Neural Frame Interpolation for Rendered Content

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

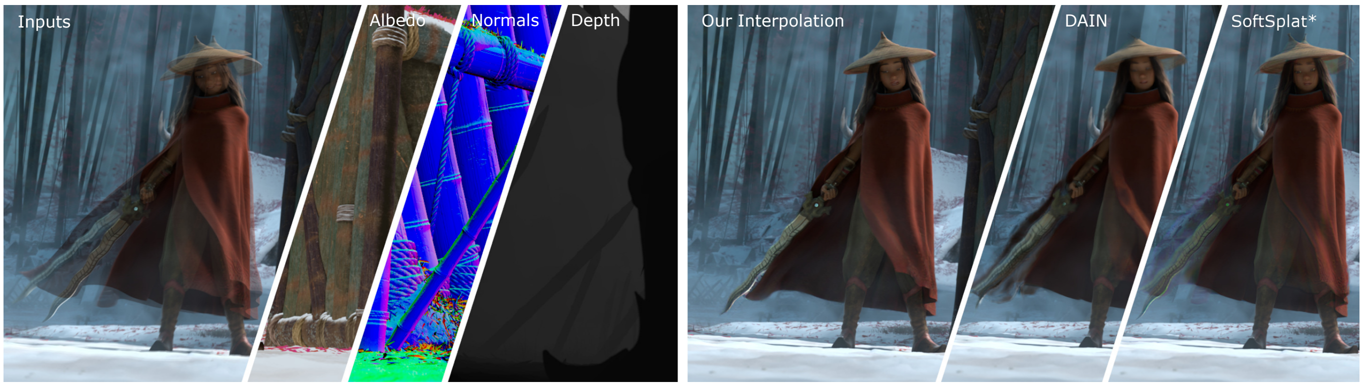

Fig. 1. Our frame interpolation method leverages auxiliary features such as albedo, depth, and normals besides color values (left). This allows us to achieve production quality results while rendering fewer pixels which is not possible with state-of-the-art frame interpolation methods working on color only (right). © 2021 Disney

The demand for creating rendered content continues to drastically grow. As it often is extremely computationally expensive and thus costly to render high-quality computer-generated images, there is a high incentive to reduce this computational burden. Recent advances in learning-based frame interpolation methods have shown exciting progress but still have not achieved the production-level quality which would be required to render fewer pixels and achieve savings in rendering times and costs. Therefore, in this paper we propose a method specifically targeted to achieve high-quality frame interpolation for rendered content. In this setting, we assume that we have full input for every n-th frame in addition to auxiliary feature buffers that are cheap to evaluate (e.g. depth, normals, albedo) for every frame. We propose solutions for leveraging such auxiliary features to obtain better motion estimates, more accurate occlusion handling, and to correctly reconstruct non-linear motion between keyframes. With this, our method is able to significantly push the state-of-the-art in frame interpolation for rendered content and we are able to obtain production-level quality results.

Authors' addresses: DisneyResearch|Studios, Stampfenbachstrasse 48, Zürich, Switzerland; ETH Zürich, Rämistrasse 101, Zürich, Switzerland; Pixar Animation Studios, 1200 Park Ave, Emeryville, CA, USA; Industrial Light & Magic, Lacon House, 84 Theobalds Road, London, United Kingdom. Corresponding authors: Karlis Martins Briedis, karlis.briedis@inf.ethz.ch; Christopher Schroers, christopher.schroers@disney.com.

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

CCS Concepts: • **Computing methodologies → Reconstruction**; **Rendering**.

Additional Key Words and Phrases: Frame Interpolation, Motion Estimation, Deep Learning

ACM Reference Format:
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 (December 2021), 13 pages. https://doi.org/10.1145/3478513.3480553

1 INTRODUCTION

Rendering high-quality computer-generated images is often extremely computationally expensive - many times requiring hundreds of core hours to render a single frame. As the demand for more content continues to grow, as well as the increase in high frame rate content in AR/VR applications, theme park rides, video games and film, addressing these high computational costs is becoming more and more important. Frame interpolation methods, where a subset of additional, in between frames is quickly computed from an existing set of frames, can greatly reduce this computational burden. By rendering many fewer frames (and thus many fewer pixels), rendering times and costs are significantly reduced for path-traced renderings, and turnaround times are improved, allowing for more artist iterations and greatly improving the artist experience.

Recent learning-based frame interpolation methods have shown significant improvement in performance over the last few years but still have not achieved production-level quality. While most prior methods have targeted interpolation of color videos and are suitable for artistic slow-motion, multi-view interpolation, frame rate conversion, we propose an interpolation specific to computer-generated content intended to save costs and decrease rendering time. By targeting only rendered content, it is possible to leverage auxiliary renderer feature buffers for guiding the interpolation. Prior work has shown that significant improvement could be obtained in other scenarios such as denoising [Bako et al. 2017; Vogels et al. 2018] or supersampling [Xiao et al. 2020].

Thus our goal is to design a frame interpolation method for rendered content that significantly outperforms prior methods operating on color only content. We achieve this by carefully integrating the information from auxiliary features in different places of the interpolation pipeline. More specifically, our contributions are:

- an optical flow method for rendered content that deals with non-linear motion between keyframes and even outperforms use of renderer generated motion vectors
- a strategy for dynamically creating a dataset for pre-training optical flow estimator for rendered content
- improved compositing and occlusion handling by leveraging auxiliary feature buffers
- obtaining production level quality results for frame interpolation of rendered content

Our method is specifically tailored to require little to no extra implementation effort when being included into existing production pipelines. The only adaptation that might be required for most renderers is the option to obtain auxiliary feature buffers without rendering the color passes.

Our paper is structured as follows: First, we will review relevant related works. Then we will describe our method and cover important implementation details. Subsequently, we conduct a comprehensive ablation study and show extensive comparisons to other state-of-the-art methods. Finally, we discuss limitations before concluding.

2 RELATED WORK

Video frame interpolation is an activate field of research with applications including frame-rate conversion, slow motion effects and compression, to name a few. In the context of rendering, the objective is to reduce computations by only rendering a subset of frames and rely on frame interpolation to estimate the missing images. Similarly to other works in denoising [Bako et al. 2017; Vogels et al. 2018], super-resolution [Xiao et al. 2020] and temporal processing [Zimmer et al. 2015], auxiliary feature buffers that are available in the renderer can be leveraged for improved results. In this section we review existing works in video frame interpolation and optical flow.

2.1 Video Frame Interpolation

Video frame interpolation has benefited from the advances in deep learning, with recent works achieving remarkable interpolation results. Frame interpolation methods can be split into four categories: phase-based, flow-based with explicit motion modeling and image warping, kernel-based with estimation of spatially varying synthesis kernels, and direct prediction employing feed-forward neural networks.

Phase-Based. [Meyer et al. 2015] model motion in the frequency domain as a partial phase shift between the inputs and use a heuristic to correct the phase difference for obtaining actual spatial motion. Further improvements were made, replacing this heuristic by a neural network parametrization [Meyer et al. 2018]. Phase based methods have shown good results for interpolation of volumetric effects but fall short in sequences with large motion.

Flow-Based. To rely on estimated motion vectors is the most straightforward approach to frame interpolation [Baker et al. 2011]. Typically the solution consists of 2 stages: motion estimation and warping/compositing. In this section we focus on the warping and compositing, as the challenges of optical flow estimation are discussed in more details later.

The key challenge for flow-based frame interpolation techniques is to deal with both occlusion/dis-occlusions and errors in flow estimation. Instead of using heuristics to solve these issues [Baker et al. 2011], methods such as [Jiang et al. 2018; Xue et al. 2019] leveraged end-to-end training of both optical flow and compositing neural network models. Typically flow-based architectures perform better in samples with large motion where other approaches would need very deep models to capture such large displacements.

Further improvements in optical flow models [Sun et al. 2018] and using forward warping [Niklaus and Liu 2018] have pushed the quality of the results one step further. Recent approaches have focused on differential forward warping [Niklaus and Liu 2020] and depth prediction [Bao et al. 2019] to better resolve occlusion boundary issues. To improve over the simple linear motion assumption made by most approaches, [Chi et al. 2020; Liu et al. 2020; Xu et al. 2019] use quadratic and cubic motion models. Enabling higher order motion modeling has been achieved by increasing the input context. However this is sensitive to the temporal gap between the input frames. Most recent flow based interpolation methods investigate recurrent residual pyramid networks [Zhang et al. 2020] and bilateral motion estimation [Park et al. 2020].

Kernel-Based. Instead of relying on optical flow, kernel-based methods combine motion estimation and image synthesis into a single convolution step [Lee et al. 2020; Niklaus et al. 2017a,b]. These methods already resulted in sharp images and may better handle challenging situations, such as brightness changes, than more traditional methods. Recently [Niklaus et al. 2021] demonstrated that improvements in network design and training procedures can significantly reduce the gap with flow based interpolation methods.

Direct. Initial attempts to predict interpolated frame with a classical feedforward CNN [Long et al. 2016] were not able to properly model motion and produced blurry outputs but recent advances have showed improvements. In particular [Choi et al. 2020; Kalluri et al. 2021] use attention or gating mechanisms to model motion and avoid optical flow computation. These approaches can reduce computation time especially in those situations where multiple frames are interpolated in one shot and are straightforward. However, the results still under-perform flow based methods.

2.2 Optical Flow Estimation

Motion estimation is a long standing problem in computer vision and a large body of work exists [Baker et al. 2011]. We focus our

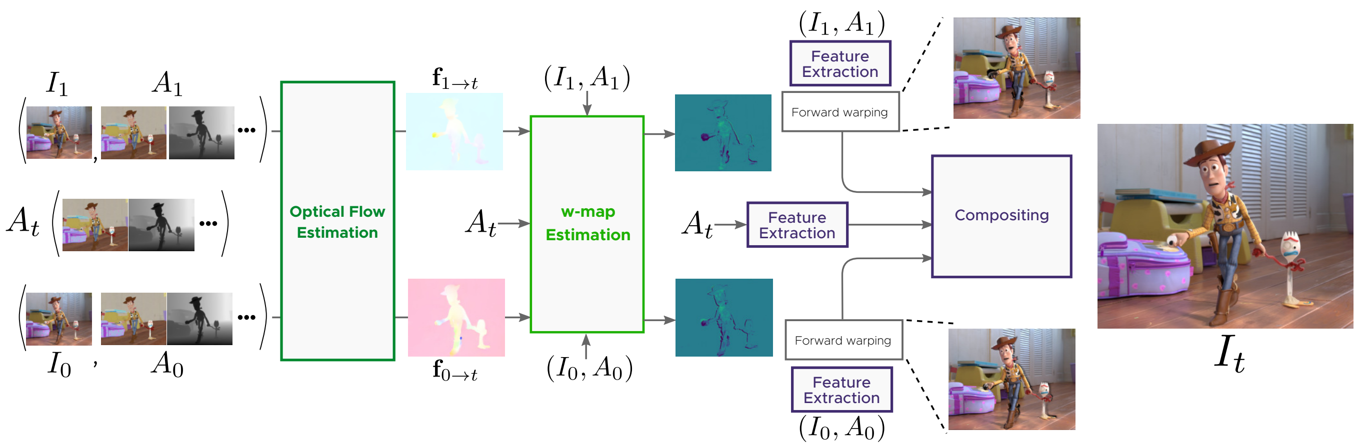

Fig. 2. **Overview.** Both color data I as well as auxiliary features A of the keyframes and the target frame are used in flow estimation, w-map estimation, and compositing. Images © 2021 Disney / Pixar

discussion on the most recent deep learning based methods and detail the important improvements we propose for rendered data. Initially, deep learning based methods have leveraged UNet like architectures [Dosovitskiy et al. 2015; Ilg et al. 2017] that demonstrated promising results despite their simplicity. An important step was made through the usage of partial cost volumes at multiple pyramid levels [Sun et al. 2018]. [Hur and Roth 2019] proposed several improvements that include sharing network weights across the pyramid levels and the estimation of an explicit occlusion map. Recently [Teed and Deng 2020] achieved state of the art results on end point error in optical flow benchmarks. The authors propose to compute a 4D correlation volume for all pairs of pixels which allows very accurate but currently only low resolution flow extraction, i.e. the estimation is performed at 1/8th of the original size. Because of the iterative strategy with the number of iterations as hyper-parameter, it is not clear how this can be trained end to end with a frame interpolation objective.

We would like to draw attention to the complexity of motion estimation, even in the context of rendered images. The work of Zimmer *et al.*[2015] is among the first methods that take interest in using rendered features for temporal processing. The authors propose a decomposition of the rendered frame into different components each accompanied by matching motion vectors. The key insight here is that accurate visual motion estimation requires estimating motion of various light paths. Such correspondences are optimized for endpoint error and there is no guarantee that rendered motion vectors produced in such a way would be the optimal choice for the task of frame interpolation. Besides, obtaining these vectors requires non-trivial implementation effort to production renderers. Similarly, [Zeng et al. 2021] propose a method for estimating motion vectors that can track shadows and glossy reflections in real-time but requires significant renderer adaptation.

Our optical flow model follows a design strategy similar to [Hur and Roth 2019; Sun et al. 2018] with the objective of frame interpolation for rendered data. We take advantage of the auxiliary buffers that are available for the interpolated frame and use them as part of the motion estimation. More precisely, each pyramid level in our optical flow model has an additional cost volume, using only auxiliary features which allows for a more precise estimation of motion with respect to the middle frame. Our interpolation results demonstrate the advantage of this strategy where we avoid the complex path space estimation [Zimmer et al. 2015] and manage to handle non-linear motion.

3 FRAME INTERPOLATION FOR RENDERED CONTENT

The currently best-performing methods for frame interpolation of color-only content are flow-based deep neural networks such as DAIN [Bao et al. 2019], QVI [Xu et al. 2019] and SoftSplat [Niklaus and Liu 2020]. These methods roughly adhere to the following strategy to reconstruct an intermediate frame at an arbitrary temporal position $t \in [0, 1]$: They first estimate motion between keyframes to derive the flow fields to the intermediate frame $\mathbf{f}_{0\to t}$ and $\mathbf{f}_{1\to t}$. Subsequently, they perform a motion compensation by warping to the intermediate frame. In the last step, this is used as input to a synthesis network to composite the final result.

We follow the same strategy in our approach for frame interpolation of rendered content and show how to refine each of the aforementioned steps leading to significantly improved reconstruction quality. We mainly achieve this by consequently incorporating additional render feature buffers in each processing step. These buffers - in our setting albedo, surface normal vectors, and depth - are available not only for keyframes but can also be efficiently evaluated for the intermediate frames to be interpolated.

Figure 2 gives an overview of our approach. In the following sections we explain in more detail how we improve the optical flow based motion estimation, the w-map estimation for occlusion handling during warping, and the compositing step.

3.1 Estimating Motion for Rendered Images

Even though a renderer has access to all information of a scene, production renderers are not always able to output correspondence vectors required to correctly register the keyframe image content on

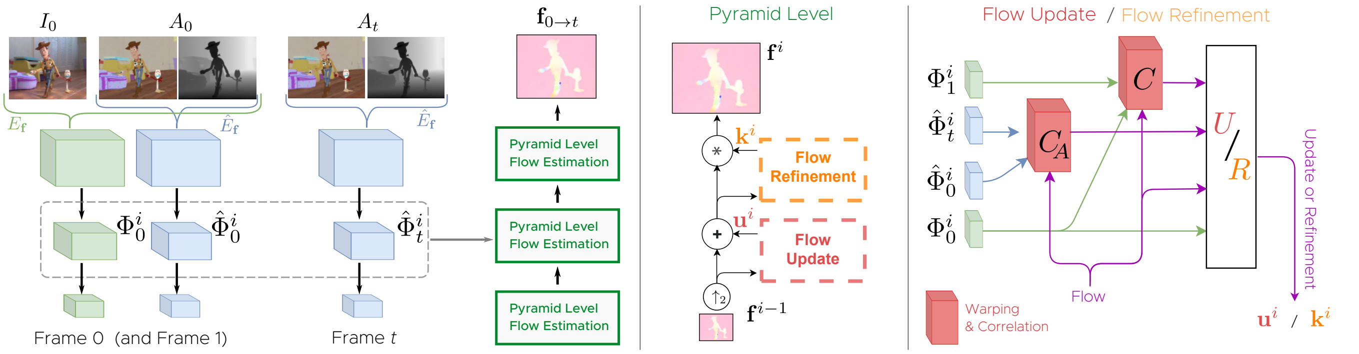

Fig. 3. **Overview of the Flow Estimation Method.** Flow Update and Flow Refinement steps have similar set of inputs, but different correlation size and final estimation networks. Images © 2021 Disney / Pixar

the unknown frame to be interpolated. In such cases, a non-trivial implementation effort would be required to extend the renderer to allow for this.

However, even when this is done, there are challenging cases when such motion vectors are not valid. In cases such as semi-transparent objects, depth of field, and motion blur, motion values can get aggregated from multiple objects which can often lead to displacement vectors that are meaningless. Another fundamentally challenging case is obtaining motion vectors for fluid simulations. As a result, even in the context of rendered data, optical flow estimation remains a crucial element for frame interpolation with great potential for significant quality improvements when properly leveraging auxiliary feature buffers.

Leveraging Auxiliary Feature Buffers. Similarly to existing deep image-based optical flow estimation networks [Hur and Roth 2019; Sun et al. 2018] we adopt a coarse-to-fine strategy that leverages feature pyramids and cost-volumes at multiple scales. In such methods, the flow estimate gets iteratively updated based on a cost volume between learned feature representations of the inputs, the cost volume providing information if some other displacement around the current estimate is a better match. In contrast to previous approaches, we compute feature pyramids not only from color but also from auxiliary feature buffers (albedo and depth) to help the correspondence matching especially in ambiguous situations and to obtain more refined motion boundaries. We denote A the set of available auxiliary features.

Thus, given two frames I_0 and I_1 corresponding to the instants $t = 0$ and $t = 1$ respectively, the goal is to understand the apparent motion by estimating the optical flow $\mathbf{f}_{0\to1}$ from I_0 to I_1 while leveraging the auxiliary features A alongside the color image data I. For this purpose, we propose to encode both I and A at each time instant into a feature pyramid, e.g. $\mathbf{\Phi}_0 = E_F(I_0, A_0)$, through a neural network encoder E_F. The feature pyramid covers different spatial resolutions where from one level to the next the resolution is halved. On each level, a 9×9 cost volume C^9 is computed which is used to extract an incremental flow update through a network U. Overall, taking into account the bilinearly upscaled and accordingly rescaled version of the flow estimate from the previous level $\tilde{\mathbf{f}}^i_{0\to1} = \uparrow_2 \mathbf{f}^{i-1}_{0\to1}$, the incremental update on the current level i is given by

$$\mathbf{u} = U\left(\Phi_0,\ C^9(\Phi_0, \mathcal{W}_{\tilde{\mathbf{f}}}(\Phi_1)),\ \tilde{\mathbf{f}}\right) \quad . \tag{1}$$

Here $\mathcal{W}$ is the backward warping function and we have used the shorthand notation $\tilde{\mathbf{f}}$ to denote the flow $\tilde{\mathbf{f}}^i_{0\to1}$. In addition, we generally drop the index i which determines the current pyramid level for notational convenience whenever possible. While a new incremental flow update $\mathbf{u}$ is computed on each level, the parameters of the network U that extracts the update are shared across all levels.

Subsequently, we refine the flow estimate of the current level $\tilde{\mathbf{f}} + \mathbf{u}$ following the exact same principle as in the update step. To make the refinement *lightweight* but still expressive, we only compute a 1×1 cost volume C^1 feeding into a small refinement network R to predict 3×3 kernels $\boldsymbol{k}$ that can be applied for refining the flow. Thus we have

$$\boldsymbol{k} = R\left(\Phi_0,\ C^1(\Phi_0, W_{\tilde{\mathbf{f}}+\mathbf{u}}(\Phi_1)),\ \tilde{\mathbf{f}} + \mathbf{u}\right) \tag{2}$$

and the final flow estimate at a given level can be obtained by applying these kernels $\boldsymbol{k}$:

$$\mathbf{f} = \left(\tilde{\mathbf{f}} + \mathbf{u}\right) \circledast \boldsymbol{k} \quad . \tag{3}$$

IRR-PWC [Hur and Roth 2019] also has a similar refinement module but the partial feature correlation C^1 is not considered during refinement.

High Resolution Flow Refinement. Due to computational costs and small numerical impact on end point error, existing optical flow methods typically limit themselves to a portion of the original resolution (e.g. 1/4 for PWC [Sun et al. 2018] and IRR-PWC [Hur and Roth 2019] or 1/8 for RAFT [Teed and Deng 2020]). To reach the full resolution they rely on bicubic upscaling, or predicted upsampling kernels by introducing additional parameters in the RAFT case .

In the setting of frame interpolation it can be beneficial to obtain accurate warps at high resolution. Therefore, to improve the accuracy of flow motion boundaries, we want to estimate motion up to the full resolution without incurring prohibitive computational cost and memory requirements. We can achieve this by only running the flow update step until a quarter resolution while maintaining the *lightweight* refinement up to the full resolution. This refinement can be especially powerful in the presence of auxiliary feature buffers since they can offer additional guidance. In this case, the refinement will not operate on $\tilde{\mathbf{f}} + \mathbf{u}$ as shown in Equation 2 for the lower levels but on $\tilde{\mathbf{f}}$ instead.

Non-Linear Motion. Most frame interpolation approaches that are based on flow assume linear motion between key-frames. They estimate the flow to the target frame $\mathbf{f}_{0\to t}$ by multiplying the key-frame flow with the time step of the intermediate frame, *i.e.*

$$\mathbf{f}_{0\to t} = t \cdot \mathbf{f}_{0\to 1}\ . \tag{4}$$

First of all such an approximation can deviate from the original artistic intent and would limit the applicability of frame interpolation on real productions. Second, this approximation causes misalignment between intermediate frame feature buffers and the warped key-frames. This misalignment hinders drawing full benefit from the additionally available data. During the model training, the misalignment between the model output and the ground truth would lead to a suboptimal loss and training process. Finally, in case of smaller artifacts remaining after the interpolation step, partial re-rendering of specific regions will be difficult due to the mismatch in motion. Such regions can, most trivially, be selected through an interactive artist input.

Our objective in this section is to leverage auxiliary features of the target frame I_t to address these issues. We modify the optical flow method to directly predict $\mathbf{f}_{0\to t}$, by additionally considering intermediate feature buffers. Essentially following the same coarse-to-fine strategy based on cost volumes but the estimation at each level i becomes

$$\mathbf{u} = U(\Phi_0,\ C^9(\Phi_0, W_{\tilde{\mathbf{f}}/t}(\Phi_1)),\ C_A^7(\hat{\Phi}_0, W_{\tilde{\mathbf{f}}}(\hat{\Phi}_t)),\ \tilde{\mathbf{f}}) \tag{5}$$

with the following changes and additions

$$\mathbf{u} = \mathbf{u}_{0\to t}, \quad \hat{\Phi}_0 = \hat{E}_F(A_0) \quad \text{and} \quad \hat{\Phi}_t = \hat{E}_F(A_t). \tag{6}$$

In analogy to Equations 2 and 3, we run a refinement considering partial features as in 5. The important difference here is that in this case we directly predict the optical flow $\mathbf{f}_{0\to t}$. This is reflected in the prediction of all the pyramid levels i. In addition to this we have partial features $\hat{\Phi}_0$ and $\hat{\Phi}_t$. These are called partial to emphasize that the corresponding encoder $\hat{E}_F$ has only access to partial data for those instants, namely the auxiliary buffers A_0 and A_t. To leverage these partial features, an additional 7×7 cost volume C_A^7 is used.

It is interesting to note here that we do not only rely on the cost volume between secondary features (C_A) but also retain the cost volume between key-frames (C). The reason for this is to allow estimation of motion that is only apparent in the full features available for key-frames. Figure 3 shows an overview of our flow estimation method.

Training The Optical Flow Model. Typically the training procedure of optical flow networks is rather involved and uses sequential trainings on specifically curated training datasets such as FlyingChairs [Dosovitskiy et al. 2015], FlyingThings [N.Mayer et al. 2016] and potentially a final fine tuning step on the Sintel dataset [Butler et al. 2012]. These datasets are not applicable to train our motion estimation network due to lack of the secondary features. The only exception to this is the Sintel dataset which does contain additional feature channels. However it is too small for the purpose of neural network training. Therefore we create our own dataset of rendered content containing all required buffers. Our main insight is that for successful flow training, synthetic image objects and their outlines do not have to align, which gives a lot more flexibility in creating a dataset. To achieve a similar complexity as FlyingChairs, we use the silhouettes available in the MSCOCO [Lin et al. 2014] dataset and textures from our frame interpolation dataset to composite pairs of frames with known motion.

We follow the training strategy of [Hur and Roth 2019; Sun et al. 2018] and optimize the mean endpoint error loss weighted across the estimation levels:

$$\mathcal{L}_{flow} = \sum_{i=1}^{n} w_i \cdot \left(\left\| \mathbf{f}^i_{0\to t} - \widehat{\mathbf{f}}^i_{0\to t} \right\|_2 + \left\| \mathbf{f}^i_{1\to t} - \widehat{\mathbf{f}}^i_{1\to t} \right\|_2 \right)\ . \tag{7}$$

Here n is the number of levels, $\widehat{\mathbf{f}}^i$ is the reference flow downscaled to the resolution of level i and $\mathbf{f}^i$ is the current network estimate at level i. We describe our dataset generation and training in more detail in Section 4.1.

3.2 Input Preprocessing

Similar as in the flow estimation, where we extract feature pyramids through an encoder E_F, we will also be extracting feature pyramids through another encoder E_C for the subsequent processing steps of occlusion handling and compositing. This is motivated by the fact that feature pyramids have also been shown to be beneficial for frame interpolation [Niklaus and Liu 2018, 2020] and here we briefly introduce the encoders that we use.

Full Context Encoder. We do not only extract features from color values, but also all feature buffers that are available for both keyframes. In this case we use color, normals, albedo, and depth. We refer to this as the *full context encoder* and independently apply it to extract context representations at 3 levels of scale (1/1, 1/2, and 1/4 of the original resolution) from both keyframes. This encoder is denoted E_C.

Partial Context Encoder. To make use of the auxiliary feature buffers A that are available also at the intermediate frame temporal position, we introduce the *partial context encoder*, which has an identical architecture as the full context encoder but is slightly smaller and has an independent set of weights. Following similar notation pattern as previously, this partial encoder is denoted $\hat{E}_C$.

3.3 Handling Occlusions in Motion Compensation

In the next processing step, the previously estimated motion is used to perform motion compensation to align the keyframe content for the final compositing step. In this process, properly handling occlusions and disocclusions is one of the most crucial aspects. Both backward and forward warping have certain advantages and drawbacks. We opt for forward warping, as, unlike backward warping, it does not require estimation of flows $\mathbf{f}_{t\to 0}, \mathbf{f}_{t\to 1}$ that are often obtained with an approximation [Jiang et al. 2018].

In the forward warping operation, every pixel in the source image I_0 is splatted to the target image I_t by mapping them with a given displacement vector $\mathbf{f}_{0\to t}$. After the warping, it is normalized by the sum of the weight contributions. For any location $\boldsymbol{y}$ on the image plane Ω, this can be described as

$$I_t(\boldsymbol{y}) = \left(\sum_{\boldsymbol{x} \in \Omega} I_0(\boldsymbol{x}) \cdot W(\boldsymbol{x}, \boldsymbol{y}) \right) \cdot \left(\sum_{\boldsymbol{x} \in \Omega} W(\boldsymbol{x}, \boldsymbol{y}) + \varepsilon \right)^{-1} \quad (8)$$

with a constant ε, the weighting function

$$W(\boldsymbol{x}, \boldsymbol{y}) = w(\boldsymbol{x}) \cdot k((\boldsymbol{x} + \mathbf{f}(\boldsymbol{x})) - \boldsymbol{y}) \ , \quad (9)$$

and an e.g. bilinear splatting kernel k. With forward warping, disocclusions are naturally taken care of resulting in empty regions that do not receive any information. However, in order to correctly deal with occlusions, it is important to have an accurate way of estimating the weighting factor $w(\boldsymbol{x})$. This weighting factor effectively determines which pixel has higher importance when multiple source pixels contribute to the same target pixel due to inaccuracies in flow or occlusions. A possible weighting factor is inverse depth, which can either be estimated from inputs [Bao et al. 2019; Niklaus and Liu 2020] or, in our case, using renderer depth values.

However, depth can suffer from noise and in addition might not be the optimal choice, e.g. when foreground objects have a large z-axis motion and move behind an object that has a higher depth value in the source frame. Therefore we estimate the weighting map referred to as *w-map* using a neural network. An overview of our weight estimation network is shown in Figure 4. Analogously to our input preprocessing step and flow pyramids, to estimate weighting for frame I_0, we first extract the full context representation $\Psi = E_C(I, A)$ from both keyframes I_0 and I_1, partial context $\hat{\Psi} = \hat{E}_C(A)$ from secondary information of the keyframe I_0 and the intermediate frame I_t. Then we backward warp $\Psi(I_1)$ and $\hat{\Psi}(I_t)$ to the time step of I_0, and use this as channel-wise concatenated input to a 3-level UNet [Ronneberger et al. 2015] with skip connections. To avoid negative contributions, outputs of the UNet need to be mapped from $[-inf, inf]$ to $[0, inf]$.

Since we have some access to the intermediate frame structure, the range of weights can be much higher than in prior methods. In our experiments this introduced numerical stability issues when using direct exponentiation as proposed in [Niklaus and Liu 2020]. Therefore we opt for

$$f(x) = \max(0,\ x)\ +\ \min\left(1,\ e^x\right)\ = ELU(x) + 1\ , \quad (10)$$

which corresponds to a shifted *ELU* [Clevert et al. 2016] activation function.

The possibility to have a very high or low confidence introduces an additional problem when performing the flow network tuning for the task of frame interpolation. By fully normalizing the outputs by the sum of weights as in Equation 8, the only parameter that can be adjusted to remove all contributions to the target pixel is adjusting the kernel parameter, *i.e.* the flow, which introduces suboptimal flow updates and caused the network to diverge. To allow for reducing such contributions in the warping step, we use a more generous padding parameter of $\varepsilon = 1$ in the denominator of Equation 8.

3.4 Compositing with Features

In the last step required, the individual warped frames need to be composited to synthesize the desired intermediate frame to be interpolated. Instead of warping the input frames directly, we use a common approach in flow estimation methods and warp the feature pyramids that provide a better representation of the inputs. Specifically, for obtaining this feature representation we follow a method as in [Niklaus and Liu 2020] since it has been shown to contribute to a better reconstruction.

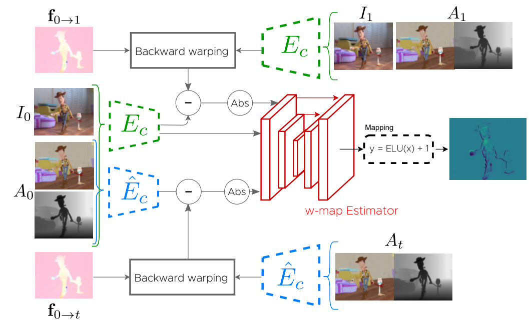

Fig. 4. **Architecture of our forward warp weighting.** Both full and partial contexts are taken into account to estimate the *w-map* required for warping frame I_0 to temporal position t. The same method is applied estimate *w-map* for I_1. Images © 2021 Disney / Pixar

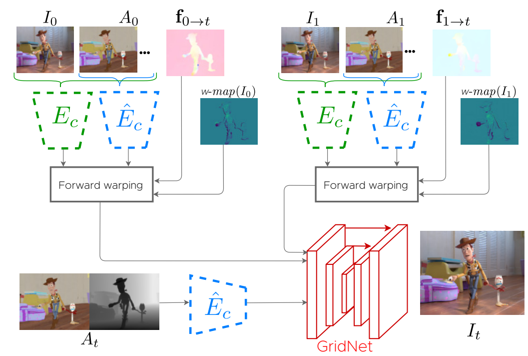

Fig. 5. **Compositing with Features.** Partial architecture of our frame interpolation method for rendered content showing the compositing piece of the method. Images © 2021 Disney / Pixar

A graphical representation of our compositing strategy is shown in Figure 5. This design enables the final frame synthesis step to compare warped inputs and to reason on their correctness by also extracting partial feature pyramids through $\hat{E}_C$ of the target frame. In accordance to previous methods that forward warp [Niklaus and Liu 2018, 2020], we use the GridNet [Fourure et al. 2017] architecture with three rows, six columns and replace the bilinear upsampling layers by transposed convolutions .

4 EXPERIMENTAL SETUP

In this section we supply more details on the implementation and training process. Roughly, the overall procedure is as follows: First the optical flow network is pre-trained by optimizing the endpoint error w.r.t. the ground truth flow vectors. In the second step, the *w-map* and compositing networks are trained with a reconstruction error on the target frame while keeping the flow network fixed. In the third step we jointly fine tune the whole method. We will start by covering the flow estimation piece and then explain the details of the remaining frame interpolation pipeline.

4.1 Optical Flow for Rendered Content

Auxiliary Feature Inputs. First, as highlighted in Section 3, we compute feature pyramids not only from color I but also from the auxiliary feature buffers A - depth and albedo. For albedo, just like color, it is reasonable to assume brightness constancy across corresponding pixels in subsequent frames. From prior works, it is well known that optical flow networks can learn to account for most brightness changes occurring in practice. The supervised training might especially be of further help for the network to establish invariances without explicitly prescribing them. In contrast to this, assuming such a constancy assumption is more problematic for depth as we expect more complex changes in subsequent frames when the camera and objects move. As a result, we would expect it to be difficult to obtain optimal results when supplying depth values directly. However, since depth edges and motion boundaries often are in alignment, depth is still a valuable feature for motion estimation. In order to account for these circumstances and to make this input more easily accessible to the network, we normalize the depth values by dividing them with the median depth of the frame sequence and invert them.

Hierarchical Flow Estimation. IRR-PWC [Hur and Roth 2019] performs flow updates on a *local* scale while refining at *full* scale, *i.e.* in this case the magnitude of the flow corresponds to that of the highest resolution, instead of the current local scale of the specific pyramid level. To avoid numerous flow rescalings, we perform all operations in the local scale and perform instance normalization[Ulyanov et al. 2016] which also harmonizes varying flow magnitudes across levels.

Dataset. Since existing optical flow training datasets are either not sufficiently big or lack the auxiliary feature buffers, we implement a strategy for *dynamically* generating flow training data based on an existing set of static rendered frames and silhouettes. In this paper we use the annotations from the MSCOCO [Lin et al. 2014] dataset outlining object silhouettes. To build a single training triplet along with the desired ground truth flows

$$\big((I_0, A_0), (I_t, A_t), (I_1, A_1), (\mathbf{f}_{0\to t}, \mathbf{f}_{1\to t})\big)\ , \tag{11}$$

we first sample a random background image including all required color and auxiliary channels from our frame interpolation training dataset. We then generate a random smooth flow field $\mathbf{f}_{t\to 1}$ by applying small global rotations, translations, and scalings. Additionally, we also create a very small resolution flow field with random flow vectors which are then upscaled to obtain smooth localised deformations in high resolution. To obtain the desired flow fields $\mathbf{f}_{0\to t}$ and $\mathbf{f}_{1\to t}$, we apply forward warping and fill holes with an outside-in strategy similar to the one used in [Bao et al. 2019]. Note that in this case the smoothness of the flow field is crucial for having negligible occlusions and obtaining precise flow field outputs. We then apply the deformations induced by the flow to all channels to obtain our background plates. In the second step, we randomly sample a silhouette as well as another image containing all required color and auxiliary channels. We then use the silhouette to extract a foreground element from this image. Here our key insight is that silhouettes and image content do *not* have to coincide for successful flow training which greatly facilitates the training process. We then estimate another smooth flow field with the same strategy as for the background, apply it to the foreground element, and paste all channels onto the background plates. For the depth values of the foreground element, instead of directly using them, we make sure that they are smaller than the ones in the background by applying a global shift. Finally, the ground truth flow fields are updated accordingly at the foreground positions. Please refer to our supplementary document for more details and visual examples of the flow data generation.

Training Details. We implement our flow model in the PyTorch framework and train it using the Adam [Kingma and Ba 2014] optimizer with a learning rate of 10^{-4} and a weight decay of $4 \cdot 10^{-4}$. We select a batch size of 4 and train for 200k iterations by minimizing the endpoint error loss as shown in Section 3.1. Training of our final flow model takes approximately 1.5 days on a single NVIDIA 2080 Ti GPU using 32-bit floating point arithmetic.

4.2 Frame Interpolation

Baseline Implementation. In principle, our proposed components are applicable for most flow-based frame interpolation methods. The approach that we follow as a baseline to integrate our contributions is as described in SoftSplat [Niklaus and Liu 2020]. We opted for such a baseline due to its strong results and lean design. Given that the implementation and model weights of SoftSplat are not publicly available, we re-implement it following the authors description. We use the simpler of the proposed importance metrics for estimating a *w-map* to handle occlusions:

$$w = \alpha |I_0 - \mathcal{W}_{\mathbf{f}_{0\to 1}}(I_1)|_1\ . \tag{12}$$

This is because the refined *w-map* variant, as suggested in SoftSplat, does not show significant gains. To be able to present a meaningful ablation study, we train such a baseline on the same dataset and training schedule as the rest of our models.

Dataset. We gather a training dataset by sampling 291 shots of 7-14 frames from 2 full-length feature animation films (*Moana*, *Ralph Breaks the Internet*), building up to 2138 triplets at 1920×804 resolution. Triplets from 13 of these shots are left out for the validation. Each training sample is generated by randomly sampling a fixed 448×256 crop from all frames of the triplet. This data is further augmented by adjusting the hue and brightness of the color values, performing random horizontal, vertical, and temporal flips, and randomly permuting the order of both surface normal and albedo channels.

The method is quantitatively evaluated on 38 diverse triplets selected from 4 feature animation films (*Incredibles 2*, *Toy Story 4*, *Frozen II*, and *Raya and the Last Dragon*) rendered with two different production renderers and with rather different visual style than the training set, further referenced as the Production set. On the other hand, we evaluate our results on publicly available sequences rendered with Blender's Cycles renderer for comparisons with future methods. All frames are rendered until little noise is left and the color values are further denoised, auxiliary feature buffers are obtained with the same sample count.

Training. We implement our models in the PyTorch framework and train using the Adamax [Kingma and Ba 2014] optimizer with a learning rate of 10^{-3} and a batch size of 4. We employ a two stage training similar to [Niklaus and Liu 2020] and fix the weights of flow network during the first stage. In the first stage we optimize for an averaged L1 loss during 217.5k iterations. In the second stage, we additionally impose a perceptual loss [Niklaus and Liu 2018] with weight of 0.4. and enable flow network updates. The second stage is optimized for an additional 72.5k iterations. Every 14.5k iterations we reduce the learning rate by a factor of 0.8. Training of our final model takes approximately 3 days on a single NVIDIA 2080 Ti graphics card using 32-bit floating point arithmetic.

Performance. It takes approximately 0.65*s* to run the interpolation network on 1280×780 inputs with the aforementioned graphics card and unoptimized implementation, making it negligible when comparing to full renders. Generation of the auxiliary feature buffers requires only a fraction (2 – 10x less time) of the hundreds of CPU core hours that are necessary for computing the full illumination of production scenes. For a better estimate, we naively extend the academic Tungsten renderer to record only albedo, depth, and normal values with the same sample count and obtain the average CPU core time it takes to render such buffers for simple scenes as 40*m* compared to 2*h*58*m* for the full render, *i.e.* almost 5× speedup. Note that significant gains could be made by reducing the sample count as feature buffers typically have noise only around object boundaries, at specular surfaces, *etc.* For more details, please refer to the supplementary document.

Summing up, it is interesting to note that with our new dataset creation strategy, the flow pre-training is much less of a burden. This is because both the flow pre-training and the frame interpolation training can be performed on the same images and all that is additionally required for the flow pre-training is a set of object silhouettes. Furthermore, our models are trained in the sRGB colorspace with the maximum range limited to 1.

5 METHOD ANALYSIS

In this section we analyse in more detail how the individual components that we propose contribute to the final result. First, we detail on the evaluation dataset and error metrics that we use for this purpose before inspecting the effect of each component one by one.

5.1 Evaluation Dataset and Metrics

Our method is evaluated on two datasets, namely Production and Blender, as described in Section 4.2.

We measure distortions between the sRGB outputs and the reference with *peak signal to noise ratio* (PSNR), *structural similarity index measure* (SSIM) and the perceptual LPIPS [Zhang et al. 2018] metric. Additionally, we report the *symmetric mean absolute percentage error* (SMAPE) [Vogels et al. 2018] computed on linear RGB outputs and reference (reported as %) and the median VMAF[1] score over the sequences.

[1] https://github.com/Netflix/vmaf/tree/v2.2.0

Table 1. Analysis for our proposed improvements on the Production evaluation set (see text for details).

		PSNR ↑	SSIM ↑	LPIPS ↓	SMAPE ↓	VMAF ↑
	Baseline	31.27	0.918	0.0717	4.092	60.58
Flow	Keyframe features	31.58	0.919	0.0707	3.991	63.34
Flow	2-frame	**35.97**	**0.952**	0.0561	**2.742**	**87.05**
Flow	2-frame w/o full refine	35.90	0.952	0.0565	2.746	86.79
Flow	Ours final	35.52	**0.952**	**0.0545**	2.794	85.34
Warp	Depth	36.14	0.953	0.0566	2.780	87.55
Warp	Feature constancy	36.55	0.956	0.0504	2.710	88.72
Warp	Ours final	**37.83**	**0.962**	**0.0496**	**2.496**	**90.51**
Comp	Direct features	38.08	0.966	**0.0454**	2.414	90.95
Comp	Ours final	**38.49**	**0.967**	0.0460	**2.380**	**92.24**

5.2 Estimating Motion

We demonstrate the effectiveness of our proposed motion estimation network by changing the flow model in incremental steps. We start from a simple IRR-PWC [Hur and Roth 2019] variant with adaptations as described in 4.1 (Baseline) until we reach our final flow method (Flow - Ours final). Results are shown in the second section of Table 1.

As first step, we incorporate auxiliary feature buffers in the flow estimation to improve correspondence matching and flow refinement (Keyframe features). This allows us to slightly improve the accuracy in all metrics and to outperform our baseline and prior art. However, such an approach does not take into account the non-linear motion that often occurs between the keyframes. By accounting for non-linear motion and using our proposed flow variant, we obtain significant improvement in all observed metrics.

An alternative to our 3-frame flow estimation is to solely rely on auxiliary buffers A_0 and A_t for the correspondence estimation. In this case, the estimation of the incremental flow update at given level is defined as

$$\mathbf{u} = U(\Phi_0,\ C^9_A(\hat{\Phi}_0, W_{\tilde{\mathbf{f}}}(\hat{\Phi}_t)),\ \tilde{\mathbf{f}})\ . \tag{13}$$

We will refer to this version as the 2-frame flow. We show that this allows to obtain very similar results as the 3-frame model, but we opt for the 3-frame because it shows on-par or better results on the structural and perceptual metrics. Additionally, the 2-frame variant is effectively a part of the 3-frame version while not having the ability to learn light-dependent motion that is not visible in auxiliary feature channels.

To show that refinement up to full resolution is beneficial, we evaluate the same 2-frame variant once with refinement to the full resolution and once with refinement stopped at a quarter of the resolution. In this case, we observe a slight decrease in all observed metrics.

5.3 Handling Occlusions

In the third section of Table 1 we show the improvement achieved by our warping method, while using our final flow estimation variant.

Table 2. Analysis of using rendered flow vectors on the BLENDER evaluation dataset. We show the performance difference on the same model with the output of our flow estimator replaced by rendered motion vectors.

	Rendered flow	PSNR ↑	SSIM ↑	LPIPS ↓
Baseline	×	31.27	0.918	0.0717
	✓	**31.87**	**0.932**	**0.0704**
Baseline trained w/ rendered flow	✓	31.49	0.931	0.0720
With our Flow	×	**34.22**	**0.962**	**0.0357**
	✓	32.56	0.931	0.0667
With our Warp	×	**36.08**	**0.966**	**0.0292**
	✓	35.34	0.943	0.0539
With our Comp	×	**36.85**	**0.971**	**0.0268**
	✓	36.01	0.95	0.0469

As depth is often seen as a good weighting for forward warping [Bao et al. 2019; Niklaus and Liu 2020], we evaluate our method by using normalized inverse depth. We do not use softmax splatting [Niklaus and Liu 2020] due to the depth range easily reaching $5 \cdot 10^5$, as then calculating it as $exp(-|\alpha| * depth)$ we would get 0 contribution for many pixels of the scene. Additionally, the scale of depth values are scene-dependent. Such adaptation allows us to slightly improve over the brightness constancy baseline approach.

As an additional experiment, we extend the idea of brightness constancy assumption for using a weighted sum for constancy assumption for each channel in color and auxiliary features:

$$w_0 = \exp\Big(\sum_{c \in \{I,A\}} \alpha^c |c - \mathcal{W}_{\mathbf{f}_{0\to1}}(c)| + \sum_{c \in \{A\}} \alpha^c |c - \mathcal{W}_{\mathbf{f}_{0\to t}}(c)| \Big)\ , \quad (14)$$

where c is the respective channel and α^c is a channel dependent learnable weighting factor initialized as -1.

We show that with such weighting we are able to achieve slight improvement over the depth approach, but it performs significantly worse than our final *w-map* estimation module.

5.4 Compositing

In the last section of Table 1 we evaluate the effectiveness of our final frame compositing approach. We compare it to using auxiliary features directly, instead of the proposed partial feature pyramids. Both variants are using our final flow and warping modules.

To do so, we extend the feature pyramid extractor of [Niklaus and Liu 2020] to process all available keyframe auxiliary features and match our full context encoder E_C. Additionally, we concatenate the warped inputs with the auxiliary features of the intermediate frame A_t before the final frame synthesis network. Overall we can observe a gain in reconstruction quality apart from LPIPS.

5.5 Comparison against Rendered Motion Vectors

As mentioned initially, not all production renderers offer to output correspondence vectors. However, since some renderers do, we compare our end-to-end trained motion estimation for the task of frame interpolation against using motion vectors extracted from the renderer. In this case we use BLENDER's CYCLES as a renderer.

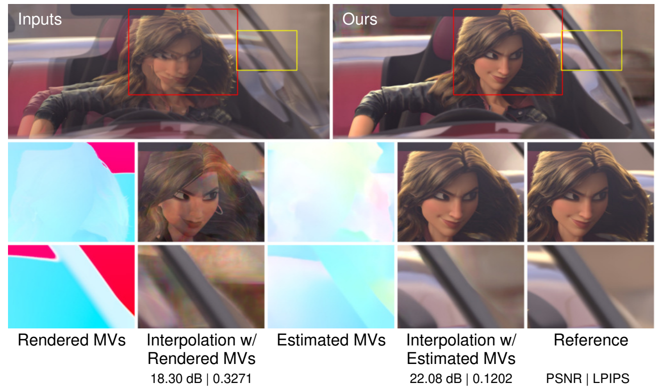

Fig. 6. Comparison between using rendered and estimated motion vectors (MVs) on a challenging sequence with an almost transparent windshield. © 2021 Disney

In Table 2 we follow the same structure as in Table 1 and show results after adding in each of our contributions. This time, we additionally evaluate with the rendered motion vectors by replacing the outputs of the optical flow estimation network. We observe a decrease of performance in all of the intermediate steps, except for the baseline. As the interpolation network might get specialized for the particular type of flow it was trained with, we also train a variant for baseline using motion vectors available to the renderer but observe even worse results than when trained with a neural flow. Such a decrease might be explained by the fact that rendering engines are not always able to produce accurate correspondence vectors in all cases.

A visual example where use of rendered motion vectors yields much worse quality outputs than our optical flow method is given in Figure 6.

6 RESULTS

In this section, we evaluate the performance of our method compared to the state-of-the-art frame interpolation methods - DAIN [Bao et al. 2019], AdaCoF [Lee et al. 2020], CAIN [Choi et al. 2020], BMBC [Park et al. 2020], and our re-implementation of SoftSplat [Niklaus and Liu 2020]. In the case of DAIN, as it was not possible to run it on the Full HD content with our available hardware, we split the inputs along width axis into two tiles with 320 pixel overlap, and linearly combine the results. We do not notice any artifacts that could be caused by such tiling.

For the evaluation, we interpolate the middle frame given two key-frames and compare against the different interpolation methods. With camera depth buffers being available, we additionally test performance of DAIN with output of depth estimator replaced with such buffer. To match the scale of depth that DAIN was originally trained with, for each frame we scale the rendered depth by $\frac{mean(\text{DAIN depth})}{mean(\text{Rendered depth})}$ to match mean values of both depth maps. We use similar error measures as in our ablation study. The full quantitative evaluation is provided in Table 3. By leveraging auxiliary features and designing an interpolation method specifically addressing rendered content, we achieve sufficiently high quality results to consider this method usable in production.

Table 3. Quantitative comparisons with prior methods.

	PRODUCTION				BLENDER			
	PSNR ↑	SSIM ↑	LPIPS ↓	VMAF ↑	PSNR ↑	SSIM ↑	LPIPS ↓	VMAF ↑
BMBC	30.08	0.904	0.107	59.26	26.65	0.872	0.169	65.12
CAIN	30.68	0.909	0.125	60.43	27.25	0.876	0.197	66.37
DAIN	31.21	0.915	0.075	70.28	28.00	0.885	0.112	67.76
DAIN w/ rendered depth	31.28	0.916	0.075	70.31	28.11	0.885	0.111	67.89
AdaCoF	30.75	0.907	0.100	56.38	27.14	0.875	0.158	65.31
SoftSplat*	31.28	0.917	0.065	67.33	28.06	0.886	0.097	68.55
Ours	**38.49**	**0.967**	**0.046**	**92.24**	**36.85**	**0.971**	**0.027**	**87.96**

Fig. 7. Interpolation of multiple in-between frames. Our method maintains high quality interpolation results for all time offsets. © 2021 Disney

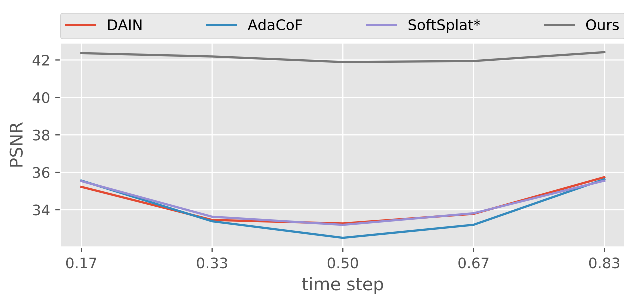

Fig. 8. Temporal consistency for 6x interpolation.

Visual comparison with prior methods is provided in Figures 9, 10, and 11. Interpolation of more challenging cases is shown in Figures 12, 13, 14, and 15. We can observe the high quality interpolation results on a large variety of scenes, with different types of content, different amounts of motion and using different renderers. In addition to this, we provide a supplementary video with results on longer video sequences.

To further analyze the temporal stability of our method, we evaluate interpolation of multiple intermediate frames on a subset of our PRODUCTION evaluation set where features for 5 intermediate frames are available. In the case of AdaCoF [Lee et al. 2020], interpolation is applied recursively to obtain all the frames. The results of this 6x interpolation evaluation are shown in Figures 7 and 8. Similarly to other methods, we show stable interpolation performance for non-middle frame interpolation but we note the important gain in quality.

7 LIMITATIONS AND DISCUSSION

Although we propose a robust method significantly outperforming prior state-of-the-art, there are still a few limitations and open areas which are beyond the scope of this paper. In this section we briefly touch upon them.

As our method relies on the use of *auxiliary feature buffers*, it can lead to sub-optimal results in sequences where such buffers do not provide a proper representation of the color outputs. This is the case for scenes containing *volumes* for example. A visualization of this scenario is shown in Figure 12. In such situations, one remedy could be to discard or zero out unhelpful auxiliary features. More principled solutions, that consider these problematic aspects during the training stage for example, would be an interesting direction for further explorations.

Even though our 3-frame optical flow estimation network in theory is capable of estimating *light-dependent motion* that is not visible in any of the auxiliary buffers, in practice there can be cases that are not correctly resolved. In these relatively rare situations that are hard to resolve, estimating a single flow vector per pixel is not sufficient due to various lighting effects [Zimmer et al. 2015], and a possible improvement could be an independent interpolation of separate passes, such as direct/indirect illumination. One such example is given in Figure 15 where due to the texture having different movement than one of the shadows, it cannot be handled correctly with the current approach. But even then, in most cases our method fails gracefully by linearly blending the inputs, as can also be seen when interpolating vanishing specular highlights (Figure 14).

In Figure 13 it is shown that even sequences with severe motion blur can be resolved well by producing output with similar levels of blur to the reference.

While we have shown that using renderer motion vectors directly can lead to worse quality than using our optical flow estimation method, often they can provide beneficial information and incorporating them would make for interesting future work.

8 CONCLUSION

In this paper we have proposed a method specifically targeted to achieve high quality frame interpolation for rendered content. Our method leverages auxiliary feature buffers from the renderer to estimate non-linear motion between keyframes that is even preferable over using rendered motion vectors in cases where those are available. Through further improvements of occlusion handling and compositing, we are able to obtain production quality results on a wide range of different scenes. This is an important step towards rendering fewer pixels to save costs and increase iteration times during the production of high quality animated content. We were able to show examples of successfully interpolating challenging sequences where only every 8th frame was given. Since we closely follow the correct non-linear motion between keyframes, there is a possibility of re-rendering small regions in case of imperfect reconstructions. Our method is designed to be easy to implement in production pipelines and it is also convenient to train due to our flow pre-training strategy that largely can operate on the same data as the frame interpolation training itself. While our method is

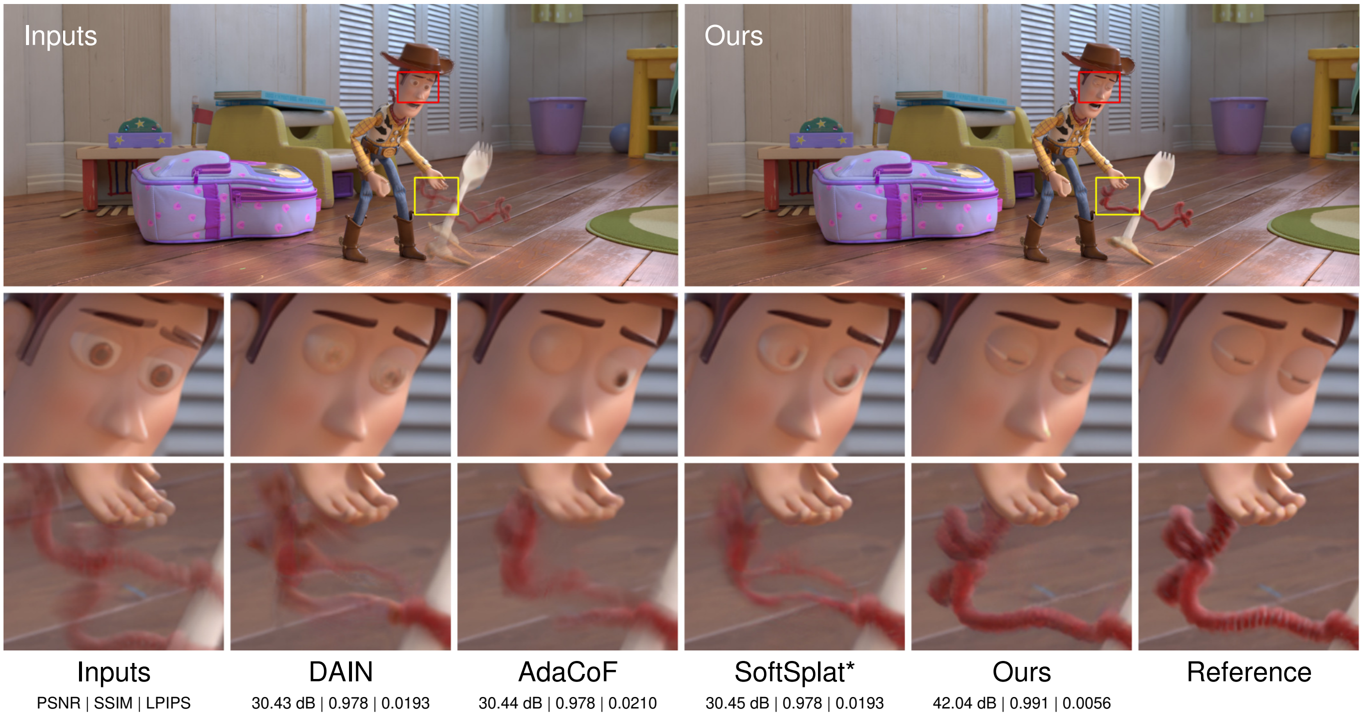

Fig. 9. Visual results on a production sequence. © 2021 Disney / Pixar

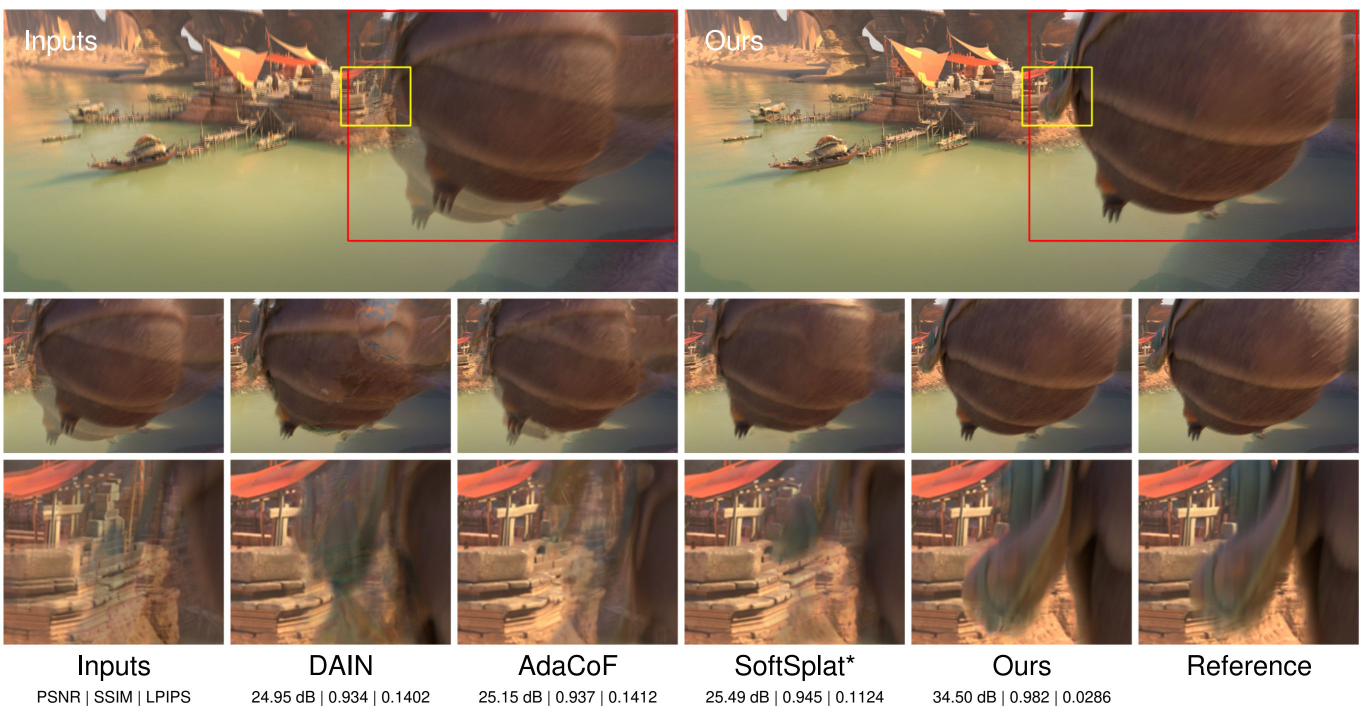

Fig. 10. Visual results on a production sequence. © 2021 Disney

Fig. 11. Visual results on a production sequence. © 2021 Disney

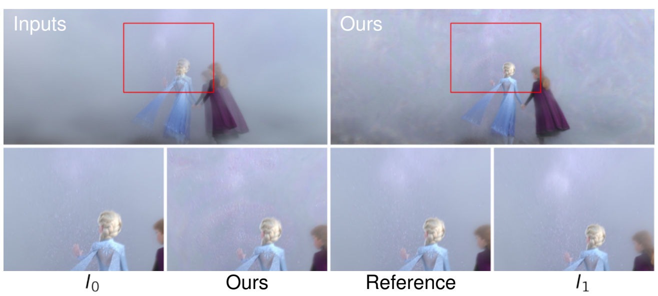

Fig. 12. Interpolation of a challenging sequence where the fog does not have meaningful auxiliary features. © 2021 Disney

Fig. 13. Interpolation of a sequence with severe motion blur. © 2021 Disney

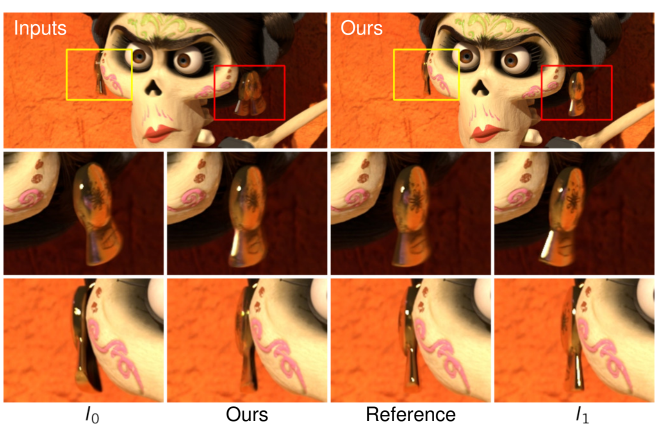

Fig. 14. Interpolation of a sequence with specular highlights. © 2021 Disney / Pixar

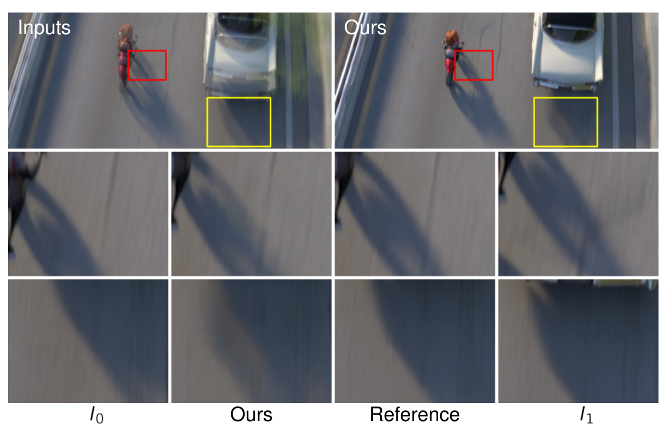

Fig. 15. Interpolation of hard shadows. When the auxiliary features do not contradict with the color, the method is capable of color-only motion compensation (middle row), otherwise a simple linear blend is used (bottom row). © 2021 Disney / Pixar

shown to perform very well on a wide variety of challenging shots, there is interesting potential for future improvements to optimize results in case of specific complex phenomena such as volumetric effects.

ACKNOWLEDGMENTS

The authors would like to thank Gerard Bahi, Markus Plack, Marios Papas, Gerhard Röthlin, Henrik Dahlberg, Simone Schaub-Meyer, and David Adler for their involvement in the project. Our method was trained and tested on production imagery but the results were not part of the released productions.

REFERENCES

Simon Baker, Daniel Scharstein, JP Lewis, Stefan Roth, Michael J Black, and Richard Szeliski. 2011. A database and evaluation methodology for optical flow. *International journal of computer vision* 92, 1 (2011), 1–31.

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*.

D. J. Butler, J. Wulff, G. B. Stanley, and M. J. Black. 2012. A naturalistic open source movie for optical flow evaluation. In *European Conf. on Computer Vision (ECCV) (Part IV, LNCS 7577)*, A. Fitzgibbon et al. (Eds.) (Ed.). Springer-Verlag, 611–625.

Zhixiang Chi, Rasoul Mohammadi Nasiri, Zheng Liu, Juwei Lu, Jin Tang, and Konstantinos N. Plataniotis. 2020. All at Once: Temporally Adaptive Multi-frame Interpolation with Advanced Motion Modeling. In *Computer Vision – ECCV 2020*, Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm (Eds.). Springer International Publishing, Cham, 107–123.

Myungsub Choi, Heewon Kim, Bohyung Han, Ning Xu, and Kyoung Mu Lee. 2020. Channel Attention Is All You Need for Video Frame Interpolation. In *AAAI*.

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

Alexey Dosovitskiy, Philipp Fischer, Eddy Ilg, Philip Häusser, Caner Hazirbas, Vladimir Golkov, Patrick van der Smagt, Daniel Cremers, and Thomas Brox. 2015. FlowNet: Learning Optical Flow with Convolutional Networks. In *2015 IEEE International Conference on Computer Vision (ICCV)*. 2758–2766. https://doi.org/10.1109/ICCV.2015.316

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).

Junhwa Hur and Stefan Roth. 2019. Iterative Residual Refinement for Joint Optical Flow and Occlusion Estimation. In *CVPR*.

Eddy Ilg, Nikolaus Mayer, Tonmoy Saikia, Margret Keuper, Alexey Dosovitskiy, and Thomas Brox. 2017. Flownet 2.0: Evolution of optical flow estimation with deep networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2462–2470.

Huaizu Jiang, Deqing Sun, Varun Jampani, Ming-Hsuan Yang, Erik Learned-Miller, and Jan Kautz. 2018. Super slomo: High quality estimation of multiple intermediate frames for video interpolation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 9000–9008.

Tarun Kalluri, Deepak Pathak, Manmohan Chandraker, and Du Tran. 2021. FLAVR: Flow-Agnostic Video Representations for Fast Frame Interpolation. (2021).

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

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)*.

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.

Yihao Liu, Liangbin Xie, Li Siyao, Wenxiu Sun, Yu Qiao, and Chao Dong. 2020. Enhanced quadratic video interpolation. In *European Conference on Computer Vision Workshops*.

Gucan Long, Laurent Kneip, Jose M Alvarez, Hongdong Li, Xiaohu Zhang, and Qifeng Yu. 2016. Learning image matching by simply watching video. In *European Conference on Computer Vision*. Springer, 434–450.

Simone Meyer, Abdelaziz Djelouah, Brian McWilliams, Alexander Sorkine-Hornung, Markus Gross, and Christopher Schroers. 2018. PhaseNet for Video Frame Interpolation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Simone Meyer, Oliver Wang, Henning Zimmer, Max Grosse, and Alexander Sorkine-Hornung. 2015. Phase-Based Frame Interpolation for Video. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 1410–1418. https://doi.org/10.1109/CVPR.2015.7298747

Simon Niklaus and Feng Liu. 2018. Context-Aware Synthesis for Video Frame Interpolation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

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

Simon Niklaus, Long Mai, and Feng Liu. 2017a. Video Frame Interpolation via Adaptive Convolution. In *IEEE Conference on Computer Vision and Pattern Recognition*.

Simon Niklaus, Long Mai, and Feng Liu. 2017b. Video Frame Interpolation via Adaptive Separable Convolution. In *IEEE International Conference on Computer Vision*.

Simon Niklaus, Long Mai, and Oliver Wang. 2021. Revisiting Adaptive Convolutions for Video Frame Interpolation. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*. 1099–1109.

N.Mayer, E.Ilg, P.Häusser, P.Fischer, D.Cremers, A.Dosovitskiy, and T.Brox. 2016. A Large Dataset to Train Convolutional Networks for Disparity, Optical Flow, and Scene Flow Estimation. In *IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*. http://lmb.informatik.uni-freiburg.de/Publications/2016/MIFDB16 arXiv:1512.02134.

Junheum Park, Keunsoo Ko, Chul Lee, and Chang-Su Kim. 2020. BMBC: Bilateral Motion Estimation with Bilateral Cost Volume for Video Interpolation. In *Computer Vision – ECCV 2020*, Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm (Eds.). Springer International Publishing, Cham, 109–125.

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]).

Deqing Sun, Xiaodong Yang, Ming-Yu Liu, and Jan Kautz. 2018. Pwc-net: Cnns for optical flow using pyramid, warping, and cost volume. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 8934–8943.

Zachary Teed and Jia Deng. 2020. RAFT: Recurrent All-Pairs Field Transforms for Optical Flow. In *Computer Vision - ECCV 2020 - 16th European Conference (Lecture Notes in Computer Science, Vol. 12347)*. Springer, 402–419. https://doi.org/10.1007/978-3-030-58536-5_24

Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. 2016. Instance normalization: The missing ingredient for fast stylization. *arXiv preprint arXiv:1607.08022* (2016).

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

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

Xiangyu Xu, Li Siyao, Wenxiu Sun, Qian Yin, and Ming-Hsuan Yang. 2019. Quadratic Video Interpolation. In *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. dÀlché-Buc, E. Fox, and R. Garnett (Eds.), Vol. 32. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2019/file/d045c59a90d7587d8d671b5f5aec4e7c-Paper.pdf

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.

Zheng Zeng, Shiqiu Liu, Jinglei Yang, Lu Wang, and Ling-Qi Yan. 2021. Temporally Reliable Motion Vectors for Real-time Ray Tracing. *Computer Graphics Forum* 40, 2 (2021), 79–90. https://doi.org/10.1111/cgf.142616

Haoxian Zhang, Yang Zhao, and Ronggang Wang. 2020. A Flexible Recurrent Residual Pyramid Network for Video Frame Interpolation. In *Computer Vision – ECCV 2020*, Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm (Eds.). Springer International Publishing, Cham, 474–491.

Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. 2018. The Unreasonable Effectiveness of Deep Features as a Perceptual Metric. In *CVPR*.

Henning Zimmer, Fabrice Rousselle, Wenzel Jakob, Oliver Wang, David Adler, Wojciech Jarosz, Olga Sorkine-Hornung, and Alexander Sorkine-Hornung. 2015. Path-space Motion Estimation and Decomposition for Robust Animation Filtering. *Computer Graphics Forum (Proceedings of EGSR)* 34, 4 (June 2015). https://doi.org/10/f7mb34