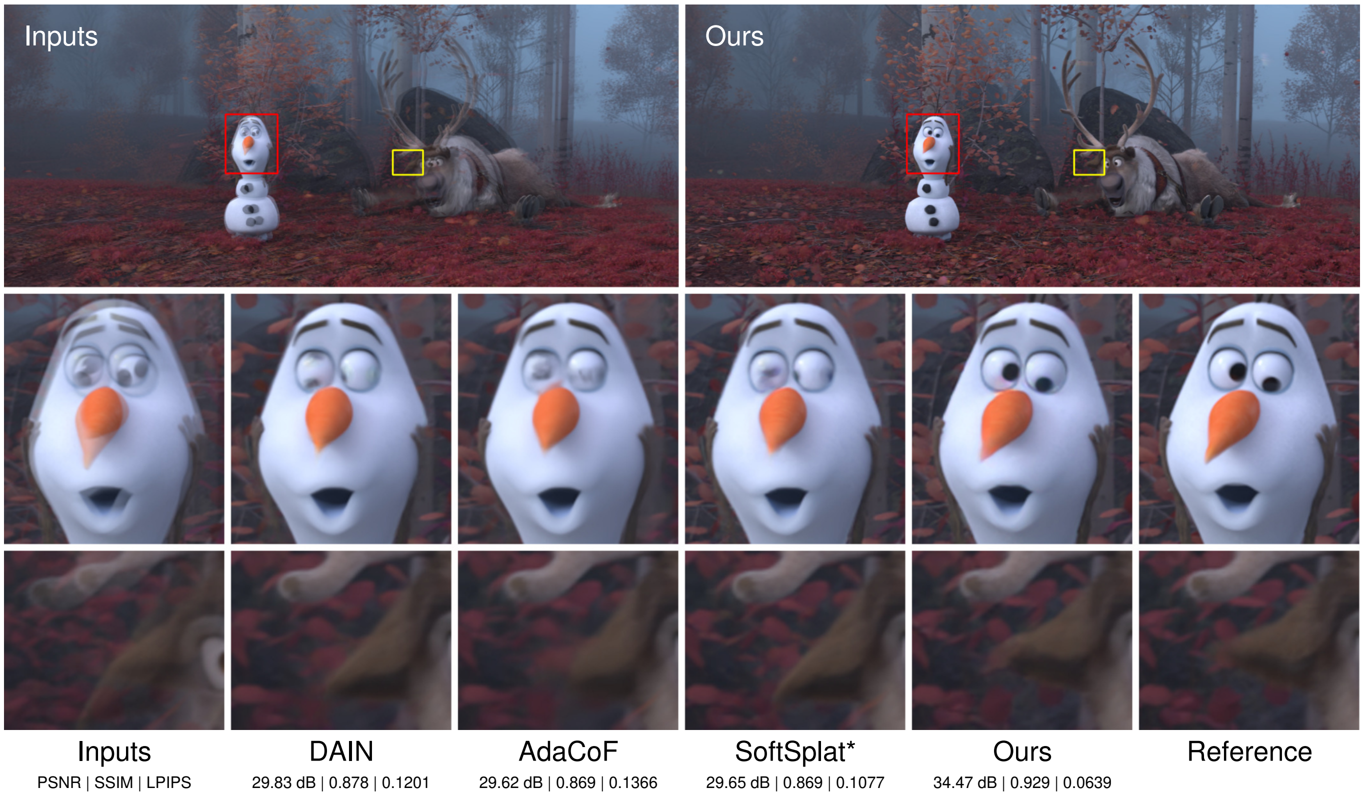
Inputs
Ours
Inputs
PSNR | SSIM | LPIPS
DAIN
29.83 dB | 0.878 | 0.1201
AdaCoF
29.62 dB | 0.869 | 0.1366
SoftSplat*
29.65 dB | 0.869 | 0.1077
Ours
34.47 dB | 0.929 | 0.0639
Reference

Fig. 8. Visual results on a production sequence. © 2021 Disney

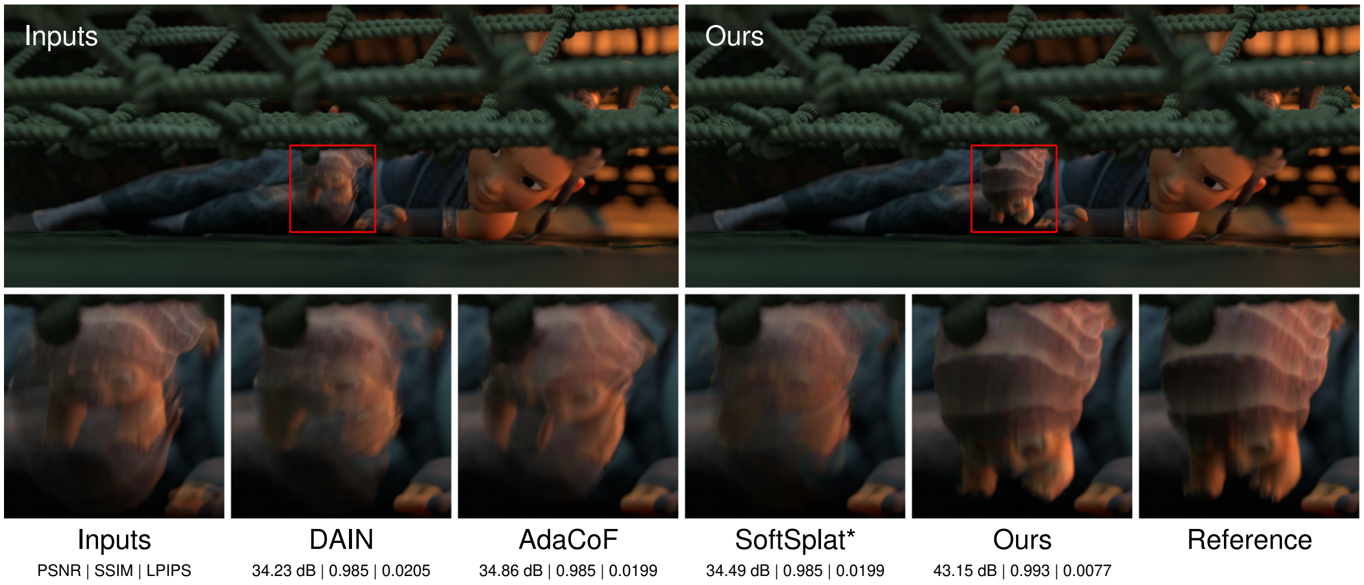
Inputs
Ours
Inputs
PSNR | SSIM | LPIPS
DAIN
34.23 dB | 0.985 | 0.0205
AdaCoF
34.86 dB | 0.985 | 0.0199
SoftSplat*
34.49 dB | 0.985 | 0.0199
Ours
43.15 dB | 0.993 | 0.0077
Reference

Fig. 9. Visual results on a production sequence. © 2021 Disney

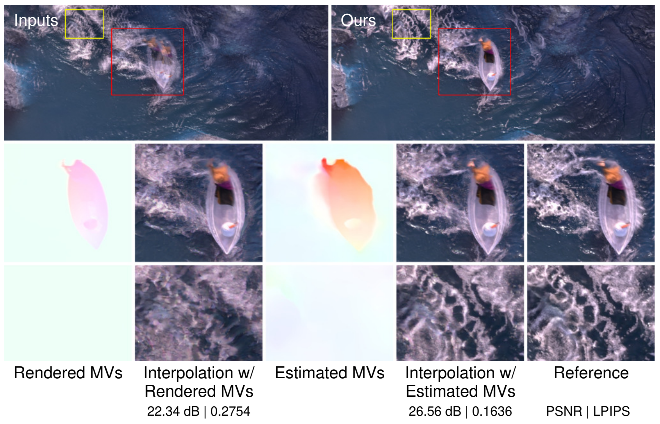

Fig. 10. Comparison between using rendered and estimated motion vectors (MVs) on a challenging sequence with fluid simulation. © 2021 Disney

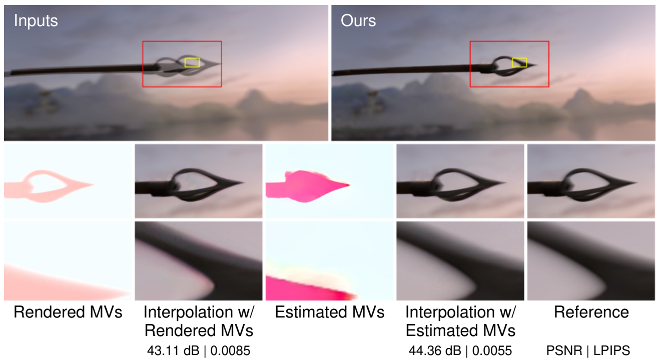

Fig. 11. Comparison between using rendered and estimated motion vectors (MVs) on a challenging sequence with motion blur. © 2021 Disney

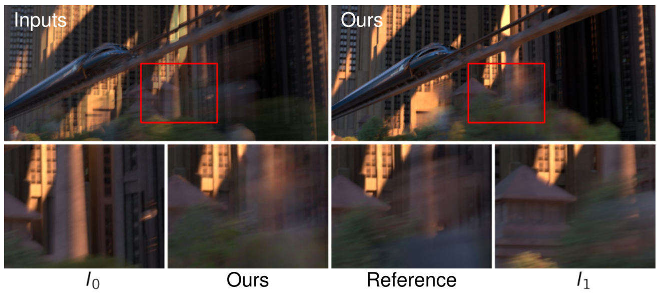

Fig. 12. Interpolation of a challenging sequence with motion blur. © 2021 Disney / Pixar

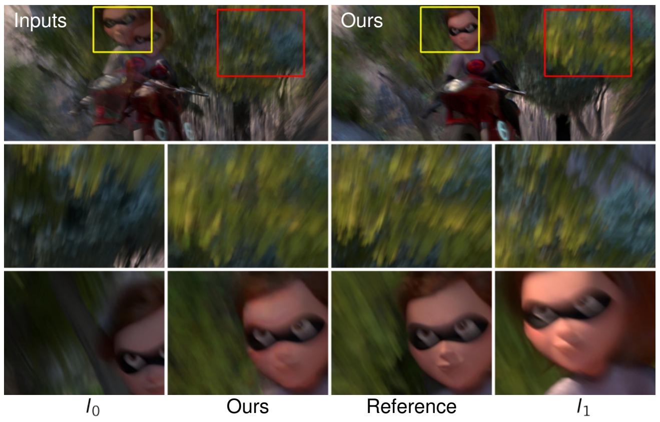

Fig. 13. Interpolation of a challenging sequence with motion blur. © 2021 Disney / Pixar

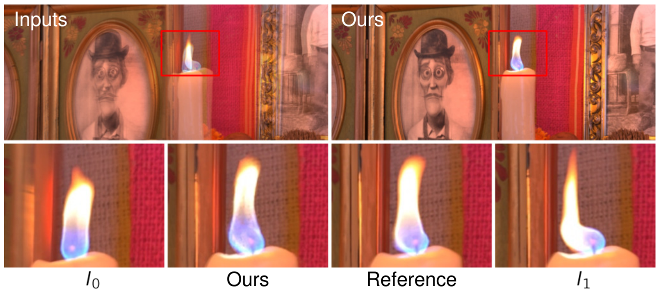

Fig. 14. Interpolation of a flame. © 2021 Disney / Pixar

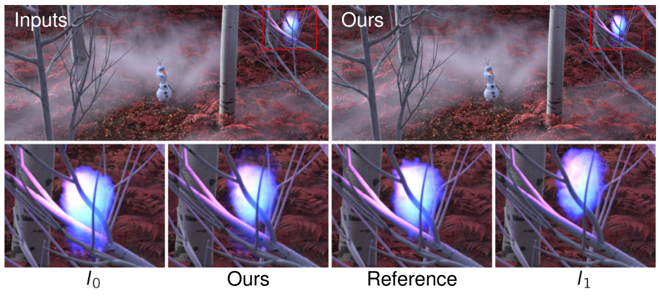

Fig. 15. Interpolation of a volumetric ball. © 2021 Disney

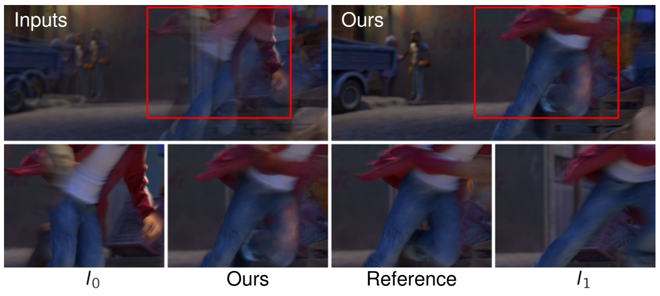

Fig. 16. Interpolation of a challenging sequence with motion blur. © 2021 Disney / Pixar

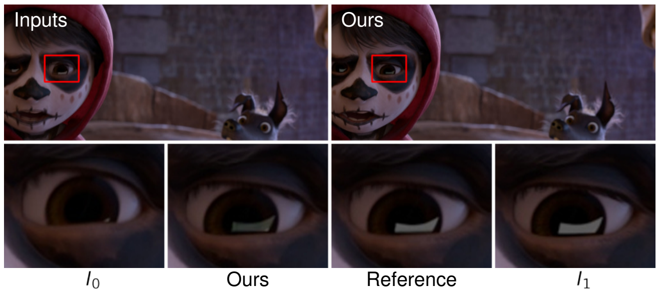

Fig. 17. Interpolation of a view-dependent specular reflection. © 2021 Disney / Pixar

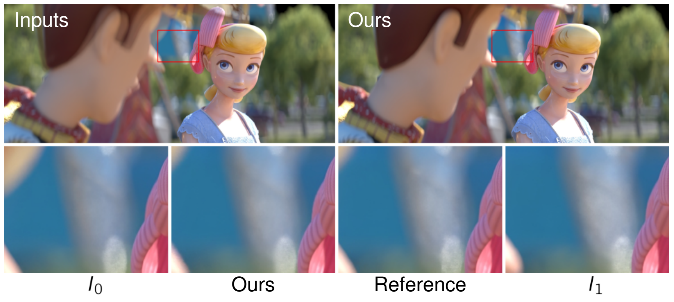

Fig. 18. Interpolation of a sequence with depth of field. © 2021 Disney / Pixar