Spatiotemporal Video Upscaling using Motion-Assisted Steering Kernel (MASK) Regression

Hiroyuki Takeda, Peter van Beek, and Peyman Milanfar

Abstract In this chapter, we present *Motion Assisted Steering Kernel* (MASK) regression, a novel multi-frame approach for interpolating video data spatially, temporally, or spatiotemporally, and for video noise reduction, including compression artifact removal. The MASK method takes both local spatial orientations and local motion vectors into account and adaptively constructs a suitable filter at every position of interest. Moreover, we present a practical algorithm based on MASK that is both robust and computationally efficient. In order to reduce the computational and memory requirements, we process each frame in a block-by-block manner, utilizing a block-based motion model. Instead of estimating the local dominant orientation by singular value decomposition, we estimate the orientations based on a technique similar to vector quantization. We develop a technique to locally adapt the regression order, which allows enhancing the denoising effect in flat areas, while effectively preserving major edges and detail in texture areas. Comparisons between MASK and other state-of-the-art video upscaling methods demonstrate the effectiveness of our approach.

Hiroyuki Takeda
Electrical Engineering Dept., University of California, 1156 High St. Santa Cruz, CA, 95064,
e-mail: htakeda@soe.ucsc.edu

Peter van Beek
Sharp Laboratories of America, 5750 NW Pacific Rim Blvd., Camas, WA, 98607
e-mail: pvanbeek@sharplabs.com

Peyman Milanfar
Electrical Engineering Dept., University of California, 1156 High St. Santa Cruz, CA, 95064,
e-mail: milanfar@soe.ucsc.edu

1 Introduction

Advances in video display technology have increased the need for high-quality and robust video interpolation and artifact removal methods. In particular, LCD flat-panel displays are currently being developed with very high spatial resolution and very high frame rates. For example, so-called "4K" resolution panels are capable of displaying 2160×4096 full color pixels. Also, LCD panels with frame rates of $120[\mathtt{Hz}]$ and $240[\mathtt{Hz}]$ are becoming available. Such displays may exceed the highest spatial resolution and frame rate of video content commonly available, namely $1080 \times 1920, 60[\mathtt{Hz}]$ progression High Definition (HD) video, in consumer applications such as HD broadcast TV and Blu-ray Disc. In such (and other) applications, the goal for spatial and temporal video interpolation reconstruction is to enhance the resolution of the input video in a manner that is visually pleasing and artifact-free. Common visual artifacts that may occur in spatial and temporal interpolation are: edge jaggedness, ringing, blurring of edges and texture detail, as well as motion blur and judder. In addition, the input video usually contains noise and other artifacts, e.g. caused by compression. Due to increasing sizes of modern video displays, as well as incorporation of new display technologies (e.g. higher brightness, wider color gamut), artifacts in the input video and those introduced by scaling are amplified, and become more visible than with past display technologies. High quality video upscaling requires resolution enhancement and sharpness enhancement as well as noise and compression artifact reduction.

A common approach for spatial image and video upscaling is to use linear filters with compact support, such as from the family of cubic filters [12]. In this chapter, our focus is on multi-frame methods, which enable resolution enhancement in spatial upscaling, and allow temporal frame interpolation (frame rate upconversion). Although many algorithms have been proposed for image and video interpolation, spatial upscaling and frame interpolation (temporal upscaling) are generally treated separately. The conventional super-resolution technique for spatial upscaling consists of image reconstruction from irregularly sampled pixels, provided by registering multiple low resolution frames onto a high resolution grid using motion estimation, see [16, 4] for overviews. A recent work by Narayanan *et al.* ([14]) proposed a video-to-video super resolution algorithm using a partition filtering technique, in which local image structures are classified into vertical, horizontal, and diagonal edges, textures, and flat areas by vector quantization [6] (involving off-line learning), and prepare a suitable filter for each structure class beforehand. Then, with the partition filter, they interpolate the missing pixels and recover a high resolution video frame. Another recent approach in [8] uses an adaptive Wiener filter and has a low computational complexity when using a global translational motion model. This is typical for many conventional super-resolution methods, which as a result often don't consider more complex motion.

For temporal upscaling, a technique called *motion compensated frame interpolation* is popular. In [5], Fujiwara *et al.* extract motion vectors from a compressed video stream for motion compensation. However, these motion vectors are often unreliable; hence they refine the motion vectors by the block matching approach with variable-size blocks. Similar to Fujiwara's work, in [9], Huang *et al.* proposed another refinement approach for motion vectors. Using the motion reliability computed from prediction errors of neighboring frames, they smooth the motion vector field by employing a vector median filter with weights decided based on the local motion reliability. In [10, 2], instead of refining the motion vector field, Kang *et al.* and Choi *et al.* proposed block matching motion estimation with overlapped and variable-size block technique in order to estimate motion as accurately as possible. However, the difficulty of the motion-based approach is that, even though the motion vector field may be refined and/or smoothed, more complex transitions (e.g. occlusions, transparency, and reflection) are not accurately treated. That is, motion errors are inevitable even after smoothing/refining motion vector fields, and, hence, an appropriate mechanism that takes care of the errors is necessary for producing artifact-free outputs.

Unlike video processing algorithms which depend directly on motion vectors, in a recent work, Protter *et al.* [18] proposed a video-to-video super-resolution method without explicit motion estimation or compensation based on the idea of Non-Local Means [1]. Although the method produces impressive spatial upscaling results even without motion estimation, the computational load is very high due to the exhaustive search (across space and time) for blocks similar to the block of interest. In a related work [24], we presented a space-time video upscaling method, called *3-D iterative steering kernel regression* (3-D ISKR), in which explicit subpixel motion estimation is again avoided. 3-D ISKR is an extension of 2-D *steering kernel regression* (SKR) proposed in [22, 21]. SKR is closely related to bilateral filtering [25, 3] and normalized convolution [17]. These methods can achieve accurate and robust image reconstruction results, due to their use of robust error norms and locally adaptive weighting functions. 2-D SKR has been applied to spatial interpolation, denoising and deblurring [17, 21, 23]. In 3-D ISKR, instead of relying on motion vectors, the 3-D kernel captures local spatial and temporal orientations based on local covariance matrices of gradients of video data. With the adaptive kernel, the method is capable of upscaling video with complex motion both in space and time.

In this chapter, we build upon the 2-D steering kernel regression framework proposed in [22], and develop a spatiotemporal (3-D) framework for processing video. Specifically, we propose an approach we call *motion-assisted steering kernel* (MASK) regression. The MASK function is a 3-D kernel, however, unlike as in 3-D ISKR, the kernel function takes spatial (2-D) orientation and the local motion trajectory into account separately, and it utilizes an analysis of the local orientation and local motion vector to steer spatiotemporal regression kernels. Subsequently, local kernel regression is applied to compute weighted least-squares optimal pixel estimates. Although 2-D kernel regression has been applied to achieve super-resolution reconstruction through fusion of multiple pre-registered frames on to a 2-D plane [22, 17], the proposed method is different in that it does not require explicit motion compensation of the video frames. Instead, we use 3-D weighting kernels that are "warped" according to estimated motion vectors, such that the regression process acts directly upon the video data. Although we consider local motion vectors in MASK, we propose an algorithm that is robust against errors in the estimated motion field. Prior multi-frame resolution-enhanced or super-resolution (SR) reconstruction methods ([4, 16]) often consider only global translational or affine motions; local motion and object occlusions are often not addressed. Many SR methods require explicit motion compensation, which may involve interpolation or rounding of displacements to grid locations. These issues can have a negative impact on accuracy and robustness. Our proposed method is capable of handling local motions, avoids explicit motion compensation, and is more robust. The proposed MASK approach is capable of simultaneous spatial interpolation with resolution enhancement, temporal video interpolation, noise reduction, and preserving high frequency components. Initial results using MASK were presented in [20].

An overview of this chapter is as follows. Firstly, we provide a review of 2-D SKR in Section 2. Then, we extend 2-D SKR to 3-D SKR and describe the MASK approach in Section 3. Subsequently, we propose a practical video upscaling algorithm based on MASK in Section 4, proposing further novel techniques to reduce computational complexity and improve robustness. We present several example results of our algorithm in Section 5 and conclude in Section 6.

2 Review of Steering Kernel Regression

This section gives an overview of SKR, which is the basis of MASK. We begin with describing the fundamental framework of SKR, called *kernel regression* (KR), in which we estimate a pixel value of interest from neighboring pixels using a weighted least-square formulation. We propose an effective weighting function for the weighted least-square estimator, called *steering kernel function*, that takes not only spatial distances between the samples of interest into account, but also the radiometric values of those samples.

2.1 Kernel Regression in 2-D

The KR framework defines its data model as

$$y_i = z(\mathbf{x}_i) + \varepsilon_i, \quad i = 1, \cdots, P, \quad \mathbf{x}_i = [x_{1i}, x_{2i}]^T, \tag{1}$$

where y_i is a noisy sample at $\mathbf{x}_i$ (Note: x_{1i} and x_{2i} are spatial coordinates), $z(\cdot)$ is the (hitherto unspecified) *regression function* to be estimated, ε_i is an i.i.d. zero mean noise, and P is the total number of samples in an arbitrary "window" around a position $\mathbf{x}$ of interest as shown in Fig. 1. As such, the kernel regression framework provides a rich mechanism for computing point-wise estimates of the regression function with minimal assumptions about global signal or noise models.

While the particular form of $z(\cdot)$ may remain unspecified, we can develop a generic local expansion of the function about a sampling point $\mathbf{x}_i$. Specifically, if the position of interest $\mathbf{x}$ is near the sample at $\mathbf{x}_i$, we have the N-th order Taylor series

$$\begin{aligned} z(\mathbf{x}_i) &\approx z(\mathbf{x}) + \{\nabla z(\mathbf{x})\}^T (\mathbf{x}_i - \mathbf{x}) + \frac{1}{2}(\mathbf{x}_i - \mathbf{x})^T \{\mathsf{H}z(\mathbf{x})\}(\mathbf{x}_i - \mathbf{x}) + \cdots \\ &= \beta_0 + \boldsymbol{\beta}_1^T (\mathbf{x}_i - \mathbf{x}) + \boldsymbol{\beta}_2^T \mathrm{vech}\left\{(\mathbf{x}_i - \mathbf{x})(\mathbf{x}_i - \mathbf{x})^T\right\} + \cdots \end{aligned} \tag{2}$$

where ∇ and H are the gradient (2×1) and Hessian (2×2) operators, respectively, and $\mathrm{vech}(\cdot)$ is the half-vectorization operator that lexicographically orders the lower triangular portion of a symmetric matrix into a column-stacked vector. Furthermore, β_0 is $z(\mathbf{x})$, which is the signal (or pixel) value of interest, and the vectors $\boldsymbol{\beta}_1$ and $\boldsymbol{\beta}_2$ are

$$\begin{aligned} \boldsymbol{\beta}_1 &= \left[\frac{\partial z(\mathbf{x})}{\partial x_1}, \quad \frac{\partial z(\mathbf{x})}{\partial x_2}\right]^T, \\ \boldsymbol{\beta}_2 &= \frac{1}{2}\left[\frac{\partial^2 z(\mathbf{x})}{\partial x_1^2}, \quad 2\frac{\partial^2 z(\mathbf{x})}{\partial x_1 \partial x_2}, \quad \frac{\partial^2 z(\mathbf{x})}{\partial x_2^2}\right]^T. \end{aligned} \tag{3}$$

Since this approach is based on *local* signal representations, a logical step to take is to estimate the parameters $\{\boldsymbol{\beta}_n\}_{n=0}^N$ from all the neighboring samples $\{y_i\}_{i=1}^P$ while giving the nearby samples higher weights than samples farther away. A (weighted) least-square formulation of the fitting problem capturing this idea is

$$\min_{\{\boldsymbol{\beta}_n\}_{n=0}^N} \sum_{i=1}^{P} \left[y_i - \beta_0 - \boldsymbol{\beta}_1^T (\mathbf{x}_i - \mathbf{x}) - \boldsymbol{\beta}_2^T \mathrm{vech}\left\{(\mathbf{x}_i - \mathbf{x})(\mathbf{x}_i - \mathbf{x})^T\right\} - \cdots\right]^2 K_{\mathbf{H}}(\mathbf{x}_i - \mathbf{x}) \tag{4}$$

with

$$K_{\mathbf{H}}(\mathbf{x}_i - \mathbf{x}) = \frac{1}{\det(\mathbf{H})} K\left(\mathbf{H}^{-1}(\mathbf{x}_i - \mathbf{x})\right), \tag{5}$$

where N is the regression order, $K(\cdot)$ is the kernel function (a radially symmetric function such as a Gaussian), and $\mathbf{H}$ is the smoothing (2×2) matrix which dictates the "footprint" of the kernel function. In the classical approach, when the pixels (y_i)

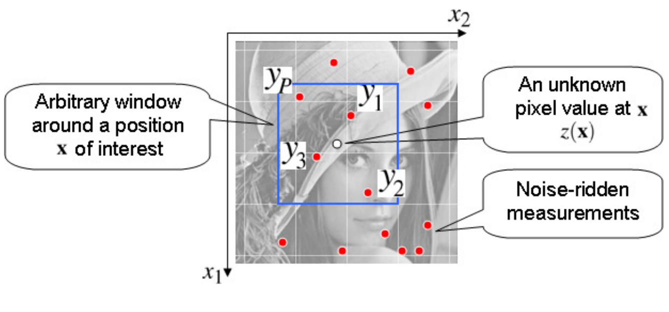

x_2
x_1
y_P
y_1
y_3
y_2
Arbitrary window around a position $\mathbf{x}$ of interest
An unknown pixel value at $\mathbf{x}$ $z(\mathbf{x})$
Noise-ridden measurements

Fig. 1 The data model for the kernel regression framework.

are equally spaced, the smoothing matrix is defined as

$$\mathbf{H} = h\mathbf{I} \tag{6}$$

for every sample, where h is called the *global smoothing parameter*. The shape of the kernel footprint is perhaps the most important factor in determining the quality of estimated signals. For example, it is desirable to use kernels with large footprints in the smooth local regions to reduce the noise effects, while relatively smaller footprints are suitable in the edge and textured regions to preserve the signal discontinuity. Furthermore, it is desirable to have kernels that adapt themselves to the local structure of the measured signal, providing, for instance, strong filtering along an edge rather than across it. This last point is indeed the motivation behind the *steering* KR framework [22] which we will review in Section 2.2.

Returning to the optimization problem (4), regardless of the regression order and the dimensionality of the regression function, we can rewrite it as a weighted least squares problem:

$$\widehat{\mathbf{b}} = \arg\min_{\mathbf{b}} \left[(\mathbf{y} - \mathbf{X}\mathbf{b})^T \mathbf{K} (\mathbf{y} - \mathbf{X}\mathbf{b}) \right], \tag{7}$$

where

$$\mathbf{y} = [y_1, \, y_2, \, \cdots, \, y_P]^T, \quad \mathbf{b} = \left[\beta_0, \, \boldsymbol{\beta}_2^T, \, \cdots, \, \boldsymbol{\beta}_N^T\right]^T, \tag{8}$$

$$\mathbf{K} = \operatorname{diag}\left[K_{\mathbf{H}}(\mathbf{x}_1 - \mathbf{x}), \, K_{\mathbf{H}}(\mathbf{x}_2 - \mathbf{x}), \cdots, \, K_{\mathbf{H}}(\mathbf{x}_P - \mathbf{x})\right], \tag{9}$$

and

$$\mathbf{X} = \begin{bmatrix} 1, & (\mathbf{x}_1 - \mathbf{x})^T, & \operatorname{vech}^T\left\{(\mathbf{x}_1 - \mathbf{x})(\mathbf{x}_1 - \mathbf{x})^T\right\}, & \cdots \\ 1, & (\mathbf{x}_2 - \mathbf{x})^T, & \operatorname{vech}^T\left\{(\mathbf{x}_2 - \mathbf{x})(\mathbf{x}_2 - \mathbf{x})^T\right\}, & \cdots \\ \vdots & \vdots & \vdots & \vdots \\ 1, & (\mathbf{x}_P - \mathbf{x})^T, & \operatorname{vech}^T\left\{(\mathbf{x}_P - \mathbf{x})(\mathbf{x}_P - \mathbf{x})^T\right\}, & \cdots \end{bmatrix} \tag{10}$$

with "diag" defining a diagonal matrix. Using the notation above, the optimization (4) provides the weighted least square estimator

$$\widehat{\mathbf{b}} = \left(\mathbf{X}^T\mathbf{K}\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{K}\,\mathbf{y} = \begin{bmatrix} \mathbf{W}_N \\ \mathbf{W}_{N,x_1} \\ \mathbf{W}_{N,x_2} \\ \vdots \end{bmatrix} \mathbf{y}, \tag{11}$$

where $\mathbf{W}_N$ is a $1 \times P$ vector that contains filter coefficients, which we call the *equivalent kernel* weights, and $\mathbf{W}_{N,x_1}$ and $\mathbf{W}_{N,x_2}$ are also $1 \times P$ vectors that compute the gradients along the x_1- and x_2-directions at the position of interest $\mathbf{x}$. The estimate of the signal (i.e. pixel) value of interest β_0 is given by a weighted *linear* combination of the nearby samples:

$$\hat{z}(\mathbf{x}) = \widehat{\beta}_0 = \mathbf{e}_1^T\widehat{\mathbf{b}} = \mathbf{W}_N\mathbf{y} = \sum_{i=1}^{P} W_i(K, \mathbf{H}, N, \mathbf{x}_i - \mathbf{x})\, y_i, \quad \sum_{i=1}^{P} W_i(\cdot) = 1, \tag{12}$$

where $\mathbf{e}_1$ is a column vector with the first element equal to one and the rest equal to zero, and we call W_i the equivalent kernel weight function for y_i (q.v. [22] or [26] for more detail). For example, for zero-th order regression (i.e. $N = 0$), the estimator (12) becomes

$$\hat{z}(\mathbf{x}) = \hat{\beta}_0 = \frac{\sum_{i=1}^{P} K_{\mathbf{H}}(\mathbf{x}_i - \mathbf{x})\, y_i}{\sum_{i=1}^{P} K_{\mathbf{H}}(\mathbf{x}_i - \mathbf{x})}, \tag{13}$$

which is the so-called *Nadaraya-Watson* estimator (NWE) [13].

What we described above is the “classic” kernel regression framework, which, as we just mentioned, yields a pointwise estimator that is always a local *linear* combination of the neighboring samples. As such, it suffers from an inherent limitation. In the next sections, we describe the framework of *steering* KR in two and three dimensions, in which the kernel weights themselves are computed from the local window, and therefore we arrive at filters with more complex (nonlinear) action on the data.

2.2 Steering Kernel Function

The steering kernel framework is based on the idea of robustly obtaining local signal structures (e.g. discontinuities in 2-D and planes in 3-D) by analyzing the radiometric (pixel value) variations locally, and feeding this structure information to the kernel function in order to affect its shape and size.

Consider the (2×2) smoothing matrix $\mathbf{H}$ in (5). As explained in the previous section, in the generic “classical” case, this matrix is a scalar multiple of the identity. This results in kernel weights which have equal effect along the x_1- and x_2-directions. However, if we properly choose this matrix locally (i.e. $\mathbf{H} \to \mathbf{H}_i$ for each y_i), the kernel function can capture local structures. More precisely, we define the smoothing matrix as a symmetric matrix

$$\mathbf{H}_i = h\mathbf{C}_i^{-\frac{1}{2}}, \tag{14}$$

which we call the *steering* matrix and where, for each given sample y_i, the matrix $\mathbf{C}_i$ is estimated as the local covariance matrix of the neighborhood spatial gradient vectors. A naive estimate of this covariance matrix may be obtained by

$$\widehat{\mathbf{C}}_i^{\text{naive}} = \mathbf{J}_i^T \mathbf{J}_i, \tag{15}$$

with

$$\mathbf{J}_i = \begin{bmatrix} z_{x_1}(\mathbf{x}_1) & z_{x_2}(\mathbf{x}_1) \\ \vdots & \vdots \\ z_{x_1}(\mathbf{x}_P) & z_{x_2}(\mathbf{x}_P) \end{bmatrix}, \tag{16}$$

where $z_{x_1}(\cdot)$ and $z_{x_2}(\cdot)$ are the first derivatives along x_1- and x_2-axes, and P is the number of samples in the local analysis window around a sampling position $\mathbf{x}_i$.

However, the naive estimate may in general be rank deficient or unstable. Therefore, instead of using the naive estimate, we can obtain the covariance matrix by using the (compact) singular value decomposition (SVD) of $\mathbf{J}_i$:

$$\mathbf{J}_i = \mathbf{U}_i \mathbf{S}_i \mathbf{V}_i^T, \tag{17}$$

where $\mathbf{S}_i = \text{diag}[s_1, s_2]$, and $\mathbf{V}_i = [\mathbf{v}_1, \mathbf{v}_2]$. The singular vectors contain direct information about the local orientation structure, and the corresponding singular values represent the energy (strength) in these respective orientation directions. Using the singular vectors and values, we compute a more stable estimate of our covariance matrix as:

$$\widehat{\mathbf{C}}_i = \gamma_i \mathbf{V}_i \begin{bmatrix} \rho_i & \\ & \frac{1}{\rho_i} \end{bmatrix} \mathbf{V}_i^T = \gamma_i \left(\rho_i \mathbf{v}_1 \mathbf{v}_1^T + \frac{1}{\rho_i} \mathbf{v}_2 \mathbf{v}_2^T \right), \tag{18}$$

where

$$\rho_i = \frac{s_1 + \lambda'}{s_2 + \lambda'}, \quad \gamma_i = \left(\frac{s_1 s_2 + \lambda''}{P} \right)^{\alpha}. \tag{19}$$

The parameters ρ_i and γ_i are the *elongation* and *scaling* parameter, respectively, and λ' and λ'' are “regularization” parameters, respectively, which dampen the effect of the noise and restrict γ_i and the denominator of ρ_i from becoming zero. The parameter α is called the *structure sensitivity* parameter. We fix $\lambda' = 0.1$, $\lambda'' = 0.1$, and $\alpha = 0.2$ in this work. More details about the effectiveness and the choice of the parameters can be found in [22]. With the above choice of the smoothing matrix and a Gaussian kernel, we now have the steering kernel function as

$$K_{\mathbf{H}_i}(\mathbf{x}_i - \mathbf{x}) = \frac{\sqrt{\det(\mathbf{C}_i)}}{2\pi h^2} \exp\left\{ -\frac{(\mathbf{x}_i - \mathbf{x})^T \mathbf{C}_i (\mathbf{x}_i - \mathbf{x})}{2h^2} \right\}. \tag{20}$$

Fig. 2 illustrates a schematic representation of the estimate of local covariance matrices and the computation of steering kernel weights. First we estimate the gradients and compute the local covariance matrix $\mathbf{C}_i$ by (18) for each pixel. Then, for example, when denoising y_{13}, we compute the steering kernel weights for each neighboring pixel with its $\mathbf{C}_i$. In this case, even though the spatial distances from y_{13} to y_1 and y_{21} are equal, the steering kernel weight for y_{21} (i.e. $K_{\mathbf{H}_{21}}(\mathbf{x}_{21} - \mathbf{x}_{13})$) is larger than the one for y_1 (i.e. $K_{\mathbf{H}_1}(\mathbf{x}_1 - \mathbf{x}_{13})$). Moreover, Fig. 3 shows visualizations of the 2-D steering kernel function for noise-free Lena image and a low PSNR[1] case (we added white Gaussian noise with standard deviation 25, the corresponding PSNR being 20.16[dB]). As shown in Fig. 3, the steering kernel weights (which are the normalized $K_{\mathbf{H}_i}(\mathbf{x}_i - \mathbf{x})$ as a function of $\mathbf{x}_i$ with $\mathbf{x}$ held fixed) illustrate the relative size of the actual weights applied to compute the estimate as in (12). We note that even for the highly noisy case, we can obtain stable estimates of local structure.

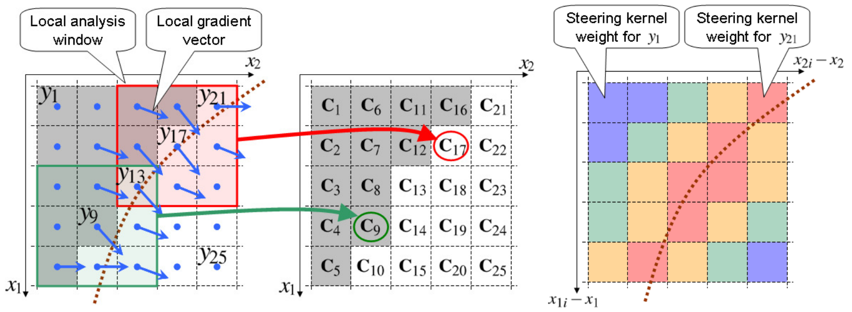
At this point, the reader may be curious to know how the above formulation would work for the case where we are interested not only in denoising, but also upscaling the images. Fig. 4 illustrates a summary of image upscaling by steering

[1] Peak Signal to Noise Ratio $= 10 \log_{10}\left(\frac{255^2}{\text{Mean Square Error}}\right)$ [dB].

kernel regression. Similar to the denoising case, we begin with computing steering (covariance) matrices, $\mathbf{C}_i$ for all the pixels, y_i, from the input image shown in Fig. 4(a) by (18) as depicted in Fig. 2(a). Once $\mathbf{C}_i$'s are available, we compute steering kernel weights by (20). For example, when we estimate the missing pixel $z(\mathbf{x})$ at $\mathbf{x}$ shown as the green box in Fig. 4(b), the steering kernel function gives high weights to the samples y_{13} and y_{17} and a small weight to y_{12}. This is because the missing pixel, $z(\mathbf{x})$, most likely lies on the same edge (shown by the red dashed curve) as y_{13} and y_{17}. Next, plugging the steering kernel weights into (11), we compute the equivalent kernel $\mathbf{W}_N$ and the estimator (12) gives the estimated pixel $\hat{z}(\mathbf{x})$ at $\mathbf{x}$. Fig. 4(c) shows the upscaled image by steering kernel regression. In [22], we introduced an iterative scheme where we recompute $\mathbf{C}_i$ from the upscaled image one more time, and, using the new covariance matrices, we estimate the missing pixels and denoise the given samples again. However, in this work, to keep the computational load low, we compute the steering matrices only once from the given samples.

3 Motion Assisted Steering Kernel Regression

SKR estimates an unknown pixel value in a single image by a weighted combination of neighboring pixels in the same image, giving larger weights to the pixels along a local orientation. In this section, we develop a multi-frame video upscaling method based on SKR by additionally utilizing local motion vectors, and we call the resulting method *motion-assisted steering kernel* (MASK) regression. The MASK approach is a 3-D kernel regression method in which the pixel of interest is estimated by a weighted combination of pixels in its spatiotemporal neighborhood,


Local analysis window
Local gradient vector
Steering kernel weight for y_1
Steering kernel weight for y_{21}
y_1
y_{21}
y_{17}
y_{13}
y_9
y_{25}
x_2
x_1
$\mathbf{C}_1$ $\mathbf{C}_6$ $\mathbf{C}_{11}$ $\mathbf{C}_{16}$ $\mathbf{C}_{21}$
$\mathbf{C}_2$ $\mathbf{C}_7$ $\mathbf{C}_{12}$ $\mathbf{C}_{17}$ $\mathbf{C}_{22}$
$\mathbf{C}_3$ $\mathbf{C}_8$ $\mathbf{C}_{13}$ $\mathbf{C}_{18}$ $\mathbf{C}_{23}$
$\mathbf{C}_4$ $\mathbf{C}_9$ $\mathbf{C}_{14}$ $\mathbf{C}_{19}$ $\mathbf{C}_{24}$
$\mathbf{C}_5$ $\mathbf{C}_{10}$ $\mathbf{C}_{15}$ $\mathbf{C}_{20}$ $\mathbf{C}_{25}$
$x_{2i}-x_2$
$x_{1i}-x_1$

(a) Covariance matrices from local gradients with 3 × 3 analysis window (b) Steering kernel weights

Fig. 2 A schematic representation of the estimates of local covariance metrics and the steering kernel weights at a local region with one dominant orientation: (a) First, we estimate the gradients and compute the local covariance matrix $\mathbf{C}_i$ by (18) for each pixel, and (b) Next, when denoising y_{13}, we compute the steering kernel weights with $\mathbf{C}_i$ for neighboring pixels. Even though, in this case, the spatial distances between y_{13} and y_1 and between y_{13} y_{21} are equal, the steering kernel weight for y_{21} (i.e. $K_{\mathbf{H}_{21}}(\mathbf{x}_{21}-\mathbf{x}_{13})$) is larger than the one for y_1 (i.e. $K_{\mathbf{H}_1}(\mathbf{x}_1-\mathbf{x}_{13})$). This is because y_{13} and y_{21} are located along the same edge.

involving multiple video frames. Hence, we first extend the 2-D kernel regression framework into a 3-D framework. Then, we present our 3-D data-adaptive kernel, the MASK function, which relies not only on local spatial orientation but also local motions. Finally, we describe the process of spatial upscaling and temporal frame interpolation based on MASK. While we focus on the principles of our approach in this section, we present a specific algorithm for video processing based on the MASK method in the next section.

3.1 Spatiotemporal Kernel Regression

For video processing, we define a spatiotemporal data model as

$$y_i = z(\mathbf{x}_i) + \varepsilon_i, \quad i = 1, \cdots, P, \quad \mathbf{x}_i = [x_{1i}, x_{2i}, t_i]^T, \tag{21}$$

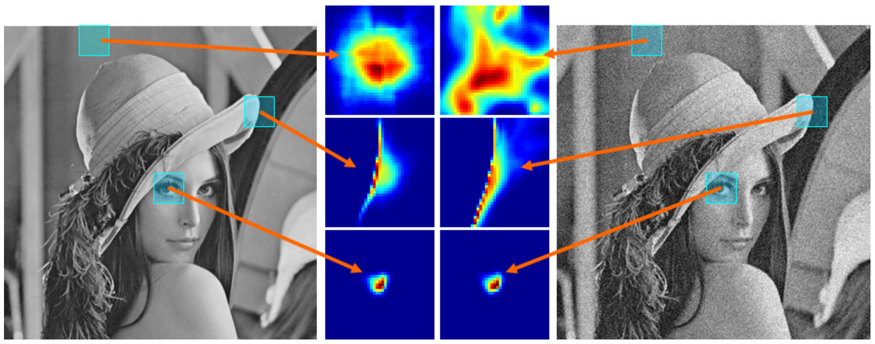
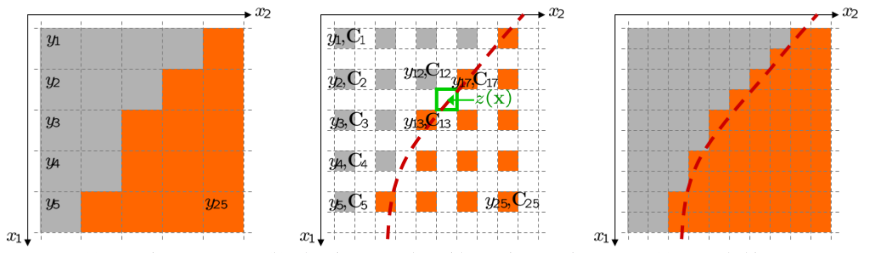


Fig. 3 Steering kernel weights for Lena image without/with noise (white Gaussian noise with standard deviation $\sigma = 25$) at flat, edge, and texture areas.



(a) Input image (b) The given samples with steering matrices (c) Upscaled image

Fig. 4 Steering kernel regression for image upscaling: (a)Input image. (b)We compute steering matrices for each pixel and then estimate. Then, estimate the missing position $\mathbf{z}(\mathbf{x})$ and denoise the given pixels y_i. The red dashed line is a speculative local orientation. (c)Upscaled image by steering kernel regression.

where y_i is a given sample (pixel) at location $\mathbf{x}_i$, x_{1i} and x_{2i} are the spatial coordinates, t_i is the temporal coordinate, $z(\cdot)$ is the *regression function*, and ε_i is i.i.d zero mean noise. P is the number of samples in a spatiotemporal neighborhood of interest, which spans multiple video frames.

Similar to the 2-D case, in order to estimate the value of $z(\cdot)$ at point $\mathbf{x}$, given the above data samples y_i, we can rely on a local N^{th} order Taylor expansion about $\mathbf{x}$. We denote the pixel value of interest $z(\mathbf{x})$ by β_0, while $\boldsymbol{\beta}_1$, $\boldsymbol{\beta}_2$, $\cdots$, $\boldsymbol{\beta}_N$ denote vectors containing the first-order, second-order, $\cdots$, N^{th} order partial derivatives of $z(\cdot)$ at $\mathbf{x}$, resulting from the Taylor expansion. For example, $\beta_0 = z(\mathbf{x})$ and $\boldsymbol{\beta}_1 = [z_{x_1}(\mathbf{x}), z_{x_2}(\mathbf{x}), z_t(\mathbf{x})]^T$.

The unknowns, $\{\boldsymbol{\beta}_n\}_{n=0}^N$, can be estimated from $\{y_i\}_{i=1}^P$ using the following weighted least-squares optimization procedure:

$$\min_{\{\boldsymbol{\beta}_n\}_{n=0}^N} \sum_{i=1}^P \left[y_i - \beta_0 - \boldsymbol{\beta}_1^T(\mathbf{x}_i - \mathbf{x}) - \boldsymbol{\beta}_2^T \mathrm{vech}\left\{(\mathbf{x}_i - \mathbf{x})(\mathbf{x}_i - \mathbf{x})^T\right\} - \cdots \right]^2 K_{\mathbf{H}_i^{\mathrm{3D}}}(\mathbf{x}_i - \mathbf{x}) \tag{22}$$

where N is the regression order and $K(\cdot)$ is a *kernel function* that weights the influence of each sample. Typically, samples near $\mathbf{x}$ are given higher weights than samples farther away.

A *3-D steering kernel* is a direct extension of the 2-D steering kernel defined in [22]. The 3×3 data-dependent steering matrix $\mathbf{H}_i^{\mathrm{3D}}$ can be defined as

$$\mathbf{H}_i^{\mathrm{3D}} = h\left(\mathbf{C}_i^{\mathrm{3D}}\right)^{-\frac{1}{2}} \tag{23}$$

where h is a global smoothing parameter and $\mathbf{C}_i^{\mathrm{3D}}$ is a 3×3 covariance matrix based on the sample variations in a local (3-D) neighborhood around sample $\mathbf{x}_i$. We can construct the matrix $\mathbf{C}_i^{\mathrm{3D}}$ parametrically as $\mathbf{C}_i^{\mathrm{3D}} = \gamma_i \mathbf{R}_i \boldsymbol{\Lambda}_i \mathbf{R}_i^T$, where $\mathbf{R}_i$ is a 3-D rotation matrix, $\boldsymbol{\Lambda}_i$ is a 3-D elongation matrix, and γ_i is a scaling parameter. We have found that such an approach performs quite well for spatial upscaling of video [24]. However, this 3-D kernel does not consider the specific spatiotemporal characteristics of video data. In particular, problems may occur in the presence of large object displacements (fast motion). This may result in either shrinking of the kernel in the temporal direction, or spatial blurring (as the kernel weights spread across unrelated data samples), both undesirable effects.

3.2 Motion Assisted Steering Kernel Function

A good choice for steering spatiotemporally is to consider local motion or optical flow vectors caused by object motion in the scene, in conjunction with spatial steering along local edges and isophotes. Spatial steering should consider the locally dominant orientation of the pixel data and should allow elongation of the kernel in this direction. Spatiotemporal steering should allow alignment of the kernel weights with the local optical flow or motion trajectory, as well as overall temporal scaling. Hence, we construct our spatiotemporal kernel as a product of a spatial- and motion-steering kernel, and a kernel that acts temporally:

$$\mathsf{K}_{\text{MASK}} \equiv \frac{1}{\det(\mathbf{H}_i^{\text{s}})} K\big((\mathbf{H}_i^{\text{s}})^{-1}\mathbf{H}_i^{\text{m}}(\mathbf{x}_i - \mathbf{x})\big)\, K_{h_{\text{t}}}(t_i - t), \tag{24}$$

where $\mathbf{H}_i^{\text{s}}$ is a 3×3 *spatial steering* matrix, $\mathbf{H}_i^{\text{m}}$ is a 3×3 *motion steering* matrix, $K_{h_{\text{t}}}(\cdot)$ is a temporal kernel function, and h_{t} is the temporal smoothing parameter which controls the temporal penalization. These data-dependent kernel components determine the steering action at sample $\mathbf{x}_i$, and are described next.

Following [22], the spatial steering matrix $\mathbf{H}_i^{\text{s}}$ is defined by:

$$\mathbf{H}_i^{\text{s}} = h_{\text{s}} \begin{bmatrix} \mathbf{C}_i & \\ & 1 \end{bmatrix}^{-\frac{1}{2}}, \tag{25}$$

where h_{s} is a global spatial smoothing parameter, and $\mathbf{C}_i$ is a 2×2 covariance matrix given by (18), which captures the sample variations in a local spatial neighborhood around $\mathbf{x}_i$. $\mathbf{C}_i$ is constructed in a parametric manner, as shown in (18).

The motion steering matrix $\mathbf{H}_i^{\text{m}}$ is constructed on the basis of a local estimate of the motion (or optical flow vector) $\mathbf{m}_i = [m_{1i}, m_{2i}]^T$ at $\mathbf{x}_i$. Namely, we warp the kernel along the local motion trajectory using the following shearing transformation:

$$\begin{cases} (x_{1i} - x_1) \leftarrow (x_{1i} - x_1) - m_{1i} \cdot (t_i - t) \\ (x_{2i} - x_2) \leftarrow (x_{2i} - x_2) - m_{2i} \cdot (t_i - t) \end{cases}.$$

Hence,

$$\mathbf{H}_i^{\text{m}} = \begin{bmatrix} 1 & 0 & -m_{1i} \\ 0 & 1 & -m_{2i} \\ 0 & 0 & 0 \end{bmatrix}. \tag{26}$$

Assuming a spatial prototype kernel was used with elliptical footprint, this results in a spatiotemporal kernel with the shape of a tube or cylinder with elliptical cross-sections at any time instance t. Most importantly, the center point of each such cross-section moves along the motion path.

The final component of (24) is a temporal kernel that provides temporal penalization. A natural approach is to give higher weights to samples in frames closer to t. An example of such a kernel is the following:

$$K_{h_t}(t_i - t) = \frac{1}{h_{\text{t}}} \exp\left(-\frac{|t_i - t|^2}{2h_{\text{t}}^2}\right), \tag{27}$$

where a temporal smoothing parameter h_t controls the relative temporal extent of the kernel. We use the temporal kernel (27) in this section to illustrate the MASK approach. However, we will introduce a more powerful adaptive temporal weighting kernel in Section 4.2, which acts to compensate for unreliable local motion vector estimates.

3.3 Spatial Upscaling and Temporal Frame Interpolation

Having introduced our choice of 3-D smoothing matrix, $\mathbf{H}_i^{3\mathrm{D}}$, using Gaussian kernel for K, we have the MASK function as

$$\begin{aligned} K_{\mathrm{MASK}}(\mathbf{x}_i - \mathbf{x}) &= \frac{1}{\det(\mathbf{H}_i^{\mathrm{s}})} K\left((\mathbf{H}_i^{\mathrm{s}})^{-1} \mathbf{H}_i^{\mathbf{m}} (\mathbf{x}_i - \mathbf{x}) \right) \cdot K_{h_t}(t_i - t) \\ &= \frac{1}{\det(\mathbf{H}_i^{\mathrm{s}})} K\left((\mathbf{H}_i^{\mathrm{s}})^{-1} \left(\mathbf{x}_i - \mathbf{x} - \begin{bmatrix} \mathbf{m}_i \\ 1 \end{bmatrix} (t_i - t) \right) \right) \cdot K_{h_{\mathrm{t}}}(t_i - t) \\ &= \frac{\sqrt{\det(\mathbf{C}_i)}}{h_{\mathrm{s}}^2 h_{\mathrm{t}}^2} \exp\left(-\frac{1}{2h_{\mathrm{s}}^2} \left\| \mathbf{x}_i - \mathbf{x} - \begin{bmatrix} \mathbf{m}_i \\ 1 \end{bmatrix} (t_i - t) \right\|_{\mathbf{C}_i}^2 \right) \\ &\qquad \cdot \exp\left(-\frac{|t_i - t|^2}{2h_{\mathrm{t}}^2} \right) \end{aligned} \tag{28}$$

where $\|\cdot\|_{\mathbf{C}_i^{\mathrm{s}}}^2$ is weighted squared L_2-norm. Figs. 5(a-i)-(a-iii) graphically describe how the proposed MASK function constructs its weights for spatial upscaling. For ease of explanation, suppose there are 5 frames at times from t_1 to t_5, and we upscale the third frame (spatial upscaling). When estimating the pixel value at $\mathbf{x} = [x_1, x_2, t]$, where $t = t_3$, first we compute 2-D steering kernel weights for each frame, as illustrated in Fig. 5(a-i), using the first Gaussian kernel function in (28). Motions are not taken into account at this stage. Second, having motion vectors, $\mathbf{m}_i$, which we estimate using the optical flow technique with the translational motion model and the frame at $t_{i=3}$ as the anchor frame, we shift the steering kernels for each frame by $\mathbf{m}_i$ as illustrated in Fig. 5(a-ii). Finally, as in Fig. 5(a-iii), the temporal kernel function penalizes the shifted steering kernels so that we give high weights to closer neighboring frames.

Local steering parameters and spatio-temporal weights are estimated at each pixel location $\mathbf{x}_i$ in a small region of support for the final regression step. Once the MASK weights are available, similar to the 2-D case, we plug them into (11), compute the equivalent kernel $\mathbf{W}_N$, and then estimate the missing pixels and denoise the given samples from the local input samples (y_i) around the position of interest $\mathbf{x}$. Similar to (12), the final spatio-temporal regression step can be expressed as follows:

$$\hat{z}(\mathbf{x}) = \sum_{i=1}^{P} W_i(\mathbf{x}; \mathbf{H}_i^{\mathrm{s}}, \mathbf{H}_i^{\mathrm{m}}, h_{\mathrm{t}}, K, N)\, y_i. \tag{29}$$

The MASK approach is also capable of upscaling video temporally (also called frame interpolation or frame rate upconversion). Fig. 5(b) illustrates the MASK weights for estimating an intermediate frame at sometime between t_3 and t_4. Fundamentally, following the same procedure as described in Figs. 5(a-i)-(a-iii), we generate MASK weights. However, for the motion vector with the unknown intermediate frame as the anchor frame, we assume that the motion between the frames at t_3 and t_4 is constant, and using the motion vectors, $\mathbf{m}_{i=1,\cdots,5}$, we linearly interpolate motion vectors $\mathbf{m}_i'$ as

$$\mathbf{m}_i' = \mathbf{m}_i + \mathbf{m}_4 (t - t_3). \tag{30}$$

Note that when $\mathbf{m}_4$ is inaccurate, the interpolated motion vectors for other frames in the temporal window ($\mathbf{m}_i'$) are also inaccurate. In that case, we would shift the kernel toward the wrong direction, and the MASK weights would be less effective for temporal upscaling. Therefore, one should incorporate a test of the reliability of $\mathbf{m}_4$ into the process, and use vectors $\mathbf{m}_i$ instead of $\mathbf{m}_i'$ if it is found to be unreliable. Our specific technique to compute the reliability of motion vectors is described in Section 4.2.

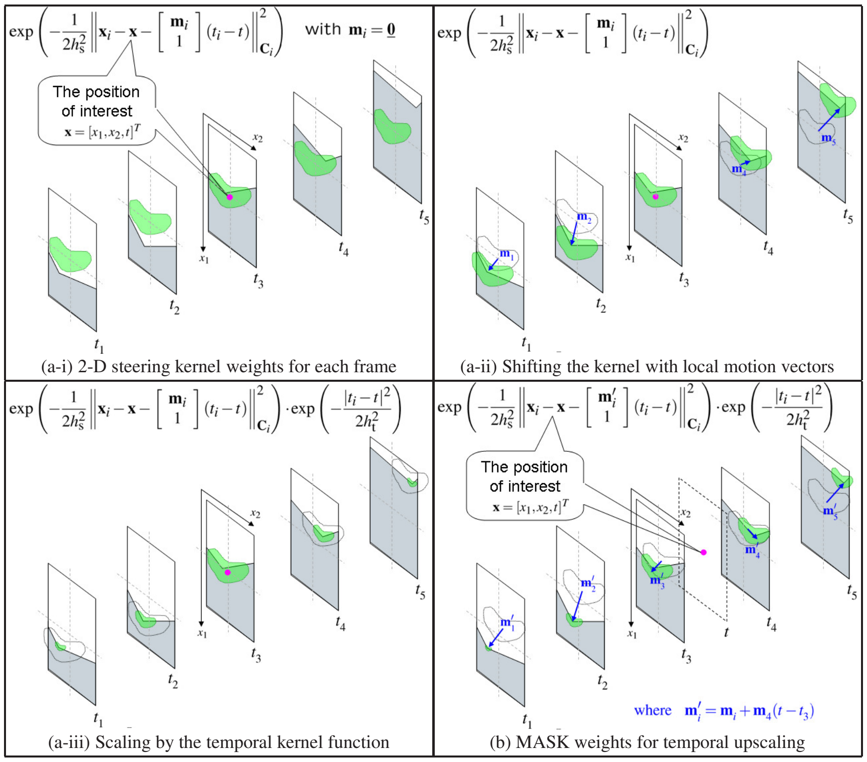

$\exp\left(-\frac{1}{2h_s^2}\left\|\mathbf{x}_i - \mathbf{x} - \begin{bmatrix}\mathbf{m}_i\\1\end{bmatrix}(t_i - t)\right\|_{\mathbf{C}_i}^2\right)$ with $\mathbf{m}_i = \underline{\mathbf{0}}$
The position of interest $\mathbf{x} = [x_1, x_2, t]^T$
x_1 x_2 t_1 t_2 t_3 t_4 t_5
(a-i) 2-D steering kernel weights for each frame
$\exp\left(-\frac{1}{2h_s^2}\left\|\mathbf{x}_i - \mathbf{x} - \begin{bmatrix}\mathbf{m}_i\\1\end{bmatrix}(t_i - t)\right\|_{\mathbf{C}_i}^2\right)$
$\mathbf{m}_1$ $\mathbf{m}_2$ $\mathbf{m}_4$ $\mathbf{m}_5$
(a-ii) Shifting the kernel with local motion vectors
$\exp\left(-\frac{1}{2h_s^2}\left\|\mathbf{x}_i - \mathbf{x} - \begin{bmatrix}\mathbf{m}_i\\1\end{bmatrix}(t_i - t)\right\|_{\mathbf{C}_i}^2\right)\cdot\exp\left(-\frac{|t_i - t|^2}{2h_t^2}\right)$
(a-iii) Scaling by the temporal kernel function
$\exp\left(-\frac{1}{2h_s^2}\left\|\mathbf{x}_i - \mathbf{x} - \begin{bmatrix}\mathbf{m}_i'\\1\end{bmatrix}(t_i - t)\right\|_{\mathbf{C}_i}^2\right)\cdot\exp\left(-\frac{|t_i - t|^2}{2h_t^2}\right)$
The position of interest $\mathbf{x} = [x_1, x_2, t]^T$
$\mathbf{m}_1'$ $\mathbf{m}_2'$ $\mathbf{m}_3'$ $\mathbf{m}_4'$ $\mathbf{m}_5'$ t
where $\mathbf{m}_i' = \mathbf{m}_i + \mathbf{m}_4(t - t_3)$
(b) MASK weights for temporal upscaling

Fig. 5 Schematic representations of the construction of MASK weights: the proposed MASK weights are constructed by the following procedure (a-i) compute 2-D steering kernel weights for each frame (with $\mathbf{m}_i = \underline{\mathbf{0}}$ at this moment), (a-ii) shift the steering kernels by the local motion vectors, and (a-iii) scale the shifted steering kernels by the temporal kernel function. Fig.(b) shows the weight construction for the estimation of an intermediate frame at time t.

4 A Practical Video Upscaling Algorithm Based on MASK

In this section, we describe a complete algorithm for spatial upscaling, denoising and enhancement, as well as temporal frame interpolation, based on the MASK approach. We introduce several techniques that enable a practical implementation of the MASK principles explained in the previous section. In particular, we develop an algorithm with reduced computational complexity and reduced memory requirements, that is suitable for both software and hardware implementation.

An overview of the proposed video interpolation and denoising algorithm based on motion-assisted spatiotemporal steering kernel regression is provided in Fig. 6. The algorithm estimates spatial and motion steering parameters using gradient-based techniques. Hence, we first compute initial estimates of the spatial and temporal derivatives, e.g. based on classic kernel regression. In this work, we obtain a quick and robust estimate of the spatial orientation angle (θ_i), elongation (ρ_i) and scaling (γ_i) parameters at $\mathbf{x}_i$ by applying a vector quantization technique to the covariance matrix obtained from the spatial gradient data. This will be described in Section 4.3. Motion vectors are estimated using the well-known Lucas and Kanade method, based on both spatial and temporal gradients in a local region. This is followed by computing estimates of the temporal motion reliability (η), and is described further in Section 4.2. Given spatial and motion steering parameters, final MASK regression is applied directly on the input video samples; further details of this step are provided in Section 4.4.

The following are further salient points for our algorithm based on MASK. We first summarize them, and then provide details in subsequent subsections.

▷ **Block-by-Block Processing**
Since the kernel-based estimator is a pointwise process, it is unnecessary to store the orientations and motion vectors of all the pixels in a video frame ($\mathbf{H}_i^s$ and $\mathbf{H}_i^m$ for all i) in memory. However, strict pixel-by-pixel processing would result in a large number of redundant computations due to the overlapping neighborhoods of nearby pixels. In order to reduce the computational load while keeping the

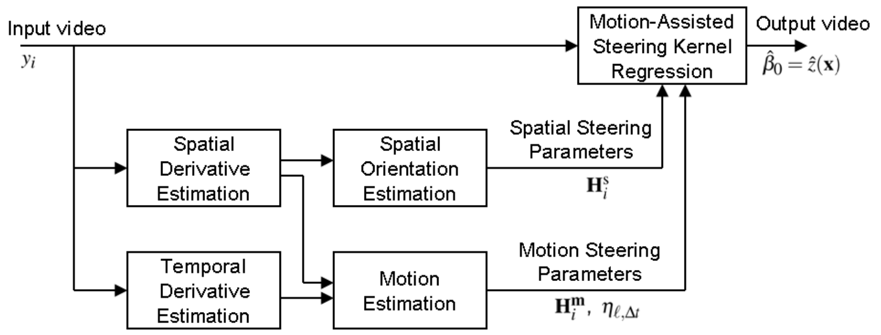

Input video
y_i
Motion-Assisted Steering Kernel Regression
Output video
$\hat{\beta}_0 = \hat{z}(\mathbf{x})$
Spatial Derivative Estimation
Spatial Orientation Estimation
Spatial Steering Parameters
$\mathbf{H}_i^s$
Temporal Derivative Estimation
Motion Estimation
Motion Steering Parameters
$\mathbf{H}_i^m$, $\eta_{\ell,\Delta t}$

Fig. 6 Illustration of video processing based on motion-assisted spatiotemporal steering kernel (MASK) regression.

required memory space small, we break the video data into small blocks (e.g. 8×8 pixels), and process the blocks one-by-one.

▷ **Adaptive Temporal Penalization**
MASK relies on motion vectors, and the visual quality of output video frames is strongly associated with the accuracy of motion estimation. Even though our motion estimation approach is able to estimate motion vectors quite accurately, the estimated vectors become unreliable when the underlying scene motion and camera projection violate the motion model. In practice, errors in motion vectors are inevitable and it is important to provide a fall-back mechanism in order to avoid visual artifacts.

▷ **Quantization of Orientation Map**
The estimation of spatial orientations or steering covariance matrices $\mathbf{C}_i^s$ in (18) involves singular value decomposition (SVD), which represents significant computational complexity. Instead of using the SVD, we use a pre-defined lookup table containing a set of candidate covariance matrices, and locally select an appropriate matrix from the table. Since the lookup table contains only stable (invertible) covariance matrices, the estimation process remains robust.

▷ **Adaptive Regression Order**
A higher regression order (e.g. $N = 2$ in this chapter) preserves high frequency components in filtered images, although it requires more computation (11). On the other hand, zeroth regression order ($N = 0$) has lower computational cost, but it has a stronger smoothing effect. Although second order regression is preferable, it is only needed at pixel locations in texture and edge regions. Moreover, in terms of noise reduction, zeroth order regression is more suitable in flat regions. We propose to adjust the order N locally, based on the scaling parameter (γ_i). Consequently, this adaptive approach keeps the total computational cost low while it preserves, and even enhances, high frequency components.

4.1 Block-by-Block Processing

The overall MASK algorithm consists of several operations (i.e. estimating spatial and temporal gradients, spatial orientations, and motions as shown in Fig. 6 and finally applying kernel regression), and it is possible to implement these in, e.g., a pixel-by-pixel process or a batch process. In a pixel-by-pixel process, we estimate gradients, orientations, and motions one-by-one, and then finally estimate a pixel value. Note that most of these operations require calculations involving other pixels in a neighborhood around the pixel of interest. Since the neighborhoods of nearby pixels may overlap significantly, frequently the same calculation would be performed multiple times. Hence, a pixel-by-pixel implementation suffers from a large computational load. On the other hand, this implementation requires very little memory. In a batch process, we estimate gradients for all pixels in an entire frame and store the results in memory, then estimate orientations of all pixels and store those results, etc. In the batch implementation, we need a large memory space to store intermediate results for all pixels in a frame; however, it avoids repeated calculations. This type of process is impractical for a hardware implementation.

As a compromise, in order to limit both the computational load and the use of memory, we process a video frame in a block-by-block manner, where each block contains, e.g., 8×8 or 16×16 pixels. Further reduction of the computational load is achieved by using a block-based motion model: we assume that, within a block, the motion of all the pixels follow a parametric model, e.g, translational or affine. In this chapter, we fix the block size to 8×8 pixels and we use the translational motion model. A variable block size and the use of other motion models are also possible, and are the subject of ongoing research.

4.2 Motion Estimation and Adaptive Temporal Penalization

As mentioned, motion estimation is based on the well-known Lucas and Kanade method [11, 19], applied in a block-by-block manner as follows. Assume we computed initial estimates of the local spatial and temporal derivatives. For example, spatial derivatives may be computed using classic kernel regression or existing derivative filtering techniques. Temporal derivatives are computed by taking the temporal difference between pixels of the current frame and one of the neighboring frames. Let $\hat{\mathbf{z}}_{x_1}$, $\hat{\mathbf{z}}_{x_2}$ and $\hat{\mathbf{z}}_t$ denote vectors containing (in lexicographical order) derivative estimates from the pixels in a local analysis window w_l associated with the ℓ-th block in the frame. This window contains and is typically centered on the block of pixels of interest, but may include additional pixels beyond the block (i.e. analysis windows from neighboring blocks may overlap). A motion vector $\mathbf{m}_l$ for block ℓ is estimated by solving the optical flow equation $[\hat{\mathbf{z}}_{x_1}, \hat{\mathbf{z}}_{x_2}]\mathbf{m}_\ell + \hat{\mathbf{z}}_t = \underline{\mathbf{0}}$ in the least-squares sense. The basic Lucas and Kanade method is applied iteratively for improved performance. As explained before, MASK uses multiple frames in a temporal window around the current frame. For every block in the current frame, a motion vector is computed to each of the neighboring frames in the temporal window. Hence, if the temporal window contains 4 neighboring frames in addition to the current frame, we compute 4 motion vectors for each block in the current frame.

In practice, a wide variety of transitions/activies will occur in natural video. Some of them are so complex that no parametric motion model matches them exactly, and motion errors are unavoidable. When there are errors in the estimated motion vectors, visually unacceptable artefacts may be introduced in the reconstructed frames due to the motion-based processing. One way to avoid such visible artifacts in upscaled frames is to adapt the temporal weighting based on the correlation between the current block and the corresponding blocks in other frames determined by the motion vectors. That is to say, before constructing MASK weights, we compute the reliability (η_ℓ) of each estimated motion vector. A simple way to define η_ℓ is to use the mean square error or mean absolute error between the block of interest and the corresponding block in the neighboring frame towards which the motion vector is pointing. Once the reliability of the estimated motion vector is available, we penalize the steering kernels by a temporal kernel K_t, a kernel function of η. Fig. 7 illustrates the temporal weighting, incorporating motion reliability. Suppose we upscale the ℓ-th block in the frame at time t using 2 previous and 2 forward frames, and there are 4 motion vectors, $\mathbf{m}_{\ell,i}$, between a block in the frame at t and the 4 neighboring frames. First, we find the blocks that the motion vectors indicate from the neighboring frames shown as $\mathbf{y}_{\ell,i}$ in Fig. 7. Then, we compute the motion reliability based on the difference between the ℓ-th block at t and other blocks and decide the temporal penalization for each neighboring block.

More specifically, we define $\eta_{\ell,\Delta t}$ and K_t as

$$\eta_{\ell,\Delta t} = \frac{\left\| \mathbf{y}_{\ell,t} - \mathbf{y}_{\ell,t+\Delta t} \right\|_{\mathrm{F}}}{M}, \tag{31}$$

$$K_{h_{\mathrm{t}}}(\eta_{\ell,\Delta t}) = \frac{1}{1 + \dfrac{\eta_{\ell,\Delta t}}{h_{\mathrm{t}}}} \tag{32}$$

where h_{t} is the (global) smoothing parameter, which controls the strength of temporal penalization, $\mathbf{y}_{\ell,t}$ is the ℓ^{th} block of the frame at time t, $t + \Delta t$ is a neighboring frame, M is the total number of pixels in a block, and $\| \cdot \|_{\mathrm{F}}$ is Frobenius norm. We replace the temporal kernel in (28) by (32). This temporal weighting technique is similar to the Adaptive Weighted Averaging (AWA) approach proposed in [15]; however, the weights in AWA are computed pixel-wise. In MASK, the temporal kernel weights are a function of radiometric distances between small pixel blocks and are computed block-wise.

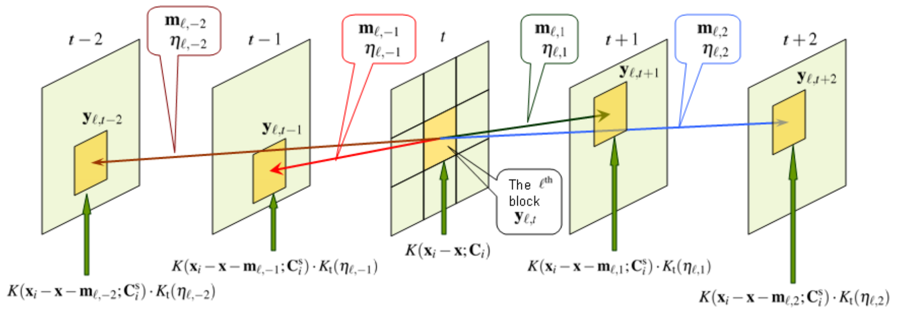

$t-2$
$\mathbf{m}_{\ell,-2}$ $\eta_{\ell,-2}$
$t-1$
$\mathbf{m}_{\ell,-1}$ $\eta_{\ell,-1}$
t
$\mathbf{m}_{\ell,1}$ $\eta_{\ell,1}$
$t+1$
$\mathbf{m}_{\ell,2}$ $\eta_{\ell,2}$
$t+2$
$\mathbf{y}_{\ell,t-2}$
$\mathbf{y}_{\ell,t-1}$
$\mathbf{y}_{\ell,t+1}$
$\mathbf{y}_{\ell,t+2}$
The ℓ^{th} block $\mathbf{y}_{\ell,t}$
$K(\mathbf{x}_i - \mathbf{x}; \mathbf{C}_i)$
$K(\mathbf{x}_i - \mathbf{x} - \mathbf{m}_{\ell,-1}; \mathbf{C}_i^{\mathrm{s}}) \cdot K_{\mathrm{t}}(\eta_{\ell,-1})$
$K(\mathbf{x}_i - \mathbf{x} - \mathbf{m}_{\ell,1}; \mathbf{C}_i^{\mathrm{s}}) \cdot K_{\mathrm{t}}(\eta_{\ell,1})$
$K(\mathbf{x}_i - \mathbf{x} - \mathbf{m}_{\ell,-2}; \mathbf{C}_i^{\mathrm{s}}) \cdot K_{\mathrm{t}}(\eta_{\ell,-2})$
$K(\mathbf{x}_i - \mathbf{x} - \mathbf{m}_{\ell,2}; \mathbf{C}_i^{\mathrm{s}}) \cdot K_{\mathrm{t}}(\eta_{\ell,2})$

Fig. 7 A schematic representation of temporal weighting in MASK for upscaling the ℓ-th block ($\mathbf{y}_{\ell,t}$) of the frame at time t. First, we locate the neighboring blocks ($\mathbf{y}_{\ell,i}$ for $i = -2, -1, 1, 2$) indicated by the motion vectors ($\mathbf{m}_{\ell,i}$). Then, we compute the motion reliability ($\eta_{\ell,i}$) based on the difference between the ℓ-th block at t and the neighboring blocks, and combine the temporal penalization by K_t with the spatial kernel function K.

4.3 Quantization of Orientation Map

The computational cost of estimating local spatial steering (covariance) matrices is high due to the SVD. In this section, using the well-known technique of *vector quantization* [6], we describe a way to obtain stable (invertible) steering matrices without using the SVD. Briefly speaking, first, we construct a look-up table which has a certain number of stable (invertible) steering matrices. Second, instead of computing the steering matrix by (18), we compute the naive covariance matrix (15), and then find the most similar steering matrix from the look-up table. The advantages of using the look-up table are that (i) we can lower the computational complexity by avoiding singular value decomposition, (ii) we can control and trade-off accuracy and computational load by designing an appropriate vector quantization scheme with almost any desired number of steering matrices in the look-up table, and (iii) we can pre-calculate kernel weights to lower the computational load further (since the steering matrices are fixed).

From (18), the elements of the spatial covariance matrix $\mathbf{C}_i$ are given by the steering parameters with the following equations:

$$\mathbf{C}_j(\gamma_j, \rho_j, \theta_j) = \begin{bmatrix} c_{11} & c_{12} \\ c_{12} & c_{22} \end{bmatrix}, \tag{33}$$

with

$$c_{11} = \gamma_j \left(\rho_j \cos^2 \theta_j + \rho_j^{-1} \sin^2 \theta_j \right) \tag{34}$$

$$c_{12} = -\gamma_j \left(\rho_j \cos \theta_j \sin \theta_j + \rho_j^{-1} \cos \theta_j \sin \theta_j \right) \tag{35}$$

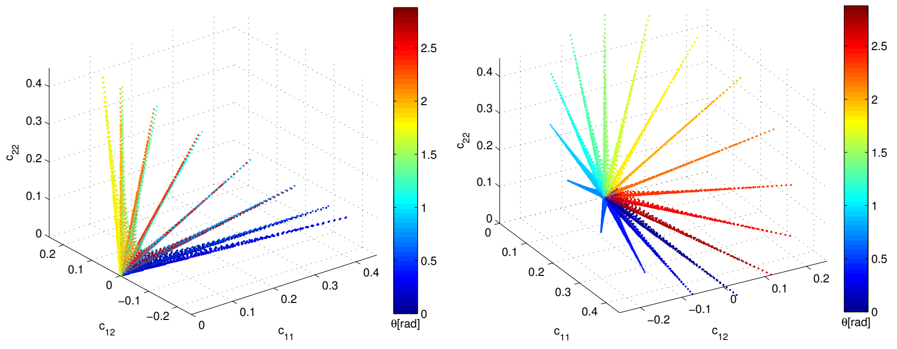
$$c_{22} = \gamma_j \left(\rho_j \sin^2 \theta_j + \rho_j^{-1} \cos^2 \theta_j \right) \tag{36}$$

where γ_j is the scaling parameter, ρ_j is the elongation parameter, and θ_j is the orientation angle parameter. Fig. 8 visualizes the relationship between the steering parameters and the values of the covariance matrix. Based on the above formulae, using a pre-defined set of the scaling, elongation, and angle parameters, we can generate a lookup table for covariance matrices, during an off-line stage.

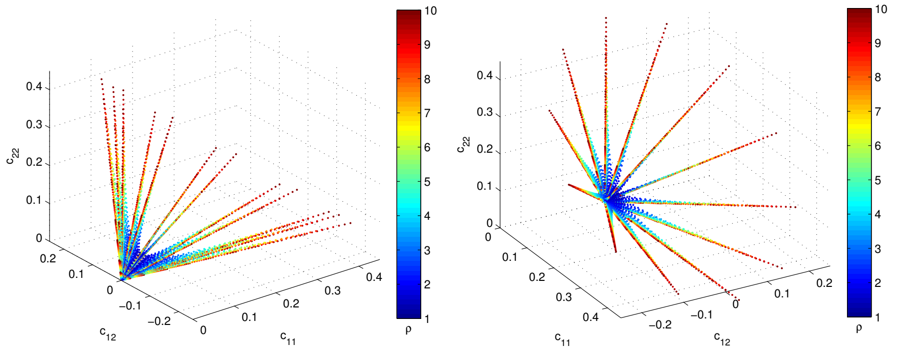
During the on-line processing stage, we compute a naive covariance matrix $\mathbf{C}_i^{\text{naive}}$ (15) and then normalize $\mathbf{C}_i^{\text{naive}}$ so that the determinant of the normalized naive covariance matrix $\det(\widetilde{\mathbf{C}}_i^{\text{naive}})$ equals 1.0:

$$\widetilde{\mathbf{C}}_i^{\text{naive}} = \frac{\mathbf{C}_i^{\text{naive}}}{\sqrt{\det\left(\mathbf{C}_i^{\text{naive}}\right)}} = \frac{1}{\gamma_i} \mathbf{C}_i^{\text{naive}}, \tag{37}$$

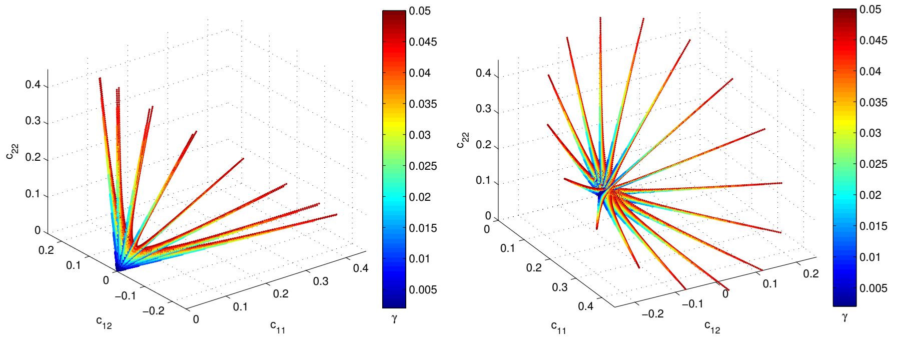
where again γ_i is the scaling parameter. This normalization eliminates the scaling parameter from the look-up table and simplifies the relationship between the elements of covariance matrices and the steering parameters, and allows us to reduce the size of the table. Table 1 shows an example of a compact lookup table. When the elongation parameter ρ_i of $\widetilde{\mathbf{C}}_i$ is smaller than 2.5, $\widehat{\mathbf{C}}_i$ is quantized as an identity matrix (i.e. the kernel spreads equally every direction). On the other hand, when $\rho_i \geq 2.5$, we quantize $\widehat{\mathbf{C}}_i$ with 8 angles. Using $\widetilde{\mathbf{C}}_i^{\text{naive}}$, we obtain the closest covariance matrix $\widetilde{\mathbf{C}}_i$ from the table. I.e.,



(a) varying the angle parameter



(b) varying the elongation parameter



(c) varying the scaling parameter

Fig. 8 The graphical relationship between the steering kernel parameters and the values of covariance matrix.

Table 1 A compact lookup table for covariance matrices.

c_{11}	c_{12}	c_{22}	ρ_j	θ_j
1.0000	0	1.0000	1.0	0
2.5000	0	0.4000	2.5	0
2.1925	1.0253	0.7075	2.5	$\frac{1}{8}\pi$
1.4500	1.4500	1.4500	2.5	$\frac{2}{8}\pi$
0.7075	1.0253	2.1925	2.5	$\frac{3}{8}\pi$
0.4000	0	2.5000	2.5	$\frac{4}{8}\pi$
0.7075	-1.0253	2.1925	2.5	$\frac{5}{8}\pi$
1.4500	-1.4500	2.1925	2.5	$\frac{6}{8}\pi$
2.1925	-1.0253	0.7075	2.5	$\frac{7}{8}\pi$

$$\widetilde{\mathbf{C}}_i = \underset{\rho_j,\theta_j}{\arg\min} \left\| \mathbf{C}(\rho_j,\theta_j) - \widetilde{\mathbf{C}}_i^{\text{naive}} \right\|_{\text{F}}, \tag{38}$$

where $\|\cdot\|_{\text{F}}$ is the Frobenius norm. The final matrix $\widehat{\mathbf{C}}_i$ is given by:

$$\widehat{\mathbf{C}}_i = \gamma_i \widetilde{\mathbf{C}}_i. \tag{39}$$

4.4 Adaptive Regression Order

As mentioned earlier, although the kernel estimator with a higher regression order preserves high frequency components, the higher order requires more computation. In this section, we discuss how we can reduce the computational complexity, while enabling adaptation of the regression order. According to [7], the second order equivalent kernel, $\mathbf{W}_2$, can be obtained approximately from the zeroth order one, $\mathbf{W}_0$, as follows. First, we know that the general kernel estimator (12) can be expressed as:

$$\hat{z}(\mathbf{x}) = \mathbf{e}_1^T \left(\mathbf{X}^T\mathbf{K}\mathbf{X}\right)^{-1} \mathbf{X}^T\mathbf{K}\,\mathbf{y} = \mathbf{W}_{\!N}\mathbf{y} \tag{40}$$

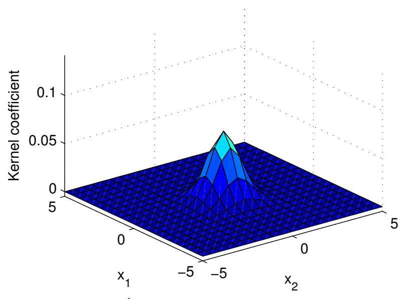
where again $\mathbf{W}_{\!N}$ is a $1 \times P$ vector containing the filter coefficients and which we call the equivalent kernel. The zeroth order equivalent kernel can be modified into $\mathbf{W}_2$ by

$$\widetilde{\mathbf{W}_2^T} = \mathbf{W}_0^T - \kappa\mathbf{L}\mathbf{W}_0^T, \tag{41}$$

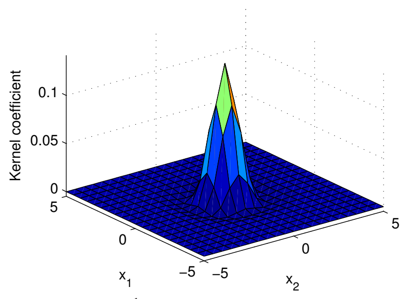
where $\mathbf{L}$ is Laplacian kernel in matrix form (we use $[1,1,1;1,-8,1;1,1,1]$ as a discrete Laplacian kernel) and κ is a regression order adaptation parameter. This operation can be seen to "sharpen" the equivalent kernel, and is equivalent to sharpening the reconstructed image. Fig. 9 shows the comparison between the actual second order equivalent kernel, $\mathbf{W}_2$, and the equivalent kernel, $\widetilde{\mathbf{W}_2}$, given by (41). In the comparison, we use the Gaussian function for K, and compute the zeroth order and the second order equivalent kernels shown in Fig. 9(a) and (b) respectively. The equivalent kernel, $\widetilde{\mathbf{W}}_2$, is shown in Fig. 9(c), and Fig. 9(d) shows the horizontal cross section of $\mathbf{W}_0$, $\mathbf{W}_2$, and $\widetilde{\mathbf{W}}_2$. As seen in Fig. 9(d), $\widetilde{\mathbf{W}}_2$ is close to the exact second order kernel $\mathbf{W}_2$.

There are two advantages brought by (41): (i) The formula simplifies the computation of the second order equivalent kernels, i.e. there is no need to generate the basis matrix, $\mathbf{X}$, or take inversion of matrices. (ii) Since the effect of the second order regression is now explicitly expressed by $\kappa\mathbf{L}\mathbf{W}_0$ in (41), the formulation allows for adjustment of the regression order across the image, but also it allows for "fractional" regression orders, providing fine control over the amount of sharpening applied locally.

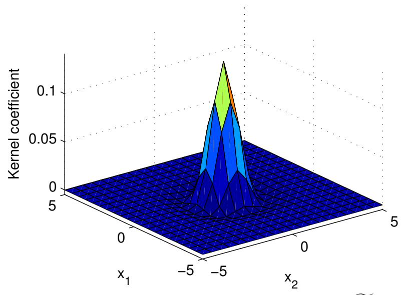
We propose a technique to automatically select the regression order parameter (κ) adaptively as follows. By setting κ near zero in flat regions and to a large value in edge and texture regions, we can expect a reduction of computational complexity, prevent amplifying noise component in flat regions, and preserve or even enhance texture regions and edges. In order to select spatially adapted regression factors, we



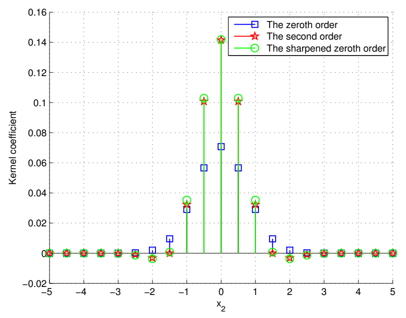
(a) The 0th order equivalent kernel ($\mathbf{W}_0$)



(b) The 2nd order equivalent kernel ($\mathbf{W}_2$)



(c) A sharpened 0th order equivalent kernel ($\widetilde{\mathbf{W}}_2$)



(d) Horizontal cross sections of the equivalent kernels

Fig. 9 Equivalent kernels given by classic kernel regression: (a) the 0th order equivalent kernel with the global smoothing parameter $h = 0.75$, (b) the 2nd order equivalent kernel ($\mathbf{W}_2$) with $h = 0.75$, (c) a sharpened 0th order equivalent kernel ($\widetilde{\mathbf{W}}_2$) with a 3×3 Laplacian kernel ($\mathbf{L} = [1,\ 1,\ 1;\ 1,\ -8,\ 1;\ 1,\ 1,\ 1]$) and $\kappa = 0.045$, and (d) Horizontal cross sections of the equivalent kernels $\mathbf{W}_0$, $\mathbf{W}_2$, and $\widetilde{\mathbf{W}}_2$. For this example, we used a Gaussian function for $K(\cdot)$.

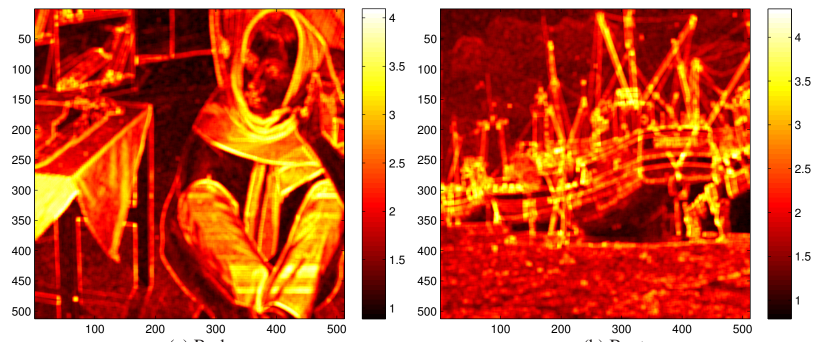
can make use of the scaling parameter γ_i, which we earlier used to normalize the covariance matrix in (37). This makes practical sense since γ_i is high in texture and edge areas and low in flat area as shown in Fig. 10. Because γ_i is already computed when computing the steering matrices, no extra computation is required. A good way to choose the regression factor (κ) locally is to make it a simple function of γ_i. Specifically, we choose our adaptive regression factor by

$$\kappa_i = 0.01\gamma_i, \tag{42}$$

where 0.01 is a global parameter controlling the overall sharpening amount. E.g. it is possible to choose a larger number if a stronger sharpening effect is desired globally. As shown in Fig. 10, with the choice of the adaptive regression order $\kappa_i = 0.01\gamma_i$ (42), the regression order becomes close to zero in the area where γ_i is close to zero, while the resulting equivalent kernel given by (41) approximately becomes a second order kernel in the area where γ_i is around 5. Setting κ too large results in overshoot of pixel values around texture and edges. We process color video in the YCbCr domain and estimate spatial orientations in the luminance component only, since the human visual system is most sensitive to orientations in the luminance component.

5 Example Video Upscaling and Denoising Results

In this section, we provide video frames generated by the proposed MASK algorithm as visual illustrations of its performance. We will provide examples of spatial upscaling, temporal frame interpolation, and denoising. We compare MASK to two other state-of-the-art multi-frame video upscaling methods: Non Local-mean



(a) Barbara (b) Boat

Fig. 10 Local scaling parameters (γ_i) for (a) Barbara image and (b) Boat image. With the choice of the adaptive regression order $\kappa_i = 0.01\gamma_i$ (42), the regression order becomes nearly zero in the areas where γ_i is close to zero, while in areas where γ_i is around 5, the resulting equivalent kernel given by (41) approximately becomes second order.

based super resolution [18] and 3-D iterative spatial steering kernel regression (3-D ISKR) [24]. The algorithm proposed in [18] consists of multi-frame fusion with Non Local-mean based weighting, as well as explicit deblurring. 3-D ISKR is an algorithm closely related to MASK involving iterative 3-D steering kernel regression; however, it does not require accurate (subpixel) motion estimation. For 3-D ISKR and MASK, we set the temporal window of support 5, and NL-based SR approach searches similar local patches across all the frames in time and the window of support 21×21 in space.

The first example shown in Fig. 11 is a visual comparison of spatial upscaling and temporal frame interpolation results, using MASK, NL-mean based SR, and 3-D ISKR. For this example, we used the Car-phone video sequence in QCIF format (144×176 pixels, 30 frames) as input, and spatially upscaled the video with an upscaling factor of $1:3$. Fig. 11(a) shows the input frame at time $t = 25$ (upscaled by pixel-replication). The upscaled results by single frame bicubic interpolation, NL-mean based SR, 3-D ISKR, and MASK are shown in Figs. 11(b)-(f), respectively. In addition, Fig. 12 shows a spatiotemporal upscaling example (both spatial upscaling and temporal frame interpolation) of the Car-phone sequence by 3-D ISKR and MASK. For this example, we estimated an intermediate frame at time $t = 25.5$ as well as spatially upscaling the intermediate frames with the upscaling factor of $1:3$. Comparing to the result by bicubic interpolation, all the adaptive methods, NL-mean based SR, 3-D ISKR, and MASK, reconstruct high-quality upscaled frames, although each has a few artifacts: jaggy artifacts on edge regions for NL-mean based SR and MASK, and overshooting artifact for 3-D ISKR.

The second example is spatio-temporal video upscaling using two color real video sequences: Spin-Calendar (504×576 pixels, 30 frames) and Texas (504×576 pixels, 30 frames). Fig. 13(a) and 14(a) show an input frame of each sequence at time $t = 5$, respectively. Spin-Calendar has relatively simple motions, namely rotations. Texas sequence contains more complicated motions, i.e., occlusion, 3-D rotation of human heads, and reflection on the helmet. Furthermore, Spin-Calendar contains camera noise, while Texas contains significant compression artifacts (e.g. blocking). Video frames that were spatially upscaled by a factor of $1:2$ using single frame bicubic interpolation, 3-D ISKR, and MASK are shown in Figs. 13(b)-(d) and 14(b)-(d), respectively. Also, Figs. 13(e)-(h) and 14(e)-(h) show selected portions of the input frame, the upscaled frame using single frame bicubic interpolation, 3-D ISKR, and MASK at a large scale. Next, we estimated an intermediate frame at time $t = 5.5$ for both Spin-Calendar and Texas sequences by 3-D ISKR and MASK, and the results are shown in Fig. 15. The intermediate frames are also spatially upscaled by the same factor ($1:2$). Again, both 3-D ISKR and MASK produce high quality frames in which camera noise and blocking artifacts are almost invisible while the important contents are preserved.

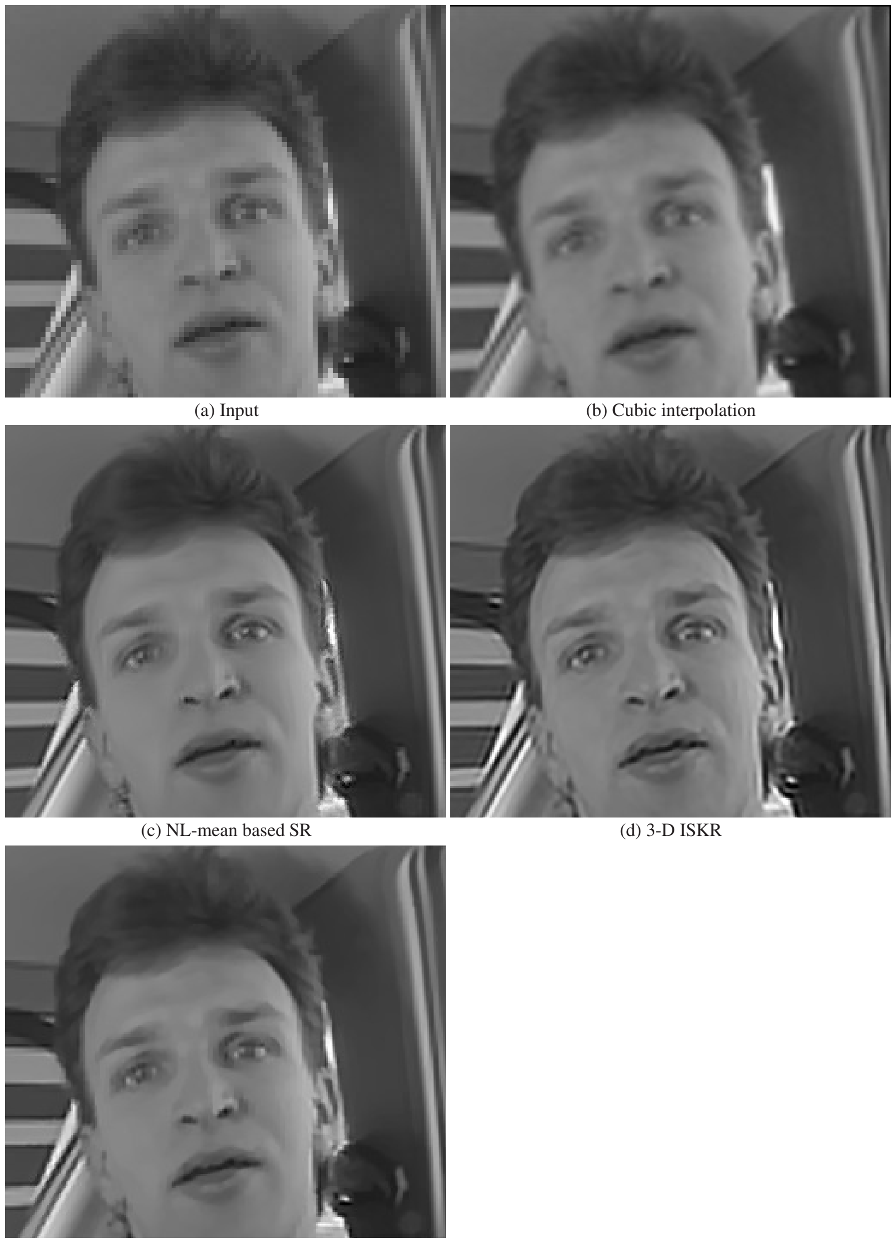


Fig. 11 Spatial upscaling of Car-phone video sequence: (a) input video frames at time $t = 25$ (144×176, 30 frames) and (b)-(f) the upscaled frames by single frame bicubic interpolation, NL-mean based SR [18], 3-D iterative SKR [24], and MASK, respectively.

6 Conclusion

In this chapter, we presented an extension of steering kernel regression for video upscaling. Our proposed algorithm is capable of spatial upscaling with resolution enhancement, temporal frame interpolation, noise reduction, as well as sharpening. In the proposed algorithm, we construct 3-D kernels based on local motion vectors, unlike our previous work [18, 24]. The algorithm includes motion estimation, but doesn't use explicit motion compensation. Instead, the spatio-temporal kernel is oriented along the local motion trajectory, and subsequent kernel regression acts directly on the pixel data. In order to avoid introducing artifacts due to motion estimation errors, we examine the motion vectors for their reliability. We apply a temporal weighting scheme, which allows us to suppress data from neighboring frames in the case of a motion error. Also, we reduce the computational cost of MASK by using a block-based motion model, using a quantized set of local orientations, and adapting the regression order. The adaptive regression order technique not only reduces the computational cost, but also provides sharpening while avoiding noise amplification.

We have presented several video upscaling examples showing that the MASK approach recovers resolution, suppresses noise and compression artifacts, and is capable of temporal frame interpolation with very few artifacts. The visual quality of the upscaled video is comparable to that of other state-of-the-art multi-frame upscaling methods, such as the Non-Local-Means based super-resolution method [18] and 3-D ISKR [24]. However, the computational complexity of MASK in terms of processing and memory requirements is significantly lower than these alternative methods. In order to improve the visual quality of MASK further, it may be nec-

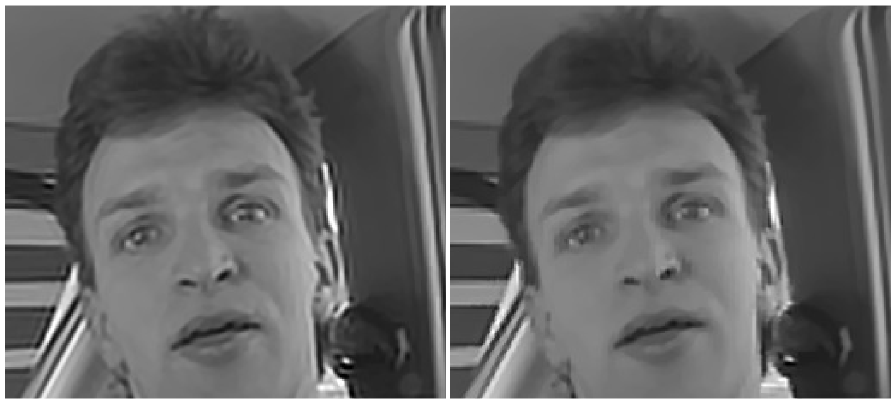


Fig. 12 Spatiotemporal upscaling of Car-phone video sequence: (a) upscaled frames by 3-D iterative SKR [24] at $t = 25.5$, and (b) upscaled frames by MASK at $t = 25.5$. In this example, we upscale Car phone sequence shown in Fig. 11(a) with the spatial upscaling factor $1:3$ and the temporal upscaling factor $1:2$.

essary to include more accurate motion estimation, for example by using smaller block sizes (currently 8×8), or extending the motion model, e.g. to an affine model.

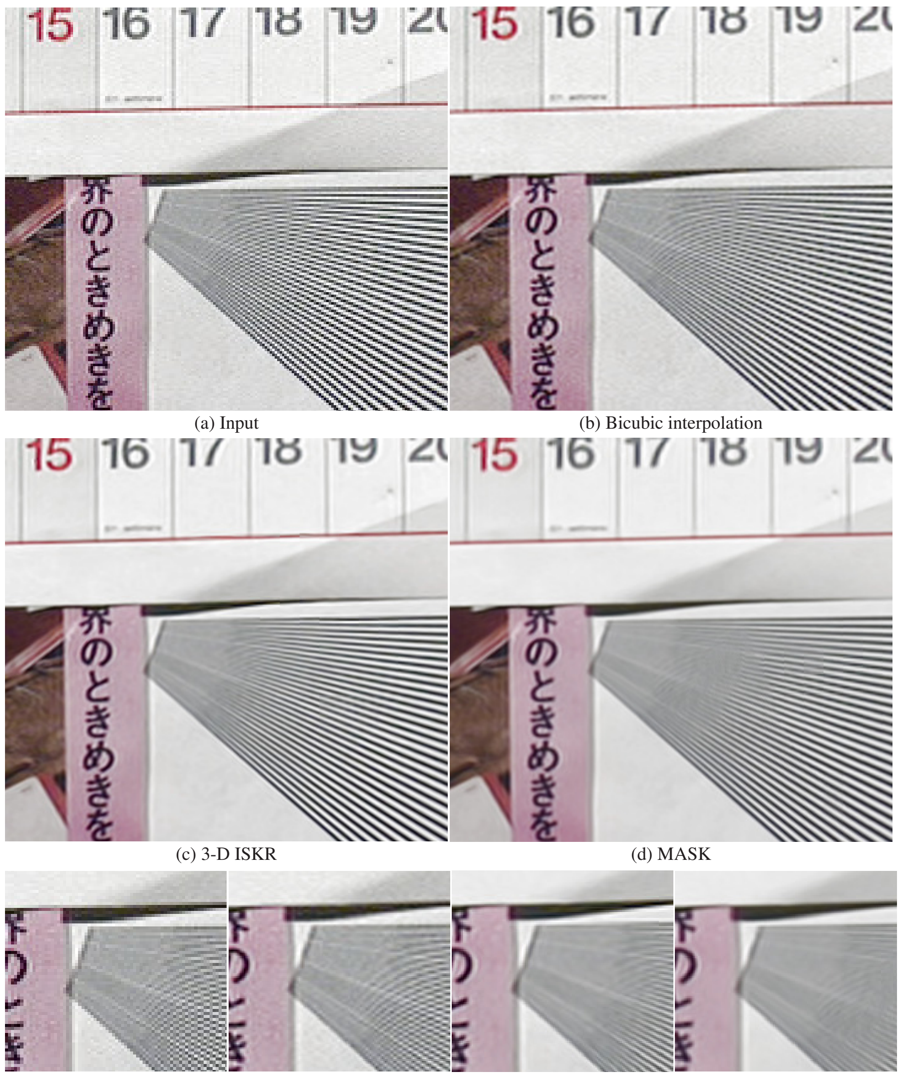


Fig. 13 Spatial upscaling of Spin-Calendar video sequence: (a) the input frame at $t = 5$, (b)-(d) the upscaled video frames by bicubic interpolation, 3-D ISKR, and MASK, respectively. (e)-(h) Enlarged images of the input frame and the upscaled frames by cubic interpolation, 3-D ISKR, and MASK, respectively.

References

1. Buades, A., Coll, B., Morel, J.M.: A Review of Image Denoising Algorithms, with a New One. In: Proc. Multiscale Modeling and Simulation, Society for Industrial and Applied Math-

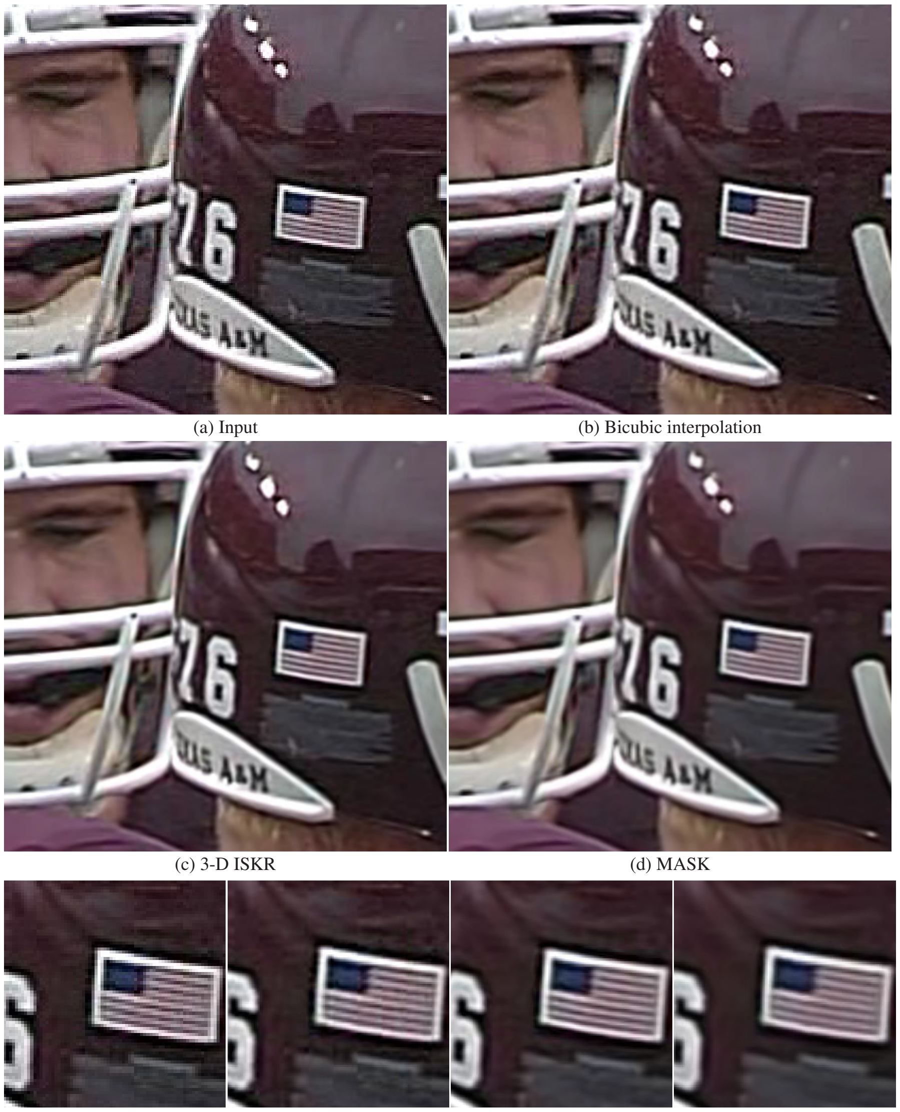


Fig. 14 Spatial upscaling of Texas video sequence: (a) the input frame at $t = 5$, (b)-(d) the upscaled video frames by bicubic interpolation, 3-D ISKR, and MASK, respectively. (e)-(h) Enlarged images of the input frame and the upscaled frames by cubic interpolation, 3-D ISKR, and MASK, respectively.

ematics (SIAM) Interdisciplinary Journal, New Orleans, LA. USA (2005)
2. Choi, B., Han, J., Kim, C., Ko, S.: Motion-Compensated Frame Interpolation Using Bilateral Motion Estimation and Adaptive Overlapped Block Motion Compensation. IEEE Transactions on Circuits and Systems for Video Technology, 17(4), 407-416 (2007)

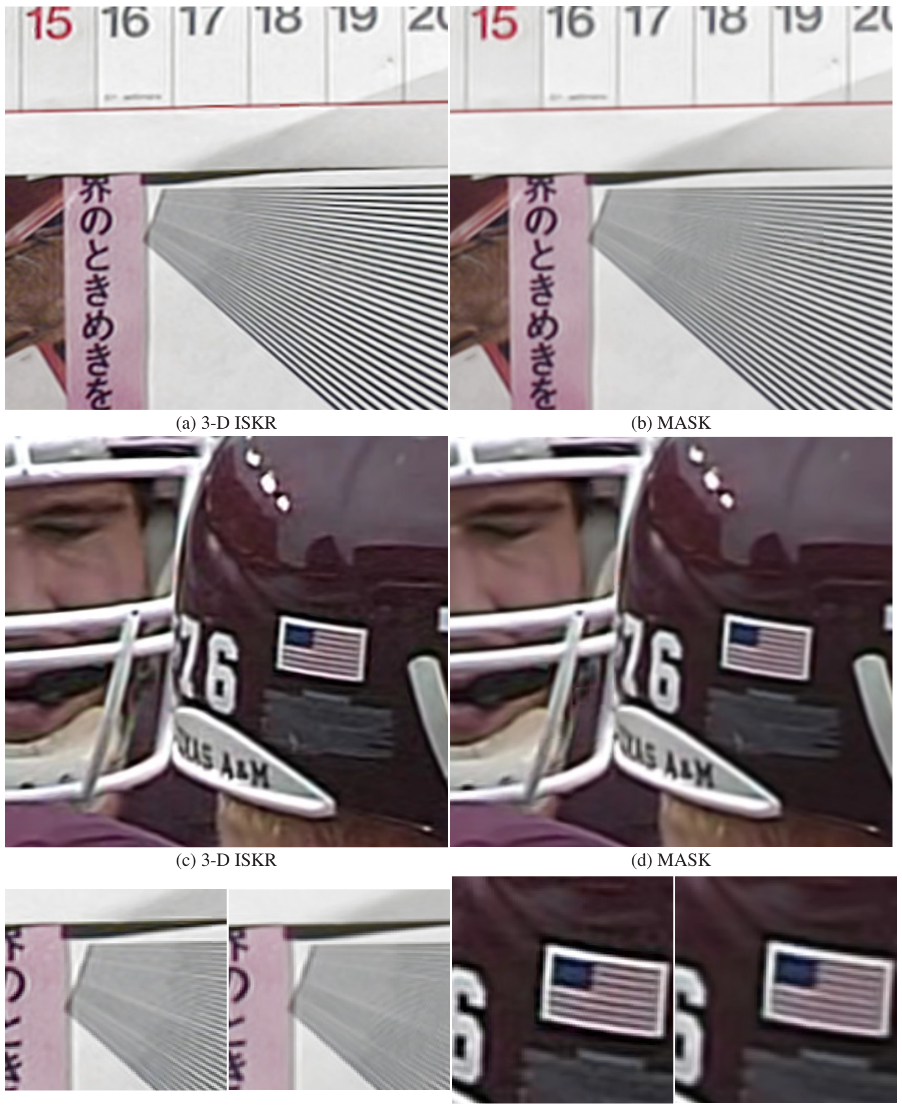


Fig. 15 Spatiotemporal upscaling of Spin-Calender and Texas video sequences: (a),(c) the estimated intermediate frames at time $t = 5.5$ by 3-D ISKR, (b),(d) the estimated intermediate frames by MASK. The frames are also spatially upscaled with the upscaling factor of 1 : 2. The images in (e)(f) and (g)(h) are the enlarged images of the upscaled frames by 3-D ISKR and MASK, respectively.

3. Elad, M.: On the Origin of the Bilateral Filter and Ways to Improve it. IEEE Transactions on Image Processing, 11(10), 1141-1150 (2002)
4. Farsiu, S., Robinson, D., Elad, M., Milanfar, P.: Advances and Challenges in Super-Resolution. International Journal of Imaging Systems and Technology, Special Issue on High Resolution Image Reconstruction (invited paper), 14(2), 47-57 (2004)
5. Fujiwara, S., Taguchi, A.: Motion-Compensated Frame Rate Up-Conversion Based on Block Matching Algorithm with Multi-Size Blocks. In: Proc. International Symposium on Intelligent Signal Processing and Communication Systems, Hong Kong, China (2005)
6. Gersho, A., Gray, R.M.: Vector Quantization and Signal Compression. Kluwer Academic Publishers, Boston (1992)
7. Haralick, R.M.: Edge and Region Analysis for Digital Image Data. Computer Graphic and Image Processing (CGIP), 1(12), 60-73 (1980)
8. Hardie, R.: A Fast Image Super-Resolution Algorithm Using an Adaptive Wiener Filter. IEEE Transactions on Image Processing, 16(12), 2953-2964 (2007)
9. Huang, A., Nguyen, T.Q.: A Multistage Motion Vector Processing Method for Motion-Compensated Frame Interpolation. IEEE Transactions on Image Processing, 17(5), 694-708 (2008)
10. Kang, S., Cho, K., Kim, Y.: Motion Compensated Frame Rate Up-Conversion Using Extended Bilateral Motion Estimation. IEEE Transactions on Consumer Electronics, 53, 1759-1767 (2007)
11. Lucas, B., Kanade, T.: An Iterative Image Registration Technique with an Application to Stereo Vision. In. Proc: DARPA Image Understanding Workshop. (1981)
12. Mitchell, D.P., Netravali, A.N.: Reconstruction Filters in Computer Graphics. Computer Graphics, 22(4), 221-228 (1988)
13. Nadaraya, E.A.: On Estimating Regression. Theory of Probability and its Applications, 141-142 (1964)
14. Narayanan, B., Hardie, R.C., Barner, K.E., Shao, M.: A Computationally Efficient Super-Resolution Algorithm for Video Processing Using Partition Filters. IEEE Transactions on Circuits and Systems for Video Technology, 17(5), 621-634 (2007)
15. Ozkan, M., Sezan, M.I., Tekalp, A.M.: Adaptive Motion-Compensated Filtering of Noisy Image Sequences. IEEE Transactions on Circuits and Systems for Video Technology, 3(4), 277-290 (2003)
16. Park, S.C., Park, M.K., Kang, M.G.: Super-Resolution Image Reconstruction: A Technical Overview. IEEE Signal Processing Magazine, 20(3), 21-36 (2003)
17. Pham, T.Q., van Vliet, L.J., Schutte, K.: Robust Fusion of Irregularly Sampled Data Using Adaptive Normalized Convolution. EURASIP Journal on Applied Signal Processing, 1-12 (2006)
18. Protter, M., Elad, M., Takeda, H., Milanfar, P.: Generalizing the Non-Local-Means to Super-resolution Reconstruction. IEEE Transactions on Image Processing, 16(2), 36-51 (2009)
19. Stiller, C., Konrad, J.: Estimating Motion in Image Sequences - A Tutorial on Modeling and Computation of 2D Motion. IEEE Signal Processing Magagine, 16(4), 70-91 (1999)
20. Takeda, H., van Beek, P., Milanfar, P.: Spatio-Temporal Video Interpolation and Denoising Using Motion-Assisted Steering Kernel (MASK) Regression. In: Proc. IEEE International Conference on Image Processing (ICIP). San Diego, CA, USA (2008)
21. Takeda, H., Farsiu, S., Milanfar, P.: Robust Kernel Regression for Restoration and Reconstruction of Images from Sparse Noisy Data. In: Proc. International Conference on Image Processing (ICIP). Atlanta, GA, USA (2006)
22. Takeda, H., Farsiu, S., Milanfar, P.: Kernel Regression for Image Processing and Reconstruction. IEEE Transactions on Image Processing, 16(2), 349-366 (2007)
23. Takeda, H., Farsiu, S., Milanfar, P.: Deblurring Using Regularized Locally-Adaptive Kernel Regression. IEEE Transactions on Image Processing, 17(4), 550-563 (2008)
24. Takeda, H., Milanfar, P., Protter, M., Elad, M.: Superresolution without Explicit Subpixel Motion Estimation. IEEE Transactions on Image Processing, 18(9), 1958-1975 (2009)
25. Tomasi, C., Manduchi, R.: Bilateral Filtering for Gray and Color Images. In: Proc. IEEE International Conference of Compute Vision. Bombay, India (1998)
26. Wand, M.P., Jones, M.C.: Kernel Smoothing. Chapman and Hall. London; New York (1995)