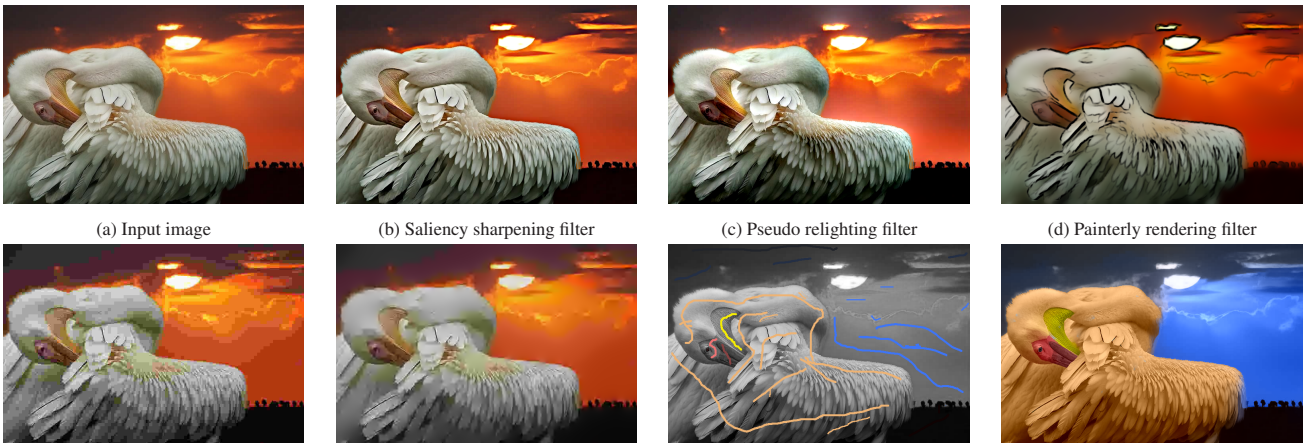
A Perceptually-Motivated Optimization-Framework for Image and Video Processing

Pravin Bhat[1] C. Lawrence Zitnick[2] Michael Cohen[1,2] Brian Curless[1]
[1]University of Washington [2]Microsoft Research



(a) Input image (b) Saliency sharpening filter (c) Pseudo relighting filter (d) Painterly rendering filter

(e) Compressed input image (f) Improved de-blocking filter (g) User input for colorization (h) Improved colorization filter

Figure 1: The figure shows some of the image enhancement filters we have created or improved upon using our optimization framework for image processing. Our formulation is designed for expressing image and video processing applications that can account for certain perceptual biases of the human vision system (HSV).

Abstract

We present an optimization framework for expressing image processing applications that can account for certain perceptual biases of the human vision system (HVS). Perception literature is ripe with studies demonstrating the HVS to be more sensitive to pixel gradients than absolute pixel values, which has led to some important work in gradient domain image filtering. Inspired by this work, our optimization framework allows image and video processing applications to easily specify both zeroth order constraints (i.e., desired pixel values) and first order constraints (i.e., desired pixel gradients in space and time) in the optimization. We introduce a spatially-varying weighting scheme for these constraints that produces perceptually pleasing results by approximating the more robust L_1-norm even when using a simple weighted least squares optimization. We also demonstrate that edge length in addition to local gradient magnitude is a useful measure of local gradient saliency. Our saliency measure is inspired by perception studies that show long coherent edges in an image, even when faint, are perceptually salient to the HVS.

Finally, we demonstrate the utility of our formulation in creating effective yet simple to implement solutions for common image processing tasks. To exercise our formulation we have created a new saliency-based sharpen filter and a pseudo image relighting application. We also revisit and improve upon filters previously defined by the gradient domain community – filters like painterly rendering, image de-blocking, and sparse data interpolation over images (e.g., colorization using optimization).

1 Introduction

To be effective image processing algorithms should account for the perceptual biases of the human visual system (HVS). In fact, most image enhancement applications in the graphics community can be binned into one of three broad categories based on the perceptual task they perform:

Content enhancement: Applications in this category enhance image content in a manner that is either beneficial or pleasing to the HVS. Examples include: sharpening, selective emphasis/de-emphasis of content in order to draw attention to the subject matter [Su et al. 2005], aesthetic enhancements such as non-photorealistic stylization [Orzan et al. 2007], and so on.

Artifact suppression: These image filters are used to hide artifacts that are distracting to the HVS. Examples include denoising, deblurring, video deflickering, suppression of compression artifacts, and hiding transition seams in image mosaics [Pérez et al. 2003].

Content preserving transformations: Image filters in this category involve transforming an image while preserving perceptual characteristics that are important to the HVS. For example, tone mapping [Fattal et al. 2002] involves transforming HDR images to LDR while preserving local contrast. The Color2Gray algorithm introduced by Gooch et al. [2005] transforms color images to grayscale by removing color from images while preserving color contrast.

In this paper, we present an optimization framework for expressing image processing applications that can account for certain perceptual biases of the human vision system (HVS). Our formulation is greatly inspired by the recent work on gradient domain image filtering where direct manipulation and use of image *gradients* has played a central role. These methods rely on the fact that gradients are integral to the way in which we perceive images. Research in human perception indicates that the HVS has lower sensitivity to absolute pixel values, and instead relies upon local contrast (edges) and ratios [Attneave 1954; Barten 1999], which more directly correlate with gradients in an image. Inspired by this body of work, our optimization framework allows image and video processing appli-

cations to easily specify both zeroth order constraints (i.e., desired pixel values) and first order constraints (i.e., desired pixel gradients in space and time) in the optimization.

In addition, we present a new measure for local gradient saliency inspired by perception studies that have shown long coherent edges in an image, even when faint, are perceptually salient to the HVS [Beaudot and Mullen 2003; Elder and Goldberg 2001]. In previous methods the saliency of a local gradient is often approximated by its magnitude as in Lischinski et al. [2006] or by a response to a local filter as in Levin et al. [Levin et al. 2004]. However, certain pixel gradients, even when faint, give rise to long coherent edges which demarcate object boundaries, shadows, surface creases, reflectance changes, and other significant visual events. To account for the perceptual importance of such gradients, our framework provides applications with the length of underlying dominant edge at each pixel and its local orientation. Thus we encourage applications to use this edge-length information to better estimate the saliency of local gradients and pixels when processing them. Needless to say, an application may choose to ignore this additional information or augment the saliency measure by using an application specific saliency detector (e.g., a face detector).

Applications can also choose to exert further control over the result by specifying the weight of each individual constraint in the optimization, which can be used in many interesting ways. For example, we present a general weighting scheme that produces visually pleasing results for most applications by approximating the more robust L_1-norm when even using a simple weighted least squares optimization.

Finally, we show how several image processing tasks can be effectively expressed in our formulation. Among the many applications we explore include saliency sharpening, pseudo-relighting, de-blocking, sparse data interpolation over images (e.g., colorization), video de-flickering and non-photorealistic rendering (NPR). Using our framework, most filters that can be applied to a single image can also be automatically applied to videos coherently by enforcing simple first-order constraints along flow lines.

In summary, our contributions include:

- an optimization framework for defining perceptually motivated image and video filters,
- a novel way of measuring local gradient saliency,
- a new constraint weighting scheme that improves results and provides more robust constraint satisfaction,
- a demonstration of our formulation in creating new, and improving upon existing, image and video processing applications.

2 Related work

Our work draws heavily from the rich body of work done in gradient domain image processing. We review some of the work in this literature that is most relevant to our optimization formulation. However, for an more extensive introduction to the gradient domain literature, the reader is referred to Agrawal and Raskar's [2007] excellent ICCV course on the topic.

One of the first gradient domain image filters was proposed by Fattal et al. [2002]. Their work casts the tonemapping problem as a pure gradient field integration problem (i.e., no zeroth order terms) by attenuating large scale gradients in a HDR image and then solving for a LDR image that best approximates this attenuated gradient field.

Perez et al. [2003] also used a pure gradient field integration approach to create seamless image composites. They modify the gradient field of a target image by overwriting it's gradients using gradients obtained from a source patch in a region selected by the user. This modified gradient field is then integrated while keeping pixel colors outside the selected region fixed using Dirichlet boundary conditions. Levin et al. [2003] used a similar approach for seamless image stitching. Levin et al. also showed that their work could be used for de-blocking compressed images. We present a similar method for image de-blocking (Section 5.6) that additionally includes zeroth order terms in the optimization to significantly improve the de-blocking quality.

Lischinski et al. [2006] generalized Levin's technique for colorizing grayscale images [Levin et al. 2004] using a mixture of both zeroth order and first order terms in the optimization. Lischinkski's method allows the user to draw a small number of brush strokes to specify local edits, such as modifications to colors, tonal values, and white balance of the image. This user-specified sparse-data is then interpolated over the image in a piecewise smooth fashion with respect to the underlying gradient field of the luminance image. We present a simple improvement to this method (Section 5.7) that significantly reduces data bleeding by using our edge-length based measure for local gradient saliency.

In their Color2Gray paper, Gooch et al. [2005] demonstrate a rather interesting application of gradient domain techniques. They investigate the problem of converting a color image to grayscale without sacrificing the color saliency that is often lost in the standard color to gray mapping (i.e., isoluminant colors mapping to the same gray value). This work uses no zeroth order terms in its optimization. However, unlike most gradient domain techniques, this technique employs more than two first-order terms for each pixel in the optimization (i.e., each pixel has a constraint defined with respect to a patch of pixels surrounding it).

Agrawal et al. [2005] have proposed a gradient projection technique to fuse gradients obtained from ambient and flash images in a manner that produces well-lit images without strong highlights. Agrawal et al. [2006] have further generalized this work to a class of edge-suppressing operations on images.

Orzan et al. [2007] used a gradient-based approach to convert photographs into painterly renditions that capture their salient features. They analyzed the multiscale output of the Canny edge detector to determine both edge importance (measured by its lifetime along the scale axis) and the characteristic edge scale. We also propose a painterly rendering filter in this paper that in comparison to Orzan's method uses zeroth order terms, a different edge saliency measure and is temporally consistent when applied to videos. A more detailed comparison is provided in Section 5.5.

The temporal constraints (i.e., first order constraints in time) used in our formulation are inspired by the work of Levin et al. [2004] and Bhat et al. [2007]. Bhat et al. showed that fusing temporal gradients defined along correspondence vectors from one video with the spatial gradients from another video can be used to combine the temporal characteristics of the former with the spatial characteristics of the later. We use similar motion-compensated temporal-constraints to encourage the temporal characteristics of the input video (e.g., temporal coherence, illumination changes) to be enforced in the filtered result. Thus, our formulation decouples the task of defining a new image filter from the task of applying that filter to a video coherently.

3 Optimization formulation for image processing

In this section we introduce our optimization formulation for image processing. In section 4 we extend this formulation to accommodate video processing. Our formulation draws heavily from the related works described in Section 2.

The task of an image processing application is to take an input image u and transform it into the final image f. Our formulation simplifies the task of writing image processing applications that can be expressed as an energy function involving zeroth and first order terms of the image f (i.e., Equation 2). For each pixel in f the application is allowed to specify a single zeroth order constraint (i.e., desired pixel value) and two first order constraints (i.e., desired pixel gradients). The application is also allowed to specify a weight for each constraint in the optimization.

An application that wishes to use our formulation has to define a function of the following form:

$$F(u, \cdots) \rightarrow [d, g, w] \tag{1}$$

Inputs: The function F takes as input the unfiltered image u and any metadata (e.g., parameter values, selection masks, edge statistics) which F may choose to use in its computation. These application specific inputs to F will be described in further detail in the applications section (Section 5). The input image u may contain multiple channels (e.g., RGB, YUV, etc). However for simplicity of exposition we will treat u as a single channel image in this section since each channel in the result f is solved for independently in practice.

Outputs: The function F returns three images – $[d, g, w]$. The image d is a single channel image that provides the data constraint for each pixel in f. The image g is a two channel image where channels g^x and g^y specify the desired x-derivative and y-derivative of f respectively. The image w is a three channel image where channels w^d, w^x, and w^y provide the weights for constraints in d, g^x, and g^y respectively.

The final result f is generated by minimizing the following energy function:

$$E(f) = \sum_{p \in f} \lambda_d E_d(p) + E_g(p) \tag{2}$$

where p is a pixel in f, E_d is our data cost function, and E_g is our gradient cost function. The constant λ_d balances the tradeoff between fidelity to data versus gradient constraints. The energy terms E_d and E_g are quadratic functions defined as follows:

$$E_d(p) = \; w^d(p) \left[f(p) - d(p)\right]^2 \tag{3}$$

and

$$E_g(p) = w^x(p) \left[f_x(p) - g^x(p)\right]^2 + w^y(p) \left[f_y(p) - g^y(p)\right]^2 \tag{4}$$

Thus, the energy terms E_d and E_g are the squared errors between the desired values specified by F and the actual values of the final image f. Each constraint also has a corresponding weight, w^d for the data constraints and w^x and w^y for the gradient constraints. These weights control the amount of influence a constraint should have on the final image. As shown later, several effects can be achieved by varying these weights, including sparse data interpolation and the suppression of haloing artifacts. Individual weights are also commonly set to zero to completely remove the effect of the corresponding constraint.

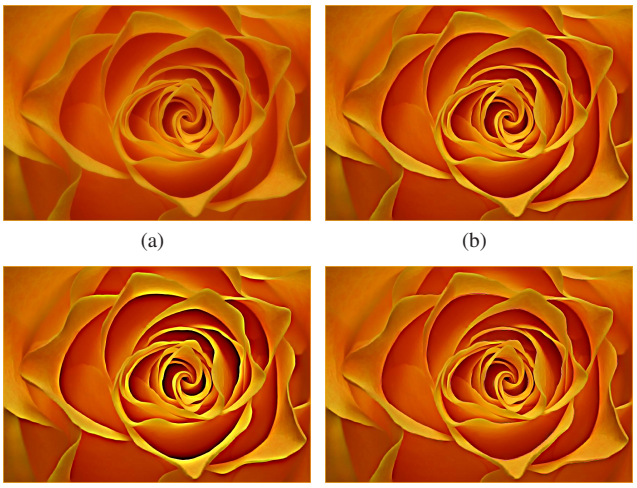


Figure 2: This figure shows the effect our robust weighting function has on the quality of the saliency sharpen filter defined in section 5.2. (a) Input image. (b) Image saliency sharpened using *robust* weighting. (c) Result with *uniform* weighting (notice the severe haloing artifacts). (d) IRLS result after solving ten weighted least-squares problems.

Since our energy function E is quadratic, its minima can be found using standard weighted least squares techniques like the conjugate gradient method [Shewchuk 1994]. To increase the runtime performance of the solver, various preconditioners may be used to better condition the optimization [Szeliski 2006]. Recently McCann and Pollard [2008] showed that a GPU accelerated conjugate gradient solver can minimize energy functions like ours in real-time for megapixel-sized images.

3.1 A simple sharpen Filter

To build the reader's intuition for image processing using zeroth and first order constraints and to provide further familiarity with our notation, in this subsection we will define a simple sharpen filter $F_{sharpen}$ using our formulation. This sharpen filter was first defined by Zeng et al. [2005] and can be shown to subsume the simple Laplacian sharpen filter (i.e, $f = u - \lambda \nabla^2 u$) commonly used in image processing. The outputs of $F_{sharpen}$ are defined as follows:

$$d = u; \; g^x = c_s \cdot u_x; \; g^y = c_s \cdot u_y;$$

$$w^d(p) = \lambda_d; \; w^x(p) = 1; \; w^y(p) = 1$$

Here, the parameter c_s is a scalar constant set to a value greater than one. The sharpening behavariour of $F_{sharpen}$ has an intuitive interpretation. That is, to increase the input image's local contrast $F_{sharpen}$ sets the desired gradients (i.e., g^x and g^y) to the gradients of the input image (i.e., u_x and u_y) multiplied by a factor, c_s, greater than one. The data constraints are set to the original image with uniform weighting, λ_d, to ensure the final result does not drift too far from the input. An example result of this sharpen filter can be seen in Figure 3.

3.2 A robust weighting scheme

Given the L_2-norm's sensitivity to large errors (i.e., outliers), the energy function E_g can produce visually unappealing results because it prefers to evenly distribute errors. This can result in haloing or pinching artifacts in regions where the desired gradient field g is hard to satisfy (see the example in Figure 2).

One solution to this problem is to use a more robust metric such as the L_1-norm, which would require the use of slower, more complicated optimization techniques like linear programming, or iteratively re-weighted least squares (IRLS).

Instead, we introduce an alternative technique that involves solving a single weighted least squares problem. By applying the appropriate weights w^x and w^y to our gradient constraints, the visual artifacts mentioned above can be considerably mitigated. While an application may choose to define its own weights, we provide a default weighting function for the gradient constraints that works well for most applications.

Our default weighting function is based on the simple prior that the gradient field of f is likely to deviate from g (thus leading to large errors in the L_2-norm) in regions where g deviates *heavily* from the gradient field of u. By reducing the weights of these constraints we can lower their influence on the resulting image as follows:

$$w^x = \frac{1}{(a \cdot |u_x - g^x| + 1)^b} \tag{5}$$

$$w^y = \frac{1}{(a \cdot |u_y - g^y| + 1)^b} \tag{6}$$

Here the parameter a is set such that it normalizes the scale of image gradients (i.e., normalized gradient range = [-1, 1]), and parameter b controls the sensitivity of Equation 4 to outliers and is typically set between 3 and 5; both parameters do not vary spatially. Figure 2 demonstrates the effect this robust weighting scheme has on the visual quality of our saliency sharpening filter (described in Section 5.2).

In Figure 2, we also compare the visual quality of our fixed weighted least squares method to that produced by the iteratively re-weighted least squares (IRLS) method after it has solved ten weighted least-squares problems. The IRLS algorithm is a general method for minimizing a robust L_2 - L_1 norm (e.g., L_1-norm) by solving successive weighted least squares problems. The method begins by weighting each constraint uniformly and in each successive weighted least squares problem the solution from the previous problem is used to downweight the outliers. One interpretation of our weighting function is that it uses the structure of our problem to approximate the weights that an IRLS solver will arrive at upon convergence.

4 Generalization of the formulation to videos

To process an input video u using a filter function F defined using the image formulation in Section 3, one could apply F to each video frame independently. Unfortunately, when the result video f is generated using this approach it often looks temporally incoherent.

To alleviate this temporal incoherence problem we are going to use a technique proposed by Bhat et al. [2007], which is also similar to the technique produced by Levin et al. [2004]. Bhat et al. showed that fusing temporal gradients defined along correspondence vectors from one video with the spatial gradients from another video can be used to combine the temporal characteristics of the former with the spatial characteristics of the later. We use similar motion-compensated temporal-constraints to cause the temporal characteristics of the input video (e.g., temporal coherence, illumination changes) to be be enforced in the filtered result. Thus, our formulation decouples the task of defining a new image filter from the task of applying that filter to a video coherently.

In addition to the input video, we require a set of motion vectors (v_x, v_y) for each pixel as input. These vectors are used to define the temporal constraints in the optimization. Although optical flow remains a difficult research problem, we have empirically found that if good motion vectors are available for 50-60% of the pixels and confidence values are available for the motion vectors, then our method produces temporally coherent results. For most results shown in the supplementary video, we use the optical flow algorithm proposed by Sand et al. [Sand and Teller 2006] to generate motion vectors. For the streaming video results (Section 4.1) we rely on the blockwise motion vectors encoded in the video [Tomar 2006].

As in the image processing case, the application's filter function F is used to obtain the desired spatial constraints (e.g., d, g^x, g^y) for each video frame. However, in the video processing case, for every pixel in the video an additional first order constraint $g^{\bar{t}}$ is used to define the desired temporal gradient between the pixel and its motion compensated neighbour in the previous frame. The value of these temporal constraints $g^{\bar{t}}$ is set equal to a function F_t of the corresponding temporal gradient of the original video, i.e. $g^{\bar{t}}(p) = F_t(u_{\bar{t}}(p))$ (Equation 10).

The motion compensated gradients are defined as follows:

$$u_{\bar{t}} = u(x, y, t) - u(x + v_x,\, y + v_y,\, t - 1) \tag{7}$$

Here $(x + v_x,\, y + v_y,\, t - 1)$ is the pixel that corresponds to p in the previous video frame. (x, y, t) is the coordinate of p in u and (v_x, v_y) is a motion vector that maps p to its corresponding pixel in the previous video frame.

Adding these constraints to our energy function $E(f)$, we get:

$$E(f) = \sum_{p \in f} \lambda_d E_d(p) + E_g(p) + \lambda_t E_t(p). \tag{8}$$

Similar to the data and gradient energy functions, $E_t(p)$ is defined as:

$$E_t(p) = w^t(p) \left[f_{\bar{t}}(p) - g^{\bar{t}}(p) \right]^2. \tag{9}$$

$$g^{\bar{t}}(p) = F_t(u_{\bar{t}}(p)) \tag{10}$$

$w^t(p)$ controls the weight given to the temporal constraint at p; a typical choice is to set $w^t(p)$ to the confidence in the accuracy of p's motion vector. The function F_t is set to the identity function by default thus causing the final video to mimic the temporal behaviour of the input video. However, certain applications may choose to modify the behaviour of F_t to better suit their needs; See the video deflickering application in section 5.4 for an example.

4.1 Generalization to streaming videos

Though the energy function defined in Equation 8 can be optimized across an entire video, as videos increase in length this may become too computationally expensive. It may also be the case that a video is streaming and the entire video may not be available. In either of these cases the energy function may be minimized by stepping

through the video one frame at a time with the values of the previous time frame fixed. That is, frame $t-1$ is first computed. Its pixel's values are then held fixed and frame t is computed. The initial frame can be computed without using temporal constraints. In this paper, only the de-blocking results for streaming *YouTube* videos were created using this approach (See supplementary video).

5 Applications and improvements

In this section we present new applications we have developed and a few previously defined applications that we have improved using our perceptually motivated formulation for image and video processing. Each of these applications was written in less than two hundred lines of C++ code using our image processing API[1]. We hope that these simple to implement, yet effective, applications will demonstrate just how intuitive and simple the solution to a perceptual image processing task can be when tackled using our formulation.

Note that all applications defined in this section, unless explicitly stated otherwise, use our robust weighting function (Section 3.2, equations 5 & 6) for defining the gradient constraint weights (i.e., w^x & w^y). Before we delve into the details of the various filters we first need to discuss our method for detecting gradient saliency since it is used by a few of our applications.

5.1 Measuring local gradient saliency

We present a new measure for local gradient saliency inspired by perception studies that have shown long coherent edges in an image, even when faint, are perceptually salient to the HSV [Beaudot and Mullen 2003; Elder and Goldberg 2001]. To account for the perceptual importance of gradients that give rise to these edges, our framework provides a long edge detector that applications can use to measure the saliency of a local gradient.

Our long edge detector detects long, coherent edges instead of simply detecting edges with a strong magnitude. The edge detector returns an image e with two channels: e^l and e^o. The e^l channel provides length of the dominant edge running through each pixel in the input image. The e^o channel provides local orientation of the dominant edge at each pixel. The long edge detector works as follows. Local edges are first detected using Freeman and Adelsons [1991] steerable filter. The edge magnitudes are then normalized in a 5×5 spatial window. Edge lengths are computed for every pixel using a message passing scheme in which the sum of the normalized edge magnitudes are summed in both directions parallel to the edge direction. The messages are weighted based on the similarity of the neighbour's edge orientation. The implementation details of our edge detector can be found in the tech report submitted as supplementary material. One could also approximate our continuously valued long edge detector by simply running the Canny edge detector [Canny 1986] on the input image and then using a flood fill algorithm at each pixel to estimate the length of the dominant edge running through the pixel.

5.2 Smart sharpening

Sharpening is one of the most commonly used image enhancement filters. Unfortunately the simple sharpen filter (see Section 3.1) intensifies all gradients in an image including gradients that give rise to noise and background texture. A better sharpen filter would only intensify those gradients that give rise to salient image features. Our saliency sharpen filter uses the long edge detector to only boost the magnitude of gradients that lie *across* long edges thus enhancing image saliency without adding to image noise or background clutter. Our saliency sharpen filter is defined as follows:

$$F_{saliency\_sharpen}(u, e) \rightarrow [d, g, w]$$

$$d(p) = u(p); w^d(p) = c_1$$

$$g^x(p) = u_x(p) + c_2 \cdot u_x(p) \cdot e^l(p) \cdot cos(e^o(p))$$

$$g^y(p) = u_y(p) + c_2 \cdot u_y(p) \cdot e^l(p) \cdot sin(e^o(p))$$

$F_{saliency\_sharpen}$ accepts as input the image to enhance (i.e, u) and the edge detector results for the image (i.e., e). The data constraints and the parameter c_1 ($c_1 > 0$) keep the enhanced image from drifting too far from the input. The desired gradient field g is defined to intensify the magnitude of gradients that lie *across* long edges and leave the other gradients unchanged. The parameter c_2 controls the overall sharpening amount ($c_2 > 0$). The term e^l (p) spatially varies the sharpening amount by the length of the underlying edge. Finally, e^o factors in local orientation of the underlying edge and thus ensures that the sharpening is done perpendicular to the local edge orientation.

(a) (b) (c)

Figure 3: A qualitative comparison of our saliency sharpen filter (Section 5.2) to the simple sharpen filter (Section 3.1). (a) Original image. (b) Simple sharpen result. (c) Saliency sharpen result. Notice how saliency sharpen enhances the image without boosting the noise or background texture.

See figure 3 and the supplementary video for a qualitative comparison of our saliency sharpen filter to the simple sharpen filter.

5.3 Pseudo image relighting

Image relighting is the process of estimating what an image would have looked like had it been captured under different lighting conditions. Previous relighting algorithms rely on estimating scene geometry in order to produce photorealistic lighting effects [Marschner and Greenberg 1997]. Instead, our relighting filter is inspired from the observation that digital artists can often create pseudo relighting effects by cleverly adding a few handcrafted intensity ramps onto the original image (e.g., Figure 4f). Our relighting filter allows the user to specify a new lighting direction on the image plane and then it simply boosts all intensity gradients that happen to be oriented along the specified direction. Integrating the gradient field modified in this manner creates the desired lighting effect by intensifying preexisting ramps in the image that happen to be aligned with the desired lighting-direction. As a result, the

[1] We plan to release the source code for our image processing API and the filters presented in this section.

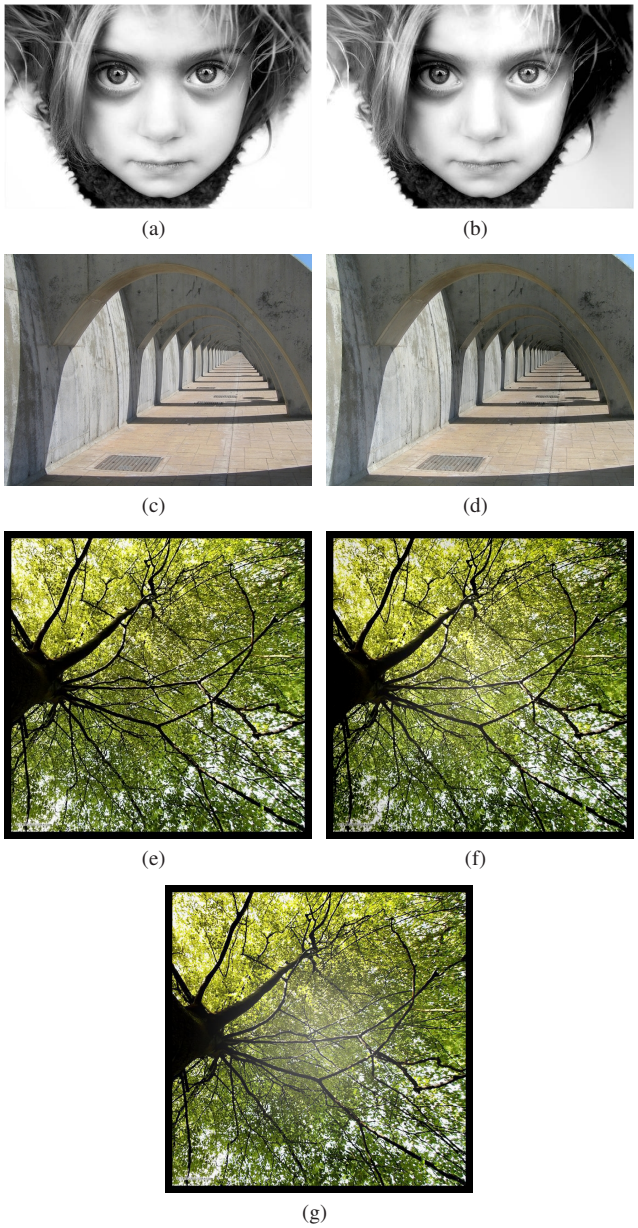


Figure 4: The figure shows some of the pseudo relighting effects created using the function $F_{relight}$. (a) Input image. (b) Image relit to simulate an additional light source to the west of the face. The effect is achieved by setting the local lighting direction for every pixel (i.e., $o(p)$ in $F_{relight}$) to point west. (c) Input image. (d) Image relit to simulate the light fading into the vanishing point, thus adding more depth to the image. The effect is achieved by setting the local lighting direction for every pixel to point away from the vanishing point. (e) Input image. (f) Image relit to simulate overhead sun light. (e) The same relighting effect attempted in Photoshop by using a radial intensity ramp. Notice that our result (f) looks more realistic in comparison to the digital artist's result (e).

relit image looks natural even though the relighting is done without computing any scene geometry. The formal definition of our pseudo relighting filter is as follows:

$$F_{relight}(u, o) \rightarrow [d, g, w]$$

Here, $F_{relight}$ accepts as input the image to relight (i.e., u) and an image containing the desired lighting angle for each pixel (i.e., o). We could have used a single angle parameter instead of o, which is an image of angles. However, by allowing the light direction to vary spatially $F_{relight}$ can be used to create a variety of relighting effects as shown in Figure 4. The following are the definitions used by $F_{relight}$ to produce $[d, g, w]$:

$$d(p) = u(p);\ w^d(p) = c_1$$
$$g^x(p) = u_x(p) + c_2 \cdot u_x(p) \cdot a(p)$$
$$g^y(p) = u_y(p) + c_2 \cdot u_y(p) \cdot a(p)$$
$$a(p) = \frac{u_x(p) \cdot cos(o(p)) + u_y(p) \cdot sin(o(p)))}{\sqrt{u_x(p)^2 + u_y(p)^2}}$$

The data constraints and the parameter c_1 ($c_1 >= 0$) keep the relit image from drifting too far from the input. The desired gradient field g is defined to boost the local gradient if it happens to be oriented along the local lighting direction. The parameter c_2 controls the maximum gradient boost ($c_2 >= 0$). The term $a(p)$ is simply computing the dot product (i.e., cosine of the angle) between the normalized local gradient and the local lighting direction.

5.4 Video deflickering

Often old videos on film will develop a temporal flicker due to film degradation, dust accumulation, or chemical exposure. A common challenge while restoring digitized footage of old videos is to remove such temporal flicker. Temporal flicker is also commonly seen in timelapse videos due to exposure and lighting changes between the long timesteps in the capture. Flicker can also be seen in regular videos shot with poorly synchronized fluorescent lamps.

Past techniques, like the VirtualDub MSU deflicker filter [Strelnikov and Vatolin 2004], have tried to eliminate flicker by simply smoothing out fluctuations in the mean brightness level of nearby video frames. However, such methods cannot get rid of spatially varying flicker caused by uneven film degradation or multiple light sources. Our method on the other hand, tries to preserve the spatial gradients of the input video while dampening temporal gradients along flow lines thus reducing spatially varying flickering without oversmoothing the input. Our deflicker filter is defined as follows:

$$F_{deflicker}(u) \rightarrow [d, g, w]$$
$$w^d(p) = 0$$
$$g^x(p) = u_x(p); g^y(p) = u_y(p)$$

$$F_t(g) = \begin{cases} 0 & if\ |g| < c \\ g & otherwise \end{cases}$$

Here u is the input video in luminance space. $F_{deflicker}$ uses no data constraints based on input video pixel values (i.e., $w^d(p) = 0$) because the input video suffers from flicker. The desired spatial gradients (i.e., g^x and g^y) are defined to replicate the spatial gradients of the input video. Now, to get rid of the temporal flicker

$F_{deflicker}$ *redefines* the function F_t that is used to compute the value of temporal constraints in Equation 10. As mentioned in Section 4, F_t is set to the identity function by default thus causing the final video to mimic the temporal behaviour of the input video. However, since the input in the deflickering application contains some temporal flicker we redefine F_t to suppress small fluctuations of luminance along flow lines. Here, parameter c controls the scale of fluctuations that are suppressed by $F_{deflicker}$. In our experiments c was set to the median magnitude of the temporal gradients along flow lines in the input flickering video.

In the supplementary video we use challenging examples to compare the deflickering performance of $F_{deflicker}$ versus two industry tools – VirtualDub MSU deflicker [Strelnikov and Vatolin 2004] and Furnace deflicker programs. The MSU deflicker program smooths out fluctuations in the mean brightness over time and the Foundry deflicker program is able to handle some spatially varying flicker. However, as can be seen from the supplementary video, $F_{deflicker}$ clearly outperforms both programs.

5.5 Painterly rendering

We will now present a filter for stylizing photographs and videos with a painterly look. Our filter is inspired from a basic technique employed by painters when depicting a scene, that is the exaggeration of salient features and the abstraction of non-salient features in the scene. Like most of our other filters, this filter also measures the saliency of a region using the length of the underlying edge detected by the long edge detector. Specifically, the filter suppresses gradients in regions with short or no edges (i.e., abstraction) and intensifies gradients across long edges (i.e., exaggeration of local contrast). Our painterly filter $F_{painterly}$ is defined as follows:

$$F_{painterly}(u, e) \rightarrow [d, g, w]$$

$$d(p) = u(p); w^d(p) = c_1$$

$$g^x(p) = u_x(p) \cdot cos(e^o(p)) \cdot n(p)$$

$$g^y(p) = u_y(p) \cdot sin(e^o(p)) \cdot n(p)$$

$$n(p) = c_2 * (1 - e^{\frac{e^l(p) * e^l(p)}{-2 * \sigma^2}})$$

Here, $F_{painterly}$ accepts as input the image to stylize (i.e., u) and the edge detector results for the image (i.e., e). The function n spatially varies the abstraction/exaggeration amount based on the underlying edge light $e^l(p)$. The parameter σ in function n controls the *abstraction* amount; Larger values of σ result in large scale objects being abstracted out of the result. The parameter c_2 ($c_2 >= 1$) controls the amount of *exaggeration* of local contrast across larger object boundaries. The data constraints and the parameter c_1 ($c_1 >= 0$) control how much the stylized image is allowed to drift from the input image. As a postprocessing step, our system can optionally overlay a simple visualization of the long edges detected in the input image on top of the result to make it look as if the artist outlined the salient edges using black brush strokes.

Now we will briefly compare our method for painterly rendering to that of Ozran et al. [2007] and Holeger et al. [2006]. Ozran's method for painterly rendering only uses a gradient field integration approach (i.e., no data constraints) and as a side effect has to use more complicated contrast equalization and blurring steps to post process their results. In contrast, the data constraints used in our method cause the overall contrast and depth of field effects (e.g., spatially varying blur) in the input image to be automatically reproduced in the result to the amount desired by the user (i.e., using the control parameter c_1). Ozran's method also does not address the problem of applying their effect to videos in a temporally coherent fashion. Unlike Ozran's method and Holeger's method, our method not only abstracts away non-salient image features but also exaggerates the contrast of salient image features, which can help excentuate the non-photorealistic look of the result. However, unlike Holeger's method, our method does not currently perform in real-time. See figure 5 and the supplementary video for a qualitative comparison of our method to Ozran and Holeger's methods.

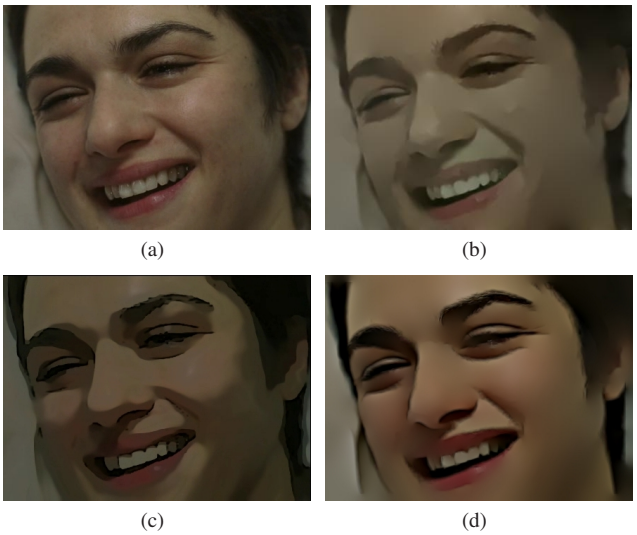


Figure 5: A comparison of our method for painterly rendering to Ozran and Holeger's methods. (a) Original image. (b) Ozran et al.'s result. (c) Holeger et al.'s result. (d) Our result. Unlike Ozran and Holeger's methods, our method not only abstracts away non-salient image features but also exaggerates the contrast of salient image features (e.g., the hairline in this case), which can help excentuate the non-photorealistic look of the result.

5.6 Improved de-blocking filter

A common problem with highly compressed images and videos is that they appear blocky because each macroblock in the image/video is compressed independently without accounting for spatial coherence across block boundaries. Perception studies have found blocking to be one of the most distracting compression artifacts ranking alongside low resolution and ringing artifacts. A good de-blocking filter can therefore improve the perceived quality of videos found on sites like *YouTube*.

Previous work in the spatial domain: There have been many attempts in the past to define high-quality de-blocking filters in the spatial domain [Averbuch et al. 2005; Castagno and Ramponi 1996; Hong et al. 1996]. The best de-blocking filters in the spatial domain tend to be similar to a bilateral filter. They take the weighted average of pixels across block boundaries in order to suppress blockiness while trying not to over-blur the image. Most of the effort in designing these filters goes into crafting a weighting kernel that can suppress block edges but not affect the true edges in the image. There are three major limitations of these spatial-domain approaches for de-blocking:

1. The de-blocking effect of these filters is localized to a few pixels near the block boundaries. For severely compressed images such de-blocking filters are unable to fully suppress the blocking artifacts.

Figure 6: A demonstration of our improvement to Levin's method for de-blocking images. (a) Original image. (b) Image after compression. (c) De-blocking result using gradient suppression but no data constraints (similar to Levin's approach); Notice how the highly compressed regions get flattened in appearance. (d) Result produced by our de-blocking method, which uses gradient suppression to reduce blockiness and data constraints to maintain fidelity to the input.

2. Increasing the size of the de-blocking kernel in order to increase the de-blocking effect invariably over-smoothes the image.
3. Applying these de-blocking filters to individual video frames results in the introduction of temporal artifacts (e.g., flickering).

De-blocking using optimization: Fortunately, the de-blocking problem can be tackled by creatively defining first order constraints in our formulation. In compressed images the gradients across macroblock boundaries (i.e, inter-block gradients) are much less reliable than the gradients inside the macroblocks (i.e., intra-block gradients) since each macroblock is compressed independently. Ergo, a straightforward de-blocking filter in our formulation would selectively edit inter-block gradients in a manner that suppresses the perceived blockiness. Our experiments show that inter-block gradients with large magnitudes usually correspond to true image gradients that simply happen to coincide with block boundaries. On the other hand, inter-block gradients with small magnitudes usually correspond to gradients with zero magnitude in the uncompressed image and form the major source of perceived blockiness in a compressed image. Therefore, our de-blocking filter selectively suppresses only those gradients that lie across block boundaries and have a small magnitude. The formal definition of our de-blocking filter is as follows:

$$F_{deblock}(u) \rightarrow [d, g, w]$$
$$d(p) = u(p); w^d(p) = c_1$$
$$g^x(p) = G(u_x(p))$$
$$g^y(p) = G(u_y(p))$$
$$G(g) = \begin{cases} g & if\ g\ is\ an\ intrablock\ gradient \\ g * S(g) & otherwise \end{cases},$$
$$S(g) = 1 - e^{\frac{g*g}{-2*\sigma^2}}$$

The data constraints and the parameter c_1 ($c_1 > 0$) keep the de-blocked image from drifting too far from the input u. The function G suppresses only those gradients that lie across block boundaries, which can be easily determined by the file format (i.e., compression type) of u. The function S suppresses gradients with magnitudes close to zero. The parameter σ controls the amount of gradient suppression that happens at block boundaries and this parameter can be learned a priori by using pairs of compressed and uncompressed images. See figures 1 & 6 and the supplementary video for image and video de-blocking results.

Now we will briefly compare our method for de-blocking to that of Levin et al. [2003], which also works by suppressing inter-block gradients. Their gradient suppression function requires access to the DCT coefficients of each macroblock, which might not be available to the application. More significantly, their approach is a pure gradient field integration approach (i.e., no data constraints). This severely affects their de-blocking quality in regions where the macroblocks only have a single DC coefficient (i.e, a single color). For example, several macroblocks in the sky and water regions of Figure 6b only have a single color. Without the use of data constraints (i.e., $c_1 = 0$), suppressing the inter-block gradients removes the image blockiness but also flattens the appearance of the result (See figure 6c). In contrast, our use of data constraints causes the colors in the macroblocks to be smoothly interpolated over the sky and water regions as shown in Figure 6d.

5.7 Improved sparse data interpolation

In their seminal work Levin et al. [2004] demonstrated an optimization approach for colorizing grayscale images using a few user drawn color scribbles. Lischinski et al. [2006] observed that Levin's work was in fact a general and a very powerful technique for interpolating sparse data over images. Lischinski pointed out that most data channels in images, and not just color channels, are best interpolated in a spatially piecewise-smooth manner with respect to the luminance channel of the image. Lischinski's work showed the generality of Levin's work by interpolating a variety of data types including tonal values, blurring amounts, and white balance corrections specified by the user using a few paint strokes. Lischinski's method maps quite easily to our formulation as follows:

$$F_{lischinski}(u, d) \rightarrow [d, g, w]$$
$$w^d(p) = \begin{cases} \infty & \text{if } d(p)\ is\ defined \\ 0 & otherwise \end{cases}$$
$$g^x(p) = 0; g^y(p) = 0$$

The function $F_{lischinski}$ accepts as input an image u that will guide the data interpolation and an image d that contains the user data (e.g., scribbles, paint strokes). The image u is grayscale or in log-luminance space depending on the data to be interpolated. The weights for the data constraints in w^d encourages the result to maintain fidelity to the user input where defined. The null gradient field in g in union with our default weighting function for gradient constraints (Section 3.2, equations 5 & 6) causes the data in d to be interpolated in a piecewise smooth manner with respect to u. In fact, the function $F_{lischinski}$ in union with our default weighting function causes the energy function in equation 2 to become equivalent to the energy function used by Lischinski's method.

Improvement: Linchiski's method interpolates sparse data in a piecewise-smooth manner with respect to the underlying image.

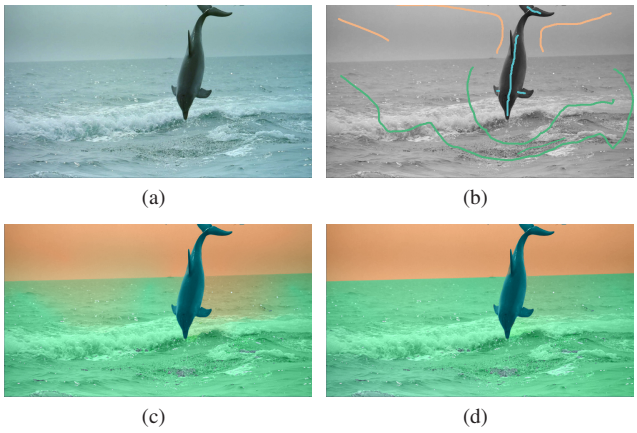


Figure 7: A demonstration of our improvement to Lischinski's method for sparse data interpolation over images. (a) Original color image. (b) User scribbles specifying the desired recolorizing of the image. (c) Colorization result produced by Lischinski's method; Notice the color bleeding between the sky and the ocean. (d) The colorization result produced by our improved method.

However, their function for estimating regions where the smoothness constraints have to be softened (i.e., to create the piecewise smooth behavior) depends on the magnitude of a single, local gradient in the image (i.e., Equations 5 & 6). We make a simple modification to Linchiski's method by using our long edge detector to more robustly detect regions that should produce a break in the smoothness of the interpolation. Thus our improvement significantly reduces the amount of data bleeding in the result (or conversely the amount of user strokes required to produce the desired result). We *redefine* the weights for the gradient constraints in $F_{lischinski}$ as follows:

$$w^x(p) = \frac{1}{(c \cdot e^l(p) \cdot cos(e^o(p)) + \epsilon)^b}$$

$$w^y(p) = \frac{1}{(c \cdot e^l(p) \cdot sin(e^o(p)) + \epsilon)^b}$$

Here, c is a scalar parameter that controls the sensitivity of the data interpolation to the underlying edge length (i.e., $e^l(p)$). The weighting functions also take into account the local edge orientation (i.e, $e^o(p)$) in order to soften the smoothness constraints across, but not along, long edges.

Figures 1 & 7 show two results created using our improved data interpolation algorithm. Also, the supplementary video shows results interpolating data over an entire video where only a few frames have been marked by the user. Figure 7 compares our improved method to Lischinski's method. The sky and water regions in this example are separated by faint local gradients causing Lischinski's method to exhibit more data bleeding in comparison to our improved method.

6 Discussion

In this paper, we have provided a perceptually-motivated optimization-framework for image and video processing. We have tried to account for certain perceptual biases of the HVS in our formulation by:

- allowing applications to define desired pixel gradients in addition to desired pixel values,
- proposing a new measure for gradient saliency that uses edge length in addition to edge strength,
- and proposed a robust weighting scheme that approximates the more robust L_1-norm in order to produce visually pleasing results.

We have demonstrated the versatility of our formulation by designing and improving a variety of image processing applications. The ease with which new applications can be developed using our framework should be apparent given the simple, intuitive solutions we arrived at for the applications we visited, which include:

- a new saliency sharpening filter,
- a new pseudo relighting filter,
- a new video deflickering filter,
- a new painterly rendering filter,
- an improved de-blocking filter, and
- an improved sparse data interpolation method.

Performance Performance is a major concern when it comes to least squares based methods for image processing. Our unoptimized C++ code currently spends a few seconds for one megapixel images and nearly one minute per video frame (at 800x600 resolution) starting from the application specific filtering to the full 3D optimization. However, there is plenty of room of improvement. Our software based conjugate gradient solver, can be significantly sped up using GPU acceleration and a preconditioner similar to the one proposed by Szeliski et al. [2006]. In fact, McCann and Pollard [2008] have recently shown that a GPU accelerated conjugate gradient solver can minimize energy functions like ours in realtime for megapixel-sized images.

Future Work There are several image processing applications that are likely to yield successful solutions when expressed using our formulation. For example, the LDR2HDR problem tackled by Rempel et al. [2007] could probably be solved with high quality results using our framework. Another interesting exploration of our formulation would be in removing compression artifacts like ringing and mosquito noise, which when combined with our deblocking filter could significantly improve the perceived quality of streaming videos (e.g., YouTube and teleconferencing videos).

Our optimization framework also has much untapped potential in the interactive image editing domain, especially when combined with learning algorithms that could automatically identify the type of pixels/gradients the user wants to manipulate given a few example brush strokes. Such interactive tools could be used to remove unwanted texture, glare, shadows, and other annoying artifacts from an image by simply drawing a few rough strokes. Conversely, such tools could also be used to selectively enhance portions of the image for dramatic emphasis. In the coming years, we hope to see the graphics community use and extend our optimization framework to create exciting new image and video processing applications.

References

AGRAWAL, A., AND RASKAR, R., 2007. Gradient domain manipulation techniques in vision and graphics.

AGRAWAL, A., RASKAR, R., NAYAR, S., AND LI, Y., 2005. Removing photography artifacts using gradient projection and flash exposure sampling.

AGRAWAL, A., RASKAR, R., AND CHELLAPPA, R. 2006. Edge suppression by gradient field transformation using cross-projection tensors. In *2006 Conference on Computer Vision and Pattern Recognition (CVPR 2006)*, 2301–2308.

ATTNEAVE, F. 1954. Some informational aspects of visual perception. *Psychol Rev 61*, 3 (May), 183–193.

AVERBUCH, A., SCHCLAR, A., AND DONOHO, D. 2005. Deblocking of block-transform compressed images using weighted sums of symmetrically aligned pixels. *IEEE Transactions on Image Processing 14*, 2 (February), 200–212.

BARTEN, P. G. 1999. *Contrast Sensitivity of the Human Eye and Its Effects on Image Quality*. International Society for Optical Engineering.

BEAUDOT, W., AND MULLEN, K., 2003. How long range is contour integration in human color vision?

BHAT, P., ZITNICK, C. L., SNAVELY, N., AGARWALA, A., AGRAWALA, M., CURLESS, B., COHEN, M., AND KANG, S. B. 2007. Using photographs to enhance videos of a static scene. In *Rendering Techniques 2007 (Proceedings Eurographics Symposium on Rendering)*, J. Kautz and S. Pattanaik, Eds., Eurographics, 327–338.

CANNY, J. 1986. A computational approach to edge detection. *IEEE Trans. Pattern Anal. Mach. Intell. 8*, 6, 679–698.

CASTAGNO, R., AND RAMPONI, G., 1996. A rational filter for the removal of blocking artifacts in image sequences coded at low bitrate.

ELDER, J. H., AND GOLDBERG, R. M. 2001. Image editing in the contour domain. *IEEE Transactions on Pattern Analysis and Machine Intelligence 23*, 3, 291–296.

FATTAL, R., LISCHINSKI, D., AND WERMAN, M. 2002. Gradient domain high dynamic range compression. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, ACM Press, New York, NY, USA, 249–256.

FREEMAN, W. T., AND ADELSON, E. H. 1991. The design and use of steerable filters. *IEEE Trans. Pattern Analysis and Machine Intelligence 13*, 9, 891–906.

GOOCH, A. A., OLSEN, S. C., TUMBLIN, J., AND GOOCH, B. 2005. Color2gray: salience-preserving color removal. *ACM Trans. Graph. 24*, 3, 634–639.

HONG, S., CHAN, Y., AND SIU, W. 1996. A practical real-time post-processing technique for block effect elimination. II: 21–24.

LEVIN, A., ZOMET, A., PELEG, S., AND WEISS, Y., 2003. Seamless image stitching in the gradient domain.

LEVIN, A., LISCHINSKI, D., AND WEISS, Y. 2004. Colorization using optimization. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Papers*, ACM Press, New York, NY, USA, 689–694.

LISCHINSKI, D., FARBMAN, Z., UYTTENDAELE, M., AND SZELISKI, R. 2006. Interactive local adjustment of tonal values. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Papers*, ACM Press, New York, NY, USA, 646–653.

MARSCHNER, S. R., AND GREENBERG, D. P. 1997. Inverse lighting for photography. In *Proceedings of the Fifth Color Imaging Conference, Society for Imaging Science and Technology*.

MCCANN, J., AND POLLARD, N. S. 2008. Real-time gradient-domain painting. *ACM Transactions on Graphics (SIGGRAPH 2008) 27*, 3 (Aug.).

ORZAN, A., BOUSSEAU, A., BARLA, P., AND THOLLOT, J. 2007. Structure-preserving manipulation of photographs. In *International Symposium on Non-Photorealistic Animation and Rendering (NPAR)*.

PÉREZ, P., GANGNET, M., AND BLAKE, A. 2003. Poisson image editing. In *SIGGRAPH '03: ACM SIGGRAPH 2003 Papers*, ACM Press, New York, NY, USA, 313–318.

REMPEL, A. G., TRENTACOSTE, M., SEETZEN, H., YOUNG, H. D., HEIDRICH, W., WHITEHEAD, L., AND WARD, G. 2007. Ldr2hdr: on-the-fly reverse tone mapping of legacy video and photographs. *ACM Trans. Graph. 26*, 3, 39.

SAND, P., AND TELLER, S. 2006. Particle video: Long-range motion estimation using point trajectories. In *CVPR '06: Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, IEEE Computer Society, Washington, DC, USA, 2195–2202.

SHEWCHUK, J. R. 1994. An introduction to the conjugate gradient method without the agonizing pain.

STRELNIKOV, K., AND VATOLIN, D., 2004. Virtualdub msu deflicker filter.

SU, S. L., DURAND, F., AND AGRAWALA, M. 2005. De-emphasis of distracting image regions using texture power maps. In *APGV '05: Proceedings of the 2nd symposium on Applied perception in graphics and visualization*, ACM, New York, NY, USA, 164–164.

SZELISKI, R. 2006. Locally adapted hierarchical basis preconditioning. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Papers*, ACM Press, New York, NY, USA, 1135–1143.

TOMAR, S. 2006. Converting video formats with ffmpeg. *Linux J. 2006*, 146, 10.

WINNEMÖLLER, H., OLSEN, S. C., AND GOOCH, B. 2006. Real-time video abstraction. *ACM Trans. Graph. 25*, 3, 1221–1226.

ZENG, Y., CHEN, W., AND PENG, Q. 2005. A novel variational image model: Towards a unified approach to image editing. *Journal of Computer Science and Technology*.