Frame Interpolation Transformer and Uncertainty Guidance

Markus Plack\textsuperscript{1}\textsuperscript{*}  
Matthias B. Hullin\textsuperscript{1}  
Karlis Martins Briedis\textsuperscript{2,3}  
Markus Gross\textsuperscript{2,3}  
Abdelaziz Djelouah\textsuperscript{3}  
Christopher Schroers\textsuperscript{3}  
\textsuperscript{1}University of Bonn  
Bonn, Germany  
\textsuperscript{2}Department of Computer Science  
ETH Zürich, Switzerland  
\textsuperscript{3}DisneyResearch|Studios  
Zürich, Switzerland  
mplack@cs.uni-bonn.de, karlis.briedis@inf.ethz.ch

\textsuperscript{*}Work done during an internship at DisneyResearch|Studios

Figure 1. Our method achieves state-of-the-art results for frame interpolation. It produces sharp textures as highlighted on both live action (left) and rendered (right \cite{15}) content. In addition to the interpolated frame, we estimate error maps that are helpful for quality checks in video production tools. More importantly, for rendered content it can be used to determine a subset of patches to render for the middle frame, which are then leveraged by our model to achieve production quality level results for a fraction of the rendering cost.

**Abstract**

Video frame interpolation has seen important progress in recent years, thanks to developments in several directions. Some works leverage better optical flow methods with improved splatting strategies or additional cues from depth, while others have investigated alternative approaches through direct predictions or transformers. Still, the problem remains unsolved in more challenging conditions such as complex lighting or large motion.

In this work, we are bridging the gap towards video production with a novel transformer-based interpolation network architecture capable of estimating the expected error together with the interpolated frame. This offers several advantages that are of key importance for frame interpolation usage: First, we obtained improved visual quality over several datasets. The improvement in terms of quality is also clearly demonstrated through a user study. Second, our method estimates error maps for the interpolated frame, which are essential for real-life applications on longer video sequences where problematic frames need to be flagged. Finally, for rendered content a partial rendering pass of the intermediate frame, guided by the predicted error, can be utilized during the interpolation to generate a new frame of superior quality. Through this error estimation, our method can produce even higher-quality intermediate frames using only a fraction of the time compared to a full rendering.
1. Introduction

Video frame interpolation (VFI) is a classical video processing problem where the aim is to restore an intermediate frame in a given video sequence. This temporal inbetweening enables many practical applications, such as video editing [38], novel-view synthesis [26], video retiming, and slow motion generation [25]. Recent advances in VFI methods [13,24,28,30,37,48,53,55] have been continuously improving the interpolation quality, but the problem remains open due to complex lighting effects and large motion that are ubiquitous in real-life videos and can introduce severe artifacts for the existing methods.

We propose a transformer-based VFI architecture that processes both source and target frames in a unified framework and compensates motion through a tightly integrated optical flow estimation and cross-backward warping. Our model improves over the current state-of-the-art as supported by our extensive quantitative experiments and a user study.

Besides the improvements in terms of results, our model also predicts the interpolation uncertainty similar to approaches for artifact detection [4,49] and adaptive sampling [29,60]. This is of key importance for usage in a production context, where working with long sequences requires a way to automatically identify problematic frames. Uncertainty estimation also benefits Computer Graphics (CG) applications, as we use it to determine which frame patches do not have sufficient quality and optionally mark them for rendering. Thanks to our novel transformer-based model, the rendered patches from the middle frame naturally fit in the same unified VFI framework, achieving high quality levels at the fraction of the cost of rendering the full middle frame. Our paradigm is more compatible with current production renderers than CG specialized VFI works [5,21,66] which require the generation of specific G-buffers for the keyframes and the intermediate frame.

In summary, our contributions are as follows.

- We introduce a novel motion-based VFI method, that treats input and target frames in the same manner through a transformer-based architecture using masks.
- Our model achieves state-of-the-art performance as shown both in quantitative experiments and a user study.
- We perform output’s uncertainty estimation subtask, which can be particularly beneficial for rendered content to achieve even better quality results.

2. Related work

While classical approaches to frame interpolation relied on optical flow and image warping [2,52,62], they have been surpassed by learning-based methods. We start our discussion with a short review of direct, phase and kernel based prediction methods, before going into more details with approaches using motion or transformers.

Direct methods were proposed using purely convolutional architectures [27,36] or combining channel attention with a deep residual network [13]. Alternatively, Meyer et al. [40] show a phase-based method based on the idea that phase-shifts can be used to represent motion, and later extended with a learning-based component [39].

Kernel-based methods, as originally introduced by Niklaus et al. [44], aim to predict kernels for all pixels that are applied in a convolutional layer. Offset prediction has been used [9,30] to reduce the necessary kernel size to handle large motion, making those methods conceptually more similar to motion-based ones. Various other extensions have been proposed, including prediction of separable kernels [45,46], time input for arbitrary frame interpolation [10], a multi-scale architecture including cost volumes [8], multi-stage networks [20], different backbones [16,54], and improving performance [50].

Most motion-based methods build on the work of optical flow estimation methods [18,57,61]. Some methods use the estimated motion between the input frames to forward splat them [23,42,43], while others aim to find the flow from the intermediate frame to the reference frames, allowing for an easy backward warping, either by estimating the flows directly [24,28,47,48,53], through other means [3,25,31,41,55], or combine both forward and backward warping approaches [17]. While most methods assume linear motion between the keyframes, others estimate non-linear motion by using more than two input frames [12,19,33,34,63] or with a learned prior [48].

Various other approaches have been proposed to improve estimation of large motion by treating small and large motion with equal priority [53], dynamically adapting the flow estimation to the motion magnitude and image resolution [55], or better strategies for feature propagation [1]. We adopt equal motion treatment by extending the scale-agnostic feature extraction [53,58]. Most recently, CG specific frame interpolation algorithms have been introduced for 2D animation [56] and 3D rendering [5].

Error estimation of the optical flow is used by Chi et al. [11] for specific treatment, proposing predefined fixed models for the various error levels. This is different from our method, that learns to predict perceptual and $L_2$-based error maps for final interpolation result.

With the introduction of the transformer [59] and its adaptation to vision tasks [22], several transformer-based frame interpolation approaches have been proposed. Liu et al. [35] use a transformer architecture that incorporates convolutions inside attention layers, but does not include any motion compensation. VFIformer [37] uses cross-scale
An overview of our method is given in Fig. 2. After extracting a feature pyramid \( \{F^l_t\}\) (Deep Feature Extraction) for each of the three frames (left) we pass a latent representation \(W_t\) along with a forward flow estimate \(F_t\) for each frame \(t\) through multiple levels of our reconstruction (center). At each level, after merging with the extracted features (Feature Merging), we update the latent representation using the initial flow estimate (Transformer Fusion I), followed by an update of the flow estimate and context vector from the new features (Flow/Context Residual) and another latent representation update using the new features and flows (Transformer Fusion II) before upsampling flow and features for the next level (Upsampling). Finally, we compute the interpolated Frame \(\hat{I}_1\) and an estimate of the error \(\hat{E}_1\) (top right).

3. Method

The goal of our method is to interpolate two keyframes \(I_0, I_2\) and find the intermediate frame \(\hat{I}_1\) along with an estimate of the error \(\hat{E}_1\). Subsequently, we analyze the error map and check if certain areas of the frame need to be rendered as we expect them to have insufficient quality. We then pass those additional masked inputs \(I_1\) to the network along with the keyframes to get a final interpolated frame. Note that our method is well equipped to handle the common problem of two-frame interpolation without any changes to the architecture or training and that the additional inputs are entirely optional, i.e., we simply set \(I_1 = 0\).

3.1. Interpolation network

Motivated by our goal to be able to handle arbitrary inputs, the overall architecture of our network is inspired by transformer architectures. This means that, opposed to common two-frame interpolation methods, there is little distinction within the network between the keyframes and the target frame. Instead, we equip each frame with a binary mask \(M_t\) indicating valid inputs to guide the interpolation. An overview of our method is given in Fig. 2.

We first extract a feature pyramid representation \(\{F^l_t\}_{l=0,...,6}\) for each of the inputs and process them in a coarse-to-fine manner with the same update blocks that share weights for the bottom 5 resolutions.

In each of the levels, we first merge the latent feature representations \(W_{l,t}\) with the respective input feature pyramid level. After that, they are updated in two transformer fusion blocks and a flow/context residual block in between that additionally updates the running flow estimates \(F_{l+1,t}\), denoting the optical flow from \(t\) to \(t+1\). Finally, the latent feature representations and flows are upsampled for processing in the next level.

In order to reduce the memory and compute costs, the processing of the topmost level is treated differently and consists of two convolutional layers.

Deep feature extraction. Our feature extraction is inspired by that of Reda et al. [32] to enable weight sharing on the lower levels of the reconstruction. We expand their idea by using a U-Net architecture instead of the original top-down approach. The reasoning behind this choice is that it more easily enables the network to capture semantically meaningful features on the upper levels of the pyramid without the need for many convolutional layers with large kernels or dilation.

First, we build image \(I^l_t\) and mask \(M^l_t\) pyramids, where image/mask \(l\) is downsampled by a factor of 2 to obtain level \(l+1\). We concatenate both and pass them through a U-Net as illustrated in Fig. 3, keeping the last three layers as features. Finally, we concatenate all input and feature tensors of the same spatial resolution to build input feature pyramids \(\{F^l_t\}_{l=0,...,6}\) for \(t \in \{0, 1, 2\}\). Note that all features from level two onward will be semantically similar.
merge the first \( \text{W} \) and thus we can use weight sharing for all following modules on those levels.

**Initialization and feature merging.** On the lowest level we initialize the optical flows \( F^0_{t,s} \) as 0 and set the latent feature representations \( \mathcal{W}^0_{t,s} \) to a learned vector that is spatially repeated.

As the first step on each level, the upsampled pixel-wise features of the previous level, or the initial values, \( \mathcal{W}^0_{t,s} \in \mathbb{R}^{D_i} \) are merged with their respective feature pyramid features \( F^i_t \in \mathbb{R}^{C_i} \), where \( \mathcal{C}_0 := 52, \mathcal{C}_1 := 148, \mathcal{C}_{i \in [2,6]} := 340, \) and \( D_i := \mathcal{C}_i + 15 \). Therefore, we only merge the first \( \mathcal{C}_i \) channels of \( \mathcal{W}^0_{t,s} \) with \( F^i_t \) while keeping the remaining 15 channels unaffected:

\[
\mathcal{W}^1_{t,s} = \left[ M^1_t F^i_t + (1 - M^1_t) \begin{bmatrix} \mathcal{W}^0_{t,s} \\ \mathcal{W}^0_{t,s} \end{bmatrix}_{0...\mathcal{C}_i-1} \right]
\]

(1)

The purpose of the directly passed through channels is similar to explicit occlusion maps employed by other methods, but we leave the choice on how to best use those additional channels to be learned by the network.

**Transformer fusion.** To update the latent feature representation of each frame \( t_0 \in \{0, 1, 2\} \), we use cross-backward warping to align the features of all other frames \( t_i \neq t_0 \) by rescaling the current flow estimate at stage \( s \) as

\[
\mathcal{W}^{t,s}_{t_i \to t_0}(x, y) = \mathcal{W}^{t,s}_{t_i}(t_0 - t_i) F^{t,s}_{t_i}(x, y)
\]

(2)

for spatial indices \((x, y)\) and using bilinear interpolation for non-integer coordinates. We treat \( \mathcal{W}^{t,s}_{t_0}(x, y) \), and \( \mathcal{W}^{t,s}_{t_2 \to t_0}(x, y) \) as tokens processed by the multhead attention module. Specifically, for each head \( i \) the per-pixel query, key and value tensors are computed as

\[
Q_i = \mathcal{W}^Q_{t_0} \mathcal{W}^{t,s}_{t_0}
\]

(3)

\[
K_i = \mathcal{W}^K_{t_0} \left[ \mathcal{W}^{t,s}_{t_1 \to t_0}, \mathcal{W}^{t,s}_{t_2 \to t_0} \right]
\]

(4)

\[
V_i = \mathcal{W}^V_{t_0} \left[ \mathcal{W}^{t,s}_{t_1 \to t_0}, \mathcal{W}^{t,s}_{t_2 \to t_0} \right]
\]

(5)

The softmax of the query/key multiplication and the residual update from the weighted sum of the values are computed as in the original transformer [59].

Since our latent feature representations have an inherent spatial structure, we opt to replace the linear layers of the standard transformer with convolutional residual layers. We use two convolutions with kernel size 3, a dropout layer before and after the second convolution and a GELU activation after the first. In addition, we use layer normalization after the multihead attention and the convolutional layers, as is common in transformer architectures. We dub those modules **multihead-attention convolutional encoders** (MACE) and stack two of them for all transformer fusion modules as shown in Fig. 4 except for the second module on the second layer, which uses four MACE modules.

**Flow residual.** Initial tests suggested that a transformer module, as used for the feature updates, is a poor choice for updating the current flow estimate. Instead, we use a convolutional module for this task. After cross-backward warping the updated features to the reference frame, we pass each pair \((\mathcal{W}^{t,s}_{t_1 \to t_0}, \mathcal{W}^{t,s}_{t_2 \to t_0})\) through a series of convolutions. The output contains the following tensors (stacked in channel dimension): Weight \( \alpha \), flow offset \( \Delta F \), and context residual \( \Delta^\alpha \) (We drop the level, time, and step indices of those...
for ease of notation). We apply softmax on the weights and update the flows and context features as

$$F_t^{l,3} = F_t^{l,2} + \frac{1}{\epsilon} \sum_v e^{\alpha_v} \Delta F_v$$

$$[W_t^{l,3}]_{C_1..D_l-1} = [W_t^{l,2}]_{C_1..D_l-1} + \frac{1}{\epsilon} \sum_v e^{\alpha_v} \Delta W_v.$$ (6)

Note how $\Delta F_v$ needs to be rescaled to a forward flow for the update of $F_t^{l,3}$.

**Miscellaneous.** For the upsampling of the flows we use parameter-free bilinear interpolation by a scaling factor of two (Denoted by $\uparrow 2$) as

$$F_t^{l,0} = 2F_t^{l+1,4} \uparrow 2.$$ (8)

The feature maps are passed through a resize convolution same as [53] to avoid checkerboard artifacts, i.e. a nearest-neighbor upsampling followed by a convolutional layer with kernel size 2 and $D_l$ output feature channels.

For the final output, we pass the latent representations $W_t^0$ together with the extracted features $F_t^0$ through two convolutional layers with kernel sizes 3 and 1 respectively. The final output has five channels of which the first three form the color image $\hat{I}_t$ and the others correspond to the color error $\hat{E}_c^t$ and the perceptual error $\hat{E}_p^t$.

### 3.2. Uncertainty estimation

To train the error outputs $\hat{E}$ of the network we compute the target error maps as follows. Let $I_t^{GT}$ be the ground truth frame at time $t$. We compute the error targets or ‘ground truth’ as

$$E_t^c = \|I_t^{GT} - \hat{I}_t\|_2$$ (9)

where $\|\cdot\|_2$ denotes the $L2$ norm along the channel dimension. The perceptual error $E_t^p$ follows the computation of LPIPS [65] without the spatial averaging. In order to prevent a detrimental influence of the error loss computations, we do not propagate gradients from the error map computations to the color output and only allow gradient flow to the error prediction of the network.

We want to use the error estimates $\hat{E}$ to find regions of the target frame that are expected to have insufficient quality, so we can render those areas and pass them to the network in a second pass to improve the quality. Assuming that most common renderers should be able to operate on a subset of rectangular tiles without a significant overhead, we average the error estimates for those tiles for which we chose a size of $16 \times 16$ pixels. Given a fixed budget for each frame, we simply select the tiles with the highest expected error and use them in the second interpolation pass.

### 3.3. Implementation and training

We follow common practice and train our network on triplets from the training set of Vimeo-90K [64]. Of the 51313 triplets of resolution $448 \times 256$ we set aside 802 for validation. For data augmentation we randomly crop windows of size 256, apply random spatial and temporal flipping and rotations in multiples of $90^\circ$. We use empty mid-
dle frames for 50% of the training samples (i.e. $I_1 = 0$) and otherwise retain between $\frac{1}{64}$ and $\frac{1}{4}$ of 16 x 16 tiles as additional input (random at first and based on the predicted error for fine-tuning).

We train our $L_1$ variant for 2.1M iterations with batch size 4 using the Adam optimizer and $L_1$ loss for the color output with weight 1.0 and for both error estimates with weight 0.01 each. We start with a learning rate of $5 \times 10^{-5}$ and reduce it every 0.75M iterations by a factor of 0.464.

For our perceptual variant ($L_S$), we follow the same schedule, but add VGG and Style loss from [53] after 1.9M iterations, at which point we set the weights of the color, VGG and style loss as 10.0, 0.25 and 40. All losses are computed only for the center frame outputs, as we assume the keyframes are given and complete.

4. Experiments

We evaluate the performance of our method on the standard interpolation task (Sec. 4.1) and the efficiency of the uncertainty guidance (Sec. 4.2). We close with an ablation study (Sec. 4.3) and a discussion of limitations (Sec. 4.4).

Metrics. We measure our results using the common evaluation metrics peak signal-to-noise ratio (PSNR), structural similarity (SSIM) and the perceptual LPIPS [65]. In addition, we perform a user study for a qualitative evaluation.

Methods. We compare our method against ABME [48], AdaCoF [30], CAIN [13], FILM ($L_1$ and $L_S$) [53], IFRNet (Large) [28], RIFE [24], VFiFormer [37], and XVFI [55].

Datasets. For the evaluation on traditional frame interpolation we use Vimeo90K [64], DAVIS [51], and SNU-FILM [13]. In addition, we evaluate on samples taken from the publicly available animated short films Big Buck Bunny [14], Cosmos Laundromat [7], Elephants Dream [6], and Sintel [15]. See supplementary material for more details and instructions to reproduce those datasets.

4.1. Traditional frame interpolation

We quantitatively evaluate our method on common datasets in Tab. 1 against the state of the art. Our $L_1$ variant shows the best PSNR and SSIM performance on all difficulty levels of SNU-FILM with a PSNR improvement of up to 0.21 dB in the hard category and a competitive performance on Vimeo90k and DAVIS. Our $L_S$ version outperforms all others in terms of LPIPS on all datasets except DAVIS and demonstrates excellent PSNR and SSIM scores within its category. We show the performance on the animated short films in Tab. 2 where each variant outperforms all others within its category with respect to all metrics and on all datasets except Cosmos Laundromat, where both nevertheless yield good results.

![Figure 6. User study on the animated short film datasets. On average, users had a normal/strong preference for our method for 48/34% of all votes. For each of the short films, we use a representative subset of 30 samples and collected a total of 3158 AB comparisons from 69 participants, most of whom are computer graphics/vision students and graduates.](image6)

![Figure 7. The closing of the eyes proves difficult to interpolate, but the expected perceptual error $\hat{E}_1^p$ closely matches the true error $E_1^p$. Passing the part of the middle frame indicated by the white box to the network we get a significantly improved interpolation. Numbers below are PSNR/LPIPS. Sample is from [15].](image7)

![Figure 8. Replacement of tiles based on random sampling, highest ground truth error, i.e. the upper boundary of achievable PSNR, and our color error estimation $\hat{E}_1^c$.](image8)
To further support our claim that our method performs well in terms of visual quality, we conduct an extensive user study. We roughly follow the approach of [42] and asked users to compare methods side by side, but included an option for a strong preference. We show one sample of each film in Fig. 5 and give the results in Fig. 6. We refer to the supplementary material for more details and results.

### 4.2. Uncertainty guided interpolation

We will demonstrate the advantages of our uncertainty guidance in two experiments by analyzing the ability of our error prediction to select appropriate patches in the interpolated image first, and secondly showing the quality improvement by passing additional patches to the network.

In Fig. 8 we demonstrate the PSNR improvement when we use our error estimation to replace a fraction of 16×16 tiles of the interpolated output by the corresponding ground truth. For comparison, we show the effect of random replacement as a baseline and a replacement of the tiles with the highest measured error as the optimal strategy. Replacing a quarter of the tiles, we achieve a PSNR improvement between 6.99 and 9.98 dB, whereas random replacement yields at most 1.27 dB.

Next we want to study the effect of additional inputs on the network output in separation from the error prediction. Therefore, we select tiles based on the true error and pass them into the network. We also compute the metrics when simply replacing the tiles in the interpolated output for our own method as a baseline and a selection of others for comparison. We plot the results in Fig. 9 which show that the perceptual quality is improved beyond the baseline approach.

We give a visual example of the full uncertainty guidance approach in Fig. 7, which shows how the correct region with high error is identified and the interpolation is improved by the additional inputs and refer to the supplementary material for additional results.
Figure 9. We show that the perceptual quality of the interpolation achieved by passing additional inputs to our method is better than the baseline approach of replacing the worst patches of the interpolation based on color error. For reference, we also show the curves when replacing the outputs of FILM and IFRNet, the two follow up methods in terms of perceptual performance.

4.3. Ablation study

For an ablation study, we train different versions of our network to show the effect of the error estimation, the deep feature extraction and the shared frame processing. We use the same training procedure and color based loss for all variants as described in Sec. 3.3. The variants without error estimation differ only in the last convolutional layer (3 instead of 5 outputs) and do not use the error losses. The deep feature representation is replaced by the feature representation proposed by Reda et al. [53] and versions without shared frame processing only update the center frame in the transformer fusion and flow/context residual modules. The results are presented in Tab. 3 and highlight the advantages of the deep feature extraction and the shared frame processing for the interpolation quality.

4.4. Limitations

Very large motion or drastic visual changes can be missed by the error prediction and are hence not recovered through a second rendering pass. We show an example of this in the supplementary material. While the shared frame processing of the network through its transformer architecture should in theory be capable of recognizing missing objects that are unlikely to be occluded, we surmise that the current training dataset lacks sufficient examples to learn such behavior.

Lastly, the current network is relatively slow and big. E.g. VFIformer is on average 44.2% faster on Vimeo90k and needs about 27.6% fewer parameters. This makes training with more than two input frames challenging, even though the architecture supports it without any changes. We hope to improve this in the future, which could allow for better results through e.g. nonlinear flow estimates, or enable using our proposed architecture for other video processing tasks such as deblurring and super-resolution.

5. Conclusion

In this work, we proposed a VFI method that incorporates optical flow motion compensation, deep feature extraction, error estimation, and shared frame processing in a transformer-based architecture. This enables our novel uncertainty-guided approach for animated content production, which can be used to greatly reduce the cost of rendering while maintaining a high visual quality as we have shown in our experiments. At the same time, our method achieves state-of-the-art results for traditional frame interpolation as demonstrated on multiple common benchmarks, and a superior visual quality confirmed by an extensive user study. Since our training procedure using masked inputs is similar to those of masked language models, a study of its properties remains an interesting direction for future work.

Acknowledgements. Markus Plack and Matthias B. Hullin are supported by the European Research Council under ERC Starting Grant “ECHO” (802192).
References


[6] (c) copyright 2006, Blender Foundation / Netherlands Media Art Institute / www.elephantsdream.org. Elephants dream, 2006. Licensed under Creative Commons Attribution 2.5 (https://creativecommons.org/licenses/by/2.5/).


[14] Copyright (C) 2008 Blender Foundation — peach.blender.org. Big buck bunny, 2008. Licensed under Creative Commons Attribution 3.0 (http://creativecommons.org/licenses/by/3.0/).

[15] Copyright (c) Blender Foundation — durian.blender.org. Sintel, 2010. Licensed under Creative Commons Attribution 3.0 (http://creativecommons.org/licenses/by/3.0/).


[26] Nima Khademi Kalantari, Ting-Chun Wang, and Ravi Ramamoorthi. Learning-based view synthesis for light field
cameras. *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia 2016)*, 35(6), 2016. 2


[53] Frisun Reda, Janne Kontkanen, Eric Tabellion, Deqing Sun, Caroline Pantofaru, and Brian Curless. FILM: Frame interpolation for large motion. In *European Conference on Computer Vision*, 2022. 2, 3, 5, 6, 8


