# Real-time Edge-Aware Image Processing with the Bilateral Grid

Jiawen Chen Sylvain Paris Frédo Durand

Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology

**Figure 1:** *The bilateral grid enables edge-aware image manipulations such as local tone mapping on high resolution images in real time. This 15 megapixel HDR panorama was tone mapped and locally refined using an edge-aware brush at 50 Hz. The inset shows the original input. The process used about 1 MB of texture memory.*

## Abstract

We present a new data structure—the *bilateral grid*, that enables fast edge-aware image processing. By working in the bilateral grid, algorithms such as bilateral filtering, edge-aware painting, and local histogram equalization become simple manipulations that are both local and independent. We parallelize our algorithms on modern GPUs to achieve real-time frame rates on high-definition video. We demonstrate our method on a variety of applications such as image editing, transfer of photographic look, and contrast enhancement of medical images.

**ACM Reference Format**
Chen, J., Paris, S., Durand, F. 2007. Real-Time Edge-Aware Image Processing with the Bilateral Grid. *ACM Trans. Graph. 26*, 3, Article 103 (July 2007), 9 pages. DOI = 10.1145/1239451.1239554 http://doi.acm.org/10.1145/1239451.1239554.



**Keywords:** Computational Photography, Edge-Aware Image Processing, Bilateral Filter, Real-time Video Processing

## 1 Introduction

Nonlinear filters have enabled novel image enhancement and manipulation techniques where image structure such as strong edges are taken into account. They are based on the observation that many meaningful image components and desirable image manipulations tend to be piecewise smooth rather than purely band-limited. For example, illumination is usually smooth except at shadow boundaries [Oh et al. 2001], and tone mapping suffers from haloing artifacts when a low-pass filter is used to drive local adjustment [Chiu et al. 1993], a problem which can be solved with nonlinear filters [Tumblin and Turk 1999; Durand and Dorsey 2002].

In particular, the bilateral filter is a simple technique that smoothes an image except at strong edges [Aurich and Weule 1995; Tomasi and Manduchi 1998; Smith and Brady 1997]. It has been used in a variety of contexts to decompose an image into a piecewise-smooth large-scale base layer and a detail layer for tone mapping [Durand and Dorsey 2002], flash/no-flash image fusion [Petschnigg et al. 2004; Eisemann and Durand 2004], and a wealth of other appli-

cations [Bennett and McMillan 2005; Xiao et al. 2006; Bae et al. 2006; Winnemöller et al. 2006]. A common drawback of nonlinear filters is speed: a direct implementation of the bilateral filter can take several minutes for a one megapixel image. However, recent work has demonstrated acceleration and obtained good performance on CPUs, on the order of one second per megapixel [Durand and Dorsey 2002; Pham and van Vliet 2005; Paris and Durand 2006; Weiss 2006]. However, these approaches still do not achieve real-time performance on high-definition content.

In this work, we dramatically accelerate and generalize the bilateral filter, enabling a variety of edge-aware image processing applications in real-time on high-resolution inputs. Building upon the technique by Paris and Durand [2006], who use linear filtering in a higher-dimensional space to achieve fast bilateral filtering, we extend their high-dimensional approach and introduce the *bilateral grid*, a new compact data structure that enables a number of edge-aware manipulations. We parallelize these operations using modern graphics hardware to achieve real-time performance at HD resolutions. In particular, our GPU bilateral filter is two orders of magnitude faster than the equivalent CPU implementation.

This paper makes the following contributions:

- **The bilateral grid**, a new data structure that naturally enables edge-aware manipulation of images.
- **Real-time bilateral filtering** on the GPU, which is two orders of magnitude faster than previous CPU techniques, enabling real-time processing of HD content.
- **Edge-aware algorithms**. We introduce a number of real-time, edge-aware algorithms including edge-preserving painting, scattered data interpolation, and local histogram equalization.

Our approach is implemented as a flexible library which is distributed in open source to facilitate research in this domain. The code is available at `http://groups.csail.mit.edu/graphics/bilagrid/`. We believe that the bilateral grid data structure is general and future research will develop many new applications.

### 1.1 Related Work

**Bilateral Filter** The bilateral filter is a nonlinear process that smoothes images while preserving their edges [Aurich and Weule 1995; Smith and Brady 1997; Tomasi and Manduchi 1998]. For an image $I$, at position $\mathbf{p}$, it is defined by:

$$bf(I)_{\mathbf{p}} = \frac{1}{W_{\mathbf{p}}} \sum_{\mathbf{q} \in N(\mathbf{p})} G_{\sigma_s}(\|\mathbf{p}-\mathbf{q}\|)\, G_{\sigma_r}(|I_{\mathbf{p}} - I_{\mathbf{q}}|)\, I_{\mathbf{q}} \quad (1a)$$

$$W_{\mathbf{p}} = \sum_{\mathbf{q} \in N(\mathbf{p})} G_{\sigma_s}(\|\mathbf{p}-\mathbf{q}\|)\, G_{\sigma_r}(|I_{\mathbf{p}} - I_{\mathbf{q}}|) \quad (1b)$$

The output is a simple weighted average over a neighborhood where the weight is the product of a Gaussian on the spatial distance ($G_{\sigma_s}$) and a Gaussian on the pixel value difference ($G_{\sigma_r}$), also called the range weight. The range weight prevents pixels on one side of a strong edge from influencing pixels on the other side since they have different values. This also makes it easy to take into account edges over a multi-channel image (such as RGB) since the Gaussian on the pixel value difference can have an arbitrary dimension.

Fast numerical schemes for approximating the bilateral filter are able to process large images in less than a second. Pham et al. [2005] describe a separable approximation that works well for the small kernels used in denoising but suffers from artifacts with the larger kernels used in other applications. Weiss [2006] maintains local image histograms, which unfortunately limits this approach to box spatial kernels instead of the smooth Gaussian kernel.

Paris and Durand [2006] extend the fast bilateral filter introduced by Durand and Dorsey [2002] and recast the computation as a higher-dimensional linear convolution followed by trilinear interpolation and a division. We generalize the ideas behind their higher-dimensional space into a data structure, the bilateral grid, that enables a number of operations, including the bilateral filter, edge-aware painting and interpolation, and local histogram equalization.

Other edge-preserving techniques include anisotropic diffusion [Perona and Malik 1990; Tumblin and Turk 1999] which is related to the bilateral filter [Durand and Dorsey 2002; Barash 2002]. Optimization [Levin et al. 2004; Lischinski et al. 2006] can also be used to interpolate values while respecting strong edges. Our work introduces an alternative to these approaches.

**High-Dimensional Image Representation** The interpretation of 2D images as higher-dimensional structures has received much attention. Sochen et al. [1998] describe images as 2D manifolds embedded in a higher-dimensional space. For instance, a color image is embedded in a 5D space: 2D for $x$ and $y$ plus 3D for color. This enables an interpretation of PDE-based image processes in terms of geometric properties such as curvature. Our bilateral grid shares the use of higher-dimensional spaces with this approach. It is nonetheless significantly different since we consider values over the entire space and not only on the manifold. Furthermore, we store homogeneous values, which allows us to handle weighted functions and the normalization of linear filters. Our approach is largely inspired by signal processing whereas Sochen et al. follow a differential geometry perspective.

Felsberg et al. [2006] describe an edge-preserving filter based on a stack of channels that can be seen as a volumetric structure similar in spirit to the bilateral grid. Each channel stores spline coefficients representing the encoded image. Edge-preserving filtering is achieved by smoothing the spline coefficients within each channel and then reconstructing the splines from the filtered coefficients. In comparison, the bilateral grid does not encode the data into splines and allows direct access. This enables faster computation; in particular, it makes it easier to leverage the computational power of modern parallel architectures.

## 2 Bilateral Grid

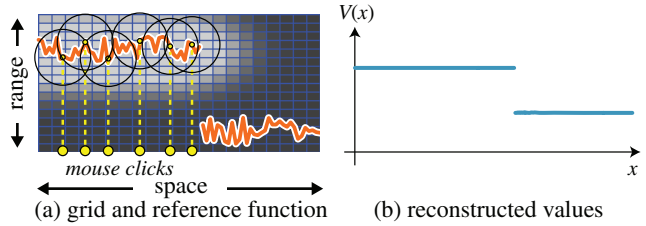
The effectiveness of the bilateral filter in respecting strong edges comes from the inclusion of the range term $G_{\sigma_r}(|I_{\mathbf{p}} - I_{\mathbf{q}}|)$ in the weighted combination of Equation 1: although two pixels across an edge are close in the spatial dimension, from the filter's perspective, they are distant because their values differ widely in the range dimension. We turn this principle into a data structure, the bilateral grid, which is a 3D array that combines the two-dimensional spatial domain with a one-dimensional range dimension, typically the image intensity. In this three-dimensional space, the extra dimension makes the Euclidean distance meaningful for edge-aware image manipulation.

The bilateral grid first appeared as an auxiliary data structure in Paris and Durand's fast bilateral filter [2006]. In that work, it was used as an algebraic re-expression of the original bilateral filter equation. Our perspective is different in that we view the bilateral grid as a primary data structure that enables a variety of edge-aware operations in real time.

### 2.1 A Simple Illustration

Before formally defining the bilateral grid, we first illustrate the concept with the simple example of an edge-aware brush. The edge-aware brush is similar to brushes offered by traditional editing packages except that when the user clicks near an edge, the brush does not paint across the edge.

When the user clicks on an image $E$ at a location $(x_u, y_u)$, the edge-aware brush paints directly in the three-dimensional bilateral grid at position $\big(x_u, y_u, E(x_u, y_u)\big)$. The spatial coordinates are determined by the click location, while the range coordinate is the intensity of the clicked pixel. The edge-aware brush has a smooth falloff in all three dimensions. The falloff along the two spatial dimensions is the same as in classical 2D brushes, but the range falloff is specific to our brush and ensures that only a limited interval of intensity values is affected. To retrieve the painted value $V$ in image space at location $(x, y)$, we read the value of the bilateral grid at position $\big(x, y, E(x, y)\big)$. We use the same terminology as Paris and Durand [2006] and call this operation *slicing*. In regions where the image $E$ is nearly constant, the edge-aware brush behaves like a classical brush with a smooth spatial falloff. Since the range variations are small, the range falloff has little influence. At edges, $E$ is discontinuous and if only one side has been painted, the range falloff ensures that the other side is unaffected, thereby creating a discontinuity in the painted values $V$. Although the grid values are smooth, the output value map is piecewise-smooth and respects the strong edges of $E$ because we slice according to $E$. Figure 2 illustrates this process on a 1D image.




(a) grid and reference function (b) reconstructed values

**Figure 2:** *Example of brush painting on a 1D image $E$. When the user clicks at location $x$, we add a smooth brush shape in the grid at location $\big(x, E(x)\big)$. The image space result of the brush operation at a position $x$ is obtained by interpolating the grid values at location $\big(x, E(x)\big)$. Although the grid is smooth, we obtain an image space result that respects the image discontinuity.*

This simple example demonstrates the edge-aware properties of the bilateral grid. Although we manipulate only smooth values and ignore the issue of edge preservation, the resulting function generated in image space is piecewise-smooth and respects the discontinuities of the reference image thanks to the final slicing operation. Furthermore, computations using the bilateral grid are generally independent and require a coarser resolution than the reference image. This makes it easy to map grid operations onto parallel architectures.

### 2.2 Definition

**Data Structure** The bilateral grid is a three dimensional array, where the first two dimensions $(x, y)$ correspond to 2D position in the image plane and form the spatial domain, while the third dimension $z$ corresponds to a reference range. Typically, the range axis is image intensity.

**Sampling** A bilateral grid is regularly sampled in each dimension. We name $s_s$ the sampling rate of the spatial axes and $s_r$ the sampling rate of the range axis. Intuitively, $s_s$ controls the amount of smoothing, while $s_r$ controls the degree of edge preservation. The number of grid cells is inversely proportional to the sampling rate: a smaller $s_s$ or $s_r$ yields a larger number of grid cells and requires more memory. In practice, most operations on the grid require only a coarse resolution, where the number of grid cells is much smaller than the number of image pixels. In our experiments, we use between 2048 and 3 million grid cells for the useful range of parameters. The required resolution depends on the operation and we will discuss it on a per-application basis.

**Data Type and Homogeneous Values** The bilateral grid can store in its cells any type of data such as scalars and vectors. For many operations on the bilateral grid, it is important to keep track of the number of pixels (or a weight $w$) that correspond to each grid cell. Hence, we store *homogeneous* quantities such as $(wV, w)$ for a scalar $V$ or $(wR, wG, wB, w)$ if we deal with RGB colors. This representation makes it easy to compute weighted averages: $(w_1V_1, w_1) + (w_2V_2, w_2) = \big((w_1V_1 + w_2V_2), (w_1 + w_2)\big)$ where normalizing by the homogeneous coordinate $(w_1 + w_2)$ yields the expected result of averaging $V_1$ and $V_2$ weighted by $w_1$ and $w_2$. Intuitively, the homogeneous coordinate $w$ encodes the importance of its associated data $V$. It is also similar to the homogeneous interpretation of premultiplied alpha colors [Blinn 1996; Willis 2006].

### 2.3 Basic Usage of a Bilateral Grid

Since the bilateral grid is inspired by the bilateral filter, we use it as the primary example of a grid operation. We describe a set of new manipulations in Section 4. In general, the bilateral grid is used in three steps. First, we create a grid from an image or other user input. Then, we perform processing inside the grid. Finally, we slice the grid to reconstruct the output (Fig. 3). Construction and slicing are symmetric operations that convert between image and grid space.

**Grid Creation** Given an image $I$ normalized to $[0, 1]$, $s_s$ and $s_r$, the spatial and range sampling rates, we construct the bilateral grid $\Gamma$ as follows:

1. *Initialization*: For all grid nodes $(i, j, k)$, $\Gamma(i, j, k) = (0, 0)$.
2. *Filling*: For each pixel at position $(x, y)$:

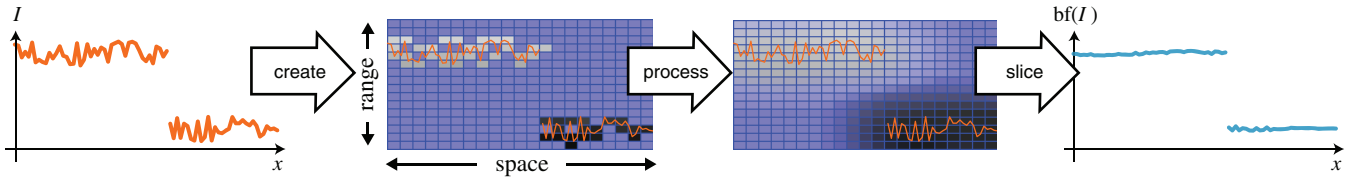
$$\Gamma\left([x/s_s],\ [y/s_s],\ [I(x, y)/s_r]\right)\ \ +=\ \ (I(x, y), 1)$$

where $[\,\cdot\,]$ is the closest-integer operator. We use the notation $\Gamma = c\,(I)$ for this construction. Note that we accumulate both the image intensity and the number of pixels into each grid cell using homogeneous coordinates.

**Processing** Any function $f$ that takes a 3D function as input can be applied to a bilateral grid $\Gamma$ to obtain a new bilateral grid $\tilde{\Gamma} = f(\Gamma)$. For the bilateral filter, $f$ is a convolution by a Gaussian kernel, where the variance along the domain and range dimensions are $\sigma_s$ and $\sigma_r$ respectively [Paris and Durand 2006].

**Extracting a 2D Map by Slicing** Slicing is the critical bilateral grid operation that yields a piecewise-smooth output. Given a bilateral grid $\Gamma$ and a reference image $E$, we extract a 2D value map $M$ by accessing the grid at $\big(x/s_s, y/s_s, E(x, y)/s_r\big)$ using trilinear interpolation. We use the notation $M = s_E(\Gamma)$. If the grid stores homogeneous values, we first interpolate the grid to retrieve the homogeneous vector; then, we divide the interpolated vector to access the actual data.

Slicing is symmetric to the creation of the grid from an image. If a grid matches the resolution and quantization of the image used




(a) input 1D image (b) grid created from 1D image (c) filtered grid and slicing image (d) filtered 1D image

**Figure 3:** *Bilateral filtering using a bilateral grid demonstrated on a 1D example. Blue grid cells are empty ($w = 0$).*

for creation, slicing will result in the same image; although in practice, the grid is much coarser than the image. Processing in the grid between creation and slicing is what enables edge-aware operations. Moreover, the particular grid operation determines the required sampling rate.

**Recap: Bilateral Filtering** Using our notation, the algorithm by Paris and Durand [2006] to approximate the bilateral filter in Equation 1 becomes (Fig. 3):

$$bf(I) \quad = \quad s_I\big( G_{\sigma_s,\sigma_r} \otimes c(I) \big) \tag{2}$$

We embed the image $I$ in a bilateral grid, convolve this grid with a 3D Gaussian kernel with spatial parameter $\sigma_s$ and range parameter $\sigma_r$, and slice it with the input image. A contribution of our work is to demonstrate that the bilateral grid extends beyond bilateral filtering and enables a variety of edge-preserving processing.

## 3 GPU Parallelization

In developing the bilateral grid, one of our major goals was to facilitate parallelization on graphics hardware. Our benchmarks showed that on a CPU, the bottleneck lies in the slicing stage, where the cost is dominated by trilinear interpolation. We take advantage of hardware texture filtering on the GPU to efficiently perform slicing. The GPU's fragment processors are also ideally suited to executing grid processing operations such as Gaussian blur.

### 3.1 Grid Creation

To create a bilateral grid from an image, we accumulate the value of each input pixel into the appropriate grid voxel. Grid creation is inherently a scatter operation since the grid position depends on a pixel's value. Since the vertex processor is the only unit that can perform a true scatter operation [Harris and Luebke 2004], we rasterize a vertex array of single pixel points and use a vertex shader to determine the output position. On modern hardware, the vertex processor can efficiently access texture memory. The vertex array consists of $(x, y)$ pixel positions; the vertex shader looks up the corresponding image value $I(x, y)$ and computes the output position. On older hardware, however, vertex texture fetch is a slow operation. Instead, we store the input image as vertex color attribute: each vertex is a record $(x, y, r, g, b)$ and we can bypass the vertex texture fetch. The disadvantage of this approach is that during slicing, where the input image needs to be accessed as a texture, we must copy the data. For this, we use the pixel buffer object extension to do a fast copy within GPU memory.

**Data Layout** We store bilateral grids as 2D textures by tiling the $z$ levels across the texture plane. This layout lets us use hardware bilinear texture filtering during the slicing stage and reduce the number of texture lookups from 8 to 2. To support homogeneous coordinates, we use four-component, 32-bit floating point textures. In this format, for typical values of $s_s = 16$ and $s_r = 0.07$ (15 intensity levels), a bilateral grid requires about 1 megabyte of texture memory per megapixel. For the extreme case of $s_s = 2$, the storage cost is about 56 megabytes per megapixel. In general, a grid requires $16 \times$ number of pixels$/({s_s}^2 \times s_r)$ bytes of texture memory. Grids that do not require homogeneous coordinates are stored as single-component floating point textures and require $1/4$ the memory. In our examples, we use between 50 kB and 40 MB.

### 3.2 Low-Pass Filtering

As described in Section 2.3, the grid processing stage of the bilateral filter is a convolution by a 3D Gaussian kernel. In constructing the grid, we set the the sampling rates $s_s$ and $s_r$ to correspond to the bandwidths of the Gaussian $\sigma_s$ and $\sigma_r$, which provides a good trade-off between accuracy and storage [Paris and Durand 2006]. Since the Gaussian kernel is separable, we convolve the grid in each dimension with a 5-tap 1D kernel using a fragment shader.

### 3.3 Slicing

After processing the bilateral grid, we slice the grid using the input image $I$ to extract the final 2D output. We slice on the GPU by rendering $I$ as a textured quadrilateral and using a fragment shader to look up the stored grid values in a texture. To perform trilinear interpolation, we enable bilinear texture filtering, fetch the two nearest $z$ levels, and interpolate between the results. By taking advantage of hardware bilinear interpolation, each output pixel requires only 2 indirect texture lookups.

### 3.4 Performance

We benchmark the grid-based bilateral filter on three generations of GPUs on the same workstation. Our GPUs consist of an NVIDIA GeForce 8800 GTX (G80), which features unified shaders, a GeForce 7800 GT (G70) and a GeForce 6800 GT (NV40). They were released in 2006, 2005, and 2004, respectively. Our CPU is an Intel Core 2 Duo E6600 (2.4 GHz, 4MB cache).

The first benchmark varies the image size while keeping the bilateral filter parameters constant ($\sigma_s = 16, \sigma_r = 0.1$). We use the same image subsampled at various resolutions and report the average runtime over 1000 iterations. Figure 4 shows that our algorithm is linear in the image size, ranging from 4.5ms for 1 megapixel to 44.7ms for 10 megapixels on the G80. We consistently measured slowdowns beyond 9 megapixels for the older GPUs, which we suspect is due to an overflow in the vertex cache. For comparison, our CPU implementation ranges from 0.2 to 1.9 seconds on the same inputs. Our GPU implementation outperforms Weiss's CPU bilateral filter [2006] by a factor of 50.

The second benchmark keeps the image size at 8 megapixels and the range kernel size $\sigma_r$ at 0.1 while varying the spatial kernel size $\sigma_s$. As with the first benchmark, we use the same image and report the average runtime over 1000 iterations. Figure 5 shows the influence of the kernel size. With the exception of a few bumps in the curve

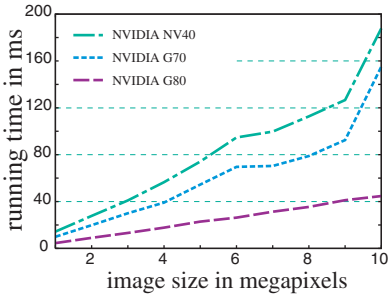



**Figure 4:** *Bilateral filter running times as a function of the image size (using $\sigma_s = 16$ and $\sigma_r = 0.1$). The memory requirements increase linearly from 625 kB at 1 megapixel to 6.25 MB at 10 megapixels.*

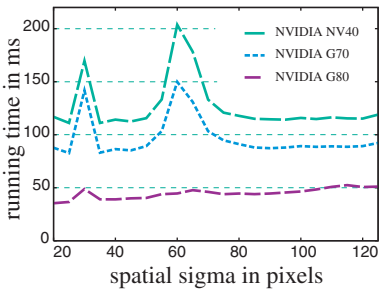



**Figure 5:** *Bilateral filter running times as a function of the spatial sigma (8 megapixels, $\sigma_r$ = 0.1). The memory requirements decrease quadratically from 3.2 MB at $\sigma_s$ = 20 down to 130 kB at $\sigma_s$ = 120.*

at $\sigma_s = 30$ and $\sigma_s = 60$ that we suspect to be due to hitting a framebuffer cache boundary, our algorithm is independent of spatial kernel size. This is due to the fact that the 3D convolution kernel on the bilateral grid is independent of $\sigma_s$. As we increase $\sigma_s$, we downsample more aggressively; therefore, the convolution kernel remains $5 \times 5 \times 5$. For comparison, our CPU implementation and Weiss's algorithm both consistently run in about 1.6s over the same values of $\sigma_s$.

All three generations of hardware achieve real-time performance (30 Hz) at 2 megapixels, the current 1080p HD resolution. The G80 attains the same framerate at 7 megapixels. On all generations of hardware, the bottleneck of our algorithm lies in the grid construction stage; the other stages are essentially free. We have verified with an OpenGL profiler that on older hardware with a fixed number of vertex processors, the algorithm is vertex limited while the rest of the pipeline is starved for data. This situation is largely improved on current hardware because it features unified shaders. The bottleneck then lies in the raster operations units.

### 3.5 Further Acceleration

On current hardware, we can run multiple bilateral filters per frame on 1080p HD video, but on older hardware, we are limited to a single filter per frame. For temporally coherent data, we propose an acceleration based on subsampling. A cell of the grid stores the weighted average of a large number of pixels and we can obtain a good estimate with only a subset of those pixels. For typical values of $\sigma_s \in [10, 50]$ and $\sigma_r \in [0.05, 0.4]$, using only $10\%$ of the input pixels produces an output with no visual artifacts. We choose the $10\%$ of pixels by rotating through a sequence of precomputed Poisson-disk patterns to obtain a good coverage. To combat "swimming" artifacts introduced by time-varying sampling patterns, we apply a temporal exponential filter with a decay constant of 5 frames. This produces results visually indistinguishable from the full bilateral filter except at hard scene transitions.

## 4 Image Manipulation with the Bilateral Grid

The bilateral grid has a variety of applications beyond bilateral filtering. The following sections introduce new ways of creating, processing and slicing a bilateral grid.

### 4.1 Cross-Bilateral Filtering

A direct extension to the bilateral filter is the *cross-bilateral filter* [Petschnigg et al. 2004; Eisemann and Durand 2004], where the notion of image data is decoupled from image edges. We define a new grid creation operator with two parameters: an image $I$, which defines the grid values, and an edge image $E$ which determines the grid position.

$$\Gamma\left(\left[\frac{x}{s_s}\right], \left[\frac{y}{s_s}\right], \left[\frac{E(x,y)}{s_r}\right]\right) \mathrel{+}= (I(x,y), 1) \qquad (3)$$

We use the notation $\Gamma = c_E(I)$ for this operator. Analogously, the slicing operator uses the edge image $E$ to query the grid and reconstruct the result:

$$cbf(I,E) = s_E\big(\, G_{\sigma_s,\sigma_r} \otimes c_E(I) \,\big) \qquad (4)$$

### 4.2 Grid Painting

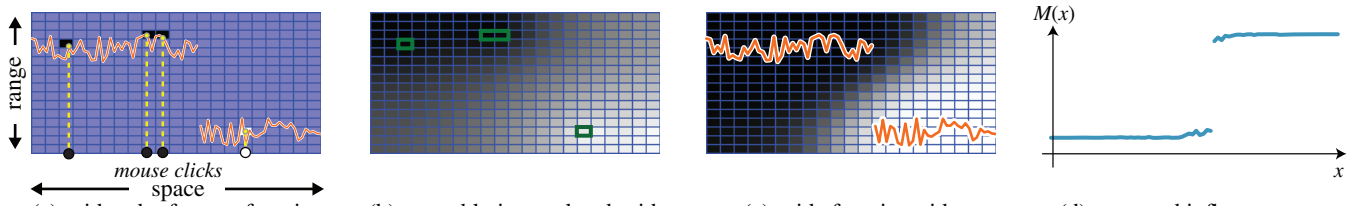
We now elaborate on the edge-preserving brush mentioned in Section 2.1. Analogous to a classical 2D brush, where users locally "paint" characteristics such as brightness, the bilateral brush further ensures that modifications do not "bleed" across image contours (Figure 2). We use a bilateral grid $\Gamma_{\text{brush}}$ to control a 2D influence map $M$ that defines the strength of the applied modification. We initialize a scalar (i.e., non-homogeneous) grid $\Gamma_{\text{brush}}$ to 0. When the user clicks on an image pixel $(x, y, I(x,y))$, we add to the grid values a 3D brush (e.g., a Gaussian) centered at position $\big(x/s_s, y/s_s, I(x,y)/s_r\big)$. We set $s_s$ to the spatial bandwidth of the Gaussian brush shape, and $s_r$ is set to the desired degree of edge preservation. We obtain $M$ by slicing $\Gamma_{\text{brush}}$ with the image $I$: $M = s_I(\Gamma_{\text{brush}})$.

(a) input and stroke (b) intermediate (c) output

**Figure 6:** *Bilateral Grid Painting allows the user to paint without bleeding across image edges. The user clicks on the input (a) and strokes the mouse. The intermediate (b) and final (c) results are shown. The entire 2 megapixel image is updated at 60 Hz. Memory usage was about 1.5 MB for a* $20 \times 20$ *brush and* $s_r$ *= 0.05.*

**GPU Implementation** We tile $\Gamma_{\text{brush}}$ as a single-component 2D texture. When the user clicks the mouse, a fragment shader renders the brush shape using blending. Slicing is identical to the case of the bilateral filter. A modern GPU can support bilateral grid painting on very large images. For a $2 \times 2$ brush with $s_r = 0.05$, the grid requires 20 MB of texture memory per megapixel; a $5 \times 5$ brush consumes less than 1 MB per megapixel.

**Results** In Figure 6, the user manipulates the hue channel of an image without creating a mask. An initial mouse click determines $(x_0, y_0, z_0)$ in the grid. Subsequent mouse strokes vary in $x$ and $y$, but $z_0$ is fixed. Hence, the brush affects only the selected intensity layer and does not cross image edges.




(a) grid and reference function (b) smoothly interpolated grid (c) grid after sigmoid (d) extracted influence map

**Figure 7:** *Edge-aware interpolation with a bilateral grid demonstrated on a 1D example. The user clicks on the image to indicate sparse constraints (a) (unconstrained cells are shown in blue). These values are interpolated into a smooth function (b) (constraints are shown in green). We filter the grid using a sigmoid to favor consistent regions (c). The resulting is sliced with the input image to obtain the influence map $M$ (d).*

### 4.3 Edge-Aware Scattered Data Interpolation

Inspired by Levin et al. [2004], Lischinski et al. [2006] introduced a scribble interface to create an influence map $M$ over an image $I$. The 2D map $M$ interpolates a set of user-provided constraints $\{M(x_i, y_i) = m_i\}$ (the scribbles) while respecting the edges of the underlying image $I$. We use a scalar bilateral grid $\Gamma_{\text{int}}$ to achieve a similar result: instead of solving a *piecewise-smooth* interpolation in the image domain, we solve a *smooth* interpolation in the grid domain and then slice.

We lift the user-provided constraints into the 3D domain: $\{\Gamma_{\text{int}}([x/s_s], [y/s_s], [I(x,y)/s_r]) = m_i\}$, and minimize the variations of the grid values:

$$\text{argmin} \int \|\text{grad}(\Gamma_{\text{int}})\|^2 \qquad (5)$$

$$\text{under the constraints:} \quad \left\{\Gamma_{\text{int}}\left(\left[\frac{x}{s_s}\right], \left[\frac{y}{s_s}\right], \left[\frac{I(x,y)}{s_r}\right]\right) = m_i\right\}$$

The 2D influence map is obtained by slicing: $M = s_I(\Gamma_{\text{int}})$. Finally, we bias the influence map toward 0 and 1 akin to Levin et al. [2007]. We achieve this by applying a sigmoid function to the grid values. Figure 7 summarizes this process and Figure 8 shows a sample result.

**Discussion** Compared to image-based approaches [Levin et al. 2004; Lischinski et al. 2006], our method does not work at the pixel level which may limit accuracy in some cases; although our experiments did not reveal any major problems. On the other hand, the bilateral grid transforms a difficult image-dependent and non-homogeneous 2D optimization into a simpler smooth and homogeneous interpolation in 3D. Furthermore, the grid resolution is decoupled from the resolution of the image, which prevents the complexity from growing with image resolution.

Another difference is that image-based techniques use the notion of "geodesic distance" over the image manifold, while we consider the Euclidean distance in a higher-dimensional space. The comparison of those two measures deserves further investigation and we believe that which method is most appropriate is an application-dependent choice.

(a) input & scribbles (b) influence map (c) output

**Figure 8:** *Fast scribble interpolation using the Bilateral Grid. The user paints scribbles over the input (a). Our algorithm extracts an influence map (b), which is used to adjust the input hue and produce the output (c). The entire 2 megapixel image is updated at 20 Hz. Memory usage was about 62 kB for $s_s = 256$ and $s_r = 0.05$.*

**GPU Implementation** Analogous to grid painting, we rasterize scribble constraints into the bilateral grid using a fragment shader. To obtain a globally smooth bilateral grid that respects the constraints, we solve Laplace's equation by extending a GPU multigrid algorithm [Goodnight et al. 2003] to handle irregular 3D domains. The domain has holes because the user-specified hard constraints create additional boundaries.

**Results** We demonstrate real-time scribble interpolation with a simple color adjustment application. The user paints scribbles over an image; white scribbles denote regions where the hue should be adjusted, while black scribbles protect the image region. In practice, the sampling rate of $\Gamma_{\text{int}}$ can be very coarse and still yield good influence maps. The example in Figure 8 was generated using a coarse grid containing about 600 variables, allowing our GPU solver for generating the influence map in real time (40 Hz). When finer adjustments are required, we can still achieve interactive rates (1 Hz) on finely sampled grids with over 500,000 variables. Refer to the supplemental video for a screen capture of an editing session.

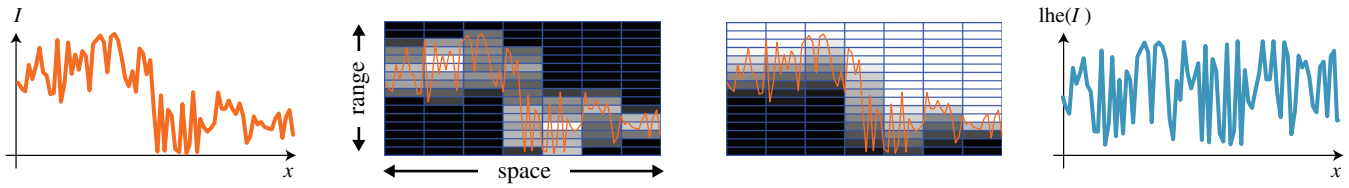
### 4.4 Local Histogram Equalization

Histogram equalization is a standard technique for enhancing the contrast of images [Gonzales and Woods 2002]. However, for some inputs, such as X-Ray and CT medical images that have high dynamic range, histogram equalization can obscure small details that span only a limited intensity range. For these cases, it is more useful to perform histogram equalization locally over image windows. We perform *local histogram equalization* efficiently using a bilateral grid, and achieve real-time performance using the GPU.

Given an input image $I$, we construct a scalar bilateral grid $\Gamma_{\text{hist}}$. We initialize $\Gamma_{\text{hist}} = 0$ and fill it with:

$$\Gamma_{\text{hist}}\left(\left[\frac{x}{s_s}\right], \left[\frac{y}{s_s}\right], \left[\frac{I(x,y)}{s_r}\right]\right) \mathrel{+}= 1 \qquad (6)$$

We denote this operator $\Gamma_{\text{hist}} = c_{\text{hist}}(I)$. $\Gamma_{\text{hist}}$ stores the number of pixels in a grid cell and can be considered a set of local histograms. For each $(x, y)$, the corresponding column splits the $s_s \times s_s$ covered pixels into intensity intervals of size $s_r$. By using the closest-integer operator when constructing $\Gamma_{\text{hist}}$, we perform a box filter in space. If a smoother spatial kernel is desired, we blur each $z$ level of the grid by a spatial kernel (e.g., Gaussian). We perform local histogram equalization by applying a standard histogram equalization to each column and slicing the resulting grid with the input image $I$.




(a) input 1D image (b) histogram grid from 1D image (c) equalized grid & slicing image (d) locally equalized 1D image

**Figure 9:** *Local histogram equalization demonstrated on a 1D image. We build a grid that counts the number of pixels in each bin (b). Each grid column corresponds to the histogram of the image region it covers. By equalizing each column, we obtain a grid (c) which leads to an image-space signal with an enhanced contrast that exploits the whole intensity range (d).*

**GPU Implementation** We construct the bilateral grid the same way as in bilateral filtering, except we can ignore the image data. Next, we execute a fragment shader that accumulates over each $(x, y)$ column of the bilateral grid. This yields a new grid where each $(x, y)$ column is an unnormalized cumulative distribution function. We run another pass to normalize the grid to between 0 and 1 by dividing out the final bin value. Finally, we slice the grid using the input image.

**Results** Our algorithm achieves results visually similar to MATLAB's `adapthisteq` (Figure 10). In both cases, low-contrast details are revealed while the organ shapes are preserved. Our method based on the bilateral grid achieves a speed-up of one order of magnitude: 100ms compared to 1.5s on a 3.7 megapixel HDR image.

## 5 Applications and Results

In this section, we describe a variety of applications which take advantage of image processing using the bilateral grid. Refer to the supplemental video for a demonstration of our results. For the video applications, decoding is performed on the CPU in a separate thread that runs in parallel with the GPU. The timings measure the average throughput for the entire pipeline. On our CPU, the largest input video (1080p resolution using the H.264 codec) takes about 25ms to decode each frame; which means that in many cases, decoding is more expensive than our edge-aware operators and becomes the pipeline bottleneck.

### 5.1 High Resolution Video Abstraction

Winnemöller et al. [2006] demonstrated a technique for stylizing and abstracting video in real time. A major bottleneck in their approach was the bilateral filter, which limited the video to DVD resolution (0.3 megapixels) and the framerate to 9 to 15 Hz. To attain this framerate, they used a separable approximation to the bilateral filter with a small kernel size and iterated the approximation to obtain a sufficiently large spatial support [Pham and van Vliet 2005]. Using the bilateral grid with our GPU acceleration technique (without the additional acceleration described in Section 3.5), we are able to perform video abstraction at 42 Hz on 1080p HD video (1.5 megapixels).

**Progressive Abstraction** To further enhance the technique, we incorporated a progressive abstraction component in the spirit of the stylization technique by DeCarlo et al. [2002] where details near a focus point are less abstracted than those far away. In our method, we build a 4-level *bilateral pyramid*—bilateral grids computed at multiples of a base $s_s$ and $s_r$. To abstract an input frame, we first compute a distance map that falls off from the user's focus point. We ensure that this map respects the image edges by cross-bilateral filtering it with the image as reference to get an *importance map*. We use the importance map to linearly interpolate between the input frame (at the focus point) and the levels of the multiscale bilateral grid. We use the result as input to the rest of the video abstraction pipeline. We found that extracting the lines from the first pyramid level yields more coherent outputs. With a 4-level pyramid, we are still able to maintain 20 Hz on 1080p video.

(a) input (1760 x 2140, HDR) (b) MATLAB's `adapthisteq` (1.5s) (c) our result (100ms)

**Figure 10:** *Local histogram equalization reveals low-contrast details by locally remapping the intensity values. The input (a) is an HDR chest X-Ray (tone mapped for display). Our algorithm (c) based on the bilateral grid has a similar visual appearance to MATLAB's* `adapthisteq` *(b) while achieving a speedup of an order of magnitude. For this example, we used $s_s$ = 243.75, $s_r$ = 0.0039; memory usage was 500 kB total for the two grids.*

### 5.2 Transfer of Photographic Look

Bae et al. [2006] introduced a method to transfer the "look" of a model photograph to an input photograph. We adapt their work to operate on videos in real time. We describe two modifications to handle the constraints inherent to video. We use a simplified pipeline that yields real-time performance while retaining most of the effects from the original process. We also specifically deal with noise and compression artifacts to handle sources such as DVDs and movies recorded using consumer-grade cameras.

**Processing Pipeline** Akin to Bae et al., we use the bilateral filter on each input frame $I$ and name the result the *base layer* $B_\mathrm{i} = bf(I)$ and its residual the *detail layer* $D_\mathrm{i} = I - B_\mathrm{i}$. We perform the same decomposition on the model image $M$ to get $B_\mathrm{m}$ and $D_\mathrm{m}$. We use histogram transfer to transform the input base $B_\mathrm{i}$ so that it matches the histogram of $B_\mathrm{m}$. We denote by $ht$ the histogram transfer operator and $B_\mathrm{o} = ht_{B_\mathrm{m}}(B_\mathrm{i})$ the base output. For the detail layer, we match the histogram of the amplitudes: $|D_\mathrm{o}| = ht_{|D_\mathrm{m}|}(|D_\mathrm{i}|)$. We obtain the detail layer of the output by using the sign of the input detail: $D_\mathrm{o} = \mathrm{sign}(D_\mathrm{i})\,|D_\mathrm{o}|$. The output frame $O$ is reconstructed by adding the base and detail layers: $O = B_\mathrm{o} + D_\mathrm{o}$.

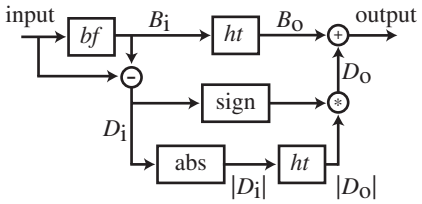



**Figure 11:** *Tone management pipeline.*

**Denoising** A number of videos are noisy or use a compression algorithm that introduces artifacts. These defects are often not noticeable in the original video but may be revealed as we increase the contrast or level of detail. A naïve solution would be to denoise the input frames before processing them but this produces "too clean" images that look unrealistic. We found that adding back the denoising residual after processing yields superior results with a more realistic appearance. In practice, since noise amplitude is low compared to scene edges, we use a bilateral filter with small sigmas.

**Discussion** Compared to the process described by Bae et al., our method directly relies on the detail amplitudes to estimate the level of texture of a frame. Although the *textureness* measure proposed by Bae et al. capture more sophisticated effects, it induces three additional bilateral filtering steps whose computational cost would prevent our algorithm to run in real time on HD sequences. Our results show that detail amplitude is a satisfying approximation. Furthermore, it provides sufficient leeway to include a denoising step that broadens the range of possible inputs.

### 5.3 Local Tone Mapping

We describe a user-driven method to locally tone map HDR images based on grid painting. We build upon Durand and Dorsey's tone mapping algorithm [2002], where the log luminance $L$ of an image is decomposed into a base layer $B = bf(L)$ and a detail layer $D = L - B$. The contrast of the base is reduced using a simple linear remapping $B' = \alpha B + \beta$ while the detail layer $D$ is unaffected. This reduces the overall dynamic range without losing local detail. The final output is obtained by taking the exponential of $B' + D$ and preserving color ratios.

Our method extends this global remapping of the base layer and lets users locally modify the remapping function using an edge-aware brush. We represent the remapping function with a grid $\Gamma_\mathrm{TM}$ initialized with a linear ramp:

$$\Gamma_\mathrm{TM}(x, y, z) = \alpha z + \beta \tag{7}$$

If $\Gamma_\mathrm{TM}$ is unedited, slicing $\Gamma_\mathrm{TM}$ with $B$ yields the same remapped base layer as Durand and Dorsey's operator: $B' = s_B(\Gamma_\mathrm{TM})$.

Users edit $\Gamma_\mathrm{TM}$ with an edge-aware brush to locally modify the grid values. The modified base layer is still obtained by slicing according to $B$. Clicking with the left button on the pixel at location $(x, y)$ adds a 3D Gaussian centered at $(x/s_\mathrm{s}, y/s_\mathrm{s}, L(x, y)/s_\mathrm{r})$ to the grid values. A right click subtracts a 3D Gaussian. In practice, we use Gaussian kernels with a user-specified amplitude $A$ and parameters $\sigma_\mathrm{s} = s_\mathrm{s}$ and $\sigma_\mathrm{r} = s_\mathrm{r}$. The spatial sampling $s_\mathrm{s}$ controls the size of the brush and the range sampling $s_\mathrm{r}$ controls its sensitivity to edges. If users hold the mouse button down, we lock the $z$ coordinate to the value of the first click $L(x_0, y_0)/s_\mathrm{r}$, thereby enabling users to paint without affecting features at different intensities.

Using our GPU algorithm for the bilateral filter that creates the base layer and for grid painting, we tone map a 15 megapixel image at 50 Hz (Figure 1). Refer to the video for a screen capture of a local tone mapping session.

## 6 Discussion and Limitations

**Memory Requirements** Our approach draws its efficiency from the coarser resolution of the bilateral grid compared to the 2D image. However, operations such as a bilateral filter with a small spatial kernel require fine sampling, which results in large memory and computation costs for our technique. In this case, Weiss's approach is more appropriate [2006]. Nonetheless, for large kernels used in computational photography applications, our method is significantly faster than previous work.

Due to memory constraints, the bilateral grid is limited to a one-dimensional range that stores image intensities and can cause problems at isoluminant edges. We found that in most cases, we achieve good results with 7 to 20 levels in $z$. A future direction of research is to consider how to efficiently store higher dimensional bilateral grids: 5D grids that can handle color edges and even 6D grids for video. Another possibility is to look at fast dimensionality reduction techniques to reduce the memory limitations.

**Interpolation** We rely on trilinear interpolation during slicing for optimal performance. Higher-order approaches can potentially yield higher-quality reconstruction. We would like to investigate the tradeoff between quality and cost in using these filters.

**Thin Features** Techniques based on the bilateral grid have the same properties as the bilateral filter at thin image features. For example, in an image with a sky seen through a window frame, the edge-aware brush affects the sky independently of the frame; that is, the brush paints across the frame without altering it. Whether or not a filter stops at thin features is a fundamental difference between bilateral filtering and diffusion-based techniques. We believe that both behaviors can be useful, depending on the application.

## 7 Conclusion

We have presented a new data structure, the bilateral grid, that enables real-time edge-preserving image manipulation. By lifting image processing into a higher dimensional space, we are able to design algorithms that naturally respect strong edges in an image. Our approach maps well onto modern graphics hardware and enables real-time processing of high-definition video.

**Acknowledgements** We thank the MIT Computer Graphics Group and the anonymous reviewers for their comments. We are especially grateful to Jonathan Ragan-Kelley for fruitful discussions on GPU programming and Tom Buehler for his assistance in making the video. This work was supported by a National Science Foundation CAREER award 0447561 "Transient Signal Processing for Realistic Imagery," an NSF Grant No. 0429739 "Parametric Analysis and Transfer of Pictorial Style," and a grant from Royal Dutch/Shell Group. Jiawen Chen is partially supported by an NSF Graduate Research Fellowship and an NVIDIA Fellowship. Frédo Durand acknowledges a Microsoft Research New Faculty Fellowship and a Sloan Fellowship.

## References

AURICH, V., AND WEULE, J. 1995. Non-linear gaussian filters performing edge preserving diffusion. In *Proceedings of the DAGM Symposium*.

BAE, S., PARIS, S., AND DURAND, F. 2006. Two-scale tone management for photographic look. *ACM Transactions on Graphics 25*, 3, 637 – 645. Proceedings of the ACM SIGGRAPH conference.

BARASH, D. 2002. A fundamental relationship between bilateral filtering, adaptive smoothing and the nonlinear diffusion equation. *IEEE Transactions on Pattern Analysis and Machine Intelligence 24*, 6, 844.

BENNETT, E. P., AND MCMILLAN, L. 2005. Video enhancement using per-pixel virtual exposures. *ACM Transactions on Graphics 24*, 3 (July), 845 – 852. Proceedings of the ACM SIGGRAPH conference.

BLINN, J. F. 1996. Fun with premultiplied alpha. *IEEE Computer Graphics and Applications 16*, 5, 86–89.

CHIU, K., HERF, M., SHIRLEY, P., SWAMY, S., WANG, C., AND ZIMMERMAN, K. 1993. Spatially nonuniform scaling functions for high contrast images. In *Proceedings of the conference on Graphics Interface*, 245–253.

DECARLO, D., AND SANTELLA, A. 2002. Stylization and abstraction of photographs. *ACM Transactions on Graphics 21*, 3. Proceedings of the ACM SIGGRAPH conference.

DURAND, F., AND DORSEY, J. 2002. Fast bilateral filtering for the display of high-dynamic-range images. *ACM Transactions on Graphics 21*, 3. Proceedings of the ACM SIGGRAPH conference.

EISEMANN, E., AND DURAND, F. 2004. Flash photography enhancement via intrinsic relighting. *ACM Transactions on Graphics 23*, 3 (July). Proceedings of the ACM SIGGRAPH conference.

FELSBERG, M., FORSSÉN, P.-E., AND SCHARR, H. 2006. Channel smoothing: Efficient robust smoothing of low-level signal features. *IEEE Transactions on Pattern Analysis and Machine Intelligence 28*, 2 (February), 209–222.

GONZALES, R. C., AND WOODS, R. E. 2002. *Digital Image Processing*. Prentice Hall. ISBN 0201180758.

GOODNIGHT, N., WOOLLEY, C., LEWIN, G., LUEBKE, D., AND HUMPHREYS, G. 2003. A multigrid solver for boundary value problems using programmable graphics hardware. In *Proceedings of the ACM SIGGRAPH / EUROGRAPHICS conference on Graphics Hardware*.

HARRIS, M., AND LUEBKE, D. 2004. GPGPU. In *Course notes of the ACM SIGGRAPH conference*.

LEVIN, A., LISCHINSKI, D., AND WEISS, Y. 2004. Colorization using optimization. *ACM Transactions on Graphics 23*, 3 (July). Proceedings of the ACM SIGGRAPH conference.

LEVIN, A., RAV-ACHA, A., AND LISCHINSKI, D. 2007. Spectral matting. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*.

LISCHINSKI, D., FARBMAN, Z., UYTTENDAELE, M., AND SZELISKI, R. 2006. Interactive local adjustment of tonal values. *ACM Transactions on Graphics 25*, 3, 646 – 653. Proceedings of the ACM SIGGRAPH conference.

OH, B. M., CHEN, M., DORSEY, J., AND DURAND, F. 2001. Image-based modeling and photo editing. In *Proceedings of the ACM SIGGRAPH conference*.

PARIS, S., AND DURAND, F. 2006. A fast approximation of the bilateral filter using a signal processing approach. In *Proceedings of the European Conference on Computer Vision*.

PERONA, P., AND MALIK, J. 1990. Scale-space and edge detection using anisotropic diffusion. *IEEE Transactions Pattern Analysis Machine Intelligence 12*, 7 (July), 629–639.

PETSCHNIGG, G., AGRAWALA, M., HOPPE, H., SZELISKI, R., COHEN, M., AND TOYAMA, K. 2004. Digital photography with flash and no-flash image pairs. *ACM Transactions on Graphics 23*, 3 (July). Proceedings of the ACM SIGGRAPH conference.

PHAM, T. Q., AND VAN VLIET, L. J. 2005. Separable bilateral filtering for fast video preprocessing. In *Proceedings of the IEEE International Conference on Multimedia and Expo*.

SMITH, S. M., AND BRADY, J. M. 1997. SUSAN – a new approach to low level image processing. *International Journal of Computer Vision 23*, 1 (May), 45–78.

SOCHEN, N., KIMMEL, R., AND MALLADI, R. 1998. A general framework for low level vision. *IEEE Transactions in Image Processing 7*, 310–318.

TOMASI, C., AND MANDUCHI, R. 1998. Bilateral filtering for gray and color images. In *Proceedings of the IEEE International Conference on Computer Vision*, 839–846.

TUMBLIN, J., AND TURK, G. 1999. LCIS: A boundary hierarchy for detail-preserving contrast reduction. In *Proceedings of the ACM SIGGRAPH conference*, 83–90.

WEISS, B. 2006. Fast median and bilateral filtering. *ACM Transactions on Graphics 25*, 3, 519 – 526. Proceedings of the ACM SIGGRAPH conference.

WILLIS, P. J. 2006. Projective alpha colour. *Computer Graphics Forum 25*, 3 (Sept.), 557–566. 0167-7055.

WINNEMÖLLER, H., OLSEN, S. C., AND GOOCH, B. 2006. Real-time video abstraction. *ACM Transactions on Graphics 25*, 3, 1221 – 1226. Proceedings of the ACM SIGGRAPH conference.

XIAO, J., CHENG, H., SAWHNEY, H., RAO, C., AND ISNARDI, M. 2006. Bilateral filtering-based optical flow estimation with occlusion detection. In *Proceedings of the European Conference on Computer Vision*.