In this supplementary, we will expand more details that are not included in the main text due to the page limitation.

S1. Algorithm Details

We introduce 3 algorithms in the main paper. In this section, we supplement the algorithm details of the confidence map back-propagation, margin fusion approach and multi-frame fusion strategy in the main paper.

Confidence map back-propagation. Algorithm S1 summarizes the strategy of confidence map back-propagation in the main paper Section 4.1. The parameters in Algorithm S1 are set to $k = 5$, $d = 10$, and $\delta_C = 0.5$.

**Algorithm S1** Back-propagation for aggregated confidence map  

**Input:** $Y$: optical flow; $C$: confidence map; $\delta_C$: threshold for confidence map; $d$: sampling interval; $k$: index of the first frame;  

**Output:** $M$: updated mask containing aggregated confidence map;  

1. set $m_{pre} = 1(C_{k+(n-1)d} - \delta_C) \in C$, put $m_{pre}$ into $M$;  
2. for $i = n - 1$; $i >= 0$; $i -= 1$ do  
3. the optical flow field $Y_{warp} = Y_{k+id}$;  
4. using $Y_{warp}$ to warp $m_{pre}$ to $m_{new} = Y_{warp}(m_{pre})$;  
5. the binarized confidence map $m_{new} = 1(C_{k+id} - \delta_C)$, where $M_{k+id} \in C$;  
6. the final mask field $M_{k+id} = m_{pre} \& m_{new}$  
7. put $M_{k+id}$ into $M$;  
8. $m_{pre} = M_{k+id}$  
9. end for

Margin fusion. The complete pipeline of the margin fusion approach is shown in Fig. S1. At first, we coarsely align the reference frame $I^s$ and the target frame $I^t$. We then crop $I^{warp}$ and $I^t$ to $I^s$ and $I^t$, and re-align them by the optical flow outpainting. Per Algorithm S2, we further calculate the mask $M_I$, which indicates the chosen regions of $I^t$. The final result $I_{result}$ is obtained by combining $I^t$, $M_I$, and $I^{warp}$. The parameters in the Algorithm S2 are set to $\delta_D = 0.2$, $\eta_t = 20$, and $k_{lin} = 11$.

Multi-frame fusion. To adaptively determine which frame and which region should be selected, the multi-frame fusion strategy is illustrated in Algorithm S3. The parameters in the Algorithm S3 are set to $\eta_u = 25k$, $\eta_r = 1.2$, and $\eta_s = 2k$.

S2. Synthetic Dataset for Training

We proposed a model-based synthetic dataset in this paper. The settings of the homography parameters are as follows: The maximum rotation angle $\theta$ is set to $10^\circ$. The range of scaling $s$ is set to $0.7 \sim 1.3$. The maximum translations $(d_x, d_y)$ in the $x$ and $y$ directions are $100$ and $70$, respectively. The maximum perspective factors in the $x$ direction and in the $y$ direction are $0.1$ and $0.15$.

For different training requirements, we apply various combinations of synthetic dataset, as shown in Fig. S2 (more visualizations can be found in the Supplementary Video). For camera pose regression, we use the large FOV video pair of stable and unstable. For training the flow smoothing network, we alternatively adopt small FOV video pairs, which simulate coarsely stabilized video. Aiming at the flow outpainting network, we take small-FOV stable videos for training and large-FOV for ground-truth supervising.

Data for Camera Pose Regression. For training the camera pose regression network, we need to generate unstable videos. For every frame, a random homography matrix produces an unstable frame. In practice, the perspective ef-
Figure S1: **Pipeline of margin fusion approach.** Given the target frame $I^t$, the reference frame $I^s$ is coarsely aligned to $I^warp$ by the predicted large-FOV flow field $Y_{large}$. Then, $I^t$ and $I^warp$ are cropped and re-aligned. Per Algorithm S2, the deduced mask $M_I^t$ is fused with $I^t$ and $I^warp$ to obtain the resulting frame.

Figure S2: **Visualization of our model-based synthetic dataset.** We designed different combinations of dataset for varying tasks.

Effects in the $x$ direction and the $y$ direction are restricted to $10^{-5} \sim 50^{-5}$. The pose between two unstable frames is parameterized by rotation, scaling, and translation.

**Data for Flow smoothing.** For training the flow smoothing network, we need to generate unstable videos with small FOV. Specifically, for the stable video, we randomly gener-
Algorithm S2 Outpainting mask Algorithm

Input: $I_t$: target frame; $I_c$: cropped target frame; $I_c^*$: cropped source frame; $I_c^{\text{warp}}$: warped frame of $I_c^*$; $M_t$: valid mask of $I_t$; $M_c$: valid mask of $I_c$; $M_c^*$: valid mask of $I_c^*$; $M_c^{\text{warp}}$: valid mask of $I_c^{\text{warp}}$;

Output: $M_{\text{label}}$: unchanged mask of $I_t$;

1: extract feature maps with VGG-16 network $f_c = VGG(I_c)$, $f_{\text{warp}} = VGG(I_c^{\text{warp}})$;
2: calculate the Euclidean distance in feature space $D = ||f_c^* - f_{\text{warp}}||_2$;
3: $M_D = D < \delta_D$;
4: labeled region $M_{\text{label}}$;
5: for $i, j = 0; i < h, j < w; i + +, j + +$ do
6: if $M_t[i, j]$ then $M_{\text{label}}[i, j] = 1$;
7: else if $M^t[i, j] & (M^c[i, j])$ then $M_{\text{label}}[i, j] = 2$;
8: else if $M^t[i, j] & M^c[i, j] & M_D[i, j]$ then $M_{\text{label}}[i, j] = 0$;
9: else $M_{\text{label}}[i, j] = -1$;
10: end if
11: end for
12: $t_{in} = M_{\text{label}}, t_{out} = 0$, flag = True;
13: while sum ($t_{in} - t_{out}$) > $\eta$ do
14: if flag then
15: $t_{in} = t_{out}$, flag = False;
16: end if
17: inflate $t_{in}$ with kernel size $k_{t_{in}}$, obtain $t_{out} = \text{inflate}(t_{in})$;
18: $t_{out}[M_{\text{label}} == 1] = 1$;
19: $t_{out}[M_{\text{label}} == -1] = -1$;
20: end while
21: $M_{\text{label}} = (t_{out} == 2)$

$\mathbb{R}^{b \times 3 \times h \times w}$ with several 2D convolutional layers, where $b$ indicates the batch dimension and $h \times w$ indicates the spatial dimensions. The final predicted parameters are obtained by a series of 1D convolutional layers. We use a batch size of 40 and train for 10k iterations. We use Adam optimizer [3] with a constant leaning rate of $10^{-4}$ for the first 4k iterations, followed by an exponential decay of 0.99995 until iteration 10k. The input resolution is set to 256 x 512. The weights in training loss Eq. (5) and Eq. (7) in the main paper are set to $\lambda_{\theta} = 1.0, \lambda_s = 1.0, \lambda_r = 1.5, \lambda_{\text{grid}} = 2.0$ for the first 6k iterations and $\lambda_{\theta} = 2.0, \lambda_s = 8.0, \lambda_r = 1.0, \lambda_{\text{grid}} = 2.0$ for the remaining 4k iterations.

Optical flow smoothing network. We use a batch size of 6 and train for 20k iterations. We use Adam optimizer [3] with a constant leaning rate of $10^{-4}$ for the first 10k iterations, followed by an exponential decay of 0.99995 until iteration 20k. The input resolution is set to 488 x 768.

Flow outpainting network. We apply an Unet architecture with gated convolution layers [7] as a flow-outpainting network. We use a batch size of 12 and train for 20k iterations. We use the Adam optimizer [3] with a constant leaning rate of $10^{-4}$. The input resolution is set to 488 x 768. The weights in training loss Eq. (14) in the main article are set to $\lambda_{\text{in}} = 2.0, \lambda_{\text{out}} = 1.0, \lambda_F = 10.0$ for the first 10k iterations and $\lambda_{\text{in}} = 0.6, \lambda_{\text{out}} = 1.0, \lambda_F = 0.0$ for the remaining 10k iterations.

S3. Implementation Details

We will illustrate the training details of different networks, including the camera pose regression network, the optical flow smoothing network, and the flow outpainting network. All networks are implemented using Pytorch.

Camera pose regression network. We first describe the architecture of the camera pose regression network. The network processes each input concatenated tensor $f_{in}$...
S4. Qualitative Evaluation

We show the results of the comparison of our method and the latest approaches in Fig. S3. Most methods [2, 4, 6, 9] suffer from a large amount of cropping, as indicated by the green checkerboard regions. Compared to full frame rendering approaches for interpolation [1] / generation [5], our method shows fewer visual artifacts. In particular, FuSta [5] would discard most of the input frame content for stabilization and deblurring, while we argue that video stabilization is based on destroying as little of the input frame content as possible. Thus, our method preserves the original content of the input frame as much as possible. We strongly recommend that the reviewers see our additional supplementary video, especially the comparison with other full-frame approaches (FuSta [5], DIFRINT [1]).

S5. More Experimental Results

Per-category Evaluation. We present the the average scores for the 6 categories in the NUS dataset [4].

Two-stage Stabilization. To illustrate our two-stage stabilization method, we conduct an interesting experiment. We tracked the position (x, y) of a fixed keypoint in 10 frames, where every two frames were spaced 5 frames apart. As shown in Fig. S5, the trajectory of the shaky keypoint converges to a fixed/stable position through two-stage stabilization.

Trade-off Evaluation. We have conducted the experiment to illustrate the trade-off between runtime speed and performance by varying the number of iterations in the probabilistic stabilization network and the number of input frames in the video outpainting network (as shown in the Table S1). Notably, our default configuration, consisting of 3 iterations and 7 input frames, was determined as the optimal balance between runtime speed and performance.

Table S1: The trade-off experiment between runtime speed and performance.

<table>
<thead>
<tr>
<th>Iter.</th>
<th>Time</th>
<th>S.↑</th>
<th>Frames</th>
<th>Time</th>
<th>D.↑</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>69ms</td>
<td>0.80</td>
<td>3</td>
<td>57ms</td>
<td>0.83</td>
</tr>
<tr>
<td>2</td>
<td>84ms</td>
<td>0.84</td>
<td>5</td>
<td>78ms</td>
<td>0.88</td>
</tr>
<tr>
<td>3</td>
<td>97ms</td>
<td>0.86</td>
<td>7</td>
<td>97ms</td>
<td>0.91</td>
</tr>
<tr>
<td>4</td>
<td>111ms</td>
<td>0.86</td>
<td>11</td>
<td>132ms</td>
<td>0.90</td>
</tr>
</tbody>
</table>

Analysis of Runtime. We attribute the faster runtime of our approach against FuSta to the following three reasons: i) The traditional pose regression algorithm used in FuSta is 10 times slower than our proposed pose regression network (see Section 6.4); ii) Our method only requires computing optical flow once per frame, while FuSta requires computing it three times and relies on additional task-specific optimization and manual adjustments (see Section 6.4); iii) In the rendering stage, FuSta takes input from 11 RGB frames and their corresponding optical flow, whereas our approach only requires 7 frames. We will highlight these reasons in the final version of the manuscript.

S6. Network Architectures

Camera pose regression network. We first describe the architecture of the camera pose regression network. Given a concatenated input tensor \( f_{in} \in \mathbb{R}^{3 \times H \times W} \), we process it with multiple down-sampled convolution layers and flatten the output feature map to \( f_{out} \in \mathbb{R}^{d \times \frac{HW}{s^2}} \), where \( d, D \) denotes the dimension of the feature channel and the spatial down-sampling ratio, respectively. The feature vector \( f_{sum} \), obtained by weighting the sum of \( f_{out} \) along the feature channel, regresses all parameters of the affine transformation, given by

\[
\begin{align*}
    w &= \psi(f_{out}), \ f_{sum} = \sum_{i=0}^{\frac{HW}{s^2}} w_i f_{out}(i, \cdot), \ \{\theta, s, d_x, d_y\} = \hat{\Theta}(f_{sum}).
\end{align*}
\]  

(S1)

Specifically, The network processes each input concatenated tensor \( f_{in} \in \mathbb{R}^{b\times3\timesH\timesW} \) with several 2D convolutional layers, as shown in Table S2, where \( b \) indicates the batch dimension and \( h \times w \) indicate the spatial dimensions. The final predicted parameters are obtained by a series of 1D convolutional layers.

Table S2: Modular architecture of camera pose regression network modules. Each convolution operator is followed by batch normalization and LeakyReLU (negative_slope=0.1), except for the last one. \( K \) refers to the kernel size, \( s \) denotes the stride, and \( p \) indicates the padding. We apply the Max-pooling layer to downsample each feature map.

<table>
<thead>
<tr>
<th>Input Size</th>
<th>Convolution Layer ((K \times K, s, p))</th>
<th>Output Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feature map extraction</td>
<td></td>
<td></td>
</tr>
<tr>
<td>input: ( \times 3 \times h \times w )</td>
<td>conv0: ((3 \times 3, 1, 1))</td>
<td>( \times 8 \times h \times w )</td>
</tr>
<tr>
<td>conv0: ( \times 8 \times h \times w )</td>
<td>conv1: ((3 \times 3, 1, 1))</td>
<td>( \times 32 \times h \times w )</td>
</tr>
<tr>
<td>conv1: ( \times 32 \times h \times w )</td>
<td>pool1: ((5 \times 5, 2, 4))</td>
<td>( \times 32 \times h \times \frac{w}{4} )</td>
</tr>
<tr>
<td>pool1: ( \times 32 \times h \times \frac{w}{4} )</td>
<td>conv2: ((3 \times 3, 1, 1))</td>
<td>( \times 64 \times h \times \frac{w}{4} )</td>
</tr>
<tr>
<td>conv2: ( \times 64 \times h \times \frac{w}{4} )</td>
<td>pool2: ((5 \times 5, 2, 4))</td>
<td>( \times 64 \times h \times \frac{w}{16} )</td>
</tr>
<tr>
<td>pool2: ( \times 64 \times h \times \frac{w}{16} )</td>
<td>conv3: ((3 \times 3, 1, 1))</td>
<td>( \times 64 \times h \times \frac{w}{16} )</td>
</tr>
<tr>
<td>Camera pose regression</td>
<td></td>
<td></td>
</tr>
<tr>
<td>input: ( \times 64 \times 1 )</td>
<td>conv1: ((1, 1, 0))</td>
<td>( \times 32 \times 1 )</td>
</tr>
<tr>
<td>conv1: ( \times 32 \times 1 )</td>
<td>conv2: ((1, 1, 0))</td>
<td>( \times 16 \times 1 )</td>
</tr>
<tr>
<td>conv2: ( \times 16 \times 1 )</td>
<td>conv3: ((1, 1, 0))</td>
<td>( \times 4 \times 1 )</td>
</tr>
</tbody>
</table>
Figure S3: Visual comparison to state-of-the-art methods. Our proposed method does not suffer from aggressive cropping of frame borders [2, 4, 6, 9] and rendering artifacts than DIFRINT [1] and FuSta [5]. Specially, we keep more of the content in the input frames than FuSta [5].

Flow outpainting network. We apply a Unet architecture with gated convolution layers [7] as a flow outpainting network, as shown in Table S3.

S7. Limitations

Although our method achieves a comparable stability score, we use only a simple Gaussian sliding window filter
Figure S4: **Per-category quantitative evaluation on NUS dataset.** We compare the cropping ratio, distortion value, and stability score with state-of-the-art methods [2, 4, 6, 8, 9, 5, 1].

Figure S5: **Illustration of our iterative optimization-based stabilization algorithm.**

Table S3: Architecture of the flow-outpainting network. Each 2D gated-convolution [7] (‘G_conv’) is followed by batch normalization and Sigmoid. The final ‘conv’ denotes the 2D convolution layer without batch normalization and Sigmoid. $K$ refers to the kernel size, $s$ denotes the stride, and $p$ indicates the padding. We apply the Maxpooling Layer for downsampling (‘down’) and bilinear interpolation for upsampling (‘up’).

<table>
<thead>
<tr>
<th>Input Size</th>
<th>Convolution Layer ($K \times K, s, p$)</th>
<th>Output Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>input: $b \times 3 \times h \times w$</td>
<td>down_0</td>
<td>$b \times 3 \times \frac{h}{4} \times \frac{w}{4}$</td>
</tr>
<tr>
<td>$G_{conv0}$: $b \times 16 \times \frac{h}{4} \times \frac{w}{4}$</td>
<td>down_1</td>
<td>$b \times 16 \times \frac{h}{8} \times \frac{w}{8}$</td>
</tr>
<tr>
<td>$G_{conv1}$: $b \times 64 \times \frac{h}{16} \times \frac{w}{16}$</td>
<td>down_2</td>
<td>$b \times 64 \times \frac{h}{32} \times \frac{w}{32}$</td>
</tr>
<tr>
<td>$G_{conv2}$: $b \times 64 \times \frac{h}{32} \times \frac{w}{32}$</td>
<td>$G_{conv3}$: $(3 \times 3, 1, 1)$</td>
<td>$b \times 64 \times \frac{h}{64} \times \frac{w}{64}$</td>
</tr>
<tr>
<td>$G_{conv4}$: $b \times 3 \times \frac{h}{64} \times \frac{w}{64}$</td>
<td>$G_{conv5}$: $(3 \times 3, 1, 1)$</td>
<td>$b \times 32 \times \frac{h}{128} \times \frac{w}{128}$</td>
</tr>
<tr>
<td>$G_{conv6}$: $b \times 2 \times \frac{h}{128} \times \frac{w}{128}$</td>
<td>$up_0$</td>
<td>$b \times 32 \times \frac{h}{256} \times \frac{w}{256}$</td>
</tr>
<tr>
<td>$G_{conv7}$: $b \times 16 \times \frac{h}{256} \times \frac{w}{256}$</td>
<td>$up_1$</td>
<td>$b \times 16 \times \frac{h}{512} \times \frac{w}{512}$</td>
</tr>
<tr>
<td>$G_{conv8}$: $b \times 3 \times \frac{h}{512} \times \frac{w}{512}$</td>
<td>$up_2$</td>
<td>$b \times 2 \times \frac{h}{1024} \times \frac{w}{1024}$</td>
</tr>
</tbody>
</table>

References


