COMPUTATIONAL VIDEO ENHANCEMENT

Eric P. Bennett

A dissertation submitted to the faculty of the University of North Carolina at Chapel Hill in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Computer Science.

Chapel Hill
2007

ABSTRACT

ERIC P. BENNETT: Computational Video Enhancement
(Under the direction of Leonard McMillan)

During a video, each scene element is often imaged many times by the sensor. I propose that by combining information from each captured frame throughout the video it is possible to enhance the entire video. This concept is the basis of computational video enhancement. In this dissertation, the viability of computational video processing is explored in addition to presenting applications where this processing method can be leveraged.

Spatio-temporal volumes are employed as a framework for efficient computational video processing, and I extend them by introducing sheared volumes. Shearing provides spatial frame warping for alignment between frames, allowing temporally-adjacent samples to be processed using traditional editing and filtering approaches. An efficient filter-graph framework is presented to support this processing along with a prototype video editing and manipulation tool utilizing that framework.

To demonstrate the integration of samples from multiple frames, I introduce methods for improving poorly exposed low-light videos to achieve improved results. This integration is guided by a tone-mapping process to determine spatially-varying optimal exposures and an adaptive spatio-temporal filter to integrate the samples. Low-light video enhancement is also addressed in the multispectral domain by combining visible and infrared samples. This is facilitated by the use of a novel multispectral edge-preserving filter to enhance only the visible spectrum video.

Finally, the temporal characteristics of videos are altered by a computational video resampling process. By resampling the video-rate footage, novel time-lapse sequences are found that optimize for user-specified characteristics. Each resulting shorter video is a more faithful summary of the original source than a traditional time-lapse video. Simultaneously, new synthetic exposures are generated to alter the output video's aliasing characteristics.

To My Father,

Robert A. Bennett,

The Original Dr. Bennett

ACKNOWLEDGMENTS

I would like to thank my advisor, Leonard McMillan, for working with me for five years and guiding my development as a researcher. I will certainly miss our enthusiastic brainstorming sessions, where Leonard's drive to explore the limits of computer science was most evident.

I would also like to thank the rest of my committee: Gary Bishop, Guido Gerig, Sing Bing Kang, Marc Pollefeys, and Carlo Tomasi. I am honored to have such a distinguished committee and I appreciate their time and feedback on my work.

I want to thank John Mason for the multispectral datasets in Section 4.3 as well as Aaron Block, Tim Malone, Terry McMahon, and John Moriconi for appearing in my videos.

My thanks as well to my collaborators on projects outside of this dissertation: Jason Stewart, Jingdan Zhang, Sing Bing Kang, Rick Szeliski, Matt Uyttendaele, and Larry Zitnick.

Discussions with friends and colleagues at UNC have also influenced my work: Aaron Block, Dave Borland, Fred Brooks, Brian Eastwood, Russ Gayle, Justin Hensley, Tyler Johnson, Luv Kohli, Anselmo Lastra, E. Scott Larsen, Brandon Lloyd, Ketan Mayer-Patel, Chris Oates, Rick Skarbez, Josh Steinhurst, Russell Taylor, Chris VanderKnyff, Jeremy Wendt, Jingyi Yu, and many more. Thanks also to my friends from around the country: Derek Jarvis, Zac Johnson, Liz Koster, John Moriconi, and Zak Vassar.

Finally, I would like to thank my parents, Robert and Marilyn, for being such wonderful inspirations throughout my life. Both teachers, they instilled in me a deep appreciation for eduction I have since carried with me. I dedicated this work to my father for being the Ph.D. role model I have always aspired to and also for introducing me to the wonders of computer technology, going all the way back to the TI-99/4A and later including some of the first digital video editors. Finally, thanks to my brother Rob and his wife Kristie for their support (and also the numerous music recommendations that have kept me listening through grad school).

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

1D	One-Dimensional
2D	Two-Dimensional
3D	Three-Dimensional
ACM	Association for Computing Machinery
A/D	Analog to Digital
ADC	Analog to Digital Converter
API	Application Programming Interface
ASTA	Adaptive Spatio-Temporal Accumulation
AVI	Audio Video Interleave
CCD	Charge-Coupled Device
CMOS	Complementary Metal-Oxide-Semiconductor
codec	Compression/Decompression
CPU	Central Processing Unit
DLP	Digital Light Processing
DV	Digital Video
FAQ	Frequently Asked Questions
FIFO	First In, First Out
fps	Frames Per Second
GB	Gigabyte
GHz	Gigahertz
GPU	Graphics Processing Unit
HDR	High Dynamic Range
IEEE	Institute of Electrical and Electronics Engineers
InGaAs	Indium Gallium Arsenide
IR	Infrared
LCD	Liquid Crystal Display
LDR	Low Dynamic Range

LED	Light-Emitting Diode
LS	Large-Scale
LSI	Linear Spatially/Shift Invariant
MiniDV	Miniature Digital Video (IEC 61834)
NIR	Near Infrared
nm	Nanometer
NTSC	National Television Systems Committee
OpenCV	Open Source Computer Vision Library
OpenGL	Open Graphics Library
PAL	Phase Alternating Line
PC	Personal Computer
PFilter	Proscenium Filter
RAM	Random Access Memory
RGB	Red Green Blue
RGBα	Red Green Blue Alpha
ROAD	Rank-Order Absolute Difference
s	Second
μ**s**	Microsecond
SAD	Sum-of-Absolute-Differences
SIGGRAPH	Special Interest Group on Graphics and Interactive Techniques
SNR	Signal-To-Noise Ratio
SSD	Sum-of-Squared-Differences
SUSAN	Smoothing over Univalue Segment Assimilating Nucleus
SWIR	Short Wave Infrared
VEC	Virtual Exposure Camera
YUV	Luminance and Chrominance Color Space

CHAPTER 1

INTRODUCTION

The popularity of digital video camcorders has ushered in a desktop digital video revolution for both professional and amateur users. High quality video can be captured, imported, and processed on commodity computers at minimal cost and with unprecedented ease. Having such video available on a computer lends itself to traditional video editing applications. Thus, along with the availability of digital video hardware came software editing tools of varying complexities for many different skill levels. However, these tools focused primarily on the temporal rearrangement of short video clips as they were imaged. The problem of making the underlying footage *better*, to significantly improve its visual quality and enhance its content, presents the next challenge and opportunity for digital video.

I first consider traditional video processing approaches for enhancing the quality of video. Simple tasks such as the global adjustment of brightness, contrast, color balance, and other characteristics are all well understood and are the limit of functionality in most video editors. Few local enhancements, such as noise reduction, are available and those that are offered are typically simple linear filters considering at most naïve frame-by-frame filtering or very small temporal neighborhoods, being an outgrowth of analog processing. However, with modern digital hardware and random access to all pixels in all video frames simultaneously, far more complex, adaptive, and non-linear approaches can now be taken that require the storage and computational power that has only recently become available.

When considering the possibilities of such processing, it is worthwhile to consider similar work on still images. Specifically, the field of *computational photography* addresses the enhancement of still images by considering information in a small number of similar images taken at different times or under different camera settings. Thus, by combining these images together with computational methods, a image superior to anything possible to capture with a physical camera is created. For instance, visual elements that appear in only one image may be combined with visual elements from other images taken minutes apart to create a realistic composite image. Or, images taken at different exposures allow a combined image with dynamic range beyond that of the sensor to be computed. Lighting may be transferred, noise reduced, spectra fused, or many other effects not possible in-camera that leverage both computation and the fact that many images can be taken at no additional cost.

Bringing the benefits of computational photography to traditional video processing, the emerging field of *computational video* addresses combining elements between frames to generate an enhanced video output. Computational video has vast potential because so much information is available for processing when capturing at video rates. For example, at 30 frames per second, 1,800 individual frames are captured every minute, and 108,000 frames are captured every hour.

Along with this additional data comes new issues that uniquely arise in computational video in comparison to computational photography. First, the output is a video sequence as opposed to a single frame. Thus, frame-to-frame enhancement decisions must be made consistently so they do not introduce artifacts when played back at full speed (*temporal coherence*). Second, the number of input images makes identification of the most useful information difficult. In addition, the infrastructure and underlying data structures must be able to support the increased quantity of data.

Overcoming these issues enables many powerful new video tools. Tasks such as video compositing and object removal can be performed in a temporally coherent manner. Videos may be adaptively filtered and enhanced using large kernels that span space, time, and even across multiple spectra. The linear progression of time can be altered and resampled to change the underlying structure of the original video.

Thus, given this potential of computational video, my thesis statement is:

Computational video enables a new class of processing tools for enhancing and improving video capture quality by leveraging information found across many frames.

1.1 Contributions

My research makes the following contributions:

- *I employ spatio-temporal video volumes as a domain for computational video operations and extend them with shearing.* Spatio-temporal volumes stack the frames of a video in chronological order to create a 3D volume, as shown in Figure 1.1, thus emphasizing relationships between adjacent spatial and temporal samples. *Shearing* is the process of spatially warping the frames of the volume to align temporally-adjacent samples to greatly assist the processing of moving objects and non-static cameras.

- *I develop the Proscenium spatio-temporal video editing framework to address the memory and processing requirements of computational video and spatio-temporal volumes.* This framework encapsulates spatio-temporal volumes within a *per-pixel, bi-directional, lazily-evaluated filter graph model.* This model also supports *virtual shears* that can be created and removed dynamically to support editing. This framework is then used to develop a prototype video editor that performs computational video manipulation tasks such as object removal, temporal filtering, and multi-frame editing.

- *I present a computational video approach to improving the quality of captured low-light, LDR (Low Dynamic Range) video by virtually extending the exposure times of each*

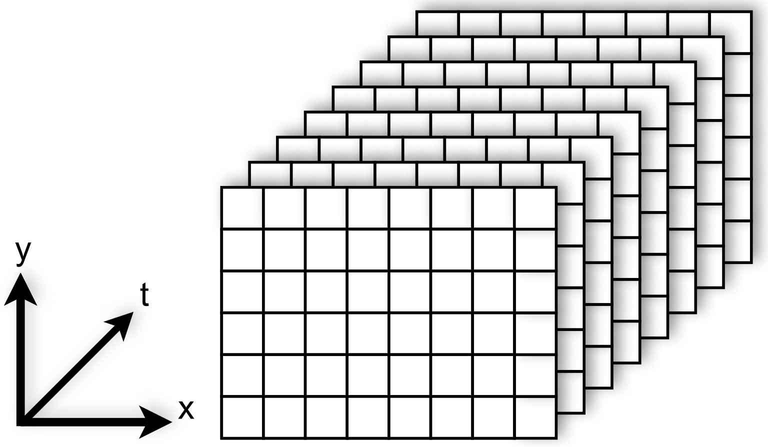

Figure 1.1: An illustration of a spatio-temporal volume constructed as a series of video frames stacked on top of each other. The samples in the volume are indexed using an (x, y, t) sampling.

sample (pixel). This is possible with the non-linear integration of information from large neighborhoods of temporally and spatially adjacent samples. Two interconnected concepts make this possible. First is a *spatially-varying LDR tone mapping algorithm* that finds the optimal exposure time in a well-exposed image. This is used as an objective for the *integration and noise reduction ASTA (Adaptive Spatio-Temporal Accumulation) filter* which extends the exposure time to find the correct original luminance had it been properly exposed. When used together, this combined method is referred to as the *Virtual Exposure Camera model.*

- To improve captured video in a different context, *I also address enhancing low-light video using spectra outside of the standard visible red, green, and blue channels.* Specifically, information from registered video in the IR spectrum is used to enhance noisy RGB footage. My approach differs from previous multispectral fusions because it enhances the RGB footage using the IR, but without introducing elements imaged only in IR. This is accomplished through *edge-preserving decomposition and cross-spectral normalization*, assisted by a novel *dual bilateral* filter. This filter preserves edges detected in both spectra but only uses samples from the visible-spectrum video.

- Having considered video manipulation and enhancement as examples of computational video, *I then consider temporal resampling of videos to alter their duration.* Specifically, the problem of time-lapse video generation from video-rate footage is considered. *A non-uniform sampling algorithm is presented that optimizes the sampling of the input video to match the user's desired duration and visual output characteristics.* This can be either to generate time-lapse videos that preserve motion, avoid fast motion, control the uniformity of the output sampling, or a combination of these characteristics. By considering the entire video as a whole, the optimal sampling can be found.

- To complement the computational video resampling, *I extend ideas from computational photography to combine the input frames together to alter the aliasing characteristics of the video output.* This *virtual shutter* combines many frames together using both common linear and non-linear filtering methods, such as low-pass, minimum, maximum, and median filtering, and more complex effects, such as compositing. Thus, new synthetic exposures are created which are related to both the low-light ASTA exposures and the spatio-temporal filtering performed in Proscenium.

1.2 Dissertation Overview

My approaches to computational video are presented as follows:

To begin, the field of computational video will be reviewed in Chapter 2. Because this field is an outgrowth of image processing, video processing, and computational photography, relevant topics from those areas are discussed as well.

In Chapter 3, the use of spatio-temporal volumes for computational video processing is examined. This is presented along with volume shearing, the Proscenium spatio-temporal video editing framework, and discussion of how computational video operations, in this case video editing and enhancement, can be performed within these volumes.

In Chapter 4, a computational approach to video enhancement is discussed for improving the quality of noisy low-light videos, showing the strengths of adaptive filtering with large spatio-temporal kernels. Visible-spectrum-only enhancement is discussed, with the Virtual Exposure Camera model, combining LDR tone mapping and the ASTA filter. Multi-spectral video enhancement is then addressed with a focus on the new dual bilateral filter.

In Chapter 5, a computational video approach to video resampling and frame combination is presented to shorten the duration of videos into time-lapse sequences. This demonstrates the ability of computational video to dramatically alter the content of an input video. The sampler is presented along with a variety of metrics to achieve many output sampling characteristics. The output frames are then generated by the virtual shutter that combines all the frames in the original video using linear and non-linear techniques.

Finally, Chapter 6 concludes my discussion of computational video and presents possible future directions for research in the field.

CHAPTER 2

PREVIOUS WORK

In this chapter, I provide an overview of the previous literature regarding computational video. Computational video is an outgrowth of video processing and computational photography, both of which are based on image processing. Thus, to consider computational video is to consider each of these component fields. Specifically, the common thread through this discussion is that the potential of computational video is derived from its ability to enhance the samples in a video by combining observations from other similar frames.

I begin this literature review by considering the established methods for processing digital video: traditional video editors and video processing frameworks. I then review recent work in spatio-temporal volumes, which present an alternate representation of video that lends itself to filtering and enhancement applications. Further investigation of traditional filtering is then discussed in the context of the classic image processing area of noise reduction. The focus then shifts to computational photography, specifically in the domains of High Dynamic Range (HDR) imaging, multispectral fusion, and multi-image combination. Given all this work as a basis, computational video is then considered, addressing both the summarization and compositing of video sources.

2.1 Video Editing

The most common interface for processing or enhancing video is through a dedicated video editing application. Therefore, I begin by discussing the goals of these applications along with those of the low-level frameworks often used to encapsulate video processing.

The primary function of most modern digital video editing systems, such as Apple Final Cut Pro, Adobe Premiere, and Avid Media Composer, is to cut raw footage into a series of clips, and then assemble those clips with transitions along some timeline into a finished video. Such editors perform cuts, cropping, editing, color-correction, and insertion of transitions between clips. Applications such as Adobe After Effects focus on making modifications primarily on individual clips (often a few seconds long). These modifications are often more complex and concentrate on the modification of video pixels and less on temporal arrangement. These operations are closer to the type of actions considered by computational video.

The visual interface used for interaction in all of these applications is a timeline plus a frame-by-frame viewer. Thus time and space dimensions are treated inherently different from each other. This does not lend itself well to the incorporation of samples and other elements from multiple frames that are necessary for computational video. Alternate representations, such as 3D spatio-temporal volumes, discussed in Chapter 3, consider the entirety of a video at once. However, there have been recent efforts in mixing standard 2D image editing user-interface metaphors with such 3D visualizations. Ideas from Adobe Photoshop, a popular commercial image-editing tool, are, in fact, frequently applied to both 3D and video applications because of its ease of use, flexibility, and rich set of tools. Furthermore, systems have been proposed to extend Photoshop's rich image-editing environment to volumetric data (Zwicker et al., 2002). By extension, such techniques may hold promise for processing video in spatio-temporal volumes.

Many video processing systems are based upon filter graphs which have been utilized for many applications in the area of image and multimedia processing (Pratt, 1997). Conceptually, the idea is that individual processing components can be ordered and arranged so that data flows from the input of a filter graph to the output; passing through the interconnected components that lie along that path. Each component (often called a filter) may modify data before passing it along to the next filter. Each filter is only responsible for processing data in a standardized manner without knowledge of what filters might be connected to it. This allows a uniform interface with components that are interconnected in any order, as in the Decorator design pattern (Gamma et al., 1995). These filters are often representative of the common operations that occur throughout video editing: color correction, smoothing, scaling, and compositing.

For instance, Apple's QuickTime (Apple Inc., 2007) framework and Microsoft's DirectShow (Microsoft Corporation, 2007) framework implement multimedia filter graphs for video and audio. At a higher level of abstraction, the Berkeley Continuous Media Toolkit (Mayer-Patel and Rowe, 1997) implements a powerful filter graph in the form of a scripting protocol, as opposed to a compiled API. Filter graphs treat video data as a stream that flows in buffers of entire frames of pixels from the graph's input to the graph's output. This is ideal when an entire frame's output is desired, but is not ideal to calculate small sub-frame regions.

As mentioned, the frame-by-frame timeline nature of these video editors and frameworks differs from the needs of computational video to consider many frames simultaneously for enhancement. Thus, a different method for thinking about video, the spatio-temporal volume, is now discussed.

2.2 Spatio-Temporal Volumes

Considering videos as three-dimensional volumes of their stacked constituent frames was first proposed in the epipolar video processing research of Bolles et al. (1987). In this work, object and scene motion are measured by taking a planar *cut* of a spatio-temporal volume containing the video. Each cut results in a still image that contains portions of many frames, allowing for correlations to be made in 2D between position and time to measure object velocity. An underlying assumption of these measurements is that the camera that captured the source video was static so that spatial positions in the video are constant through time.

These volumes and planar cuts were later realized as an interactive visualization in "Interactive Video Cubism" (Fels et al., 2000). These spatio-temporal cuts (originally planar and later spherical) again provide a view into the video, but do not modify the underlying data. More advanced applications are then explored in (Klein et al., 2001) and (Klein et al., 2002), which both use multiple spatio-temporal cuts to create artistic video interpretations of the source material. In an extreme case, "Making Space For Time in Time-Lapse Photography" (Terry et al., 2004) scales the cut plane concept to visualize the total contents of multiple days of video footage in a single image. Alternatively, Zomet et al. (2003) identified particular slices through videos with specific camera motions that correspond to camera projections different from the projection of the imaging device. These works demonstrate a wide range of visualizations that motivate the possibilities of spatio-temporal processing.

The spatial-alignment of individual frames is essential to video processing within a spatio-temporal volume. As will be further discussed in Section 3.1, this alignment guarantees that a consistent spatial pixel location in the volume is always imaging the same scene element. Thus, enhancement algorithms can access other samples of the same object at the same (x, y) coordinate but at different t values (time). I call this spatial remapping of the volume *shearing*. Planar cuts of a sheared volume represent non-planar cuts of the original spatio-temporal volume. Non-planar spatio-temporal cuts have since been used as an interface to specify multi-frame compositing operations (Wang et al., 2005).

It follows then that the parameters for shearing are derived from stabilizing a scene element through time. Shearing can be used to stabilize each element one-at-a-time, allowing each visual element to be edited, processed, or enhanced independently. This approach is influenced by the video layers concepts of Wang and Adelson (1994) who developed the notion that general planes of motion in a video should be edited independently.

To solve for a spatio-temporal volume's underlying shear function, a variety of stabilization methods exist. Buehler et al. (2001) demonstrate how foreground and background stabilization can be used to generate novel videos with refined camera and object motions. Their work relies on extensive offline analysis for dense feature tracking, local warping, and iterative smoothing of the source sequence. More recently, Sand and Teller (2004) presented a "Video Matching" method for aligning slightly different video sources. This alignment is between videos, and is able to robustly handle cases of missing scene elements between those videos. At a lower-level, single visual elements (trackable points) or sparse sets of points can be tracked through a video using a feature tracker such as the Lucas-Kanade algorithm (Shi and Tomasi, 1994). An efficient implementation of this technique is publicly available as part of the OpenCV toolkit (Bouguet, 2000).

Spatio-temporal volumes can serve as the basis for a wide range of computational video techniques, as further explored in Chapter 3. I next consider another class of video processing techniques that combine information from local neighborhoods to create an improved or enhanced result. For example, noise filtering can be considered as a computational video method.

2.3 Noise Filtering

A unique strength of computational video is its ability to improve or enhance every pixel value of a video by considering the additional temporal and spatial information within a local neighborhood of samples. Similarly, the standard concept of filtering changes the value of a signal's sample based upon a kernel of surrounding samples, using either linear or non-linear methods. Such filtering is an essential part of computational video.

One particularly useful class of filters is for noise reduction. The noise apparent in images and video sources is due to many factors, including sensor noise, low signal strength, and data corruption (noise characteristics, measurement, and modeling are discussed in Section 4.1). Here, an introduction to 2D spatial noise filtering is presented along with an overview of the video filtering literature.

2.3.1 Spatial Filtering

Noise filtering methods have a long history throughout the signal processing literature. First, I discuss noise filters for processing still images and then I consider noise reduction methods for video in the next section as an extension of those methods.

The most basic noise filter to consider is Gaussian smoothing (and its discrete approximations), which results in a spatial low-pass filter of the image (Bovik, 2000). While Gaussian smoothing is effective at removing random shot noise (noise characteristics are discussed in more detail in Section 4.1), high-frequencies are also removed, thus blurring the image edges and textures. The formula for a general n-dimensional Gaussian filter with equal support in all dimensions (*i.e.*, a 1D Gaussian falloff based on Euclidean proximity) is given below:

$$J_s = \frac{\sum\limits_{p \in \Omega} g(\|p - s\|, \sigma_h) I_p}{\sum\limits_{p \in \Omega} g(\|p - s\|, \sigma_h)}, \tag{2.1}$$

$$g(x, \sigma) \equiv \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-x^2}{2\sigma^2}}. \tag{2.2}$$

Gaussian smoothing is an LSI (Linear Spatially/Shift Invariant) filter, incorporating information from nearby samples in its 2D kernel Ω and weighting their importance uniformly based on the proximity of each pixel to the filter's center (its output pixel). It can be considered as a "domain" filter of the kernel's samples. The weighting is center biased and falls off as a normalized Gaussian function (Eq. 2.2), hence its name.

An alternate class of filters, called range filters, combine samples whose weights are based instead upon photometric proximity rather than their spatial proximity. One popular range filter, the Sigma filter (Lee, 1983), combines samples within the spatial kernel Ω that are within 2σ of the value at the kernel's center. The value of σ can be determined by the standard deviation of the original image itself. Sigma filtering is very sensitive to the size of Ω because faraway samples can make equivalent contributions to those at the center of the kernel, potentially corrupting edges due to its non-regularizing formulation.

To overcome the edge blurring artifacts of the Gaussian filter and the potential edge corruption artifacts of the Sigma filter, edge-preserving filters can be used from the anisotropic diffusion and bilateral filter families. These filters attempt to filter pixels within smooth areas while avoiding crossing over contours of significant change (*i.e.*, edges). Anisotropic diffusion of images (Perona and Malik, 1990) provides an iterative filtering method that adapts to the image's gradient, based upon heat flow equations:

$$I_t = div((c(x, y, t)\nabla I) = c(x, y, t)\Delta I + \nabla c \cdot \nabla I, \tag{2.3}$$

$$c(x, y, t) = \upsilon\left(||\nabla I(x, y, t)||\right), \tag{2.4}$$

$$\upsilon(\nabla I(x, y, t)) = e^{(-(||\nabla I||/K)^2)} \;\; or \;\; \upsilon(\nabla I(x, y, t)) = \frac{1}{1 + \left(\frac{||\nabla I||}{K}\right)^2}. \tag{2.5}$$

The value of K determines which edges are preserved and which are smoothed, based on their gradient magnitudes. In the discrete implementation of anisotropic diffusion given by Perona and Malik (1990), only the four adjacent pixels are considered at each step. Thus, to smooth large neighborhoods many iterations of the diffusion process are required. This results in the most significant downside to anisotropic diffusion: its slow execution speed.

Alternatively, bilateral filtering (Tomasi and Manduchi, 1998) provides a single-pass noise removal process that shares many of the advantages of anisotropic diffusion, as discussed by Barash et al. (2002). It is simultaneously a range and a domain filter, weighting samples based on both spatial proximity and photometric similarity. The bilateral filter is also a specific instance of the more general range and domain SUSAN filter of Smith and Brady (1997). Bilateral filtering requires two parameters: σ_h, the Gaussian spatial falloff, and σ_i, the Gaussian intensity difference falloff. The bilateral filter is formulated as:

$$J_s = \frac{\sum\limits_{p \in \Omega} g(\|p - s\|, \sigma_h) g(D(p,s), \sigma_i) I_p}{\sum\limits_{p \in \Omega} g(\|p - s\|, \sigma_h) g(D(p,s), \sigma_i)}, \tag{2.6}$$

$$D(p,s) \equiv I_p - I_s. \tag{2.7}$$

The utility of bilateral filtering goes beyond applications in edge-preserving noise reduction filtering. The output of the bilateral filter contains smooth regions separated by sharp edges, which are commonly called the *large-scale features*. These large-scale features can be considered a piecewise-constant approximation of the original image. The differences between the original image and the bilateral filter's output are the *detail features* which contain the textures, as shown in Figure 2.1. The utility of this large-scale/detail decomposition was originally discussed in terms of High Dynamic Range (HDR) processing by Durand and Dorsey (2002), where each component was processed separately. More generally, the decomposition and recomposition of image components is related to earlier work by Peli and Lim (1982), who used high-pass and low-pass frequency separation of signals in a similar manner.

There have also been many extensions to the bilateral filter. Boomgaard and Weijer (2002) posed the question of how to improve the robustness of the bilateral filter's noise-handling by considering alternate dissimilarity values (Eq. 2.7) that measure photometric differences. The trilateral filter (Choudhury and Tumblin, 2003) takes a different approach to improving the bilateral filter model by biasing its kernel away from edges and dynamically choosing the kernel's size in an attempt to model signals as piecewise-linear rather than piecewise-constant

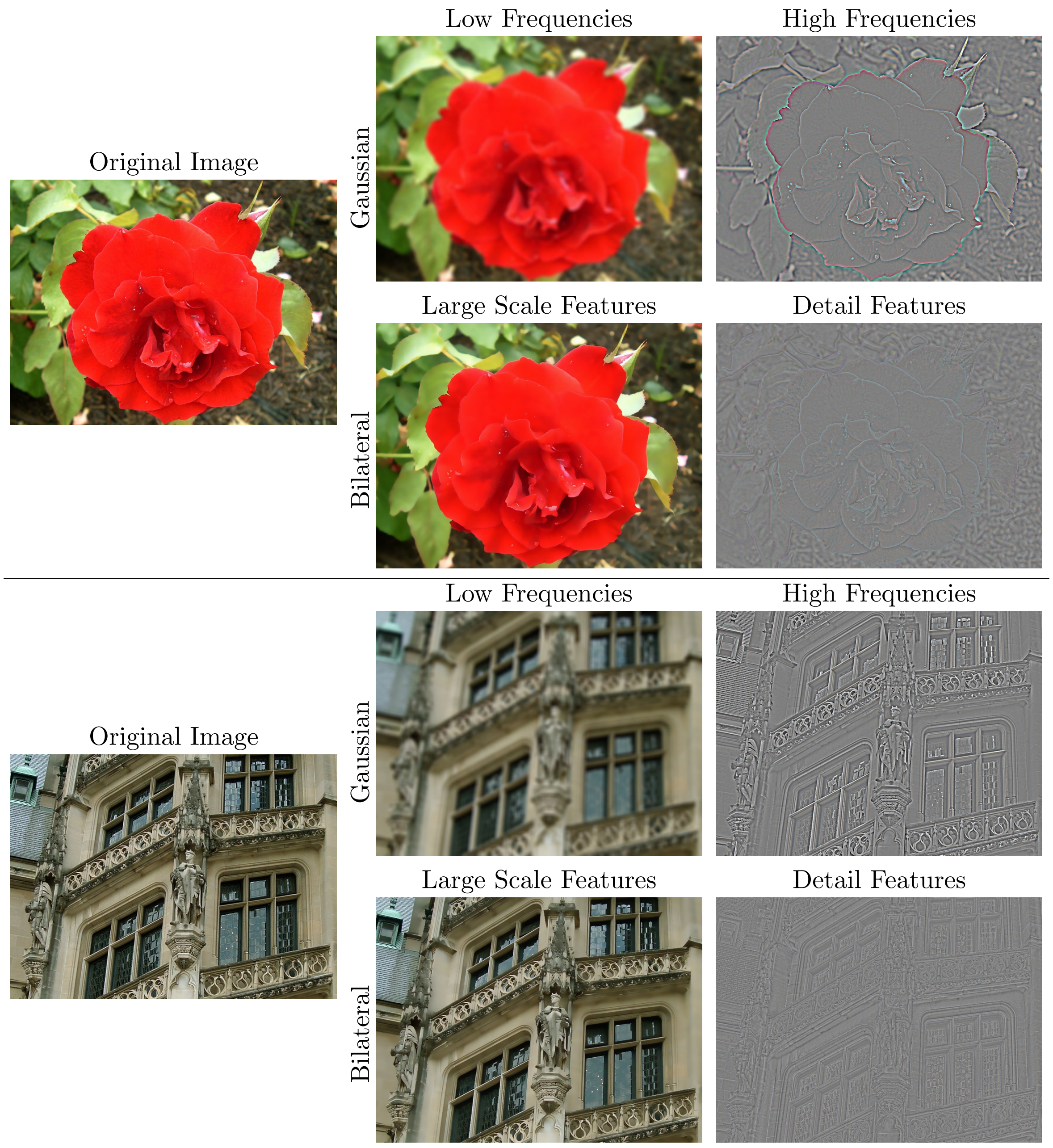

Figure 2.1: Images comparing high/low frequency decomposition from Gaussian filtering with large-scale/detail feature separation with the edge-preserving bilateral filter. In the Gaussian decomposition, the sharp edges are captured in the high-frequencies, whereas in the bilateral decomposition they are part of the large-scale features, keeping only the subtle textures in the detail features.

functions in the limit. Although the quality of filtering is improved, performance is decreased. If speed is required, a fast bilateral implementation ($O(log\ r)$, where r is the kernel radius) is presented by Weiss (2006) that assumes no spatial falloff (*i.e.*, $\sigma_h = \infty$).

To specifically address the reduction of severe shot noise and salt-and-pepper noise, statistically-motivated rank order filters were introduced. These filters sort the values in the kernel Ω and use that information to select only certain values to use, such as the median, minimum, or maximum. The most popular of these is the median filter (Tukey, 1971), which chooses the median intensity from all intensities in the kernel. This reduces noise, but can corrupt edges by growing or shrinking them (dilation or erosion) similar to Sigma filtering, as they are influenced by the kernel's samples.

The "bilateral median" filter described by Francis and Jager (2003) combines bilateral filtering and median filtering to reject outliers from the bilateral kernel. Garnett et al. (2005) used a robust Rank Order Absolute Difference (ROAD) metric to robustly detect if the bilateral kernel itself is centered on a sample of shot noise. This metric is the sum of the absolute differences of the n most similar neighboring pixels to the kernel's center value I_s. The original bilateral filter preserves shot noise because it is considerably different from its neighbors, but the ROAD metric distinguishes shot noise from actual edges that appear in neighboring pixels.

To demonstrate the effects of many of these standard filters, comparisons are provided in Figures 2.2 and 2.3. These filters are designed for 2D still images, so their kernels are entirely spatial. For video noise reduction, however, 3D kernels that utilize temporal information become possible.

2.3.2 Video Filtering

When processing videos, as opposed to still images, noise filtering can be performed considering information from adjacent temporal samples to introduce more samples into the kernel and improve consistency in the filtering from frame-to-frame (temporal coherence). For instance, Dubois and Sabri (1984) perform non-linear temporal noise filtering assisted by optical flow displacement estimation. Each pixel is combined temporally using a recursive low-pass

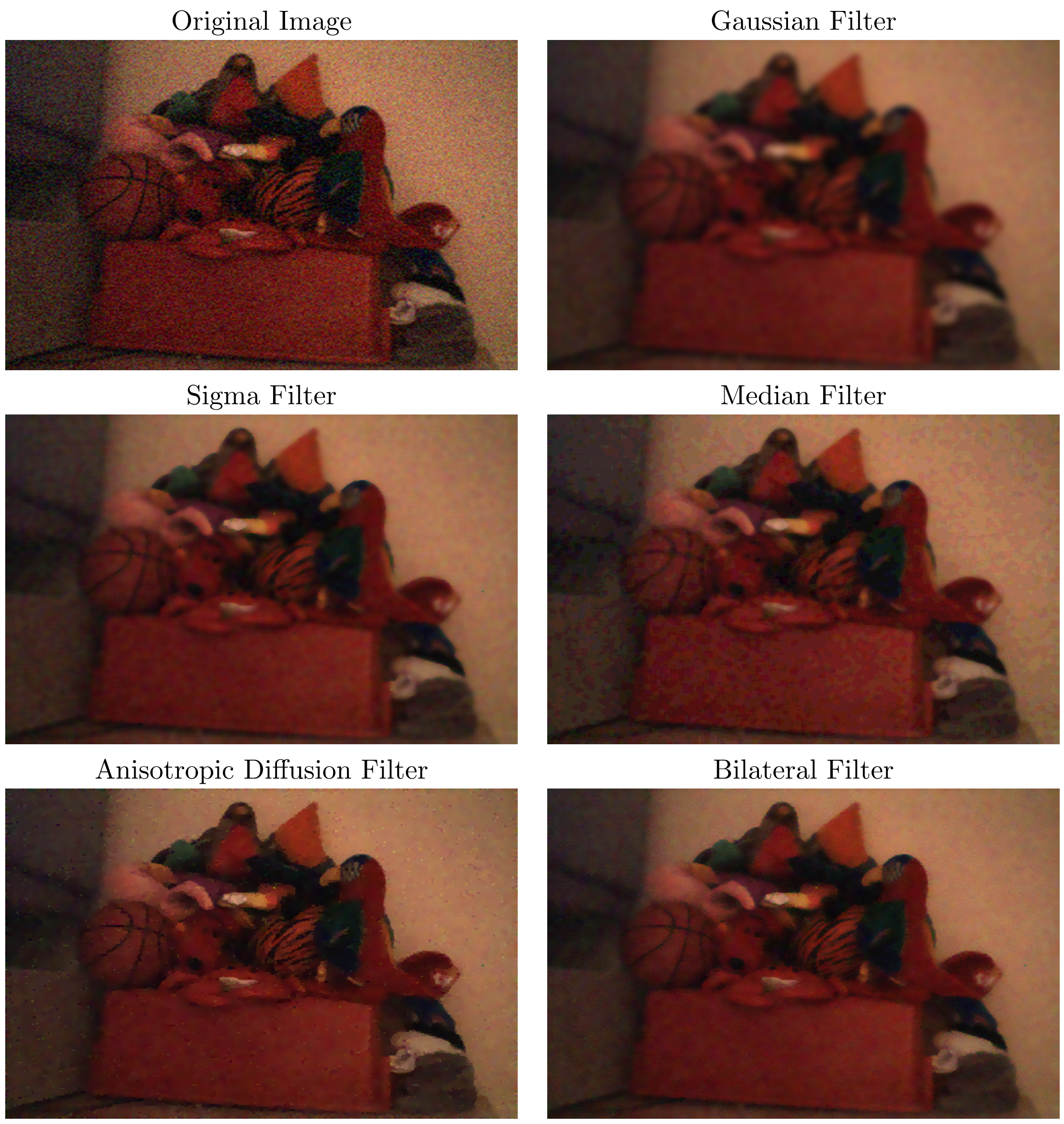

Figure 2.2: Comparison of existing spatial filtering techniques. The original image exhibits a high noise level in both luminance and chrominance. The Gaussian filter ($\sigma = 5.0$) blurs the edges but removes the noise. The Sigma filter ($\sigma = 30$) preserves some of the edges but corruption begins to occur at this Ω kernel size (11x11). The 11x11 median filter preserves many eges, but introduces new distortion in blotchy regions. Anisotropic diffusion (50 iterations with $K = 7$) does an excellent job, save some stray pixels in the background. The bilateral filter ($\sigma_h = 5$ and $\sigma_i = 30$) also does an excellent job filtering in a single pass with very smooth regions within objects.

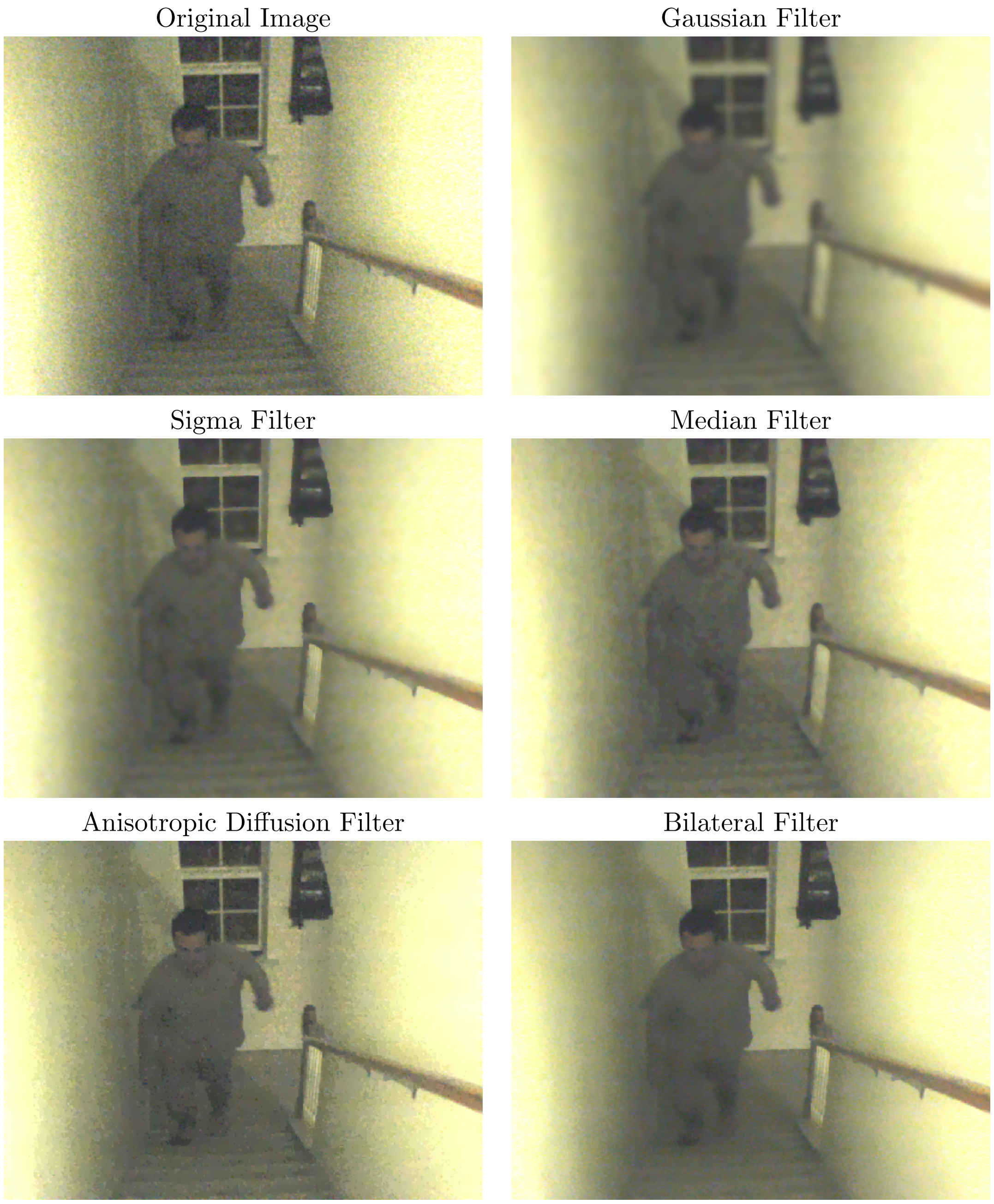

Figure 2.3: Additional comparison of spatial filtering techniques with a surveillance data source of a stairwell. In very low-light, the original amplified image suffers from very high noise levels. The Gaussian filter ($\sigma = 3.0$) blurs the edges, making the scene difficult to interpret. The Sigma filter ($\sigma = 20$ with a Ω kernel size = 7x7) does a good job, but is somewhat blotchy. The 5x5 median filter also results in a blotchy image. Anisotropic Diffusion (50 iterations with $K = 4$) has some trouble reducing the large amount of shot noise, especially on the walls. The bilateral filter ($\sigma_h = 5$ and $\sigma_i = 15$) does the best overall noise reduction, although detail is lost in the face region, making identification difficult.

temporal filter weighted by the reliability of the displacement estimate. This method requires well-exposed, easy-to-track video to correctly filter.

Jostschulte et al. (1998) presented a spatio-temporal shot noise filter that first spatially and then temporally filters video while preserving edges that match a template set. A motion-sensing algorithm is used to vary the amount of temporal filtering. Thus, its filtering can be considered as a 2D spatial filter augmented by temporally-adjacent samples, thus time and space are treated differently. Alternatively, "Spatio-Temporal Anisotropic Diffusion" (Lee and Kang, 1998) uses a three-dimensional kernel to remove video noise, treating temporal and spatial dimensions similarly. This approach is a 3D variant of the 2D anisotropic diffusion already discussed in Section 2.3.1.

As opposed to filtering in the local spatio-temporal neighborhood, the NL-means noise reduction of Buades et al. (2005) searches for matching neighborhoods, which may or may not be spatially aligned or temporally continuous, to attenuate noise. These neighborhoods are used to refine the information about the correct underlying values in the original, noisy neighborhood. In contrast, by finding a statistical mapping of a known video to a set of many small spatio-temporal training patches "Video Epitomes" (Cheung et al., 2005) allows the simultaneous enhancement of many similar video regions. This is used for noise reduction in addition to super-resolution and the estimation of dropped video frames.

Video noise filtering is effective because each pixel can incorporate information from many frames to estimate its true value using images (frames) that are very similar to it. This is analogous to HDR capture, where each luminance value is combined from a set of registered images with varying exposures.

2.4 HDR Processing

The real world has a much higher luminance dynamic range than the standard 8-bit sensors and displays in the imaging pipeline, only capable of a 256:1 maximum dynamic range. High Dynamic Range (HDR) imagery that can represent more realistic dynamic ranges, often on order of 10,000:1 or 100:000:1, has thus been long recognized as essential for accurately modeling light transport (Ward, 1991).

There are two problems that quickly become evident working with HDR. First, capture of HDR imagery is impeded by sensors and digitizing precision, requiring multiple images at different exposures to properly image all visual elements. Even worse, HDR video capture requires the use of specialized imaging hardware. Another problem is display on 8-bit devices, requiring a remapping of the image luminances (tone mapping) to match the display's low dynamic output range. Thus, HDR processing is a class of computational photography problem because the correct answers cannot be obtained through a single image or standard image capture process. Instead, through the combination of multiple images and computation, a more accurate and useful result is obtained.

2.4.1 HDR Capture

Methods for real-world HDR capture are now considered for both still images and video. Special consideration is given to the handling of low luminance levels (common in low-light video) in addition to more typical bright HDR luminance levels. Further low-light imaging characteristics are addressed in detail in Chapter 4.

Debevec and Malik (1997) developed computational methods for assembling individual still HDR images from a series of photographs with increasingly long exposure times, using a common photographic process known as bracketing. The exposure curve of the camera used to capture these images can be solved for, allowing the determination of a luminance estimate at each photosite, which is no longer limited by the precision of the source images. This work relies upon the stillness of the scene over a period of time sufficient to capture each of the bracketed exposures, making it impractical for video.

Researchers have more recently also constructed prototype HDR video capture systems. Kang et al. (2003) built a system based on a camera that can sequence through different exposure settings for each frame. Once the images are registered using optical flow, it is possible to combine exposures to increase the dynamic range. The small number of exposures combined into each output frame suggests that a high signal-to-noise ratio (SNR) is assumed for all exposure settings, and therefore, it is designed for well-lit HDR scenes.

Nayar and Branzoi (2003) present a system whereby a computer controlled LCD panel is placed in front of the CCD. The LCD's per-pixel transparency is varied to modulate the exposure of image regions based on the previous frame's luminance. Along with the hardware, they also discuss a local and global tone mapping approach that addresses temporal coherence issues. Using LCDs implies attenuation of some minimum percentage of the incoming light, which complicates capturing dark pixels. Nayar and Branzoi (2004) also introduce a variant to this method using a DLP micromirror array to modulate the exposure, via time-division multiplexing (like a camera shutter), throughout the image. In theory, such systems could provide continuous exposure control at each pixel given the additional hardware requirements. Note that both of these approaches rely on causal filtering to determine the exposure of each pixel at capture time. If any pixel is over or underexposed because of an incorrect exposure time, the photosite's measurement becomes unusable.

Instead of blocking light reaching the sensor, the sensor itself may be modified for HDR video capture. Acosta-Serafini et al. (2004) describe an HDR camera that selectively resets a pixel based on a prediction of when it will saturate. Here, the reset interval and the digitized pixel intensity level combine to form a floating-point intensity value. They primarily focus on high-speed HDR sensing and do not specifically address low-light capture. Liu and El Gamal (2003) combine high-speed samples to reduce noise and improve dynamic range by using specific imaging device features such as high-speed non-destructive reads. Their filtering is based on a combination of linear filters and motion detection at each individual pixel. Bidermann et al. (2003) describe an HDR high-speed CMOS imaging platform with per-pixel ADCs and storage, which could use the underlying algorithms of Liu and El Gamal (2003) to capture HDR in well-lit scenes.

2.4.2 HDR Tone Mappings

Using the discussed HDR capture techniques to create images with dynamic ranges larger than 256:1, the images must then be tone mapped for display while minimizing loss of perceived detail and contrast.

The tone mapping problem was formalized by Tumblin and Rushmeier (1993) and has led to a variety of spatially uniform (Drago et al., 2003) (logarithmic mapping) compressions and spatially varying (Tumblin and Turk, 1999) (Durand and Dorsey, 2002)(Fattal et al., 2002) tone mappings. By spatially varying the tone mapping, a technique may locally adapt to maximize displayable contrast. For example, Retinex approaches, such as the multiscale Retinex (Jobson et al., 1997), suggest that a Gaussian-like kernel can be convolved at each point in the image and subtracted from the original image in log space, resulting in a more "viewable" version of an HDR still image. The Retinex approach is fast due to it being non-iterative, but it can generate unwanted edge blurring artifacts because of its underlying Gaussian nature.

The tone mapper of Durand and Dorsey (2002) presents a similar system, but uses bilateral filtering to maintain sharp edges and to reduce fringing artifacts. It operates by separating the large-scale features from the detail features and processes them separately. As shown in Figure 2.4, this processing is done in the log-luminance domain, which models the detail features as modulations to the large-scale features (contrast modifications), instead of being additive. Once the separation has occurred, the large-scale features are attenuated by some factor $k < 1$, then recombined into the output. The resulting image then has a reduced dynamic range. The technique works because, after compression, all of the uniform regions fall within an 8-bit range. The lower magnitude textures are also visible because they are modulating luminances in that displayable range. The results still look plausible because the contrast ratios are unchanged in smooth regions, only the absolute luminances are modified, and by Retinex theory (Jobson et al., 1997), the human visual system is not sensitive to smooth absolute luminance changes over large spatial areas.

Other HDR work includes that of Pattanaik et al. (2000) who present an approach that mimics the time dependent local adaptation of the human visual system. They also discuss temporal coherence issues to avoid introducing frame-by-frame tone mapping "flicker" when processing videos. In "Gradient Domain HDR Compression" (Fattal et al., 2002), the gradient field of an image is locally attenuated and then reintegrated. It should be noted that they also describe a method for improving images that already use the display's full 8-bit dynamic range. This is the closest problem addressed in the literature to the reverse problem of increasing the dynamic range of LDR images, discussed in Section 4.2.2.

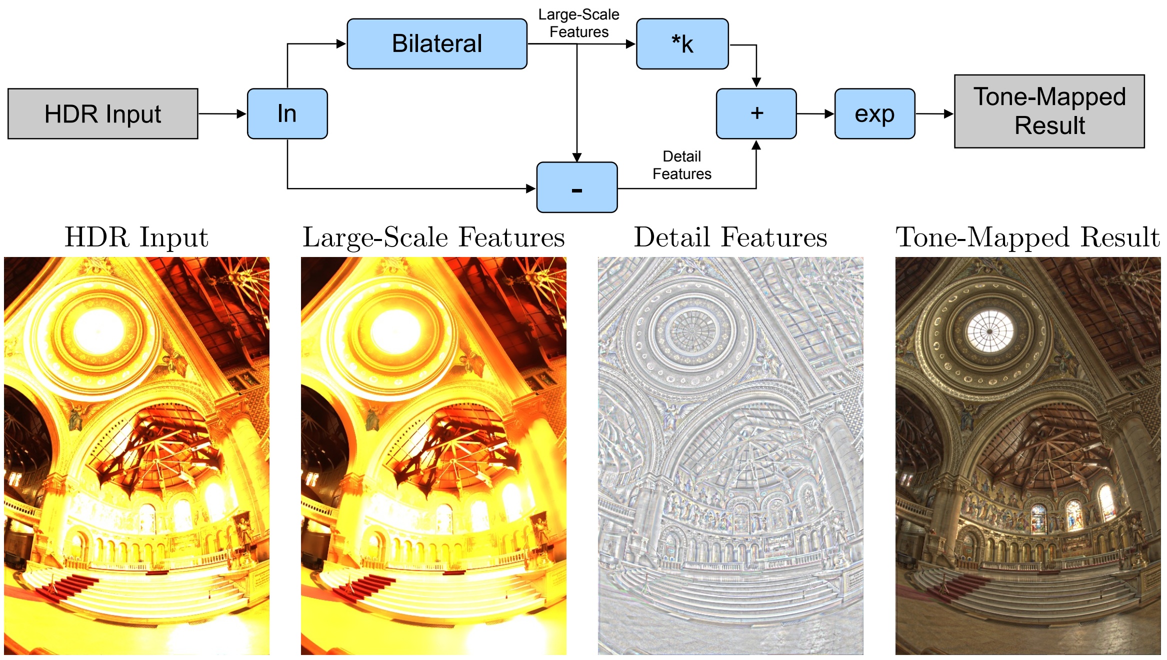

Figure 2.4: Illustration of the HDR tone mapping pipeline described by Durand and Dorsey (2002). The result of the bilateral filter, the large-scale features, are attenuated by a factor k while everything else, the detail features, remain unchanged. The tone mapping is performed in the log domain so that the detail features modulate the large-scale features (images shown have been converted to the linear domain for display). Values of $k < 1$ cause dynamic range compression (as originally published) while values of $k > 1$ expand the dynamic range.

By using multiple images, HDR capture and tone mapping create resulting images that were not otherwise possible to capture. Another approach that combines multiple images is multispectral fusion. Because these images are now each from a distinctive spectrum, their fusion creates enhanced results not visible to the human eye.

2.5 Multispectral Fusion Techniques

Multispectral fusion involves the depiction of multiple, potentially non-visible, spectral bands as a visible image. The goal can be to communicate information from all sources, or to augment a poor quality signal in one band with a higher quality one (such as augmenting noisy night-vision video with heat-detection imagery). In this section, two classic multispectral applications are summarized: remote sensing (aerial and satellite imagery) and night-vision.

To fuse amplified night-vision data with multiple IR bands, Fay et al. (2000) introduce a neural network to create false-color (pseudo-color) images from a learned opponent-color importance model. Many other false-color fusion models have been suggested in the remote sensing community. A summary of popular techniques is provided by Pohl and Genderen (1998). Another common fusion approach is to combine pixel intensities across spatial scales using multiresolution Laplacian or wavelet pyramid decompositions, as in (Toet, 1990) and (Li et al., 1994). Also, physically-based models that incorporate more than per-pixel image processing have also been suggested (Nandhakumar and Aggarwal, 1997).

Therrien et al. (1997) introduce a method to decompose visible and IR sources into their respective high and low frequencies, and processes them in a decomposition/recomposition framework inspired by Peli and Lim (1982). A non-linear mapping is applied to each set of spectral bands to fuse them into the result. Therrien et al. (1997) address normalizing luminance responses between spectra to overcome differences in surface reflectivity between bands. These normalized luminances are mapped to a new space defined by a Sammon mapping (Sammon, 1969). Issues regarding the temporal coherence of such a mapping in video are not mentioned.

IR colorization algorithms, such as those by Welsh (2002) and Toet (2005), attempt to learn a mapping from IR to natural chrominance to construct a plausible colorized output. For that reason, colorization can be considered a class of fusion that estimates chrominance based on image priors from IR footage. However, acquiring a prior and performing accurate matching from prior to IR is often difficult and does not guarantee temporal coherence for video processing.

2.6 Computational Photography

Having considered combining elements from multiple spectra together (often imaged simultaneously), we next consider combining exposures taken with the same imager, but at different times. This allows for existing imagers, such as digital cameras and camcorders, to achieve enhanced results with the assistance of computational methods.

Therefore, I now present techniques that are typically labeled as computational photography, combining visual elements from images taken with a standard camera at different times to create an optimal composite image. Specifically, multiple images are used as input, and a still image is the result. These combination techniques draw on the wealth of algorithms and techniques that have already been discussed in this chapter.

"Image Stacks" (Cohen et al., 2003) considered the idea of using multiple temporally adjacent frames to enhance knowledge about a pixel's true or desired value. Multiple images are registered and then each pixel of the output image is computed as a function of its temporal neighbors. "Image Stacks" includes methods for automatically selecting and combining image regions using compositing and min, max, median, and temporal low-pass filtering. An α-blended "over" compositing (Porter and Duff, 1984) (Eq. 2.8) is also used to illustrate the passage of time by compositing so that recent motions occlude past motions:

$$Over = \alpha \cdot Foreground + (1 - \alpha) \cdot Background. \tag{2.8}$$

The concept of combining sequential frames of motion together can be traced back to the stroboscopic multiple exposure photographs of Harold Edgerton (Edgerton and Killian, 1979) and the motion studies of Muybridge (Muybridge, 1955) and Marey (Braun, 1995), all performed using traditional film capture. The combination of images to create motion studies was also addressed by Freeman and Zhang (2003) with the use of stereo depth data to influence each pixel's compositing order.

As an extension to "Video Stacks", "Interactive Digital Photomontage" (Agarwala et al., 2004) allows the user to choose individual parts of a few images through a simple drawing interface to quickly specify the best composite image. This fusion is accomplished using both graph-cut (Kwatra et al., 2003) and gradient domain algorithms (see below).

"Space-Time Scene Manifolds" (Wexler and Simakov, 2005) combined the concept of using spatio-temporal volumes (Section 2.2) with a similar goal of combining multiple sequential video frames captured with a non-static camera. The resulting optimized non-planar cuts are chosen to maximize the visual information in the resulting cut image while simultaneously mosaicing the constituent video frames.

The seamless integration of scene elements from multiple images has been explored in the gradient domain using Poisson solvers to reintegrate processed gradient fields. Poisson solvers have already been mentioned in the context of HDR tone mapping (Section 2.4.2). Techniques such as Poisson Image Editing (Perez et al., 2003), and day/night fusion (Raskar et al., 2004) generate gradient fields that contain visual elements from multiple images.

The ratio-image work of Liu et al. (2001) transfers illumination between multiple images with the assistance of known illumination priors. The capture of these priors requires images taken in controlled lighting environments, which are suited for still images, but not for video.

New variants of the bilateral filter, discussed in Section 2.3.1, have been developed for use in computational photography. The "joint bilateral filter" uses a second image as the source of comparison for edge identification, thus transferring its edges to the original image. This filter was used by Petschnigg et al. (2004) and by Eisemann and Durand (2004) (who refer to it as "the cross bilateral filter"). Both of these papers consider the problem of combining the qualities of an image captured with the use of a flash with the "look" of a noisy image captured under ambient illumination. A joint bilateral filter is created by changing Equation 2.7 to instead perform its photometric comparisons in a second image, indicated as I':

$$D(p, s) \equiv I'_p - I'_s. \tag{2.9}$$

The extent of noise removal depends on how well exposed a given region is in the flash image. These papers also address flash shadows, which introduce visible edge differences between the sources. These flash shadows resemble some of the differences seen between RGB and IR spectra mentioned in Section 2.5.

These approaches are all designed to output a single image as a function of its input frames. In the final section, computational video techniques that result in video outputs are considered.

2.7 Computational Video

Computational video is a more recent field than computational photography, so its seminal literature is not yet established. In this dissertation, computational video involves processing video inputs to create video results that modify and combine elements across many frames.

This is accomplished using techniques enabled by modern computational power and the simultaneous processing of many frames enabled by the low cost of storage. One type of computational video is the temporal resampling of a video to change its duration. Computational video techniques are thus allowed to warp both space and time, compressing or extending the video's duration and intermixing frames.

2.7.1 Video Summarization

Temporal resampling is commonly used for multimedia summarization. Early work in Video Skimming (Smith and Kanade, 1997) looked for short, representative video segments that, when pieced together, could tell the story of the video in a reduced period of time. Segments were chosen based on characteristics including scene change detection, camera motion, object recognition, and audio. The documentary films they targeted had distinct scene changes providing the algorithms additional hints. An alternate summarization approach proposed by Hua et al. (2003) searched for video segments that contain scene and camera motion between shot boundaries and combined them to match the rhythm of an audio source to create music videos.

Using similar documentary films to (Smith and Kanade, 1997) and standard, uniformly-sampled summarization (fast-forward), Wildemuth et al. (2003) explored how fast videos can be played back while remaining coherent to the viewer. The result was that showing 1 out of every 64 frames typically allowed the viewer to comprehend most of the content. Note that in addition to documentary-style video sources, summarization can also be applied to static-camera, time-lapse sources. In particular, time-lapse has been shown to have many applications for viewing slowly changing processes. Time-lapse techniques are regularly used in fields as varied as biological microscopy (Riddle, 1979) and cinematic effects (Kinsman, 2006).

"Video Summarization By Curve Simplification" (DeMenthon et al., 1998) presents an algorithm to choose a non-uniform temporal sampling based upon simplification of tracked motion-path curves. These motion curves can be considered a subset of all motion activity in the scene. The sampling is derived from the use of the greedy Douglas-Peucker curve-fitting algorithm (Douglas and Peucker, 1973). Note that slower, but optimal, dynamic programming-based curve-fitting solutions (Perez and Vidal, 1994) are possible.

In “Video Summarization Using MPEG-7 Motion Activity and Audio Descriptors” (Divakaran et al., 2003), short video sub-clips identified as containing significant motion are played at real-time speeds and assembled into a shorter video. The combined duration of these essential sub-clips forms the lower bound of the output video’s duration. Longer videos are constructed by padding the result with less interesting frames.

Rav-Acha et al. (2006) summarize long videos by allowing events to occur without the strict chronological ordering of the source footage, thus events may overlap. The events are identified and then combined in a manner determined via a spatio-temporal Markov random field optimization and simulated annealing.

2.7.2 Temporal Resampling and Compositing

Computational video considers a wider range of temporal resamplings and frame combinations. For instance, a class of operations exist that extend the length of videos by repeating segments. “Video Textures” (Schödl et al., 2000) looks for transitions within a video that are least noticeable in an attempt to indefinitely extend its playing time. A dynamic programming solver is used along with a pairwise error metric to evaluate potential jumps.

Other approaches have also attempted to warp time, both globally (full frame) and locally (region-by-region). “Flow-Based Video Synthesis and Editing” (Bhat et al., 2004) rearranges repeating patterns of natural phenomena, such as waterfalls, that have reoccurring flow characteristics, to extend their play time. “Evolving Time Fronts” (Rav-Acha et al., 2005a) plays videos with differing speeds in multiple image regions for the effect of altering their outcomes or for artistic effects. “Dynamosaics” (Rav-Acha et al., 2005b) carries this idea further by improving the blending between regions with graph-cuts (Boykov et al., 1999). “Panoramic Video Textures” (Agarwala et al., 2005) also finds frame-to-frame jumps within non-static camera videos to create panoramas.

Videos can also be processed to alter their component visual elements. “Motion Magnification” (Liu et al., 2005) renders videos with amplified object motion vectors while not changing the underlying temporal sampling, thus increasing apparent object velocity. Alternatively, “Space-Time Super-Resolution” (Shectman et al., 2005) creates a composite of multiple videos using their relative frame rates and spatial positions to maximize spatial and temporal information while reducing overall aliasing.

As a parallel to these computational video methods, user-assisted, temporally-aware video compositing algorithms have also been developed. “Video Matting of Complex Scenes” (Chuang et al., 2002) approaches the problem using Bayesian methods. “Interactive Video Cutout” (Wang et al., 2005) combines compositing with spline-based, non-planar spatio-temporal volume cuts in its interface for extracting foreground elements. In this interface, the user can specify hints to the alpha compositing engine across multiple frames by painting directly onto the cut plane. The VideoShop project (Wang et al., 2007) composites multi-frame visual elements from video clips together in the gradient domain through the use of a 3D multigrid Poisson solver.

2.8 Summary

A wide range of literature related to computational video and its foundations was reviewed to act as a background for the techniques and applications presented throughout the remainder of this dissertation. The common element in all of these works is the visualization, combination, and enhancement of still images and videos to create improved outputs by utilizing all the information in the source material. As a first step to building efficient computational video tools, a framework is now presented for working with a new class of spatio-temporal volumes that enable video editing and processing.

CHAPTER 3

SPATIO-TEMPORAL VIDEO PROCESSING

In this chapter, I explore approaches to computational video where videos are abstracted as spatio-temporal volumes. Computational video often relies on using nearby spatial and temporal samples in order to enhance the underlying video. Likewise, spatio-temporal volumes provide a representation for accessing and organizing these samples. Furthermore, I extend spatio-temporal volumes to include *sheared* volumes that spatially align frames to bring together important scene elements into a common spatio-temporal neighborhood. This shearing can be done in relation to the background to stabilize a moving camera, or in relation to a moving object to stabilize its motion.

Frequently, shears are not meant to be permanent. Therefore it is desirable after processing that the original camera and object motions be restored. Thus, modifications made to these sheared volumes must be applied to the source data and are formulated as mapping functions, thus making the shear *virtual*. Furthermore, there are significant performance issues involved in manipulating and visualizing uncompressed spatio-temporal video volumes in sheared and un-sheared states. To handle these issues, an efficient graph-based processing framework called Proscenium is introduced to encapsulate and abstract away the details of such processing. Finally, as a proof-of-concept, a simple video editor is developed that visualizes, edits, and filters in a manner that leverages the capabilities of spatio-temporal volumes (Figure 3.1).

The chapter begins with discussions of previous spatio-temporal volumes and then an overview of the new shearing extension. Then, spatio-temporal video processing with Proscenium is described from the bottom up starting at the lowest level, the data representation,

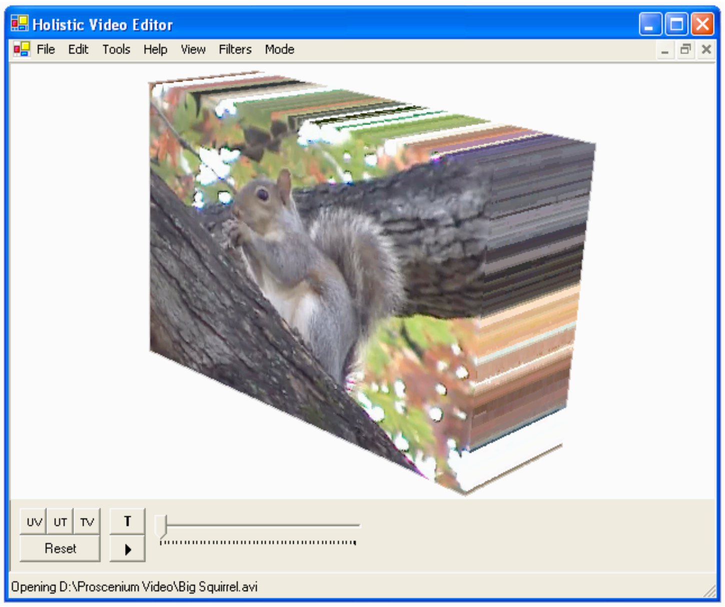
Holistic Video Editor
File Edit Tools Help View Filters Mode
UV UT TV
T
Reset
Opening D:\Proscenium Video\Big Squirrel.avi

Figure 3.1: Screenshot from the prototype application depicting an 8 second video segment as a spatio-temporal volume. The prototype editing system utilizes the Proscenium framework to enable the shearing, modifying, and enhancing of such volumes.

on up through implementing shearing and the filter-graph framework, then culminating with the prototype video editor.

3.1 Spatio-Temporal Volumes

A spatio-temporal volume (Bolles et al., 1987) can be conceptualized as the stacking of the individual frames of a video in chronological order into a right rectangular prism with dimensions of width, height, and time. Thus, any sample in the volume can be accessed with a unique positional index (x, y, t). In this manner, the spatio-temporal volume treats the time dimension similarly to the two spatial dimensions and allows for 3D volume processing.

Note, however, sampling is different in the spatial dimensions versus the temporal dimension. Spatial sampling is usually uniform, with the same aspect ratio vertically as horizontally. However, the same cannot be said about space and time measurements. Spatially, an edge between two objects is assumed to occur between two pixels. Temporally, however, an edge due to motion may move many pixels from frame-to-frame. Thus, unless the frame rate is sufficiently high that no motion results in a translation of more than one pixel, the temporal dimension is undersampled compared to the spatial dimensions. This relationship must be considered when using information from adjacent temporal samples as opposed to adjacent spatial samples.

Given a fixed sampling, it also makes sense to consider interpolation between the known samples (both spatial interpolation and temporal interpolation). Visualizing the volume as a solid as opposed to a sequence of frames (Figure 3.2) makes this a natural extension. Interpolation may be nearest-neighbor, trilinear, or via some other reconstruction filter.

From a visualization standpoint, spatio-temporal volumes are useful because all frames are shown simultaneously, removing the need for a standard timeline-based interface. However, only pixels on the edge of the volume can be seen, as it is a solid. Thus, the first tool developed for spatio-temporal volumes was the axis-aligned *planar slice* (cut plane) that allows the user to see inside the volume (Bolles et al., 1987), as shown in Figure 3.3. Arbitrary planar cuts were later introduced by (Fels et al., 2000). These cut planes can reveal motion patterns in planar slices that allow traits, such as object velocity, to be measured. They also allow for novel artistic views of the video to be made by sweeping the plane through the volume.

Although cut planes are useful visualizations, there is an inherent assumption that the volume being cut is from a fixed camera. In particular, a moving camera implies that adjacent

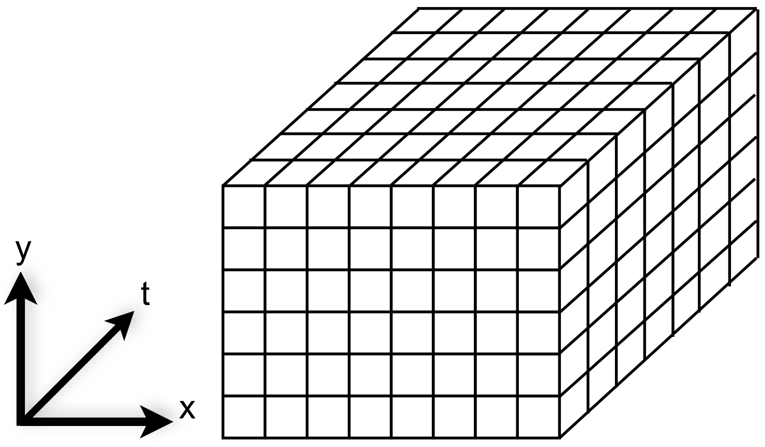

Figure 3.2: An illustration of a spatio-temporal volume as a solid. Although samples in the volume are typically indexed using integers, values at non-integer indices are interpolated from the original underlying samples.

temporal samples in the volume are not imaging the same incoming rays. To overcome this difficulty, spatio-temporal volumes are extended in this work such that each constituent frame may be spatially warped via a homography, a process referred to as *shearing*. Although the volume is no longer a right rectangular prism, the new frame-to-frame alignment greatly expands the capabilities of spatio-temporal video enhancement. This shearing operation is fundamental to the use of spatio-temporal volumes and is shown in Figure 3.4. Once sheared, cut planes can again be used, except now each plane exposes pixels that were not originally planar in the input volume. Also, note that the volume can be sheared to temporally-align any scene element, such as moving foreground elements.

Besides being a useful visualization of the data, spatio-temporal volumes also serve as a platform for enforcing *temporal coherence* in video processing algorithms. Temporal coherence ensures that moving objects look natural when played back at full speed because of consistency in the way they appear from frame-to-frame. Maintaining this coherence across a multiframe enhancement involves making changes to multiple frames in the same manner, avoiding variation. Performing similar edits to objects (especially moving objects) is difficult to do by hand or with algorithms that operate on a frame-by-frame basis.

Now that the underlying concepts of spatio-temporal volumes have been established, we

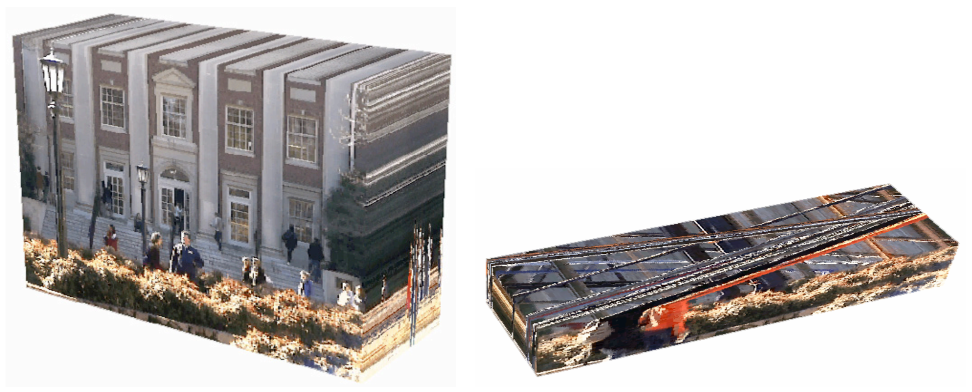

Figure 3.3: Visualization of a spatio-temporal volume and a spatio-temporal cut plane. On the left, a 10 second video is presented as a spatio-temporal volume. The front of the volume shows the first frame, the right side shows the right-most vertical line through time, and the top shows the top-most scanline through time. On the right, the volume has been rotated and been cut using two planar cuts. The first, parallel to the front face, has shortened the video. The second has revealed a different scanline which shows the motion of people walking during the duration of the video.

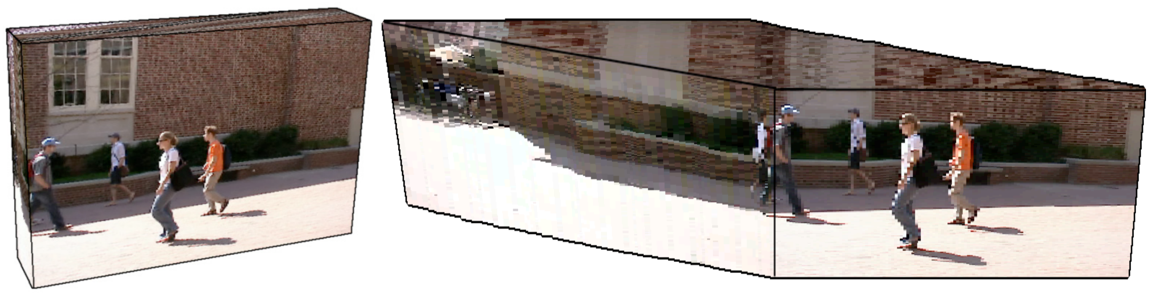

Figure 3.4: Illustration of the shearing (spatial warping) of a spatio-temporal volume. On the left is a spatio-temporal volume where the camera rotates from right to left. On the right is the same video presented in a sheared volume where the background is static through time. Thus, any given (x, y) is always imaging the same scene element. However, (x, y) samples may no longer exist for all values of t.

transition to considering the logistics of building a shared set of tools to encapsulate these concepts.

3.2 A Spatio-Temporal Video Editing Framework

I have developed a spatio-temporal video editing framework called Proscenium for processing with spatio-temporal volumes that abstracts away the representation details. Specifically, Proscenium handles the underlying storage along with shearing and filter-to-filter interactions. Given these capabilities, it is simple to build interactive full-featured computational video applications.

To begin, the low-level storage of spatio-temporal volumes is addressed. This is followed by a discussion of shearing to support the alignment of spatial elements through time. The requirements of shearing then lead to the development of a filter-graph system tuned to spatio-temporal processing. Finally, examples of filters that fit within this framework are presented.

3.2.1 Spatio-Temporal Volume Representation

At a base level, each source video segment is represented as a three dimensional array addressed in spatial dimensions by x and y and in time by frame number t. If the dimensions of the video volume are constant, a 3D array may be used, however the ability to insert additional

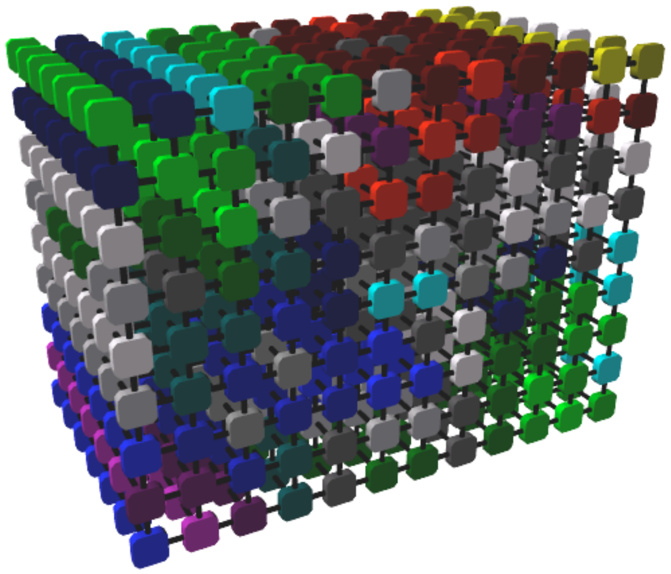

Figure 3.5: Illustration of a spatio-temporal volume that specifically indicates the interconnections between each sample and its neighbors. When considering spatio-temporal video processing (particularly filtering), the information in nearby spatial and temporal samples is essential to accurate video reconstruction and temporal coherence (consistent changes from frame-to-frame).

frames in the t dimension implies storing the video as a list of individual 2D frames.

For working with multichannel color videos, the typical representation for storing pixels is as unsigned bytes with RGBα (0-255) values (32 bits/pixel). The α component serves as either a traditional blending factor (Porter and Duff, 1984) or as a pixel-specific auxiliary variable for Boolean or more complex operations. The color $(0, 0, 0, 0)$ is reserved as being *empty*, meaning that nothing is present at a pixel location. Although not used in this implementation, floating point values and alternate color spaces are easily supported.

The spatio-temporal volumes of Proscenium support both discrete and continuous sample access. Discrete addressing is analogous to standard integer-indexed array access of the underlying samples, whereas continuous access allows for fractional addressing and implies interpolation. Although discrete addressing is most efficient, continuous access becomes important when accessing pixels within a sheared volume. The quality of the continuous sampling's interpolation (nearest-neighbor, trilinear, etc.) can be either application dependent or user-specified.

By definition, spatio-temporal volumes are right rectangular prisms, and each of the constituent frames is assumed to have the same spatial size and orientation. In the next section, that assumption is removed.

3.2.2 Spatio-Temporal Volume Shearing

The fundamental innovation introduced by my work to existing spatio-temporal volumes is the ability to virtually shear these volumes, thereby spatially warping each of the volume's frames. The benefit of this operation is that it enables the alignment of objects within spatio-temporal neighborhoods, allowing information in that neighborhood to be used to enhance its constituent samples. I now present the specifics of how shearing is implemented with minimal data movement along with the user interface details that will be used later in Section 3.3.

Specifically, shearing (S) is a subset of all possible volumetric warps in which all values in the temporal t dimension remain constant, as shown in Figure 3.6. Specifically, each frame in the volume has a unique spatial mapping S_t between its sheared position and its position in the original volume from the class of general 2D projective transforms. To allow lookups in both directions, the inverse transform S_t^{-1} is also maintained. These projective transform mappings model a wide range of spatial changes including frame-to frame translations, rotations, scales, skews, and any combination of these. This class of transforms is easily specified and sufficient for the alignment of rigid nearly-planar objects. As discussed as future research (Setion 6.1) more general, non-planar mappings are also a possibility for shearing.

The formulation of the shearing transform for frame t is shown below. The value of t itself is not included inside the matrix to avoid projective normalization. By convention, the first frame of a movie remains stationary (an identity transform), while all other frames generate their correspondences in relation to that frame:

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} S_{t,0} & S_{t,1} & S_{t,2} \\ S_{t,3} & S_{t,4} & S_{t,5} \\ S_{t,6} & S_{t,7} & 1 \end{bmatrix} \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} \quad \text{and } t = t'. \tag{3.1}$$

The user specifies the shear by choosing scene features that should remain spatially constant. This is done using small sets of user-defined inter-frame correspondences. Although

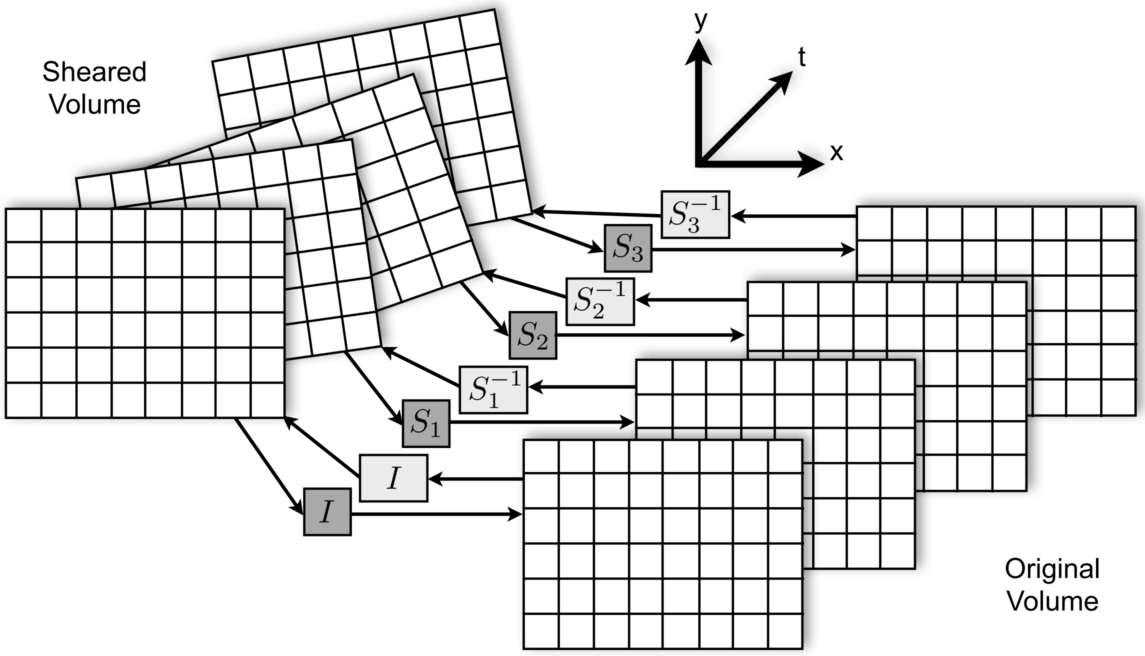

Figure 3.6: Illustration of the shearing process. On the right, the original un-sheared volume stored in memory is shown. On the left, the sheared volume is shown, where each frame has its own 2D projective transform. The shear is applied virtually, so all pixel lookups are processed through a transform into the original volume. To allow lookups in both directions, the inverse transform is stored as well. The first frame is, by convention, stationary, and thus uses an identity transform.

full-projective transforms are supported, when only a single correspondence is specified, frame-to-frame translations are assumed. When two correspondences are provided a similitude transformation is found (a translation and rotation with a global scale). With three corresponding points affine transformations of the images are computed. Four correspondences specify a unique projective transformation. When more than 4 correspondences are provided the closest projective transformation (in the least-squares sense) is computed. All of the coefficients for the transforms can be found using linear methods (Wolberg, 1995) and are solved here in real-time using Gauss-Jordan elimination (Press et al., 1992).

Having a user in-the-loop to select correspondences to specify the shear transforms makes for an straightforward user interface. However, the process of selecting correspondences in every frame can be tedious. So, instead, the user selects corresponding points at appropriate intervals in the video, and those points are linearly interpolated or tracked using an image-based feature tracker (Bouguet, 2000) in the intervening frames.

Thus, the goal is to provide all of the benefits of computer-vision based tracking without its shortcomings. When it works, feature-based methods provide excellent tracking of visual elements from frame-to-frame, saving the user time and often providing more consistent tracking than rough user-specification. However, occlusion, noise, and homogeneous image regions all can cause these methods to fail, often making a mistake in one frame that causes all subsequent frames to track the wrong path. In these cases, the user's job becomes to intervene when necessary, because the user has a better understanding of the overall content of the video and the purpose of the shear. Often with correction of a small number of frames containing errors, the tracker can perform the remainder of the tracking correctly. This approach is similar to the interface in the compositing work of Chuang et al. (2002) that allowed the user to continuously refine the assumptions made by the compositing solver.

If the intention of shearing the volume is to permanently change its shape, a resampling of the volume using the planar transforms may be used and the original video discarded. However, it is more often the case that shearing is performed to aid in editing and enhancement, with the intention that the shear will be removed, restoring the original video. Thus, having a mechanism to "virtually" shear the volume to allow visualization and processing is needed with the capability to apply edits to the sheared volume, yet have them be propagated to the un-sheared volume. To handle that task, and other, more general purpose video processing tasks, Proscenium's dynamic filter graphs are used. Multiple examples of such workflows are presented in the Results (Section 3.4).

3.2.3 Proscenium Filter Graphs

Now, the underlying bi-directional filter graphs used by Proscenium are discussed. These graphs extend existing video processing filter graphs by tuning them to account for the pixel access patterns common in spatio-temporal processing. These dynamically constructed graphs are then populated with individual PFilters (Proscenium Filters) that conform to a common interaction interface. PFilters each encapsulate a single video processing task, such as color correction, cropping, or shearing. By combining these PFilters together, complex interactions,

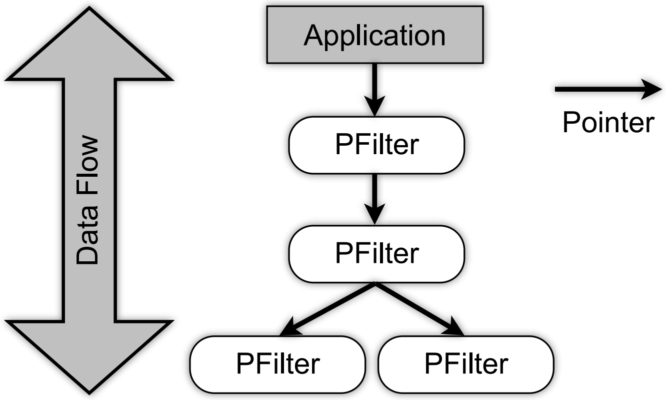

Figure 3.7: Diagram illustrating the interconnections of PFilters in a Proscenium filter graph. Each filter is only aware of those filters that comprise its inputs.

such as manipulating and enhancing sheared volumes, are efficiently enabled.

The primary intuition when constructing a system for spatio-temporal video processing is that the access patterns are inherently different from those of frame-at-a-time video processing. Traditional filter graph implementations, such as QuickTime (Apple Inc., 2007) and DirectShow (Microsoft Corporation, 2007), treat pixel data in large buffers of entire frames. Working in spatio-temporal volumes often implies requiring access to individual pixels in small regions of the volume that span many spatial pixels and many temporal frames. Thus, the Proscenium framework's filter graphs never process anything larger than a single pixel at a time to maximize flexibility.

To accomplish this, Proscenium's graph is fully bi-directional, instead of being a traditional directed acyclic graph that forces data to flow from the input of the system to the output. Bi-directionality allows the application at the output of the graph to request only the pixels it needs for interactive display or processing. By only solving volume samples as they are needed, these graphs are inherently lazily evaluated. The filter graph is also designed to insert and remove PFilters (Proscenium Filters) at run-time without rebuilding the entire graph.

All actions begin with a request from the application that it wants a pixel at some (x, y, t). This request is sent to the output of the filter graph (not the input) which decides what actions to take (Figure 3.7). The most basic action is to say nothing was found and return an *empty* pixel. It can also return a constant, such as a background color. Finally, it can request pixel

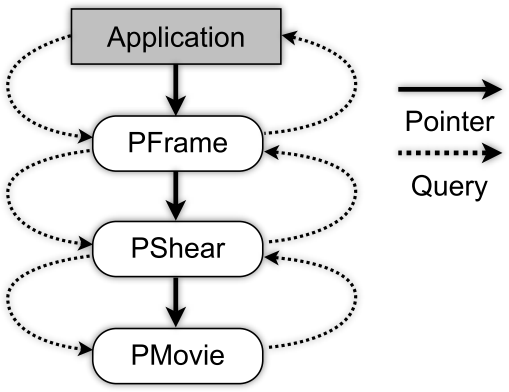

Figure 3.8: Illustration of the flow of pixel queries through an example filter graph. Data requests traverse down the graph until the source data is reached, then are passed back up the chain, possibly being modified along the way. This particular filter graph stabilizes a video then crops its edges, creating a synthetic camera motion.

data from any of the filters at its inputs, modify that color, and return it to the requestor. The proper ordering of requests (Figure 3.8) is crucial to achieve the correct functionality in the filters described later. There is an added performance bonus in this ordering of operations that if the final PFilter (the first queried) is able to return a value without querying the other PFilters, it foregoes the trouble of having to traverse through the entire graph.

Each PFilter implements a common interface for interacting with the graph. It specifies its spatial dimensions, `pixelWidth` and `pixelHeight`, along with its temporal dimension, `numFrames`. Discrete accesses to pixel values are made with `getActualPixel(int x,y,t)`, continuous accesses to pixel values are made with `getPixel(float x,y,t)`, and edits are made with `setActualPixel(int, x,y,t, Color newColor)`. For complete details, the PFilter class structure specification is given in Appendix A.1. Examples of such filters are now presented to illustrate the types of processing these graphs support in addition to "virtual" shearing.

3.2.4 Spatio-Temporal Operators

To make more concrete the types of spatio-temporal operations that can be part of a Proscenium filter graph, a series of PFilters are now described. These particular PFilters represent only a small subset of the filters that have been designed and are included to be instructional.

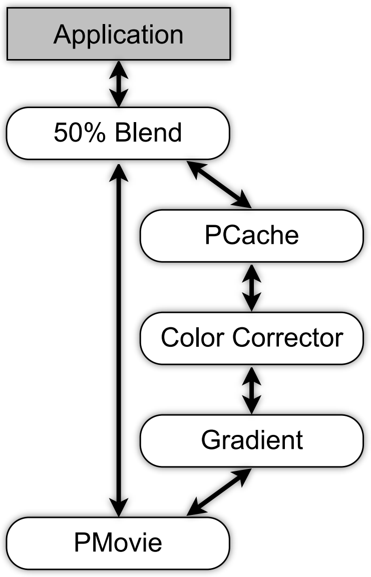

Figure 3.9: This filter graph depicts the effect of tinting the edges in a video. The raw color is mixed with a color corrected grayscale gradient edge, whose output is cached to avoid later re-calculation. The per-pixel request model handles bifurcated graphs without need for any further abstraction.

However, interesting editing permutations are possible with just a few, as in Figure 3.9.

Most PFilters can be categorized into a few useful groups. First are those PFilters that alter the color value of the pixel that was passed into it through its input, such as for color or contrast adjustment. Next are those filters that pull their values from pixels in the spatio-temporal neighborhood surrounding the requested pixel, such as the background restoration filter. Finally are those filters that alter the shape of the individual frames as they pass through the PFilter, such as for performing virtual shears.

Here is a summary of the example PFilters. Full specifications, technical descriptions, and pseudocode are included in Appendix A.2.

- **Simple Color Correction (PCorrect)**: Given an (x, y, t) coordinate, it returns the pixel at (x, y, t) in its source filter with an adjusted color value.

- **Video Framing (PFrame)**: Crops a video's width, height, and/or duration by substituting alternate `pixelWidth`, `pixelHeight`, and `numFrames` values while dynamically offsetting the underlying volume.

- **Shearing (PShear)**: Implements the virtual shearing of the spatio-temporal volume (Figure 3.6). Thus, given an (x', y', t') in the sheared volume, it uses an inverse projective transform to find (x, y, t) in the original, un-sheared volume. This hides the fact that the underlying volume has a different shape. All pixel-level sampling issues (for both reading and writing) are handled within this filter. Edits made to the filter graph pass through this filter, modifying the underlying data so that when the stabilization is removed, the edits remain.

- **Background Restoration (PBackground)**: Given an (x, y) pair, returns the background plate seen at that (x, y) location through the entire video. This background plate is calculated through either the median statistic or *edge filling*, the process of returning the temporally nearest pixel value in a warped volume to fill in the edges of panoramas.

- **Video Caching (PCache)**: Acts as a cache within a filter graph to remember pixel values that have already been solved by the portion of the graph connected to its input.

- **Video Input & Output (PMovie & PAVIOut)**: Provides a method to bring video from disk into the filter graph and write the results back to disk, respectively.

By themselves, these filters are not very interesting. However, now they are combined into a prototype system for video editing that can achieve far more sophisticated results.

3.3 Spatio-Temporal Video Editing

The remainder of this chapter explores the flexibility of spatio-temporal volumes to directly manipulate video in a manner influenced by existing video-editing applications. To do so, the construction of a prototype spatio-temporal video editor is described. With this application, the performance of spatio-temporal processing can also be measured.

The goal of building this prototype is two-fold. First is to demonstrate that the Proscenium framework provides the baseline capabilities of existing spatio-temporal visualizations as well as the functionality of existing video editors. Second is to enable new interactive computational video tools for manipulating video with similar flexibility and ease of use as current 2D image editing applications by harnessing relationships between pixels in adjacent frames. Establishing these inter-frame temporal relationships relies on the ability to shear the volume, visualize its effect, and perform edits into the sheared volume. In the interest of flexibility, no constraints are placed on the type of source footage that can be edited.

The implementation of such a system introduces issues relating to performance, interactivity, and data flow. Such selected issues are now discussed.

Because spatio-temporal video processing is intended for users from seasoned professionals to amateurs, it is important that it can scale well from workstation class hardware down to mid-range consumer desktops. System utilization for the underlying Proscenium framework can be broken down into three major areas: CPU, Video, and RAM. The CPU performs the filter graph functions and is usually the limiting factor for the rate pixels' colors are calculated via calls to `getActualPixel()`. Thus, the faster the CPU, the more filters can be applied while still maintaining a sense of interactivity. The quality of the displayed interactive volume can be subsampled to increase speed, leaving the complete full resolution rendering of the video for the time of output. The video card determines the refresh rate when slicing and manipulating the volume. Specifically, the volume is visualized as a stack of flat, textured polygons drawn from back to front to ensure correct z ordering for α-transparency (the orientation of planes is also dynamically swapped to appear solid from all sides). RAM is an issue because movies must be stored uncompressed for fast random access. This is a disadvantage of supporting visualization of arbitrary cut planes, because they show small parts of many frames. If video were kept compressed on disk, as in most video editors, a whole frame must be decompressed in order to read just a few pixels. Fortunately, most special effects shots are short, allowing them to fit in RAM uncompressed.

To provide an interactive environment, the application architecture in Figure 3.10 is used. The application controls two concurrent threads: one for visualization and another for applying user edits. The visualization thread operates by initiating `getActualPixel()` filter graph calls to fill in the subsampled volume seen by the user. These requests are made at the head of the filter graph and are propagated down to the source footage. The application can

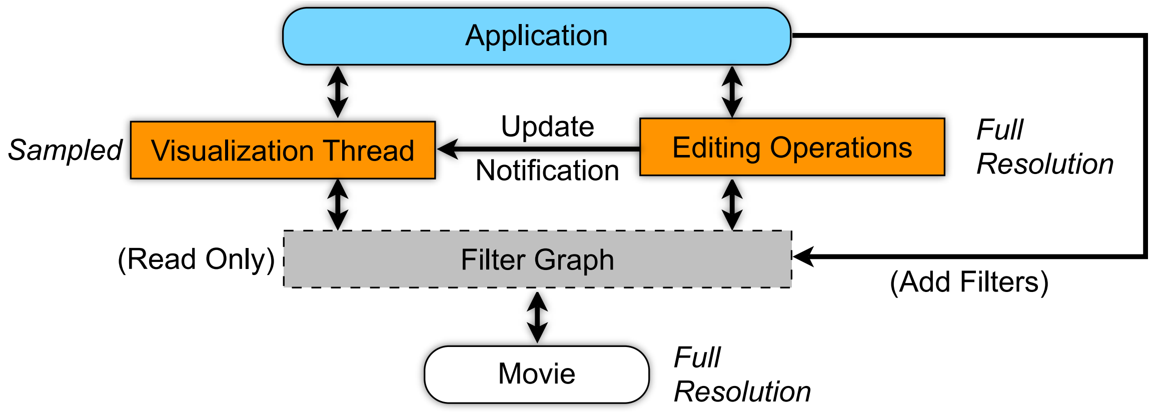

Figure 3.10: A view of the application architecture, highlighting data flow and thread responsibilities. One thread handles the sampled 3D visualization while the other performs edits within the filter graph at its full resolution.

insert filters into the graph at any time at the expense of having to invalidate the pixels in the subsampled view and start again.

However, when users perform edits that are not filters (such as painting), those edits must occur at the full resolution of the source footage, thus using the second thread. This is where sampling issues akin to those mentioned for PShear in Appendix A.2 reoccur. The problem is caused because the user is interacting with a subsampled view of the transformed data. If changes were made to the subsampled data and written back into the volume directly, the result would not change all the pixels in the affected region (or if supersampling occurred, multiple overwrites may result). Thus, edits must occur at the full resolution of the video's filter graph output.

When an edit occurs, the visualization thread must be notified of invalidated areas, so it can reprocess those pixels through the filter graph to rebuild the polygon texture set. If the edit is small, the invalidation area is subsequently small, but if the edit involves adding a full-frame filter, such as a color correction, all pixels will change, requiring a total recalculation. To make the system more interactive during total invalidations, a background thread dynamically adjusts the level of detail of the textures. When the user initially applies a global modification filter, a low resolution set of textures is quickly built and displayed. While the user examines the new volume the textures refine themselves until a sufficiently high resolution is reached.

Given the construction of this application, it can then be used to perform computational video manipulations within spatio-temporal video volume visualizations.

3.4 Results

To demonstrate the possibilities of spatio-temporal video editing, a series of editing workflows are now described enabled by Proscenium. Timing information is provided throughout on a uniprocessor Pentium IV at 2.4 GHz with 1 GB of RAM. The majority of computation is the interface calling `getActualPixel()` to calculate pixels in the filter graph. Thus, the more pixels in the interactively subsampled volume texture, the longer the delay. To account for this relation, timing data for requesting pixels is measured in time per-pixel, but calculations altering the data or rendering final output are given as total processing times.

3.4.1 Object Removal

For the first example, I show the steps involved in the removal of a Frisbee being thrown in a 720x480 120 frame video clip (Figure 3.11) where the camera loosely tracks its path from right to left. The difficulty of removing the Frisbee from the video is two-fold because neither the Frisbee nor the background is stationary over the course of the video. Each element will be sheared separately and edits will be made in each sheared state. Once the edits are complete, the shears will be removed, but the edits will remain.

- First, the video is sheared to stabilize the Frisbee. Loading the video from disk takes 3.3s, and display is .92μs/pixel. Once the Frisbee is mid-air, correspondence points are placed on it in every few frames and tracked. The transform matrices for each frame are calculated (.01s) and applied through the PShear filter (Figure 3.12). Display of the wider view takes 1.53μs/pixel.

- The next step involves deleting the Frisbee by selecting a bounding box around it, which is easily done because of the shearing. These selected pixels are then erased (resulting in a hole in the video). At this point, all PFilters, including the PShear, are removed, restoring the original volume (although now with a hole in it).

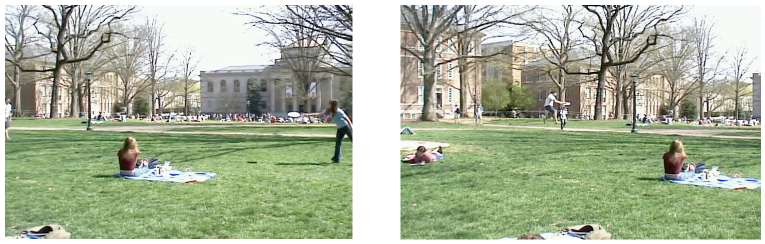

Figure 3.11: Two original frames from a 3 second DV resolution video clip of two people throwing a Frisbee with a panning camera that follows the Frisbee.

- The hole must be filled to restore the background, but the PBackground filter requires the background to be static to build the background plate. The solution is to again shear the volume so the background becomes static, as shown in Figure 3.13. The PBackground filter is now run (64.7s). Note the results of the background replacement are propagated back to the original video volume, replacing the *empty* pixels.
- Finally, the shear is removed, restoring the original camera motion.

The resulting video is shown in Figure 3.14. Note that due to the efficient bi-directional filter graph, RAM usage for the edit was only the original uncompressed movie size plus 25-50% more of garbage collected scratch space.

3.4.2 Aspect Widening

A different effect that can be applied to the Frisbee footage is to expand its aspect ratio from its original 4:3 aspect ratio to 16:9 (Figure 3.15). When the background was stabilized in the previous section, the spatio-temporal volume resembled a panorama. This stabilized volume can be used to to add pixels to the left and right of the original frames, in effect making a solid panoramic volume.

- After stabilization and filling (this time using the *edge filling* filter at 2.14μs/pixel after a 34s pre-calculation), the original video is inlaid into the full width of the panorama; now with the aspect ratio of the full panorama, which is wider than even 16:9.

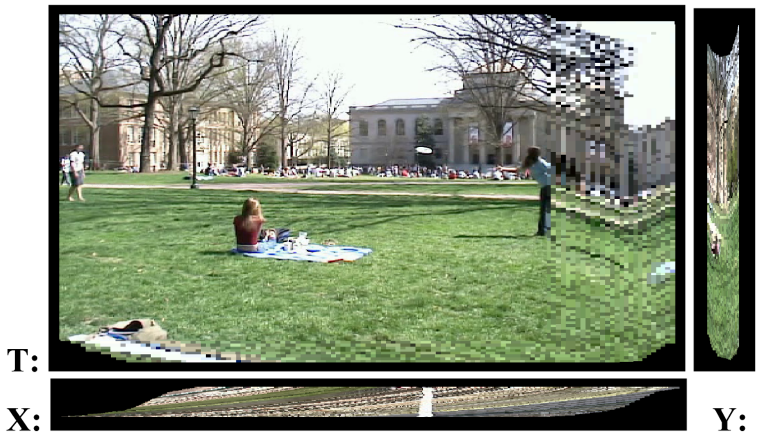

Figure 3.12: Sheared volume after the Frisbee has been stabilized. Three visualizations of the volume are shown above. The top-left image shows the sheared volume at a given time. The right image shows a fixed column through time and the bottom image shows a fixed scan line after the Frisbee has been stabilized.

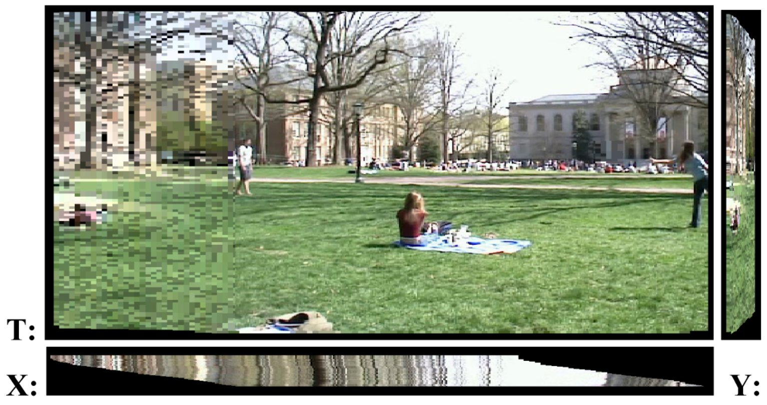

Figure 3.13: A shearing of the volume with the background stabilized is shown. A constant time slice is shown in the upper-left, a fixed column is shown in the upper-right, and the bottom is a fixed scan line through the aligned volume.

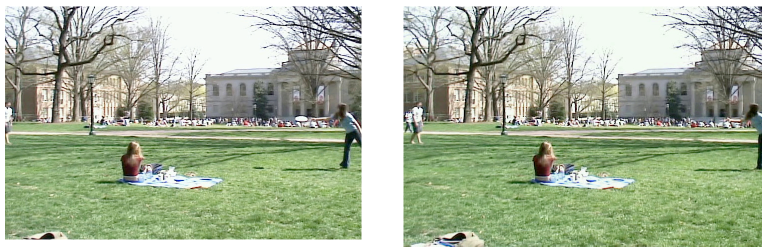

Figure 3.14: Before (left) and after (right) removal of a white Frisbee by stabilizing the Frisbee, cutting it from the video, and replacing those pixels with the background.

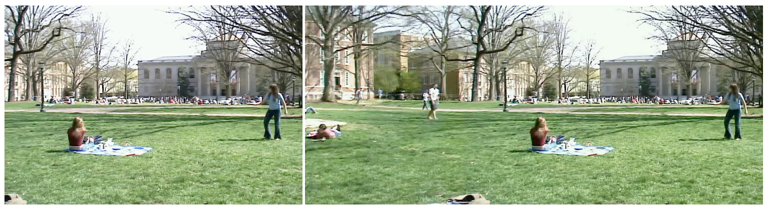

Figure 3.15: Before and after widening the aspect ratio of the Frisbee sequence by background stabilization and background replacement.

- The user could crop the panoramic video to 16:9, but because the video is stabilized, there would be no camera movement. Fortunately, the user can instead shear the volume prior to cropping (2.60μs/pixel), thus adding a synthetic camera motion.
- After shearing, a 16:9 crop can be made (3.21μs/pixel) completing the effect.

3.4.3 Temporal Painting

A common task in post-production is to paint directly onto a moving scene object. This involves performing the same edit to multiple frames while maintaining temporal coherence. To demonstrate temporal painting in spatio-temporal volumes across multiple sheared frames, video of Pixar's walking teapot toy is painted upon (720x480 with 77 frames). The teapot has a logo on it with a bouncing ball that can be painted to "glow" red, as if it had an LED behind it (Figure 3.16).

- The teapot in the video moves from left to right and shows perspective distortion as it bounces up and down while moving. By specifying a shear with three automatically tracked points (.05s to compute and 1.68μs/pixel to display), the distortion around the ball in the logo can be removed, making it appear the same way and in the same location in all frames.
- Once in the sheared domain, a temporal paint brush with a feathered edge and a temporal depth equal to that of the movie is chosen. By drawing a single brush stroke in the first frame, every frame in the sequence is modified (5s).

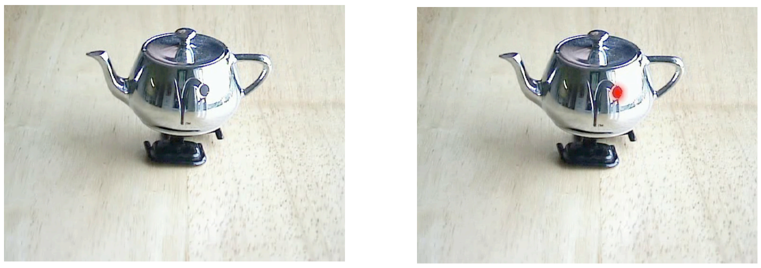

Figure 3.16: A frame from the walking teapot sequence before and after temporal painting of all frames simultaneously with a feathered brush while sheared to provide object stabilization.

- The shear can then be removed and the resulting video will have a glowing LED yet retain its original motion. This entire process (including final rendering) can be completed in a minute and a half, and it handles all perspective distortion of the fictional LED, which an artist would have to painstakingly tweak frame-by-frame to appear correct.

These results have demonstrated both common traditional video manipulation functionality and also more complex computational video operations that, due to shearing, exhibit temporal coherence across the entire edit. Thus, the potential of using spatio-temporal volumes as the basis for advanced video editing is established.

3.5 Summary

In this chapter, computational video effects were generated by leveraging the strengths of spatio-temporal volumes. Essential to this process was the novel capability to shear the volume, which aligns similar image regions in multiple frames. Once sheared, information from adjacent frames may be used, such as to easily propagate paint strokes or to build background mattes.

To illustrate in a working system that spatio-temporal volumes can exhibit capabilities beyond those of traditional video editors, a framework for spatio-temporal volumes and an associated video-editing application were introduced. The framework, Proscenium, demonstrated how bi-directional filter graphs are well suited to spatio-temporal data access patterns. The video editor then utilized these tools to perform edits (which were subsequently measured for performance).

Continuing the line of reasoning that the power of computational video comes from combining information in nearby samples, enhanced here with shearing, another problem is now addressed. By combining many samples in badly-exposed, low-light video, the desired well-exposed, noise-free pixel values can be estimated. These samples are obtained from local spatio-temporal neighborhoods and also cross-spectral neighborhoods to filter with as much information as possible.

CHAPTER 4

LOW-LIGHT VIDEO ENHANCEMENT

In this chapter, two complete computational video approaches are presented to the problem of enhancing low-light video[1]. Low-light video, or Low Dynamic Range (LDR) video, contains high noise levels and significant quantization noise because it only uses a few of the bits of dynamic range the sensor offers. The problem domain of enhancing LDR footage is an excellent test case for proving the capabilities of computational video processing because its enhancement is aided by additional information from nearby spatial and temporal samples. The first approach presented here considers the problem of enhancing a low-light visible-spectrum video by itself, while the second presents a fusion of both visible and non-visible spectra videos, integrating samples from both spectra. Besides both enhancing low-light video, these approaches are also unified by their use and extension of concepts of the edge-preserving bilateral filter (Tomasi and Manduchi, 1998) for a variety of filtering and decomposition tasks.

Visible-spectrum-only enhancement is performed inside a processing model called the Virtual Exposure Camera (VEC) which provides simultaneous tone mapping and noise reduction. Specifically, noise reduction is performed with a novel computational video filter called the Adaptive Spatio-Temporal Accumulation (ASTA) filter. Alternatively, the multispectral fusion in Section 4.3 decomposes each of the input videos and combines the best elements of each into the output video. These underlying components are also enhanced with a new multispectral filtering technique called the *dual bilateral* filter.

To begin, the characteristics of LDR videos are discussed, followed by descriptions and results of each of the two enhancement techniques.

[1]This low-light video enhancement work was sponsored through DARPA-funded, AFRL-managed Agreement FA8650-04-2-6543.

4.1 LDR Video Characteristics

The LDR videos considered in this chapter have a small signal-to-noise ratio and low precision. To be specific, in addition to typical low-light videos themselves, the class of LDR videos also includes videos with “peaky” histograms. Such videos are composed of elements that span a dynamic range larger than the sensor’s, leading to low precision renditions of all elements.

Many common applications result in LDR videos. For instance, filming theatrical lighting is difficult because background scenery is seldom well exposed in comparison to the spotlights placed on the actors. LDR video also results from high speed imaging, where fast shutter speeds are necessary. Small aperture video to increase depth-of-field can also lead to LDR video. Poor lighting scenarios, such as is common in surveillance applications can also result in underexposed videos.

The consideration of LDR video characteristics is really a matter of considering the characteristics of the underlying sensors. There are a variety of noise sources in CCD (Charge Coupled Device) and CMOS (Complementary Metal Oxide Semiconductor) sensors that confound imaging in low-light situations, such as readout, photon shot, dark current, and fixed pattern noise in addition to photon response non-uniformities. These factors, along with noise measurement techniques are discussed by Reibel et al. (2003).

Readout noise results from the amplification and A/D (Analog to Digital) conversion of the analog electronics to digital measurements. Photon shot noise results from the photon light transport process being a Poisson distribution in nature. Dark current thermal noise results from heat on the sensor causing incorrect pixel readings, while fixed-pattern noise is always present. Non-uniformities change the ratio of the number of photons hitting the photosite to the measured luminance at that photosite.

Combining these concepts, a noise model is assumed similar to that of (Tsin et al., 2001). At a high level, an image (I) can be decomposed into the actual signal (E), fixed pattern noise (N_f), and temporal Poisson shot noise which is approximated with zero-mean Gaussian distributions. Thermal dark current sensor noise (N_c) is modeled with constant variance while shot noise (N_s) is modeled with a variance dependent on exposure time and intensity:

$$I = E + N_s + N_c + N_f. \tag{4.1}$$

To simplify this model, a total noise variance, σ_i, is calculated for a sensor and the sum of temporal noise is labeled as N_t:

$$I = E + N_t + N_f. \tag{4.2}$$

It is assumed that dark current noise and fixed pattern noise N_f can be removed via subtraction of a reference dark image at the same temperature and exposure settings. Photon shot and readout noise are the primary problems, but they are assumed zero-mean, so if multiple samples of the same pixel from temporally adjacent frames (M) can be found, their average will reduce the contribution of the per-frame error $N_{t(m)}$ (computing the mean of n samples improves the precision of the luminance readings by a $\sqrt{n}$ factor). However, a significant problem for dealing with dark areas captured with CCDs is that the amplitude of sensor read noise is independent of exposure whereas photon shot noise varies linearly with exposure time. Therefore, read noise is more significant than shot noise at very dark pixels and, likewise, the SNR is comparatively smaller.

$$\lim_{M\to\infty} \frac{1}{M} \left[\sum_{m=0}^{M} \left[E + N_{t(m)} + N_f \right] - N_f \right] = E. \tag{4.3}$$

As an aside, the term “salt-and-pepper noise” is often used to describe significant dark current and photon shot noise that makes random pixels in the image appear very different (either brighter or darker) than their correct value. Salt and pepper noise also may indicate actual corruption of the image itself, either by electronic file corruption or physical occluders in the imaging system. In this work, the term “shot noise” is generically used to refer to photon shot noise, but may also include contributions from read noise and temporal thermal noise if they are particularly strong.

Another source of noise is due to quantization, which occurs at the lowest end of a sensor’s dynamic range. With each additional bit used for luminance, the number of intermediate values is doubled. Thus, far more subtle detail is visible in brighter objects. This is noticeable when a linear gain factor is applied to a dark image, resulting in significant stair-stepping of luminance values because intermediate values could not be represented in so few bits.

In this work, it is assumed that quantization errors can be overcome by averaging many samples, as in Equation 4.3, to achieve sub-integer luminances. However, a more robust model (PQ-noise) is presented by Alter et al. (2006) where it is shown that the averaged fractional luminance is biased toward integer results, as a function of the number of photons required to achieve each luminance level. In effect, the quantization error does not completely cancel out. In that work, a robust non-Euclidean luminance dissimilarity metric is also described that could be used to supplement the dissimilarity values throughout this chapter at the expense of significantly increased computation.

Note that these model assumptions are not true for compressed video footage, where quantization is non-uniform both spatially and across frequencies. In this work, a linear camera response is also assumed, which is true for raw CCD samples, but not for the hidden post-processing found in many camcorders.

Given LDR video, the problem now becomes enhancing it to reduce the sensor noise and quantization errors. Now, a computational enhancement method is presented that extends the exposure time of each sample while simultaneously reducing noise.

4.2 The Virtual Exposure Camera for the Enhancement of LDR Video

The Virtual Exposure Camera (VEC) is a model for analyzing and enhancing LDR video with computational video concepts. The VEC model identifies poorly exposed regions of video and increases precision by simulating longer exposure times. This simulation involves temporal integration (filtering) of the contributions of as many pixel values as would have been captured over the interval of the longer exposure. Figure 4.2 illustrates the high-level VEC processing model, which will now be discussed. The specifics of implementation of its

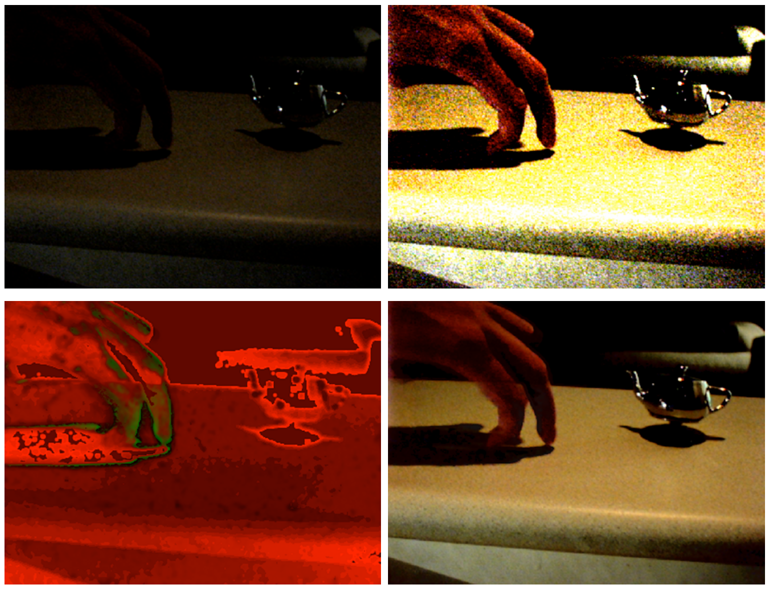

Figure 4.1: A frame from a video processed using the Virtual Exposures Camera (VEC) model. Upper Left: original frame; Upper Right: histogram stretched version; Bottom Left: red = number of temporal pixels integrated, green = number of spatial pixels integrated; Bottom Right: the VEC result after filtering and tone mapping.

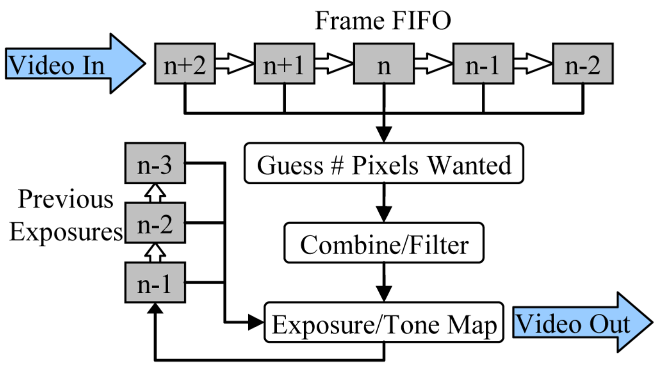

Figure 4.2: The VEC model for processing LDR video. Since no single frame contains sufficient information for noise reduction and tone mapping, spatio-temporal processing is done with knowledge of recent frames and how tone mapping was applied. Rudimentary tone mapping is performed before filtering to guide the adaptive filter's settings.

components are presented in the following sections.

An important relationship is that the amount of filtering ("Combine/Filter" in Figure 4.2) necessary at each pixel relies on how bright that pixel eventually will be displayed, as to avoid amplifying pixels with sensor and quantization noise (*i.e.*, improving that pixel's precision). This is equivalent to determining how many pixels are additively combined to achieve each new per-pixel exposure time, called the gain factor λ. For example, if the desired exposure time is 10 times greater than the input exposure (*i.e.*, $\lambda = 10$), this is equivalent to additively combining 10 similar exposures, or averaging 10 exposures together then amplifying the value by 10. The former is similar to holding the shutter open across exposures to achieve a longer exposure, while the latter implies filtering to improve precision and remove noise prior to amplification, although both are equivalent.

For the VEC, λ is defined as the ratio of the output luminance of a pixel to its input luminance. The luminance increases because, at the end of the VEC pipeline, a tone mapping algorithm ("Exposure/Tone Map" in Figure 4.2) is applied to the result, targeted at improving the visibility of poorly exposed areas and further reducing noise.

However, the tone-mapped result cannot be known prior to actually performing the filtering, which in turn needs to know the tone-mapped result to determine λ. To overcome this circular dependence, an estimate of the post-filtered, tone-mapped luminance is used to determine λ. The approximated tone mapping is found by applying a spatially uniform tone-mapping function $m(x, \psi)$ to a Gaussian blurred version of the pixel (to approximate the ASTA output). This tone mapping then determines λ which is used by ASTA to actually filter the pixel. The result of that filtering is then tone mapped to find the VEC result.

As a matter of implementation, the VEC model treats video as a FIFO queue of frames, where filtering occurs in the current frame but with knowledge of the frames that come before or after it (in a real-time, low latency system, the future might not be known). This allows information from temporally adjacent samples to be integrated into the current frame (as in any computational video approach) but without having to keep the entire video in memory at any given time. As in Proscenium (Chapter 3), pixels are indexed using (x,y,t) notation, with t being the frame number.

To begin analyzing the VEC process, the noise reduction and extended exposure filter will be discussed. Then, in Section 4.2.2, the tone mapping approach is presented.

4.2.1 The ASTA Filter

The underlying noise reduction filter used in my approach for enhancing LDR videos, the Adaptive Spatio-Temporal Accumulation (ASTA) filter, seeks out similar pixels to integrate together to simultaneously extend the exposure time and reduce noise. Two major factors affect how ASTA filters: how many pixels it wants to combine (λ) and if these pixels are in an area of the image with motion. ASTA adapts by transitioning between temporal-only and spatial-only bilateral-inspired filtering while choosing its support based on local illumination.

To begin, the progression of filters that are part of ASTA are presented (temporal and spatial bilateral filters), followed by extensions to those filters in Sections 4.2.1.3 and 4.2.1.4, and then a presentation of the ASTA filter in Section 4.2.1.5.

4.2.1.1 The Spatial Bilateral Filter

ASTA is based on the edge-preserving bilateral filter (Tomasi and Manduchi, 1998). As detailed in the Previous Work, the bilateral filter performs a Gaussian smoothing that attenuates the contributions of pixels by how different their intensities are from the intensity at the center of the kernel. Although a simple subtractive difference is often used to measure this difference of intensities, here this notion is generalized to include non-photometric differences which are also treated as dissimilarity values. A dissimilarity value is any relationship that satisfies the following properties: $D(x, x) = 0$ and $D(x, y) = D(y, x)$. A dissimilarity is metric if the triangle inequality holds: $D(x, y) + D(y, z) \geq D(x, z)$.

As was shown in Section 2.3.1, the spatial bilateral filter centered at a pixel s with a subtractive dissimilarity value $D(p, s)$, is formulated as follows:

$$J_s = \frac{\sum\limits_{p \in \Omega} g(\|p - s\|, \sigma_h) g(D(p, s), \sigma_i) I_p}{\sum\limits_{p \in \Omega} g(\|p - s\|, \sigma_h) g(D(p, s), \sigma_i)}, \tag{4.4}$$

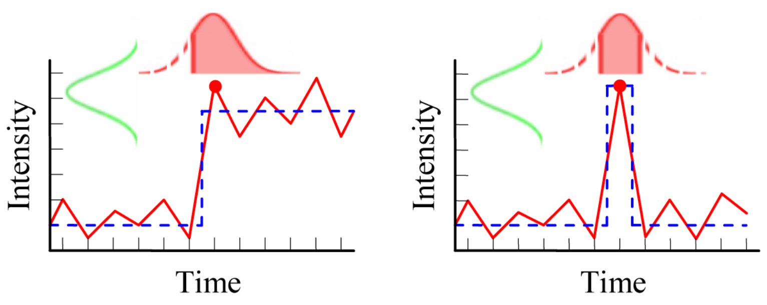

Figure 4.3: Left: The bilateral filter recovers the signal (blue) from the noisy input (red). Right: The bilateral filter is unable to attenuate the shot noise because no other pixels fall within the intensity dissimilarity Gaussian.

$$D(p, s) \equiv I_p - I_s, \tag{4.5}$$

$$\text{and } g(x, \sigma) \equiv \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-x^2}{2\sigma^2}}. \tag{4.6}$$

Here, the spatial kernel Ω is also formalized as

$$\Omega = \begin{bmatrix} p_x = [s_x - k,\ s_x + k] \\ p_y = [s_y - k,\ s_y + k] \end{bmatrix}. \tag{4.7}$$

Three variables control the bilateral filter's operation. First, σ_h controls how quickly the spatial Gaussian falls off. The second, σ_i, controls the Gaussian dissimilarity weighting. It attenuates the contributions of neighboring pixels that are that are too different and is typically chosen based on an estimate of the signal's SNR. Finally, k determines the kernel size.

The bilateral filter does a good job of smoothing out small signal variations (imperfections) while preserving edges, but it is incapable of removing shot noise from a signal (see Figure 4.3). When the filter kernel is centered on an outlier pixel, the intensity Gaussian will exclude all other values, leaving it unchanged, which accentuates it compared to the otherwise cleaned signal.

As an alternate approach to spatial filtering, applying the bilateral filter temporally, enabled by random access to frames by computational video, is now discussed.

4.2.1.2 Bilateral Filtering in Time

In the case of a fixed camera, the best estimate of a pixel's true value is predicted by those pixels at the same location in different frames. In the absence of motion, a simple average of all pixels at each (x, y) coordinate through time gives an optimal answer, assuming zero-mean noise. However, averaging in the presence of motion creates ghosting artifacts. The solution is to consider changes in a pixel's value through time due to motion as "temporal edges". Because a bilateral filter maintains edges while providing noise reduction in areas with small amplitude noise it will not filter across motion. Because of this, a temporal 1D-bilateral filter is used as the basis of the ASTA noise reduction filter. Formally, the temporal 1D-bilateral filter is created by using a standard bilateral filter (Equation 4.4) with the following Ω kernel:

$$\Omega = \begin{bmatrix} p_x = s_x \\ p_y = s_y \\ p_z = [s_z - k,\ s_z + k] \end{bmatrix}. \tag{4.8}$$

A difficulty of applying a temporal bilateral filter is choosing an appropriate value for σ_i (the dissimilarity falloff) that simultaneously removes noise while preserving motion based entirely on differences of pixel luminances. If σ_i is too large, ghosting still results, and if σ_i is too small, noise will remain. Therefore, an ideal value of σ_i often does not exist for noisy videos.

4.2.1.3 Alternate Dissimilarity Values

As a solution to the typical bilateral filter's inability to remove shot noise (both with spatial and temporal bilaterals) mentioned in Section 4.2.1.1, alternate dissimilarity values $D(p, s)$ for the bilateral filter can be used. Specifically, instead of using a simple intensity difference, an arbitrary function may be substituted that returns a value for each pair of pixels in a video or image that may or may not be solely intensity-based.

For example, the dissimilarity value could be the difference between p and some statistic of the local spatial neighborhood around s, making the filter more robust to shot noise. The problem of choosing the intensity at the bilateral filter's center as the sole reference was discussed by Boomgaard and Weijer (2002), but no suggestion of an alternative statistic was given. A wide variety of other statistics could be applied to choose the s pixel's intensity, such as local minima, local maxima, or even other bilateral filters. Even measures not directly associated with luminance at all could be used.

I propose a median-centered bilateral filter in which the value of s would be determined by a small kernel median filter centered at the bilateral kernel's center to improve quality in noisy image areas. This is useful for spatial filtering (and will be used later as part of ASTA), but does not consider temporal filtering, which is preferred when no motion is present. Thus, I now discuss an improvement targeted at temporal bilateral filtering.

4.2.1.4 Spatial Neighborhood Dissimilarity Value

For the purposes of improving temporal bilateral filtering, I propose a specific new dissimilarity value. This dissimilarity value compares the local spatial neighborhoods centered at the same pixel in different frames to improve the distinction between noise and motion. Equation 4.9 shows this normalized Gaussian weighted dissimilarity for an $n \times n$ neighborhood and a temporal edge tolerance of σ_e.

$$D(p_{xyt}, s_{xyt}) \equiv \frac{\displaystyle\sum_{x=s_x-n}^{s_x+n} \sum_{y=s_y-n}^{s_y+n} g\left(||x-p_x,\ y-p_y||, \sigma_e\right) |I_{x,y,p_t} - I_{x,y,s_t}|}{\displaystyle\sum_{x=s_x-n}^{s_x+n} \sum_{y=s_y-n}^{s_y+n} g\left(||x-p_x,\ y-p_y||, \sigma_e\right)}. \tag{4.9}$$

The difference between two pixels' intensities does not provide enough information to judge if they are significantly different. However, by comparing spatial neighborhoods, a judgment can be reached. Thus if only a small percentage of pixels change, it is assumed to just be noise so averaging into the filter occurs. However, if many pixels change, it is assumed to be a more significant event, so no blending occurs. For clarification, despite the fact that neighborhoods are being are compared, only the pixels at the center of each neighborhood are ultimately blended together. This neighborhood size, often between 3×3 and 5×5, can be varied depending on noise characteristics, as can σ_e (usually between 2 and 6). This neighborhood dissimilarity value is inspired by correspondence measures frequently used in

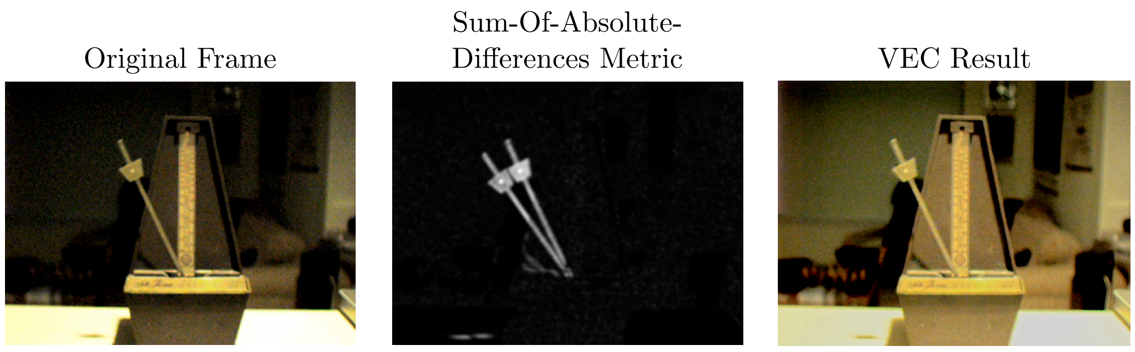

Figure 4.4: Illustration of the spatial neighborhood dissimilarity value used in temporal filtering. The original frame is shown on the left. Each (x,y) for a pair of nearby frames are shown in the middle image. Two metronome arms are seen because the dissimilarity value is based on absolute value. The right image is the same frame processed using VEC with both ASTA and the tone mapper.

stereo imaging. Both Sum of Absolute Differences (SAD) and Sum of Squared Differences (SSD) have been implemented and achieve similar results, although SSD occasionally creates artificially sharp edges. Figure 4.4 illustrates the SAD version.

4.2.1.5 Implementing ASTA

The ASTA filter is now presented which adaptively combines the benefits of spatial and temporal bilateral filtering. Specifically, the filter adapts from the temporal bilateral filter (with the spatial neighborhood dissimilarity value) to the median-centered spatial bilateral filter to achieve the exposure target λ determined by the VEC model.

First, consider that if only temporal bilateral filtering with the spatial neighborhood dissimilarity value is used, and it is applied to an area of high motion, only the center pixel of the kernel will make a sizable contribution to the result. In this case, the filter would not integrate enough pixels to achieve the desired gain factor and the result will be unchanged.

To overcome this problem, ASTA adapts to its surroundings by finding enough pixels to integrate even in the presence of motion. For a static pixel, ASTA reduces to a temporal bilateral filter with the spatial neighborhood difference dissimilarity value. However, if it does not find enough similar pixels to achieve the desired exposure it transitions to a spatial-only median-centered bilateral filter, as shown in Figure 4.5. Like Yee et al. (2001), ASTA also exploits the psychophysical phenomenon that in areas of motion, the human visual system's ability to perceive high frequencies is reduced. Thus, in areas with insufficient temporal information due to motion, it transitions to spatial filtering.

In its implementation, ASTA can be conceptualized as a voting scheme, where each vote is a measure of the support of the filter. Before ASTA is run on a pixel, the total number of required votes λ is already known. The temporal bilateral filter gathers some votes, and if they are not sufficient, more votes are gathered from the spatial bilateral filter.

The actual vote target is defined as $\lambda \times g(0, \sigma_h) \times g(0, \sigma_i)$. The factor $g(0, \sigma_h) \times g(0, \sigma_i)$ is used as the definition of a vote because it is the contribution to the denominator of the bilateral filter from a pixel that is an exact match in space and intensity ($D(x, y) = 0$). The larger the dissimilarity value, the lower its contribution to the denominator is. Thus, by analyzing the denominator of a bilateral filter, it can be determined if a sufficient number of votes were tallied. ASTA is thus formalized in Equation 4.10. The terms n and d represent the numerator and denominator of Equation 4.4, respectively.

$$\begin{aligned}
\tfrac{n_T}{d_T} &= temporalBilateral(x, y, t, \sigma_h, \sigma_i), \\
\tfrac{n_S}{d_S} &= spatialBilateral(x, y, t, \sigma'_h, \sigma'_i), \\
\omega &= \lambda \times g(0, \sigma_h) \times g(0, \sigma_i), \\
ASTA(x, y, t, \lambda, \sigma_i, \sigma'_i) &= \begin{cases} \frac{n_T}{d_T}, & d_T \geq \omega \\ \frac{n_T + n_S}{d_T + d_S}, & d_T < \omega \; AND \; d_T + d_S < \omega \\ \frac{n_T + n_S \frac{\omega - d_T}{d_S}}{\omega}, & d_T < \omega \; AND \; d_T + d_S \geq \omega. \end{cases}
\end{aligned} \tag{4.10}$$

ASTA adapts its filtering settings based on the number of pixels it wants to combine. First, not every pixel could ever get a full vote, because even though it may have the same neighborhood it is attenuated by the distance Gaussian. Therefore, the temporal filter kernel size and Gaussian σ_h are chosen dynamically such that if every comparison were a perfect match, $d_T \approx 2 \times \omega$. Similarly, if the vote count for the temporal bilateral filter comes up short, the spatial bilateral filter attempts to have the remaining number of votes fall within the area of one standard deviation of its distance Gaussian by dynamically choosing σ'_h. The remaining sigmas, σ_i for the temporal bilateral (and σ_e for its dissimilarity value) and σ'_i for

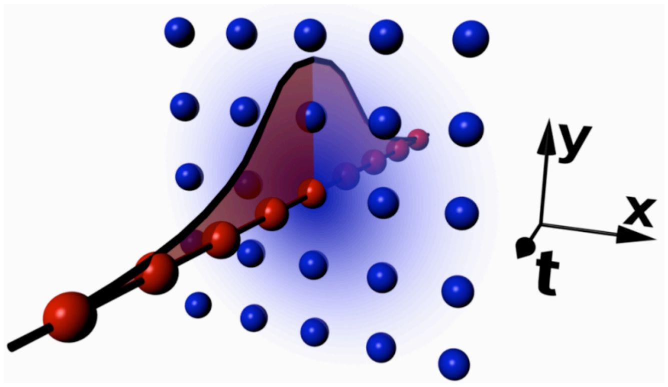

Figure 4.5: Illustration of the temporal-only and spatial-only nature of ASTA. The temporally filtered red pixels are preferred to be integrated into the filter, but if not enough are similar to the center of the kernel, the blue spatial pixels begin to be integrated.

the spatial bilateral, are held constant in each video's processing.

Temporal bilateral filters are run on the image's luminance and mapped to each channel, but only spatial filtering is done on each color channel. Furthermore, spatial filtering is done in the log domain, whereas temporal filtering is not. This is because, in very dark areas, it is difficult to choose σ_i in the log domain for the spatial neighborhood dissimilarity due to the combined contributions of noise fluctuations that each become much larger in log space.

So far, it has been assumed that the camera used to capture footage is stationary, assuring spatial correspondences for background pixels. For moving cameras, the shearing operations described for Proscenium are used. The camera motion is removed by shearing the volume, then the ASTA algorithm can be applied, and the shear removed to achieve the desired result with the original motion.

Now, the problem becomes finding λ, which is a function of the LDR tone mapping operation. Note, the tone mapping is used again after running ASTA to refine the result because more accurate tone mapping can be performed with decreased noise.

4.2.2 LDR Tone Mapping

The VEC's tone mapping approach is now presented which determines the per-pixel amplification of imaged pixels. Most tone mapping approaches attenuate luminance levels, however, with LDR, luminances need to be increased. Here, the HDR tone mapping approach of

Durand and Dorsey (2002) is modified to consider the specific issues of LDR videos, shown in Figure 4.6.

The VEC's tone mapping approach considers the notion that SNR varies with intensity (as shown in Section 4.2.3). Thus, details in dark regions are less accurate than those in brighter regions. A tone mapper specialized for underexposed video should therefore associate a confidence level for details based on their luminous intensity. For instance, in the brightest areas of a video where the CCD received a reasonable exposure, the mix of details and large-scale features should be adjusted to achieve the tone mapping. In darker areas the details should be attenuated to suppress noise.

The tone-mapping approach of Durand and Dorsey (2002), discussed in Section 2.4.2, separates an image into details and large-scale features. Subtracting the original log-image from a bilaterally filtered log-image provides an estimate of the image details. Durand and Dorsey then attenuate the large-scale features by a uniform scale factor in the log domain to reduce the overall contrast of the HDR image, but leave the details untouched. This is not a problem for low-noise source images.

However, for LDR tone-mapping, details and large-scale features need to be processed with different pipelines. Specifically, detail features need to be attenuated based on their estimated accuracy, as determined by local luminance, and large-scale features need to be adjusted to achieve the desired contrast. These two signals are then remixed into the output.

For this purpose, a non-linear mapping function is introduced to serve both mappings. This non-linear mapping function, with independent parameters, is used to both attenuate image details and to adjust the contrast of large-scale features. It obeys the Weber-Fechner law of just-noticeable difference response in human perception but provides a parameter to adapt the logarithmic mapping in a way similar to the logmap function of Drago et al. (2003) and Stockham (1972). The mapping is given by:

$$m(x,\psi) = \frac{log\left(\frac{x}{xMax}(\psi - 1) + 1\right)}{log(\psi)}. \tag{4.11}$$

The white level of the input luminance is set by $xMax$ and ψ controls the attenuation profile. As shown in Figure 4.7, the shape of the detail attenuation and contrast mapping

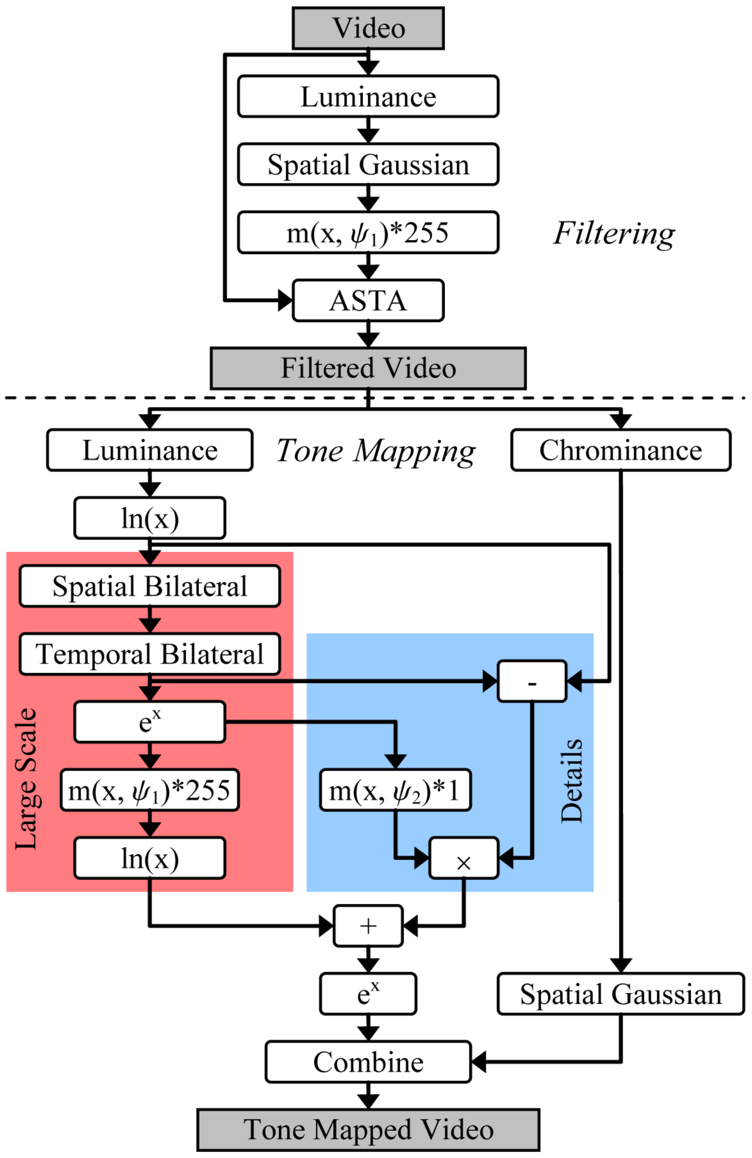

Figure 4.6: A flowchart of the entire process for the Virtual Exposure Camera, including detail of the LDR tone-mapping process. The highlighted areas show the different processing paths of large scale and detail features.

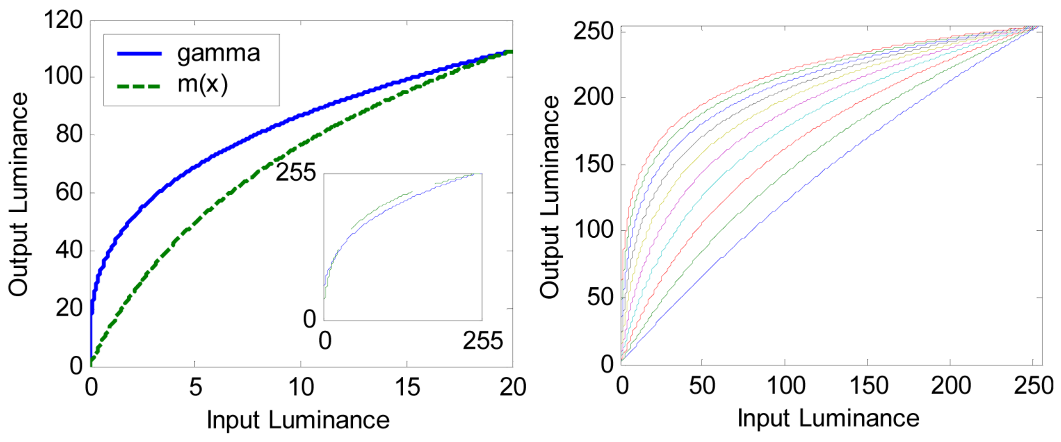

Figure 4.7: Plots showing the non-linear LDR tone mapping function. The left plot shows how the function does not have as steep a slope for luminances near 0 as does gamma correction as to not over-accentuate dark regions (γ=2.0 for gamma correction, $\psi = 64$ for $m(x, \psi)$). The inset shows that over the rest of 0-255, they are mostly similar. The right plot shows a family of $m(x, \psi)$ curves of $\psi = 2$ (the most linear) through $\psi = 1024$ (the most curved).

function, $m(x, \psi)$, is similar to a traditional gamma function, but it exhibits better behavior near the origin. As noted by Drago (2003) the high slope of standard gamma correction for low intensities can result in loss of detail in shadow regions. This is particularly troublesome for underexposed images like the LDR sources considered here.

Given the large-scale/detail feature separation of Durand and Dorsey (2002) and the $m(x, \psi)$ mapping function, the LDR tone mapping process shown in Figure 4.6 can be constructed. The tone mapping begins by extracting the luminance of each frame and the chrominance ratio of each color component as discussed by Eisemann and Durand (2004). A bilateral filter is then applied to the log-image to extract the large-scale features. For temporal coherence in the absence of motion, a temporal bilateral filter, with narrow support (a small σ_i), is applied to maintain similar tone-mapped values from frame-to-frame. This result is then subtracted from the log luminance of the original frame to yield the detail features.

To expand the dynamic range of the large-scale features (which are primarily in the lowest bits of the imager), the linear intensities of the large-scale features are uniformly tone mapped using Equation 4.11, with a ψ_1 of approximately 40.

Separately, the log-intensities of the details are attenuated based on the brightness of the linear large-scale features. Again, this is important to remove details from very dark regions that contain no useful information due to low SNR. If attenuated linearly, a pixel with a brightness of $.5 \times xMax$ would have half of its high frequency masked. However, since the confidence of details degrades most at dark values, details are also attenuated based on the curve. For simplicity, the same curve from Equation 4.11 is used, but with a different ψ_2 (often around 700.0), resulting in a desirably steep roll off for low intensities.

Finally, the log large-scale features and log detail features are recombined to generate the final output luminance. Separately, noise in the chrominance is attenuated via standard Gaussian blurring, taking advantage of the low spatial acuity for chrominance in the human visual system. Finally, the luminance and chrominance ratios are then recombined into the result. The results exhibit better overall use of the dynamic range with reduced detail features in originally dark areas where they were assumed to not be correct.

4.2.3 Results

In this section, the quality of the VEC model is evaluated both qualitatively and quantitatively. However, straightforward evaluation is confounded by the coupled nature of the tone mapping and ASTA filtering because they are designed to work together, yet no appropriate comparison method exists. Furthermore, the video contains motion, which makes comparison to average frames misleading. Alternatively, for completely static scenes, ASTA degenerates to 1D temporal bilateral filtering, which eliminates the benefits of its adaptive nature. Thus, in order to convey the quality of enhancement, a number of metrics are discussed.

All color video footage was captured using a Sony DFW-V500 4:2:2 uncompressed video camera and a Point Grey Research Color Flea at 30 frames per second. The high-speed grayscale footage used for the "Marshmallows" video was captured using a Point Grey Research Dragonfly Express operating at 120 frames per second.

First, examples are shown of stills taken from video sequences with moving foreground objects under low light. Figure 4.1 depicts the entire VEC process for a noisy piece of video of "walking fingers" with a single, dim light source. The pseudo-color image demonstrates how ASTA adapts its integration strategy in different areas of each frame. Figures 4.8 and 4.9 then show additional examples of the VEC system. Figure 4.8 illustrates the processing of a typical LDR frame, and Figure 4.9 shows initial poor utilization of dynamic range.

Next, I illustrate how the VEC model affects the underlying histograms. Figure 4.10 shows the histograms of raw and processed VEC results. ASTA does not noticeably change the shape of the histogram from the original, but the tone mapped result shows the enhanced dynamic range of the VEC approach. Also, the quantization artifacts are removed through the VEC process, as seen in the histogram stretched version, because the ASTA filtering improved the underlying precision of the video.

To further quantify the type of noisy LDR videos the VEC system is designed to handle, the histograms and standard deviations at each luminance level are calculated. Table 4.2 gives a full breakdown of per-luminance errors, summarized in Table 4.1, for four LDR videos (stills are shown in Figure 4.11). This data is also represented as plots in Figures 4.12 and 4.13. Given these values, it is clear that sensor noise dominates over quantization error. The standard deviation of the noise grows slowly and linearly, meaning that it dominates dark pixels, but becomes much less significant as intensities increase, as assumed in Section 4.1.

To arrive at these statistics without captured ground truth, this analysis was performed on dark, static portions of these videos that contain very dark details that would most benefit from the VEC system, but are also the noisiest areas. Given these regions, temporal averaging is used to find the ground truth. As noted before, the downside to this style of analysis is that ASTA tends to become a temporal-only filter in static scenes, which disregards its ability to switch to spatial filtering. However, its spatial filtering element, bilateral filtering, has already been analyzed in the literature (Tomasi and Manduchi, 1998).

Finally, to present the quantitative improvement of these videos, post-enhancement statistics are presented in Table 4.3 for two representative videos. The video of marshmallows is enhanced with the entire pipeline, hence it has a larger mean luminance because of tone mapping. The video of a parking lot used tone mapping to configure the ASTA filter, but the post-filtering tone mapping was never applied, hence it has a similar mean luminance.

A complete video enhancement process has now been presented that used only that video as a source of information. Temporal and spatial filtering were used to find information, because each pixel did not contain sufficient information to create a well-exposed image.

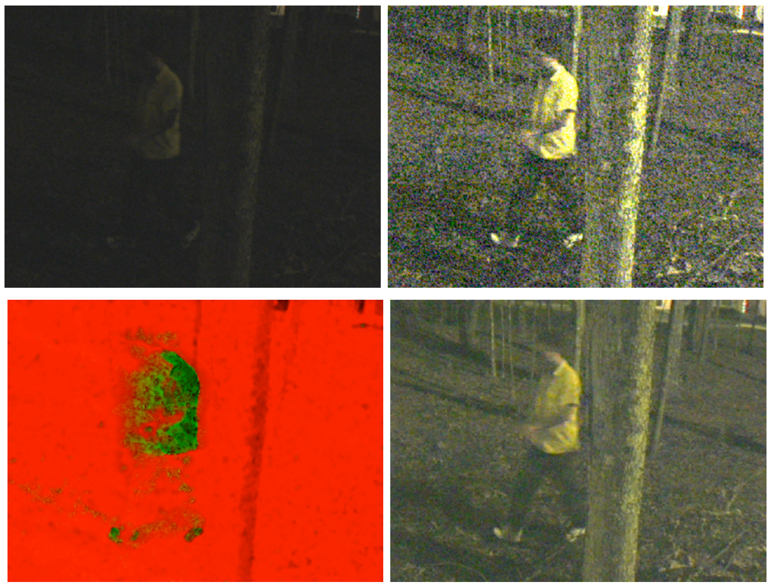

Figure 4.8: A frame from a video processed using the VEC system. Upper Left: original frame; Upper Right: histogram stretched version; Bottom Left: red = number of temporal pixels integrated, green = number of spatial pixels integrated; Bottom Right: the VEC result after filtering and tone mapping.

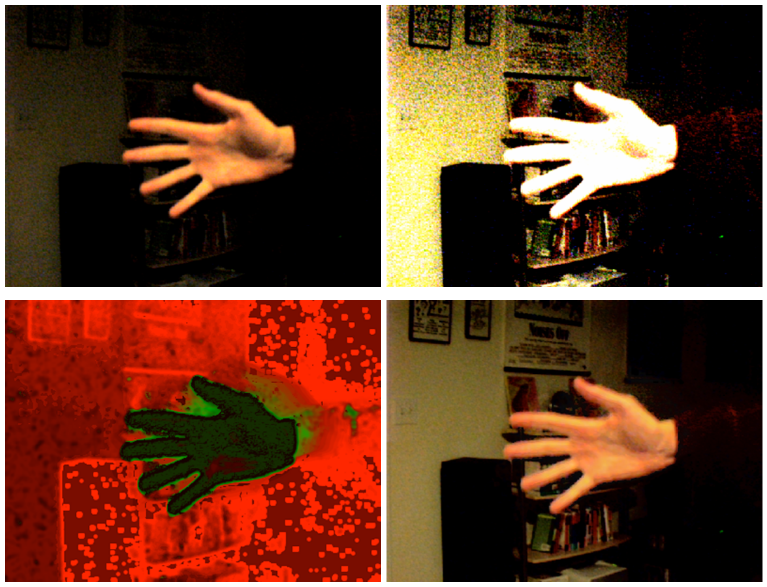

Figure 4.9: A frame from a video processed using the VEC system. Upper Left: original frame; Upper Right: histogram stretched version; Bottom Left: red = number of temporal pixels integrated, green = number of spatial pixels integrated; Bottom Right: the VEC result after filtering and tone mapping.

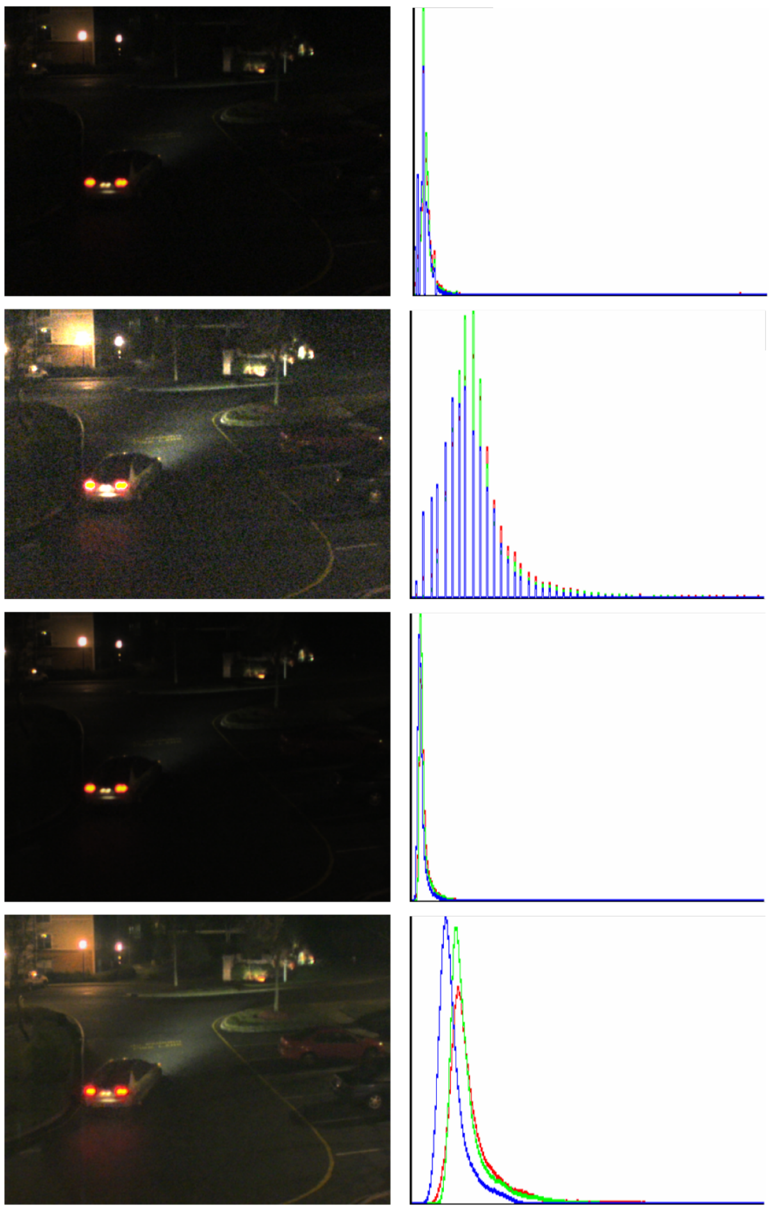

Figure 4.10: Inspection of color histograms in the VEC process. From top to bottom: the original video frame and its histogram; a histogram stretched frame and its histogram showing quantization error; an ASTA processed frame and its histogram which is similar to the unfiltered histogram; the tone mapped ASTA frame and its stretched histogram without quantization error. Note that the vertical scale in these histograms is separately stretched to show maximum detail in each.

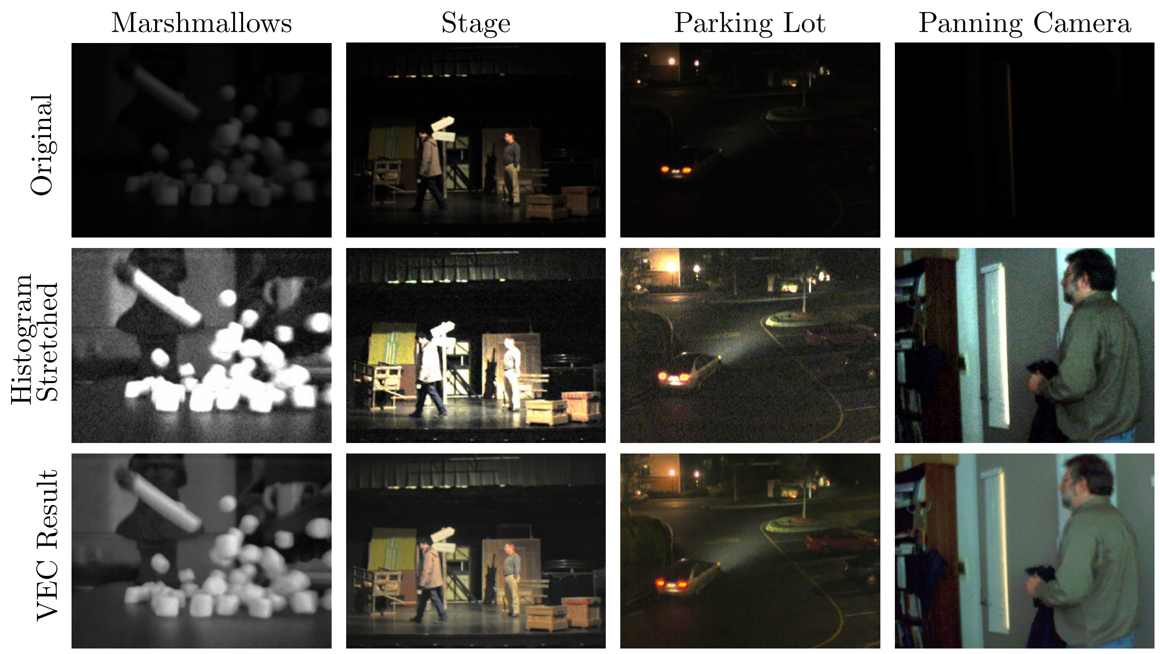

Figure 4.11: Images of four LDR videos with no enhancement, with histogram stretching, and after being run through the VEC system. The marshmallows video shows a dart hitting a stack of marshmallow, filmed at 120 fps. The second shows people walking around a stage. This video is an excellent example of a “peaky” histogram where a few image elements are well exposed, but nothing else is. The third shows a nighttime surveillance video of a parking lot. Finally, a video with a moving camera is shown, enhanced using video shearing to align samples spatially between frames.

Unprocessed Video Footage Statistics

	Marshmallows	**Stage**	**Parking Lot**	**Panning Camera**
Mean	6.535	2.182	2.964	5.345
Standard Deviation	2.536	1.130	1.742	2.437
SNR (Ratio)	2.577	1.931	1.701	2.193
SNR (dB)	8.223	5.717	4.616	6.821
Frames Analyzed	29	30	23	25

Table 4.1: Video noise statistics for the four example videos shown in Figure 4.11. Specifically, these statistics are for dark, static regions in the video, for which the ground truth was determined by temporal averaging.

	Marshmallows		**Stage**		**Parking Lot**		**Panning Camera**	
Mean	**StdDev**	**%**	**StdDev**	**%**	**StdDev**	**%**	**StdDev**	**%**
0	2.469	0.17	0.966	30.34	1.562	0.64	3.688	1.12
1	2.445	3.13	1.043	61.6	1.659	15.05	2.464	15.31
2	2.437	13.8	1.176	75.96	1.730	61.71	2.419	44.98
3	2.451	30.39	1.275	86.24	1.747	88.37	2.368	58.4
4	2.462	42.31	1.330	92.63	1.713	95.82	2.300	65.61
5	2.487	49.18	1.357	95.92	1.713	97.35	2.254	70.21
6	2.485	57.85	1.373	97.26	1.744	98.03	2.249	73.81
7	2.499	69.18	1.408	98.3	1.794	98.59	2.259	77.18
8	2.522	76.92	1.452	98.88	1.793	99.12	2.293	79.64
9	2.589	83.82	1.479	99.27	1.785	99.53	2.338	82.13
10	2.619	90.78	1.537	99.55	1.822	99.83	2.381	84.77
11	2.639	93.52	1.565	99.65	1.864	99.91	2.383	87.64
12	2.685	94.67	1.573	99.69	2.050	99.94	2.390	90.07
13	2.705	95.85	1.520	99.7	1.844	99.95	2.381	92.43
14	2.726	96.75	1.589	99.72	1.880	99.96	2.352	94.72
15	2.756	97.73	1.553	99.74	1.915	99.97	2.361	97.88
16	2.746	99.14	1.706	99.77	2.075	99.97	2.399	99.75
17	2.762	99.9	1.835	99.8	1.533	99.97	2.481	99.97
18	2.747	100	1.646	99.84	2.198	99.98	3.504	99.99
19	3.128	100	1.398	99.86	1.872	99.98	3.217	99.99
20	None		1.682	99.92	1.796	99.99	4.637	100
21	None		1.740	99.97	2.320	99.99	3.141	100
22	None		1.766	99.99	2.221	99.99	4.650	100
23	None		1.208	100	2.045	99.99	None	
24	None		None		1.837	99.99	None	
25	None		None		0.000	99.99	None	
≥26	None		None		2.016	100	None	

Table 4.2: Breakdown of image statistics for the videos shown in Figure 4.11. Percentages shown are for the cumulative percentage of pixel luminances at or less than that value, but standard deviations are for that luminance bucket only. The intuition is that although noise does increase slowly along with luminance, the noise is a far greater percentage of the luminance at dark samples than at bright samples. After processing, the exposures exhibit an expanded dynamic range due to tone mapping, shown in the histograms of Figure 4.10.

VEC Footage Processing Statistics

	Marshmallows		**Parking Lot**	
	Original	**Full VEC**	**Original**	**ASTA Only**
Mean	6.535	53.673	2.964	2.959
Standard Deviation	2.536	1.298	1.742	0.283
SNR (Ratio)	2.577	41.350	1.701	10.456
SNR (dB)	8.223	32.330	4.616	20.388
Frames Analyzed	29		23	

Table 4.3: Video noise statistics before and after processing of two datasets shown in Figure 4.11. The marshmallows video was run through the entire VEC pipeline, resulting in tone mapping which significantly brightens the footage. The parking lot video was only run though the ASTA noise reduction, hence the similar mean but improved SNR. Again, all statistics are derived from dark, static regions of the video.

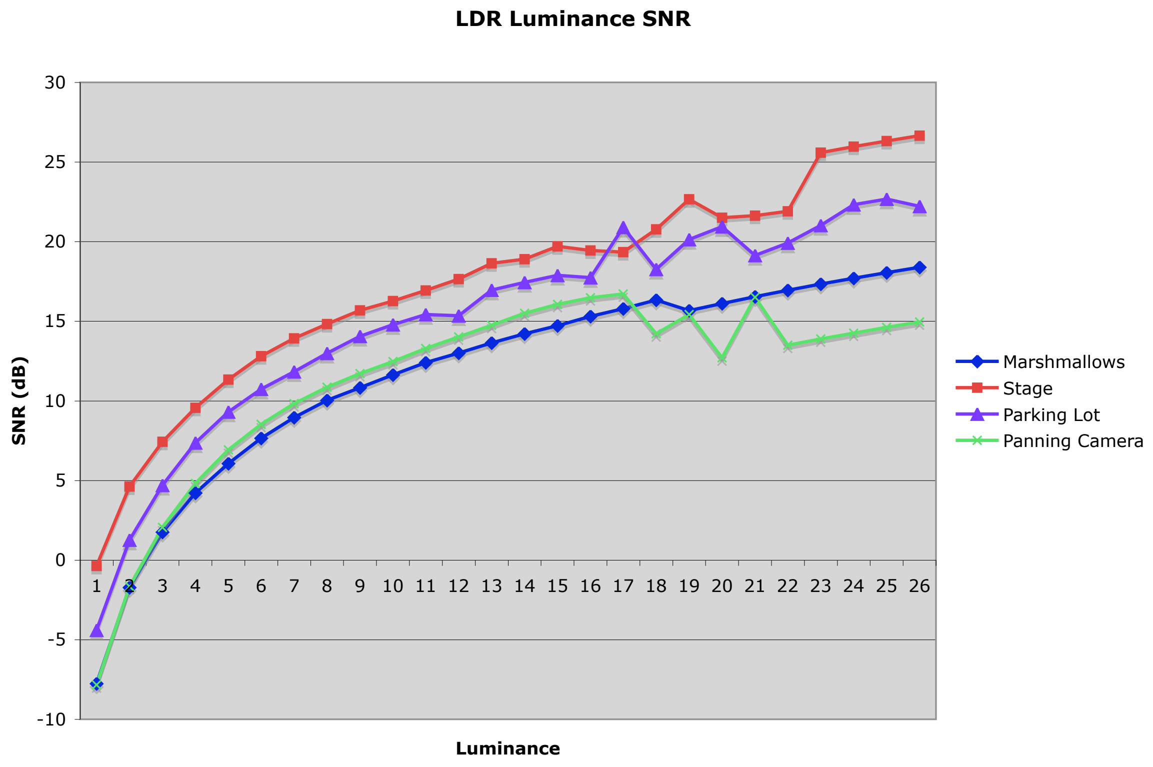

Figure 4.12: A plot of the SNR data from Table 4.2 showing the SNR of pixels at each luminance level in four videos. The data in the table has been converted to SNR to clearly show that as luminance rises, the noise level becomes less and less significant.

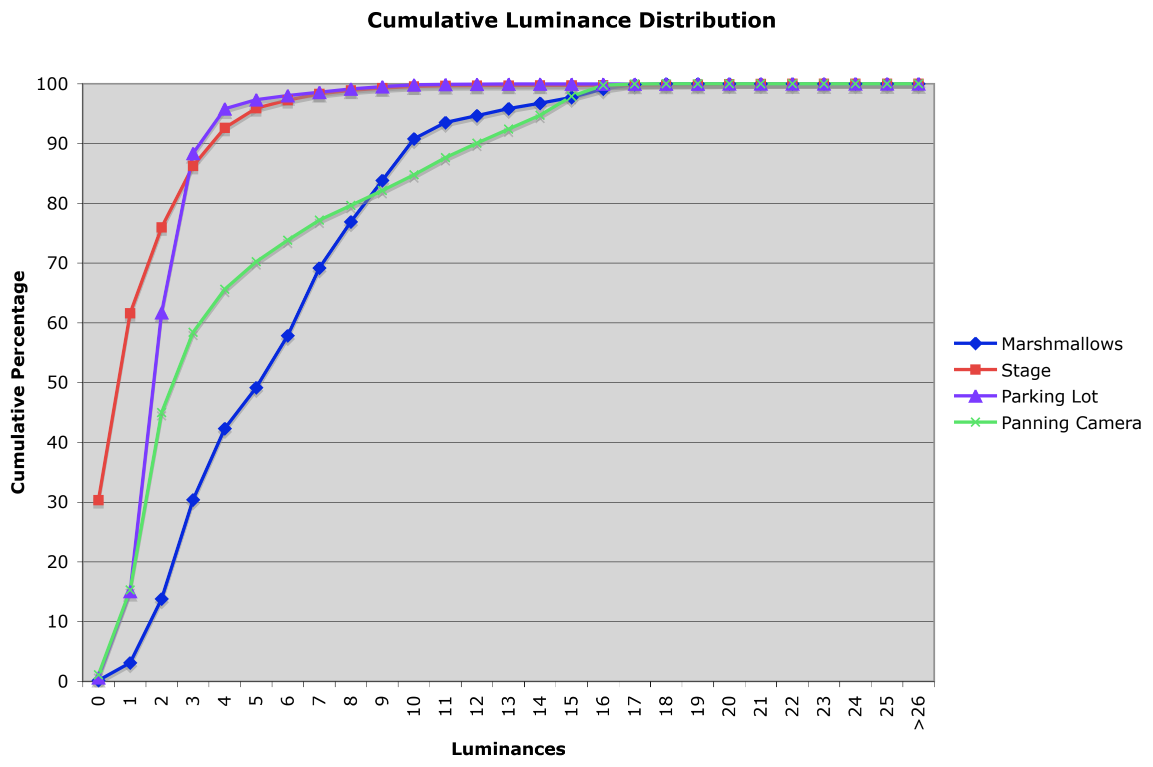

Figure 4.13: A plot of the cumulative distribution of luminances (histogram) of the four videos, using data from Table 4.2. This clearly shows that roughly only 3 bits of precision (0-7) are being used to capture the LDR videos.

4.3 Multispectral Low-Light Video Enhancement

In the remainder of this chapter, the alternate problem of enhancing visible-spectrum video with additional information from a non-visible spectrum video is considered.

A significant issue in night-vision imaging is that, while IR imagery provides a bright and relatively low-noise view of a dark environment, it can be difficult to interpret due to inconsistencies with visible-spectrum imagery. Therefore, attempts have been made to correct for the differences between IR and the visible-spectrum. The first difference is that the relative responses in IR do not match the visible spectrum. This problem is due to differing material reflectivities, heat emissions, and sensor sensitivities in the IR and visible spectra. These differing relative responses between surfaces hinder the human visual system's ability to perceive and identify objects. The other difference is the IR spectrum's lack of natural color. Unfortunately, colorization (chromatic interpretation) of IR footage and correction of relative luminance responses are difficult because there exists no one-to-one mapping between IR intensities and corresponding visible-spectrum luminances and chrominances.

In contrast, visible-spectrum video is easy to interpret due to its natural relative luminances and chrominances, but visible-spectrum sensors typically fail in low-light and night-vision situations due to poor sensor sensitivity. To achieve sufficient responses, long exposure times must be used, making them impractical for video applications.

Because RGB video has desirable perceptual characteristics, I now present a fusion technique that enhances visible-light video using information from a registered and synchronized IR video sensor (Figure 4.14). The goal is to create video that appears as if it was imaged only in the visible spectrum and under more ideal exposure conditions than actually existed. This differs from most multispectral fusion approaches that combine elements from all sensors, creating a mixed spectral representation (Pohl and Genderen, 1998). It also differs from learning-based methods that rely on sparse priors of the visible-light spectrum to enhance IR (Welsh et al., 2002) because an IR/RGB pair is captured for every frame.

The fusion decomposes the visible-spectrum and IR-spectrum videos into low frequencies, edges, and textures (detail features). Specifically, the visible spectrum considered here is 400-700 nm and the IR spectrum is considered as either Short Wave Infrared (SWIR, 900-1700

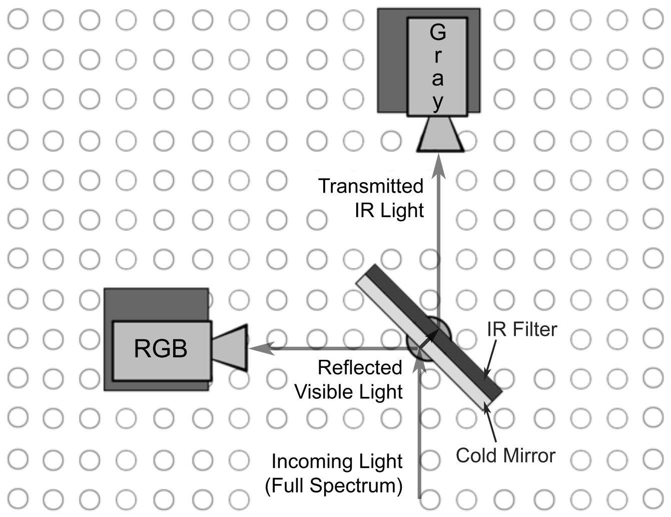

Figure 4.14: Diagram of the prototype multispectral imaging system mounted on an optical bench. The incoming optical path is split with a cold mirror, which provides an efficient separation of spectra, along with an IR pass filter to verify only IR reaches the IR sensor.

nm) or Near Infrared (NIR, 700-2500 nm). These decompositions are enhanced and fused back together in a manner that corrects for their inherent spectral differences.

4.3.1 Fusion Overview

The video fusion can be broken down into four distinct stages:

1. Noise reduction of the RGB video
2. IR video normalization using ratio images
3. Decomposition of input videos into RGB luminance low frequencies, edges, and IR detail features
4. Fusion of multispectral components into the RGB output

The LDR visible spectrum's noise is reduced using bilateral filtering techniques (Section 4.3.2, "Prefilter" in Figure 4.15). This noise reduction improves the accuracy of the decompositions, particularly in static image areas. It also filters chrominance, which is provided by the RGB and is processed in a separate pipeline (Figure 4.18).

Many visible-spectrum textures are corrupted by video noise and must instead be acquired from the IR video. However, the IR textures cannot be transferred directly due to relative

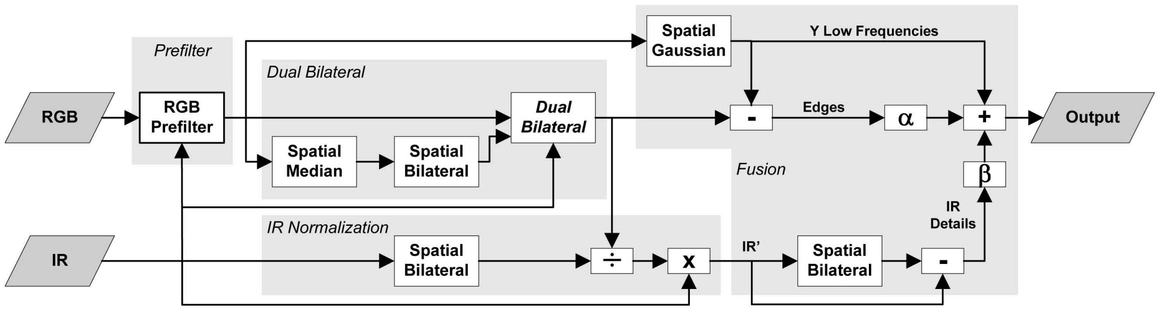

Figure 4.15: Illustration of the luminance processing of the fusion technique. The RGB luminance signal (Y) provides the low frequencies. Assisted by the IR signal, the edges are extracted as well. The IR signal is normalized by a ratio of bilateral filters (large-scale features) then its detail features (textures) are isolated. The right side of the diagram demonstrates the linear combination of image components weighted by α and β.

luminance differences. Thus, the IR video is normalized to exhibit similar relative luminances to the RGB image (Section 4.3.3.1, "IR Normalization" in Figure 4.15).

To extract sharp RGB edges a novel filter called the *dual bilateral filter* is introduced (Section 4.3.3.2, "Dual Bilateral" in Figure 4.15). This filter uses shared edge-detection information from both spectra simultaneously while considering sensor noise tolerances. It also enables more robust IR normalization.

Finally, the extracted components are fused into a single video stream that contains reduced noise, sharp edges, natural colors, and visible-spectrum-like luminances (Section 4.3.4, "Fusion" in Figure 4.15).

4.3.2 RGB Video Noise Reduction

To begin, the visible spectrum video is filtered to improve the signal-to-noise ratio (SNR) of static objects and to provide improved color reproduction. This allows for more accurate decomposition and fusion later in the pipeline (Figure 4.15).

The ASTA filter in Section 4.2.1 provides a great starting place for the design of an algorithm, especially its use of the spatial neighborhood dissimilarity value (Section 4.2.1.4). However, for this multispectral fusion, the fallover to spatial filtering is not used, only the 1D filtering aspect of the algorithm is included. Instead, it is improved in a different manner by using additional IR information.

To further improve the filtering, ideas are incorporated from the joint bilateral filter introduced in (Petschnigg et al., 2004) and (Eisemann and Durand, 2004). Joint bilateral filters allow a second image to shape the kernel's weights. Thus, all dissimilarity comparisons are made in one image, but the filter weights are applied to another. As was presented in Section 2.5, this is accomplished by using a new dissimilarity value, in this case, based on the IR video source:

$$D(p, s) \equiv I_p^{IR} - I_s^{IR}. \tag{4.12}$$

For ASTA-style temporal filtering, this means that the $n \times n$-neighborhood SSD motion detection should occur in the IR video to determine the visible image's filter support. This is accomplished by modifying Equation 4.9 as follows:

$$D(p_{xyt}, s_{xyt}) \equiv \frac{\displaystyle\sum_{x=s_x-n}^{s_x+n} \sum_{y=s_y-n}^{s_y+n} g\left(||x - p_x,\ y - p_y||, \sigma_e\right) \left| I_{x,y,p_t}^{IR} - I_{x,y,s_t}^{IR} \right|}{\displaystyle\sum_{x=s_x-n}^{s_x+n} \sum_{y=s_y-n}^{s_y+n} g\left(||x - p_x,\ y - p_y||, \sigma_e\right)}. \tag{4.13}$$

The complete noise reduction technique is a temporal-only joint bilateral filter that uses $n \times n$ SSD neighborhood dissimilarities calculated in the IR video to filter the visible video. This de-noises the static regions of the RGB video and improves color reproduction.

In most cases, visible-spectrum motion can be detected in the IR video even in the presence of significant relative luminance differences between spectra. However, if the above SSD bilateral neighborhood motion detection fails, the system can be made more robust by replacing it with Equation 4.19, discussed in Section 4.3.3.2.

4.3.3 Video Decomposition Techniques

In this section, methods are described to decompose pre-filtered visible and IR videos into separate components. These components are then assembled (in Section 4.3.4) into the final fusion result. First, a per-pixel scaling of the IR video is introduced that normalizes it to resemble the visible light video. This allows the detail features to be acquired from the IR and appear correct when fused with RGB components. However, this normalization mapping requires knowledge of the large-scale features from the visible imagery, which cannot be robustly extracted using existing bilateral filters because of the remaining confounding noise. Therefore, an extension to the bilateral filter (the *dual bilateral filter*) is given to address this problem. Because of its robustness, this new filter is also used to extract the image components that provide sharp edges in the final fusion.

From this point on, the notation "Y" is used to refer to only the luminance channel of the LDR visible-spectrum input. The chrominance channels, U and V, are separated from the RGB video in YUV color space after pre-filtering (Section 4.3.2) and processed separately in Section 4.3.4.

4.3.3.1 Y and IR Video Normalization

Before decomposing the input videos for fusion, their characteristics are adjusted to more closely resemble the desired system output. To prepare the dark and underexposed Y, it is histogram stretched to the display's full range, often 0-255, or to an HDR range.

Since the eventual goal is to combine IR detail features with visual elements from the visible image, the IR video, from which those detail features are extracted, is remapped to better resemble the stretched Y video. These sources differ in both absolute and relative luminances, so features transferred from IR to visible may not smoothly fuse. Therefore, by correcting these luminance differences by modulating the IR per-pixel image statistics they then resemble those of the Y video.

The concept of ratio images, discussed by Liu et al. (Liu et al., 2001), resembles this normalization process. In their application, images were captured of two faces in neutral poses ($\mathcal{A}$ and $\mathcal{B}$). By assuming a Lambertian illumination model, given a new expression on the first person's face ($\mathcal{A}'$) a similar expression could be simulated on the face of the second person ($\mathcal{B}'$) at each pixel (x, y) with the following modulation:

$$\mathcal{B}'(x,y) = \mathcal{B}(x,y)\frac{\mathcal{A}'(x,y)}{\mathcal{A}(x,y)}. \tag{4.14}$$

In this IR normalization there exists no neutral pose image standard. Instead, to correct differing relative responses, the ratio is the surface-to-surface luminance ratio. Since relative response differences are characteristic of surface types, it follows that their ratios in uniform image regions are ideal for normalization. Uniform regions of the Y and IR videos can be approximated with the spatial bilateral result, the large-scale features (Y_{LS} and IR_{LS}).

Thus, the following formulation normalizes the IR video:

$$IR'(x,y) = IR(x,y)\frac{Y_{LS}(x,y)}{IR_{LS}(x,y)}. \tag{4.15}$$

This normalization is also similar to the per-pixel log-space texture transfers of Durand and Dorsey (2002) and Eisemann et al. (2004) and to the linear-space modulation of Petschnigg et al. (2004). However, this IR normalization is applied to the original images, not to just a single component (such as their detail features). Normalization is crucial because of the significant relative luminance differences between image sources. Normalizing the entire image before decomposition may substantially change the image structure, meaning that pre-normalized large-scale features may become detail features after normalization, and vice versa.

Spatial bilateral filters are run on both the visible and IR videos to obtain Y_{LS} and IR_{LS} respectively. For the well-exposed, relatively noise-free IR video, spatial bilateral filtering extracts the large-scale features as expected. However, directly recovering the large-scale features from the Y video using spatial bilateral filtering fails because it is confounded by the remaining noise. Recall, from Section 4.3.2, that many samples are required to significantly reduce noise and sufficient samples were unavailable in moving regions. Although adapting to spatial filtering to handle these scenarios is an option (as was done in ASTA), here sensor readings from both video sources are simultaneously used to accurately reconstruct the visible video large-scale features for the normalization.

4.3.3.2 Dual Bilateral Filtering

The registered IR video can be used to filter the visible video while preserving edges in order to extract the large-scale features. However, the IR joint bilateral filter, discussed in Section 4.3.2, cannot be directly used because of the inherent differences in spatial edges between the two sources (Figure 4.23). As noted in Section 4.3, features often appear in one spectra but not the other. The goal is to maintain all features present in the visible spectrum to avoid smoothing across edges. Therefore, multiple measurements are used to infer edges from the two non-ideal videos sources: the IR video, with its unnatural relative luminances, and the noisy Y video.

Therefore, I introduce a bilateral filter that includes edge information from multiple sensors, each with its own estimated variance, to extract the Y large-scale features. Sensor noise variance estimates are determined through analysis of fixed-camera, static-scene videos. In the noisy visible video, edges must be significantly pronounced to be considered reliable. The less-noisy IR edges need not be as strong to be considered reliable.

This information is combined in the bilateral kernel as follows. The Gaussian distributions used by the bilateral filter's dissimilarity values, shown in Equations 4.5 and 4.12, can each be recast as the Gaussian probability of both samples p and s lying in the same uniform region given a difference in intensity, denoted $U_{p,s}$:

$$P(U_{p,s}|Y) = g\left(I_p^Y - I_s^Y, \sigma_Y\right), \tag{4.16}$$

$$P(U_{p,s}|IR) = g\left(I_p^{IR} - I_s^{IR}, \sigma_{IR}\right). \tag{4.17}$$

This requires estimating the probability of samples p and s being in the same uniform region (*i.e.*, no edge separating them) given samples from both sensors, $P(U_{p,s}|Y, IR)$. If the noise sources in Equations 4.16 and 4.17 are considered independent, it can be inferred that:

$$P(U_{p,s}|Y, IR) = P(U_{p,s}|Y)P(U_{p,s}|IR). \tag{4.18}$$

From Equation 4.18 it is clear that $P(U_{p,s}|Y, IR)$ will be low if either (or both) $P(U_{p,s}|Y)$ or $P(U_{p,s}|IR)$ are low due to detection of a large photometric difference (an edge). So, I substitute Equations 4.16, 4.17, and 4.18 into Equation 4.4 to derive a *dual bilateral* filter which uses sensor measurements from both spectra to create a combined dissimilarity value:

$$J_s = \frac{\sum\limits_{p \in \Omega} g(\|p-s\|, \sigma) P(U_{p,s}|Y) P(U_{p,s}|IR) I_p}{\sum\limits_{p \in \Omega} g(\|p-s\|, \sigma) P(U_{p,s}|Y) P(U_{p,s}|IR)}. \tag{4.19}$$

This dual bilateral is now used to extract the large-scale features from the visible-spectrum video. The advantages of this approach beyond joint bilateral filtering are illustrated in Figure 4.23.

In the presence of appreciable single-pixel shot noise, the $P(U_{p,s}|Y)$ measure can be confounded, resulting in edges being detected where none exist. It is therefore assumed that no single-pixel detail in the noisy Y video should be considered an edge. To incorporate this notion, the $P(U_{p,s}|Y)$ term in Equation 4.19 should be calculated using a median-filtered Y video that eliminates this shot noise (the Y video filtered by the dual bilateral is unaffected). If desired, any remaining Gaussian temporal noise in the Y edge-detection source can be further attenuated via bilateral filtering. This additional filtering is depicted prior to the dual bilateral in Figure 4.15.

This framework also supports additional sensors by multiplications of $P(U_{p,s}|Sensor)$ in both the numerator and denominator. Because of the normalized bilateral form, any associated scalars will cancel out.

4.3.4 Multispectral Bilateral Video Fusion

The final step is to gather the necessary image components and fuse them together into the result. However, first an optimal fusion for creating enhanced RGB visible-spectrum images is considered. To reiterate, the goal is to reconstruct the RGB source in an enhanced manner with the assistance of the IR imager only as needed.

Figure 4.16 illustrates two methods for decomposing images: Gaussian decomposition into low and high frequencies and edge-preserving decomposition into large-scale features and detail features. The image's sharp edges lie in the area indicated by the dashed lines. To construct the fusion, RGB luminance low frequencies, IR detail features, edges, and chrominance are combined together. The rationale for these filtering and fusion choices are now summarized.

Even in the presence of noise, the RGB luminance video (Y) contains low frequencies of sufficient quality. These provide correct relative luminances for large, uniform image regions. The low frequencies ($LowFreq$) can be extracted by Gaussian smoothing the pre-filtered RGB

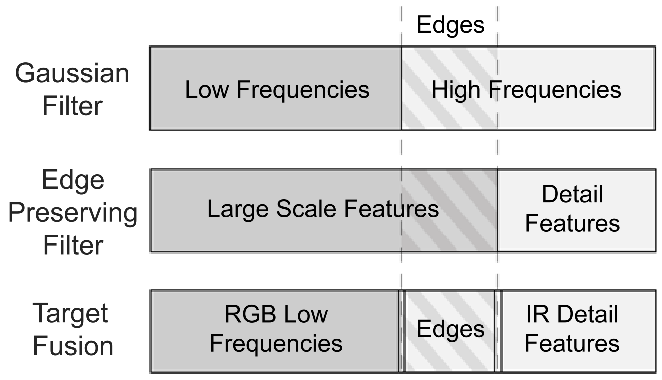

Figure 4.16: Illustration of two common image decomposition methods and how those components are combined by the fusion method. Gaussian smoothing of an image extracts its low frequencies while the remainder of the image constitutes the high frequencies. Similarly, edge preserving filtering extracts large-scale features and details. Edges are separated out (the image components present in the high frequencies but not in the details) and used in the output fusion.

luminance from Section 4.3.2 ($Y_{Gaussian}$).

Because the Y details are most corrupted by visible-spectrum sensor noise, evidence for them is sought in the normalized IR footage (IR'). Detail features ($Details$) are obtained by subtracting the IR spatial bilateral's large-scale features ($IR'_{Bilateral}$) from its unfiltered image (IR') (Figure 4.16). The IR' detail features are used for the entire output image, including static regions, because, from (Grossberg and Nayar, 2004), the minimum signal recoverable from a video source is the mean of the dark current noise at any pixel. Therefore, there are textures in dark areas of the visible-spectrum video that luminance averaging cannot reconstruct. In this case, the better-exposed IR' footage provides those unrecoverable details.

Obtaining accurate edges ($Edges$) is crucial to the sharpness of the fusion output image, yet the visible-spectrum edges were corrupted by noise during capture. On the other hand, not all the edges are present in the IR footage, preventing a direct IR edge transfer. However, the dual bilateral filter in Section 4.3.3.2 can extract enhanced visible-spectrum large-scale features with additional IR measurements ($Y^{IR'}_{DualBilateral}$). The edge components are isolated by subtracting a Gaussian with matching support ($Y_{Gaussian}$). Considering the image deconstruction model (Figure 4.16), the edges complete the fusion along with the RGB luminance low frequencies and the IR detail features.

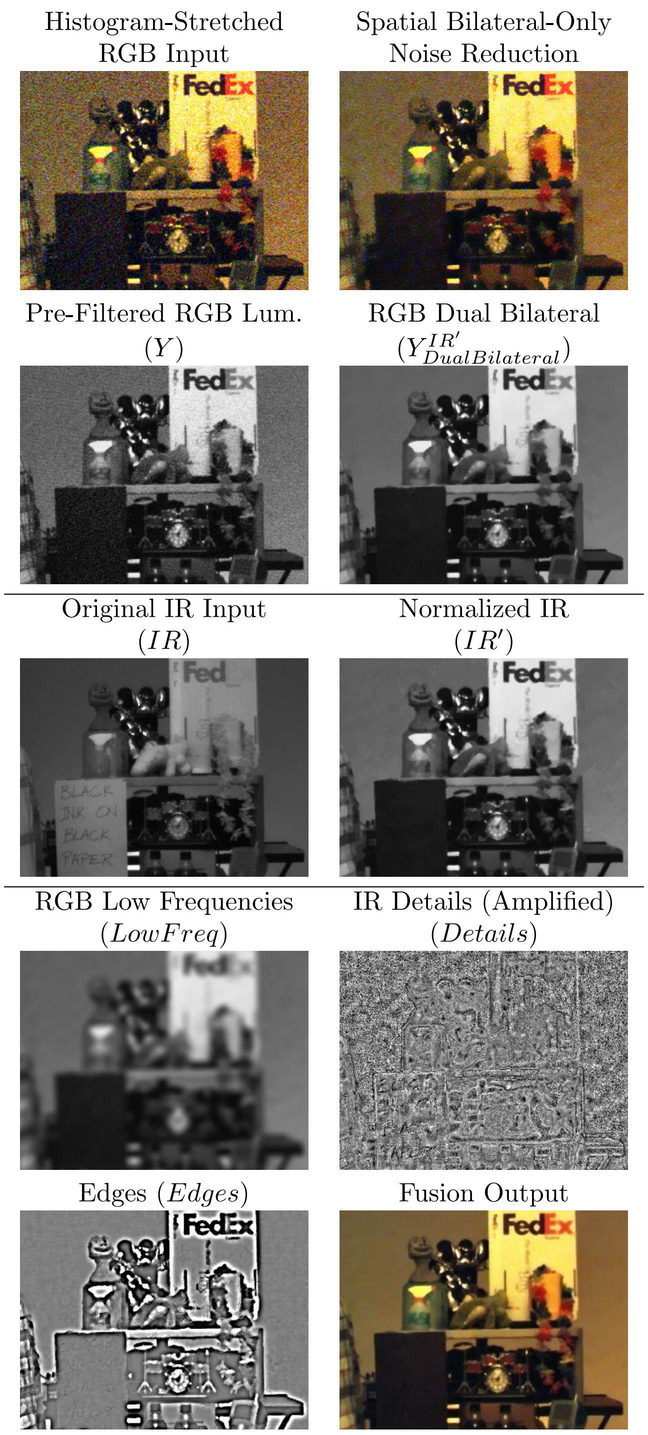
Histogram-Stretched RGB Input
Spatial Bilateral-Only Noise Reduction
Pre-Filtered RGB Lum. (Y)
RGB Dual Bilateral $(Y^{IR'}_{DualBilateral})$
Original IR Input (IR)
Normalized IR (IR')
RGB Low Frequencies $(LowFreq)$
IR Details (Amplified) $(Details)$
Edges $(Edges)$
Fusion Output

Figure 4.17: Illustration of images at various stages of the multispectral fusion processing pipeline associated with the variables used in Section 4.3.4. Specifically, note the quality of the dual bilateral, the proper relative luminances of the normalized IR, and the image components which constitute the final fused output. For comparison, the spatial bilateral-only noise reduction is also shown. Note that although at this size the normalized IR and dual bilateral Y images appear similar, the dual bilateral lacks texture details found in the IR'.

Equations 4.20 and 4.21 detail this entire luminance fusion process (this pipeline is also shown in Figure 4.15 and depicted with step-by-step images in Figure 4.17):

$$\begin{aligned} LowFreq &\equiv Y_{Gaussian}, \\ Edges &\equiv Y^{IR'}_{DualBilateral} - Y_{Gaussian}, \\ Details &\equiv IR' - IR'_{Bilateral}, \end{aligned} \tag{4.20}$$

$$Y' = LowFreq + \alpha(Edges) + \beta(Details). \tag{4.21}$$

A linear combination of the image components determines the final reconstruction. For the examples in the next section, α was set at 1.0 and β was varied between 1.0 and 1.2 depending on the texture content. Values of α greater than 1.0 result in sharper edges but would lead to ringing artifacts. When $\alpha = 1.0$, it is unnecessary to decompose $LowFreq$ and $Edges$, as $Y^{IR'}_{DualBilateral}$ contains both. Subsequently, Equation 4.21 becomes:

$$Y' = Y^{IR'}_{DualBilateral} + \beta(Details). \tag{4.22}$$

The UV chrominance is obtained from the pre-filtered RGB from Section 4.3.2. Gaussian smoothing is used to remove chrominance noise (especially in the non-static areas not significantly improved by pre-filtering). The full chrominance pipeline is shown in Figure 4.18. Although it is possible to filter the UV in the same manner as the luminance (*i.e.*, using the detected edges to limit filtering across edges) doing so limits each pixel's support compared to Gaussian smoothing. Insufficient support leads to noise artifacts and local "blotchiness". Likewise, I chose to trade off sharpness for lower chrominance noise and thus rely on the low spatial chrominance sensitivity of the human visual system to limit blurring artifacts.

4.3.5 Results

Having discussed the foundations of performing low-light, LDR enhancement using multiple spectra, now the results are discussed. First, the specifics of capture are presented followed by the processing of the actual multispectral datasets.

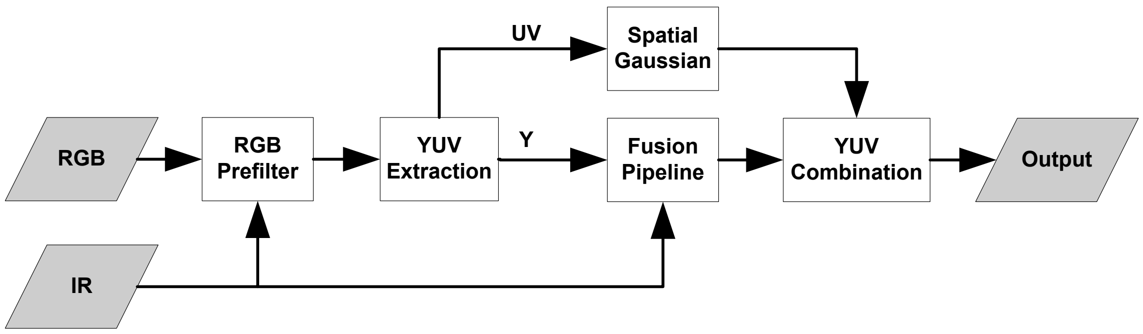

Figure 4.18: Diagram of chrominance processing in the multispectral fusion pipeline. After the RGB temporal noise pre-filtering, the signal is converted to YUV. The Y component goes through the pipeline in Figure 4.15, while the U and V channels are Gaussian smoothed to remove any noise where it was not removed by the pre-filtering (*i.e.*, in areas of motion). Note, pre-filtering is shown in both figures to illustrate when in the overall pipeline the luminance and chrominance signals are split, but pre-filtering is performed only once.

The source RGB and IR videos are captured using two synchronized (genlocked) video cameras sharing a common optical path. Two PointGrey Flea cameras (one grayscale and one RGB) are used with the grayscale camera covered by a longpass filter passing only IR light (780nm 50% cutoff, Edmund Optics #NT32-767). The two cameras are arranged as shown in Figures 4.14 and 4.21. A cold mirror (reflects ~90% of the visible spectrum, transmits ~80% of the IR spectrum, Edmund Optics #NT43-961) is used as a beamsplitter because the spectral sensitivities of these sensors are mutually exclusive. This increases the number of photons reaching the appropriate CCD over traditional beamsplitting. Since each camera has its own lens and sensor alignment, their optical paths may differ slightly. Therefore, a least-squares feature-based homography transform (Wolberg, 1995) is used to register the RGB video to the IR video prior to processing. The RGB sensor has a higher resolution, so some downsampling occurs during the registration. A benefit of this two sensor setup is that in well-lit environments, this same capture rig can also capture visible RGB footage.

For performing RGB and IR fusion, a combined sensor with sensitivity to all four channels would be ideal, but instead multiple registered sensors are used along with a beamsplitter to achieve a similar result. This type of multi-sensor configuration was recently used for multi-sensor matting (McGuire et al., 2005). However, their system was configured using similar cameras at differing focuses, as opposed to using cameras with varying spectral sensitivities.

Because the IR sensor is an unmodified off-the-shelf imager, it is significantly less sensitive to IR than specialized IR sensors, such as InGaAs SWIR sensors. Such high-end sensors would be ideal for this fusion algorithm. Yet even with the current setup, the IR sensor is sensitive up to roughly 1000 nm and provides sufficiently bright imagery for fusion. Also, there is an added benefit from having similar Flea camera bodies, allowing for accurate alignment between imagers.

The noise reduction filters in Sections 4.2.1.1, 4.3.2, and 4.3.3.2 rely upon σ values derived from sensor noise characteristics in static environments. Experimentally, I found an average σ_Y of 8.78 for the RGB sensor and σ_{IR} of 2.25 for the IR sensor. However, I chose values of σ_Y=7.5 and σ_{IR}=2.5 to account for subsequent median and bilateral processing.

The first example, shown in Figure 4.19, shows a frame from a video sequence processed using the multispectral fusion method. In this video, a person walks across the camera's view. Note that the plaid shirt, the plush crocodile, the flowers, and the writing on the paper in the IR video do not match the RGB footage (Figure 4.23). With the fusion, the result contains preserved details and reduced noise in all image regions. Figure 4.22 shows the improvement in signal quality (mean variance) without loss of sharpness for a frame of this video. The second example video, shown in Figure 4.20, shows the reconstruction of a moving robot. This video poses similar problems to the previous example in addition to proper handling of specular highlights.

Finally, Figure 4.17 illustrates the stages of the full processing pipeline by showing images as they are filtered and fused through the system. The images were taken from a 20 frame video with no motion.

4.4 Summary

In this chapter, two computational video methods were presented that take different approaches to solve the problem of enhancing low-light, LDR videos. The first approach improved LDR videos with the Virtual Exposure Camera, which is a combination of an adaptive spatio-temporal filter (ASTA) and a tone-mapping objective. The second approach addressed the usage of an additional registered non-visible spectrum video source that had better visual

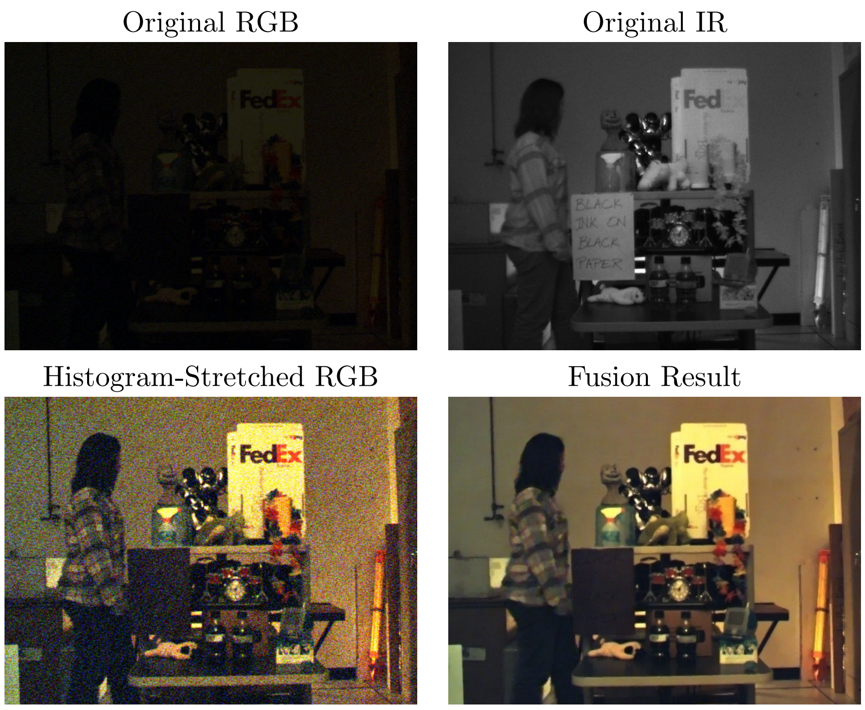

Figure 4.19: Result 1 - Upper Left: A frame from an RGB video of a person walking; Upper Right: the same frame from the IR video; Lower Left: the RGB frame histogram stretched to show noise and detail; Lower Right, the fusion result. Notice the IR video captures neither the vertical stripes on the shirt, the crocodile's luminance, nor the plush dog's luminance. Furthermore, note the IR-only writing on the sign. These problem areas are all properly handled by the multispectral fusion.

characteristics. By using multispectral noise reduction and decomposition the fusion appears to be imaged only in the visible spectrum. Thus, by considering information in multiple frames (and multiple spectra) these methods are capable of improving image quality.

The results of these enhancements are videos of the exact same duration as the original (*i.e.*, each sample in the input is enhanced and included in the output). In the next chapter, an algorithm addressing the problem of significantly altering the video (by drastically shortening its duration) is presented that combines spatial analysis with temporal modification.

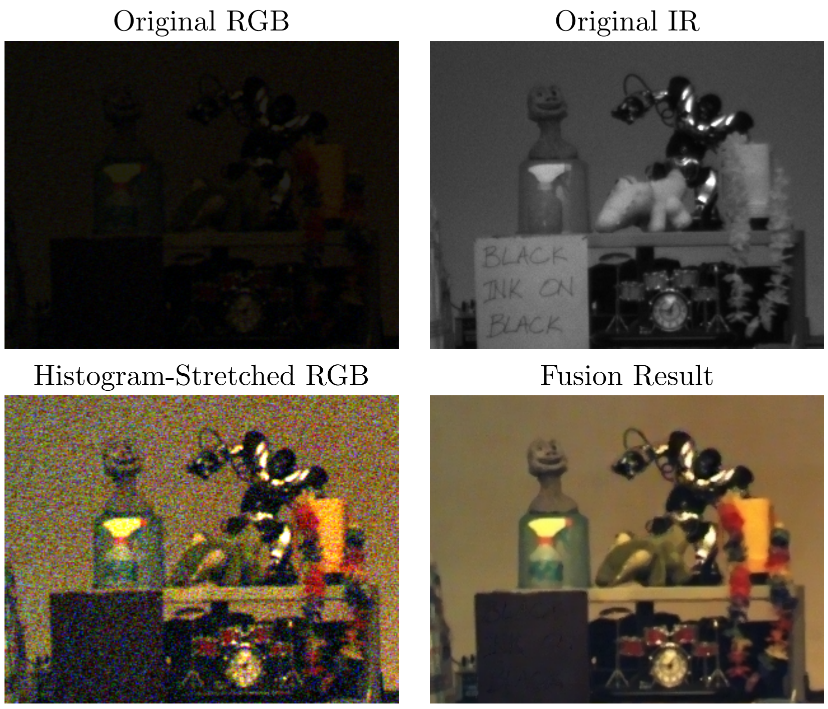

Figure 4.20: Result 2 - Upper Left: A frame from an RGB video of a moving robot; Upper Right: the same frame from the IR video; Lower Left: the RGB frame histogram stretched to show noise and detail; Lower Right: the fusion result.

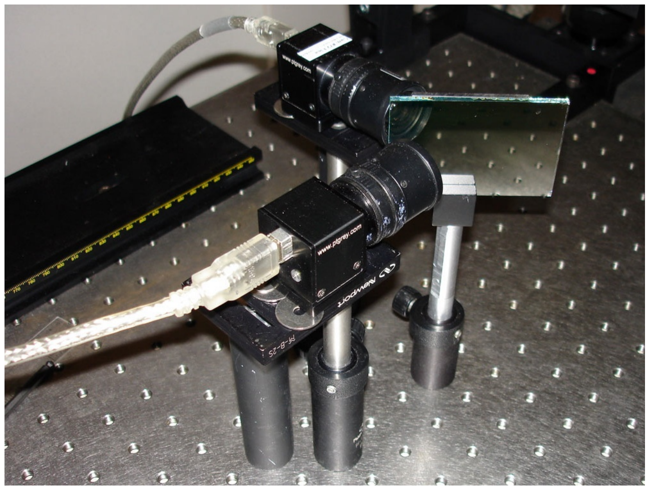

Figure 4.21: A photograph of the multispectral fusion capture setup with a Point Grey Color Flea capturing the visible-spectrum RGB and a filtered Point Grey Grayscale Flea capturing the non-visible IR spectrum.

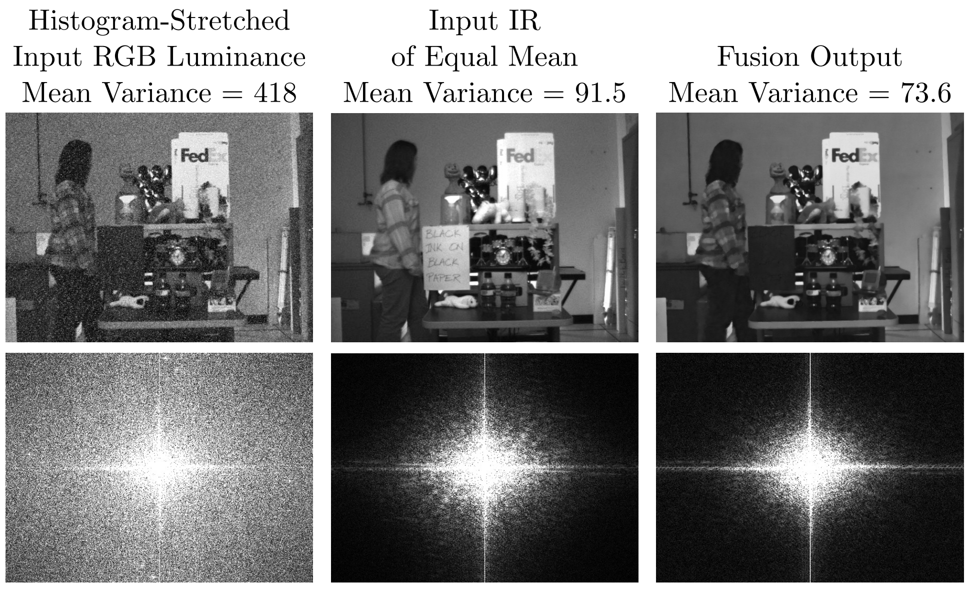

Figure 4.22: Comparison of the mean spatial variance within a 3×3 window and power spectrum of each of the input images and the fused output. The original noisy RGB luminance input (left) is shown with its mean variance and spectral noise. As in the multispectral fusion process, it is histogram stretched to use more of the display's range. The less-noisy IR input (middle) exhibits less high-frequency noise and a lower mean variance than the visible spectrum sensor. For a fair comparison, the histogram of the IR was also stretched to match the visible-spectrum mean, a step not part of our fusion. The fusion result (right) is significantly improved with reduced noise and mean variance while still preserving high-frequency content. These statistics are similar to the IR video, yet are achieved with a visible-spectrum-like response.

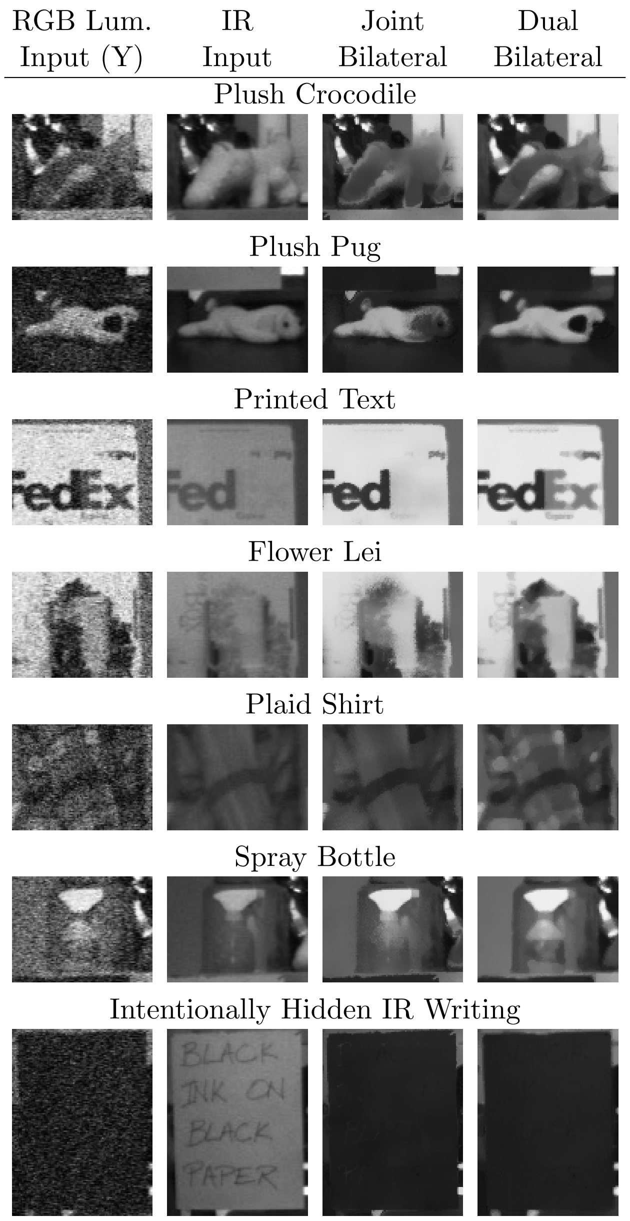

Figure 4.23: Illustration of the difference in quality between the joint bilateral filter ((Petschnigg et al., 2004), (Eisemann and Durand, 2004)) and the new dual bilateral filter, each configured for the best output image quality. The desired output is a enhanced version of the RGB luminance (Y) that preserves all edges. Because the joint bilateral filter relies on IR edges to filter the Y, it cannot correctly handle edges absent in the IR due to relative luminance response differences. This results in blurring across the non-detected edges in the result. However, the dual bilateral filter detects edges in both inputs (weighted by sensor noise measurements) and is thus better at preserving edges only seen in the Y video. Again, note that the target filter output should resemble the visible spectrum, meaning objects visible only in IR should not be included.

CHAPTER 5

COMPUTATIONAL TIME-LAPSE VIDEO

In this chapter, I consider the application of computational video techniques to altering the duration of a video by temporal resampling. It has already been established that over the course of a video, the same objects are imaged repeatedly, implying an underlying redundancy across video frames. Thus, those videos should be representable in fewer frames with reduced redundancy, thus shortening their duration. An extreme case of this, condensing standard-frame-rate videos into very short time-lapse videos is now presented that can yield superior time-lapse results to traditional capture methods.

I now present such a system for generating time-lapse videos with improved sampling, reconstruction, and enhanced artistic control beyond traditional time-lapse capture methods. Traditional time-lapse methods use uniform temporal sampling rates and short, fixed-length exposures to capture each frame. As is the case with any periodic sampling method, the sampling rate must be sufficiently high to capture the highest-frequency changes, otherwise aliasing will occur. This places an upper bound on the capture interval of the time-lapse output if it is to be free of aliasing artifacts. When shorter versions of the output are desired, the filmmaker is forced to make a tradeoff between aliasing, which exhibits itself as popping artifacts in time-lapse videos, missing motions, or pre-filtering, which introduces blurring.

My computational time-lapse video approach simplifies time-lapse capture by removing the need to specify a sampling rate and exposure time a priori. This is accomplished through non-uniform sampling to select salient output frames and non-linear integration of multiple frames to simulate normalized long exposures. However, it requires the input footage be initially captured at video rates. As a post-process, it has knowledge of the entire video, so sampling and exposure settings are based on the entire video's content.

This system relies on two techniques. The first chooses the constituent frames in the output time-lapse by calculating an optimal non-uniform downsampling of the video-rate footage. By downsampling after capture, the most important frames can be selected with knowledge of all motions and changes that occur during the video. Furthermore, it corrects the case where the interesting action occurs out of phase with the uniform sampling. This sampling is derived from a dynamic programming approach to curve approximation that can easily be tuned to support any error metric defined between a pair of frames. I discuss error metrics to achieve a variety of effects, including maximizing change, minimizing change, and controlling sampling uniformity.

The second technique simulates a virtual camera shutter by combining sequential frames together to simulate extended exposure times. This *virtual shutter* acts as a non-linear down-sampling pre-filter that reduces aliasing artifacts. One virtual shutter setting provides *motion tails* that depict dense motion paths over time similar to the multiple-exposure motion studies of Muybridge (1955). However, by leveraging non-linear frame combinations possible through computational video, combinations not achievable via traditional photography are achieved, including "over" compositing, maximum, and median operations. These filters are extentions to the original spatio-temporal filters introduced in the Proscenium architecture in Section 3.2.4 and also related to the synthetic exposures generated by the Adaptive Spatio-Temporal Accumulation (ASTA) filter described in Section 4.2.1.

Independently, each of these two techniques improves the quality of time-lapse videos. When used in combination they provide a flexible tool for constructing time-lapse videos that are both non-uniformly sampled and have variable non-linear exposures, neither possible without computational video methods.

5.1 Traditional Time-Lapse Techniques

To contrast my computational time-lapse video approach against existing approaches, the two most common methods of configuring a film-based time-lapse capture setup are now presented. The most common employs a device called an *intervalometer* that takes a fixed-

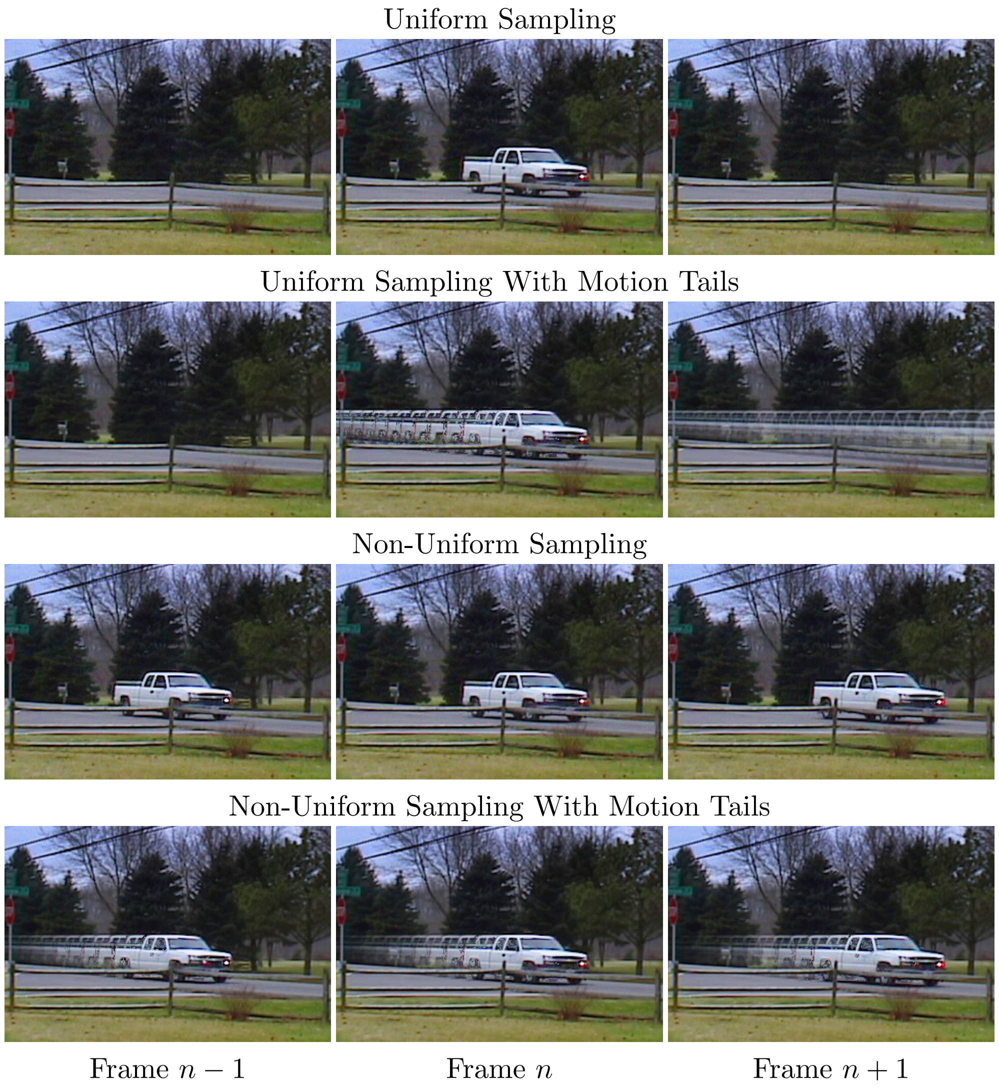
Uniform Sampling
Uniform Sampling With Motion Tails
Non-Uniform Sampling
Non-Uniform Sampling With Motion Tails
Frame $n-1$
Frame n
Frame $n+1$

Figure 5.1: Sequences of three consecutive time-lapse frames that illustrate the sampling issues of traditional time-lapse methods and the computational time-lapse techniques presented here that reduce aliasing. In a standard uniform sampling, the truck appears in only a single frame, resulting in “popping”. Adding motion tails makes the truck appear more noticeable and in multiple frames. Using non-uniform sampling chooses additional output frames containing the truck. Finally, a result is shown combining both techniques. The motion tails are shortened because less motion occurs between frames.

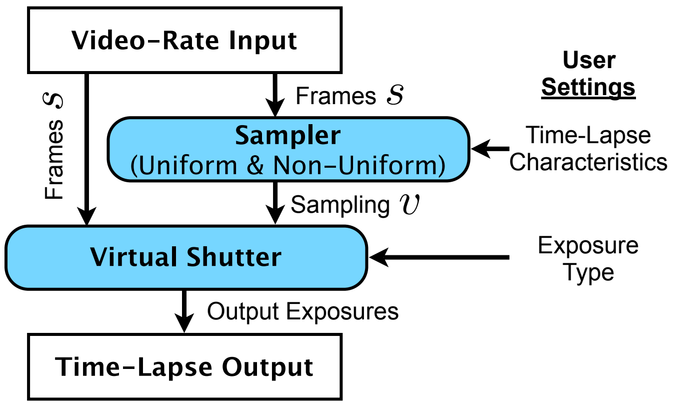

Figure 5.2: Diagram of the flow of the computational time-lapse video system. Video-rate footage (s) is input into the sampler, which chooses the uniform or non-uniform sampling (v) that best matches the user's desired duration and characteristics. This sampling determines the times and durations of the virtual shutter's non-linear synthetic exposures.

length exposure every few seconds, with sampling interval T_s (Equation 5.1). Doing so risks missing motions that occur between exposures or undersampling fast motions which result in aliasing upon playback. A less common approach, used in low-light conditions, takes longer exposures throughout the entire time step. As the exposure length increases, so does motion blur. Motion blur guarantees high-frequency motion will be imaged, but that motion will be blurred and often fades into the background.

$$T_s = \frac{EventDuration}{totalOutputFrames - 1}. \tag{5.1}$$

As digital still photographers have long known, it is better to capture at high resolutions and perform spatial downsampling and/or cropping as a post-process, which allows multiple alternatives to be explored and tried non-destructively. I argue that a similar philosophy can be applied to the construction of time-lapse videos. This means capturing at a higher frame-rate than the T_s step dictates, then temporally downsampling in a controlled post-processing environment with global knowledge of the video. The entire computational time-lapse video approach assumes a video-rate input and transforms it into a shorter video through a combination of downsampling and integration techniques. Until recently, capturing long videos at high resolutions was impractical due to storage requirements, but it is now possible to store days of video on most commodity PCs.

In computational time-lapse, time is discretely sampled into frames in the original video-rate footage. Taking a single short exposure at each time step (*i.e.*, one sample from many samples) is a non-optimal downsampling, potentially generating temporal aliasing. The discrete sampling interval T_d is a re-expression of T_s:

$$T_d = \frac{totalInputFrames}{totalOutputFrames - 1}. \tag{5.2}$$

Considering this uniform downsampling as modulation with an impulse train, the train's phase (the offset of the first sample frame) can drastically alter the results. Leaving the shutter open for a longer fraction of each time step is analogous to convolving the source signal with a low-pass filter prior to downsampling. While this decreases aliasing, it is not ideal for conveying motion. The computational time-lapse video methods instead provide flexibility both in which frames are chosen and how samples are integrated to simulate an exposure.

Another class of time-lapse systems uses feedback instead of an intervalometer. Using a CCD attached to a computer, the last recorded exposure is compared with the most recent exposure. If they are dissimilar enough, the new exposure is recorded and the process repeated. Similarly, some surveillance cameras use motion detectors to trigger video-rate recording. However, the methods presented in this chapter generate videos of user-specified duration whereas these other approaches cannot.

Now, having considered traditional sampling at time of capture, we can consider the problem of optimally sampling a completely known signal in a post-process, first in 1D then in video.

5.2 Non-Uniform Temporal Sampling

In this section, I develop a non-uniform temporal sampling algorithm to select the frames of a time-lapse video. The user specifies the duration and visual characteristics of the time-lapse output then the sampler chooses the frames accordingly. To explain this technique, I revisit the 1D piecewise linear signal approximation technique of Perez and Vidal (1994) that forms the basis of the sampler. This algorithm is then extended to video, modeled with a unique high-dimensional vector signal through time. Finally, alternate error metrics and algorithmic optimizations are discussed.

5.2.1 1D Signal Approximation

To understand the computational time-lapse video sampling approach, first consider the problem of finding a piecewise-linear approximation of a sampled 1D signal s of length N. The goal is to find the optimal subset, v, of M samples from s, that, when linearly interpolated between their original positions, best reconstruct s. The signal is approximated by interpolating samples already present in s, thus only the indices of v need to calculated. In computational geometry, this problem is known as $min - \epsilon$ (Perez and Vidal, 1994).

The quality of a reconstruction can be measured by the sum of squared errors between all samples s_k in the original signal s and their values in the interpolated signal. Therefore, the optimal solution has the lowest total error over all of its linearly-interpolated segments. The error incurred by each piecewise segment is calculated with the metric $\Delta(i,j)$: the error given two sample indices i and j from the original signal s that form a segment's endpoints. $\Delta(i,j)$ is a sum-of-squared error metric between all points on that interpolated segment (modeled with slope b_{ij} and y-intercept a_{ij}) and the s_k samples they approximate:

$$\Delta(i,j) = \sum_{k=i}^{j} \left((a_{ij} + b_{ij}k) - s_k\right)^2 , \tag{5.3}$$

$$a_{ij} = s_i - b_{ij}i \, , \qquad b_{ij} = (s_j - s_i)/(j - i). \tag{5.4}$$

We now seek $\hat{D}(s,M)$, the optimal vertex set v, that results in the minimum error reconstruction of s when $M = |v|$. The reconstruction error is the total of the individual $\Delta(i,j)$ segment errors:

$$\hat{D}(s,M) = min_v \sum_{n=1}^{M-1} \Delta(v_n, v_{n+1}). \tag{5.5}$$

The solution of $\hat{D}(s, M)$ can be expressed via the following recursive algorithm (Perez and Vidal, 1994):

$$D(n,m) = \begin{cases} min_{m-1 \leq i \leq n-1}(D(i, m-1) + \Delta(i,n)), & n \geq m > 1 \\ 0, & n = m = 1 \\ +\infty, & otherwise \end{cases} \tag{5.6}$$

$$\hat{D}(s, M) = D(N, M). \tag{5.7}$$

This recursion can be efficiently solved with dynamic programming. The time complexity of a naïve implementation is $\Theta(M \cdot N^3)$, derived from computing the $D(N, M)$ memoization table of size $M \times N$. Solving each $D(N, M)$ requires finding the minimum of up to N different segment possibilities. Evaluation of the underlying errors requires solving up to N points along that segment. For further analysis of the time complexity and a pseudo-code implementation for curve approximation consult (Perez and Vidal, 1994). Having established this optimal sampling in 1D, I now extend it to video signals.

5.2.2 Min-Error Video Time-Lapse Sampling

One definition of an "optimal" time-lapse video is the video that retains as much of the original motion and change as possible. However, this is not the only desirable time-lapse output and in the following sections I discuss alternate time-lapse characteristics.

Now, consider finding the optimal M frame time-lapse video of an N frame 1 pixel movie. This movie is modeled as a 1D signal of its luminance over time. To create a time-lapse version, it must be determined which samples should be included in the output.

The curve-approximation algorithm from the previous section can be used to choose samples, as it includes samples that adaptively model a signal to achieve a duration M. It guarantees that as many signal changes as possible will be included by the interpolated samples v. Playing back the v samples sequentially without interpolation gives a time-lapse video that represents the best non-uniformly sampled frames to approximate the full-length video.

I extend this idea to select the set of v frames that generate the optimal 2D time-lapse video. A straightforward analogy to finding the optimal piecewise-linear approximation to a 1D curve is to find a set of frames that, when piecewise-linearly interpolated, best approximate the original movie. This frame interpolation can be thought of as a "cross-fade movie" where each of the v frames is blended together to create the interim frames. Again, these frames include as much motion and change as possible in M frames.

I now modify the Perez and Vidal (1994) algorithm to process 2D videos. In Equation 5.3, the function $\Delta(i,j)$ specifies the error introduced when an interpolated segment exists between frames i and j in a 1D signal. Because the interim frames between i and j are omitted in the resulting time-lapse output, $\Delta(i,j)$ can also be thought of as the cost of jumping between samples. The video is treated as a vector signal s_k^{xy} (where k is a frame at time t) evaluated under this metric. Like all computational video approaches, this method relies heavily on random access to all of the video's samples to generate these error metrics:

$$\Delta(i,j)^{xy} = \sum_{k=i}^{j} \left((a_{ij}^{xy} + b_{ij}^{xy} k) - s_k^{xy} \right)^2 . \tag{5.8}$$

Here, the error of a video segment (the segment's error between each of its original frames and the interpolated "cross-fade movie" frames) is the sum of the segment errors of all the frame's pixels. $\Delta(i,j)$ is from now on referred to as the *min-error* metric:

$$\Delta(i,j) = \sum_{x} \sum_{y} \Delta(i,j)^{xy}. \tag{5.9}$$

As before, a dynamic programming solver is used to identify the v frames whose corresponding interpolated segments have the lowest total error. These frames then become the time-lapse output.

Although this solution produces useful results as-is, "shape" controls are included to modify $\Delta(i,j)$. This way, the user can affect the impact of particularly low or high errors on the final sampling. Specifically, I propose the following modified form:

$$\Delta'(i,j) = (\beta \Delta(i,j))^{\alpha} . \tag{5.10}$$

In many cases, α and β are near 1, and adjusting them affects sample clustering. For videos with significant scene or sensor noise an advanced form, with a per-frame threshold term γ, can be used:

$$\Delta'(i,j) = \left(\beta \times MAX\left(0, \Delta(i,j) - MAX(0,(j-i-1)\gamma)\right)\right)^{\alpha}. \tag{5.11}$$

Finally, the Perez and Vidal (1994) solution must be modified to force time to monotonically increase by adjusting Equation 5.6 to incur an error of $+\infty$ when $n = m$, except when n and m are both 1. This creates degenerate solutions for $M > N$, but that would imply a non-time-lapse video output.

5.2.3 Min-Change Error Metric

The default algorithm generates time-lapse videos that preserve as much motion and change as the duration M permits. Now, the construction of time-lapse videos is considered that avoids including frames that are significantly different from other sampled frames. This means choosing similar frames that minimize temporal aliasing by avoiding objects that "pop" in and out.

The resulting v can be controlled by introducing the *min-change* error metric $\delta(i,j)$ that may be used in place of $\Delta(i,j)$, the cost of a segment being included in v. Because dissimilar frames should not be included sequentially in v, $\delta(i,j)$ is penalized based on the dissimilarity between frames i and j. In other words, a low-error segment is now defined as one whose end frames are similar and thus jumping between i and j will be minimally noticeable, *i.e.*,

$$\delta(i,j)^{xy} = \left(s_j^{xy} - s_i^{xy}\right)^2. \tag{5.12}$$

When the full-frame $\delta(i,j)$ is calculated, the result is a sum-of-squared difference between the two end frames only (the interim frames do not affect the error of the segment/jump). This calculation is reminiscent of the metric used for jump evaluation in Video Textures (Schödl et al., 2000).

As in Section 5.2.2, Equation 5.10 (substituted with $\delta(i,j)$) may be used to "shape" the errors to refine v. As before, α and β are near 1. If an offset is required, a per-jump γ error offset form is suggested:

$$\delta'(i,j) = \left(\beta \times MAX\left(0, \delta(i,j) - \gamma\right)\right)^{\alpha}. \tag{5.13}$$

5.2.4 Enforcing Uniformity

Up to this point, it has been assumed that uniform sampling was undesirable. However, if samplings become very non-uniform, as in the case of the minimum change error metric, samples tend to cluster in close temporal proximity and everything else is ignored. This results in the playback of a segment rather than a summation. Thus, I now introduce an error metric that can be used in addition to the previous metrics to control the uniformity of the sampling.

As in Section 5.1, the uniform discrete sampling interval T_d can be determined. A uniformity metric should attempt to include segments in v that are T_d apart and penalize others by how far they stray from T_d. This is done with another error metric, $\Upsilon(i,j)$, this time with a normalized penalty based on dissimilarity from T_d:

$$\Upsilon(i,j) = \frac{j - i - T_d}{T_d}. \tag{5.14}$$

This metric alone is useless, since it results in a uniform sampling. When used in conjunction with either the min-error $\Delta(i,j)$ or min-change $\delta(i,j)$ metrics from Sections 5.2.2 and 5.2.3 it acts like a spring force pulling the solution towards uniformity. Below, linearly-weighted combinations are presented to accomplish this that are called the *combined uniform* error metrics:

$$\Delta(i,j)_{combined} = \lambda\Delta'(i,j) + (1-\lambda)\Upsilon(i,j), \tag{5.15}$$

$$\delta(i,j)_{combined} = \lambda\delta'(i,j) + (1-\lambda)\Upsilon(i,j). \tag{5.16}$$

Note, the shape variables α, β, and γ must act to normalize $\Delta'(i,j)$ and $\delta'(i,j)$ between

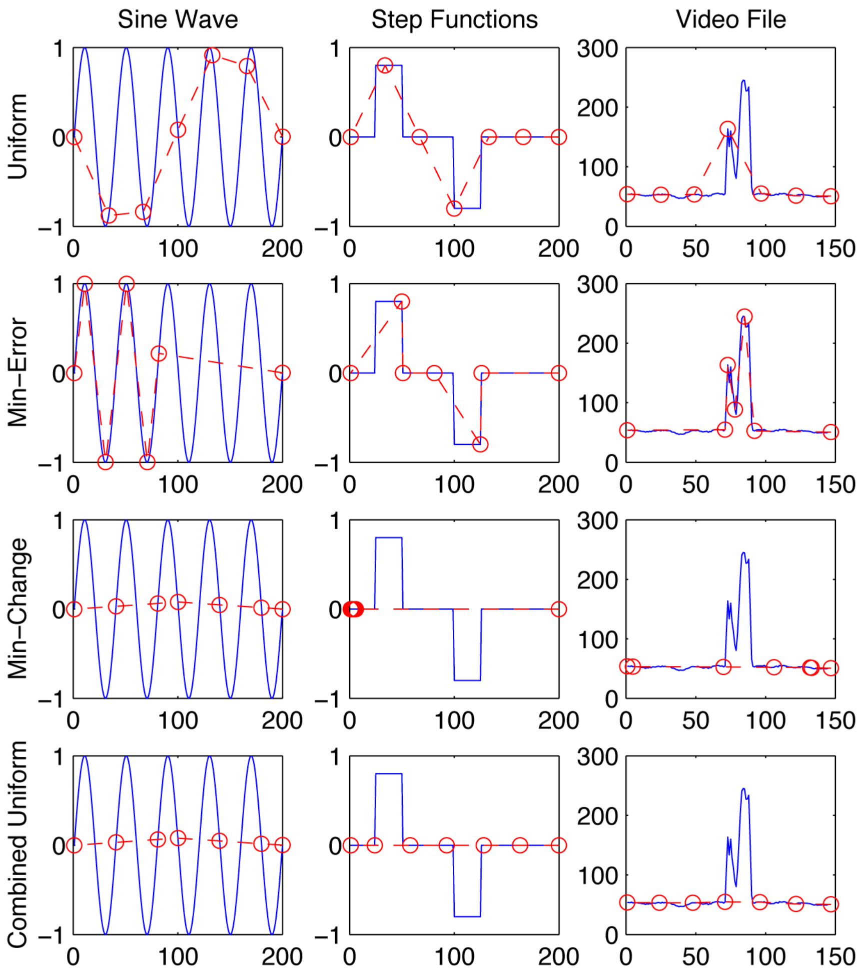

Figure 5.3: Plots showing the uniform and non-uniform samplings v (circles) of the time-lapse sampler on a variety of test signals. For each test, the number of output samples, M, is 7. First, the uniform sampling always selects a constant interval. The min-error metric minimizes the total sum-of-squared approximation error for the entire signal. The min-change metric minimizes the total squared change between samples, sometimes resulting in significant clustering in areas of no change. Finally, the combined uniform metric enhances the min-change metric by pulling the solution toward uniformity.

0 and 1 to be of relative scale to $\Upsilon(i,j)$. This and all of the computational time-lapse video metrics are compared in 1D in Figure 5.3.

5.2.5 Efficient Calculation

Given the min-error, min-change, and combined uniform metrics, the user has a wide range of time-lapse characteristics to select from. However, these approaches are designed for optimality, not for speed. To accelerate the process, the efficiency of calculation is now discussed.

Perez and Vidal noted the complexity of the original min-error 1D line approximation solution (Section 5.2.1) can be improved to $\Theta(M \cdot N^2)$ by incrementally solving $\Delta(i,j)$ for sequential values of j when i is fixed. Equation 5.3 can be expanded as follows:

$$\begin{aligned}\Delta(i,j) = (j-i)a_{ij}^2 + 2a_{ij}b_{ij}\sum_{k=i}^{j} k - 2a_{ij}\sum_{k=i}^{j} s_k \\ +b_{ij}^2\sum_{k=i}^{j} k^2 + 2\sum_{k=i}^{j} s_k^2 - 2b_{ij}\sum_{k=i}^{j} ks_k.\end{aligned} \tag{5.17}$$

In this form, each of the $\sum_{k=i}^{j}$ cumulative sums can be reused to find $\Delta(i,j+1)$ by recalculating $a_{i,j+1}$ and $b_{i,j+1}$ and adding the $k = j+1$ values to the sums. This requires storage of 5 double precision variables between iterations.

My additional optimization only requires the storage of 2 double precision variables and an 8-bit luminance while significantly reducing the number of calculations. Doing so involves subtracting s_i from all s and subtracting i from all indices, effectively moving the start of all line segments to $(0,0)$. This eliminates a, which is always 0, but requires storage of s_i. The solution of the slope b_{ij} remains unchanged, as now shown:

$$\Delta(i,j) = 2b_{ij}\sum_{k=0}^{j-i}\left(k(s_{k+i}-s_i)\right) + \sum_{k=0}^{j-i}(s_{k+i}-s_i)^2 + b_{ij}^2\sum_{k=0}^{j-i} k^2, \tag{5.18}$$

$$\sum_{k=0}^{j-i} k^2 = \frac{(j-i) + 3(j-i)^2 + 2(j-i)^3}{6}. \tag{5.19}$$

Although this is unnecessary for a single 1D signal, it is beneficial when processing millions of video pixels.

Even with this optimization, the slowest part of the solution method remains calculating $\Delta(i, j)$ for each segment. To accelerate the process, I introduce a final constraint: no two sequential samples in v may be more than q frames apart from each other. In the array containing all $\Delta(i, j)$ for every i, j pair, this is equivalent to solving a band of $\Delta(i, j)$ values q wide (only $i < j$ is solved, as time must move forward). This drops the time complexity to $\Theta(M \cdot N \cdot q)$.

Enforcing a maximum sampling interval q no longer guarantees an optimal solution because the system cannot generate time-lapse results with any sampling interval larger than q. Care must be taken when choosing q because it impacts both the minimum and maximum segment lengths. For example, if two adjacent samples are chosen in v, then a future segment must become longer to cover the resulting gap. I typically chose a q 3 to 4 times larger than T_d to account for this.

The values of $\Delta(i, j)$ are pre-calculated and cached for later use. This allows the user to interactively choose the "shape" variables α, β, γ, and λ without re-solving the underlying metric. Again, these variables are purely for giving the user fine controls to tweak the sampling.

A complete time-lapse sampling system has now been presented. However, many video frames are not included in the final video, although they may contain useful information. Use of the min-error metric implies as much motion and change should be included in the output as possible, but the user-specified duration may not be sufficient to capture all the motion. Thus, information from the skipped frames should be propagated to the included frames. Similarly, use of the min-change metric implies that motion should be minimized in the output, thus any remaining post-sampling motion that could not be sampled around could be removed and replaced with the background from temporally adjacent frames. Both of these scenarios imply a method, inspired by computational photography frame combination, to create optimal exposures using information from the skipped frames.

5.3 Virtual Shutter

The virtual shutter computationally combines nearby frames and enables effects not possible with traditional optical cameras. Each output exposure is a combination of a sliding window of video-rate exposures from the input signal s. In these new exposures frames are combined to decrease aliasing, accentuate or remove changing scene elements, and highlight motion paths.

5.3.1 Virtual Shutter Features

The primary reason to use a post-process to create exposures is it allows the system to arbitrarily choose any start and end time for each exposure. Thus, it is no longer constrained by the sampling interval. In a traditional camera, an exposure must end before the next exposure begins, fixing the maximum exposure time to the sampling interval. However, in a computational video post-process with access to all frames, exposures can be created longer than the sampling interval allows. Each input video-rate frame acts as a short, discrete time-step that is, in turn, integrated into much longer time-lapse exposures.

Furthermore, with a post-process, exposures can be made whose lengths adapt to the non-uniform sampling from Section 5.2. Since the sampling indices of the time-lapse video result from the sampling post-process, there is no way to know the corresponding exposure lengths at capture time. Thus, it is necessary to use a post-process to adapt exposures to the sampling. Note that the virtual shutter may also be used independently of Section 5.2, with uniform sampling.

Each exposure interval is a function of the input sampling v of s. The user specifies how many sampling intervals the output should integrate over: Ψ. Values of $\Psi > 1$ imply overlaps between exposures, causing the same input frame to be integrated into multiple exposures. Thus, an output frame t integrates the frames in s within the interval $v_{(t-\Psi)}$ to v_t. This is a causal, non-symmetric process, as frames in the future are not included in the exposure.

Creating exposures as a post-process frees the system from the linear integration of CCDs or film, similar to the low-light enhancement in Chapter 4. Instead, the user has the flexibility to integrate each discrete time step (frame in s) with any weighting or non-linear function,

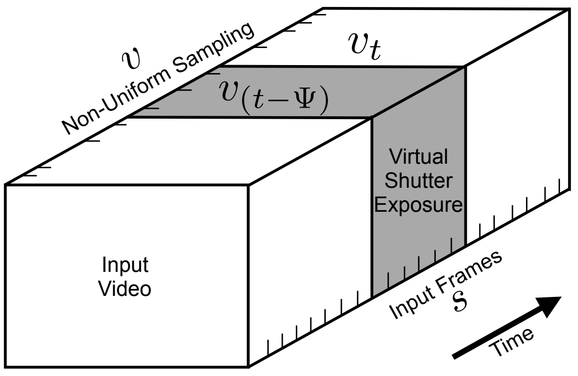

Figure 5.4: Illustration of the virtual shutter sliding a window of frames through the video that are combined into synthetic exposures. Each exposure ends at a sample v and extends Ψ indices in v back in time, allowing for adaptively-sized exposure times.

such as weighting by chronological order. Because the individual exposures s are short (the input exposures were 1-10 ms, although 33 ms exposures were used at night) they are assumed to not be saturated, temporally aliased, or contain motion blur, all common problems with long or multiple exposures. Long and multiple exposures are digitally simulated to avoid these problems, thus handling a wider variety of illumination capture conditions.

The virtual shutter can be considered a general purpose spatio-temporal filter based on established computational photography concepts. Previous work in computational photography has focused on processing a series of images as input and outputting a single frame. Alternatively, the virtual shutter slides a window of input frames through the video, creating an exposure for each v frame (Figure 5.4). Each individual exposure created by the virtual shutter is related to the image combination work in "Image Stacks" (Cohen et al., 2003). However, I extend this work with an adaptively-sized temporal window and by creating exposures that better illustrate events as part of a complete time-lapse video, thus bringing it into the computational video domain.

5.3.2 Virtual Shutter Filters

The computational time-lapse video system currently supports a variety of virtual shutter effects targeted at the specific needs of time-lapse video sources:

- **Maximum Virtual Shutter:** The maximum virtual shutter simulates film's non-linearity due to saturation. Nighttime time-lapse videos of car headlights are effective because the headlights saturate the film, creating bright, uniform streaks. Here, a similar effect is created by choosing the maximum value at each pixel, thus accentuating the prominence of bright foreground objects against dark backgrounds:

$$VS_t^{xy} = MAX_{f=v_{(t-\Psi)}}^{v_t} s_f^{xy}. \tag{5.20}$$

- **Minimum Virtual Shutter:** The minimum virtual shutter alternatively chooses the darkest pixels within the exposure window. Although no photographic analog exists, this frequently removes bright foreground phenomena, such as fog or smoke:

$$VS_t^{xy} = MIN_{f=v_{(t-\Psi)}}^{v_t} s_f^{xy}. \tag{5.21}$$

- **Median Virtual Shutter:** The median virtual shutter creates an exposure of the most representative pixels, even if those pixels never appeared simultaneously. This is another effect unachievable with a real camera. It is best used as a complement to the min-change non-uniform sampling (Section 5.2.3) to remove any residual "popping" that could not be sampled around. This is actually an extension to the PBackground filter discussed in 3.2.4, except that it runs the median over only a subset of all video frames:

$$VS_t^{xy} = MEDIAN_{f=v_{(t-\Psi)}}^{v_t} s_f^{xy}. \tag{5.22}$$

- **Extended Exposure Virtual Shutter:** The extended exposure virtual shutter simulates holding the shutter open between exposures. This is accomplished with low-pass filtering to mimic the response of a camera along with normalization to mimic aperture adjustments to avoid saturation:

$$VS_t^{xy} = \frac{1}{v_t - v_{(t-\Psi)} + 1} \sum_{f=v_{(t-\Psi)}}^{v_t} s_f^{xy}. \tag{5.23}$$

This represents a discrete approximation of the resulting motion blur, which can be further enhanced by the use of motion estimation techniques, such as in the work of Brostow and Essa (2001). The progression of time can be depicted by placing more weight on the most recent samples, indicating chronology:

$$VS_t^{xy} = \frac{1}{\sum \omega(t)} \sum_{f=v_{(t-\Psi)}}^{v_t} \omega(f) s_f^{xy}. \tag{5.24}$$

I chose the $\omega(f)$ weighting to be an adaptive exponential falloff curve whose tail length matches the exposure window. Here, I set μ to 30 and allow the user to configure the curve's falloff with ζ:

$$\omega(f) = \zeta^{\kappa}, \quad \kappa = \mu \cdot \frac{v_t - f}{v_t - v_{(t-\Psi)}}, \quad 0 < \zeta \leq 1, \quad 1 < \mu. \tag{5.25}$$

- **Motion Tails Virtual Shutter** Another virtual shutter effect uses compositing to simulate dense stroboscopic motion studies that illustrate paths of motion. Using just the extended exposure virtual shutter on fast moving objects causes them to fade into the background, as the background is seen far more often than the motion. To overcome this issue, the foreground is separated from the background then composited back into the background image. This borrows concepts from the "Image Stacks" (Cohen et al., 2003) matte filter that used the median image as a background plate and image subtraction to extract foreground elements. These images were then composited using the "over" operation (Porter and Duff, 1984) into the final image. Because the compositing was performed in temporal order, more recent motions occluded older motions.

  I extend this idea by considering it over a sliding window of frames in the video that contribute to the exposure. If $\Psi > 1$, the resulting motion tails will overlap. To better show the direction of motion in the video output, the "over" α blending term is adjusted using $\omega(f)$ (Equation 5.25). This fades the older frames into the background and makes the newer frames more opaque. My experience is that this $\omega(f)$ is particularly important for visualizing fast moving actions with long tails.

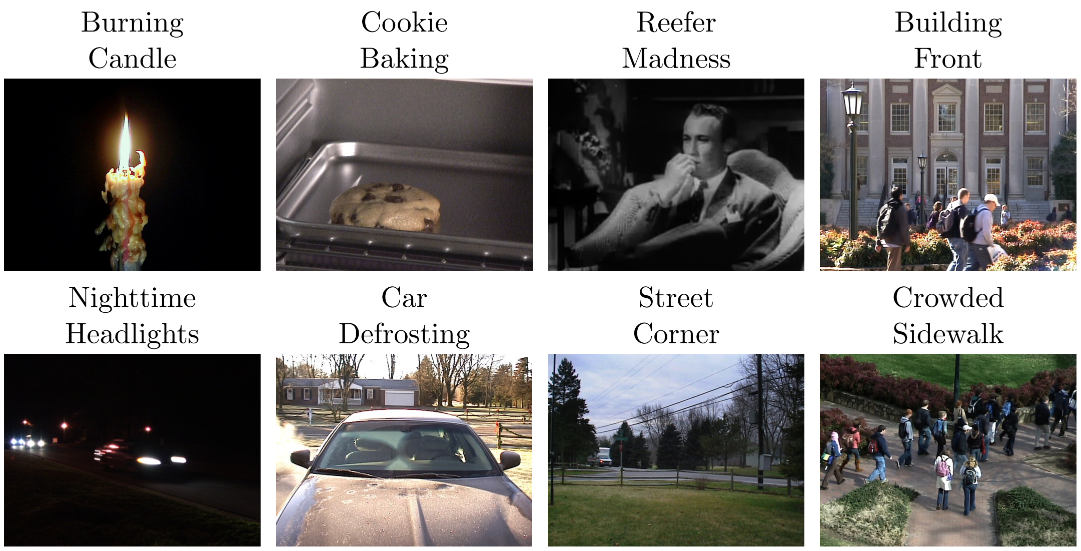

Figure 5.5: Unprocessed input frames from each of the example videos discussed in Section 5.4. From left to right: Top: a 13 second time-lapse of a candle burning with wax dripping, an 8 second sequence of a cookie baking, a 20 second summarization of the film "Reefer Madness", a 5 second time-lapse of people walking, Bottom: a 6 second time-lapse of car headlights, a 10 second sequence of a car defrosting, a 7 second time-lapse of a street corner, and a 15 second time-lapse of a busy sidewalk during a class change.

5.4 Results

Having discussed the concepts of non-uniform sampling and the virtual shutter, implementation of those methods as a complete system is now presented along with the results of eight example videos. The computational time-lapse system is implemented as a series of applications that mirror Figure 5.2's structure. The first application generates the $\Delta(i, j)$ min-error (Section 5.2.2) and $\delta(i, j)$ min-change (Section 5.2.3) metrics from the source video s (the $\Upsilon(i, j)$ uniformity metric from Section 5.2.4 is calculated on-the-fly). I also have the ability to selectively choose sub-regions within the video to analyze to "focus" the sampler on particular visual elements. Also, the errors can be calculated at lower resolutions without noticeable degradation. These errors are then used by the non-uniform sampler application to perform dynamic programming optimization. Here, the user may specify the "shape" variables along with a visualization of their impact. The resulting sampling v is fed into the virtual shutter application where the exposure effects of Section 5.3.2 are exported as a video.

For the remainder of this section, I describe the processing behind each of a number of test sequences (Figure 5.5) and their results. The system settings and runtimes are detailed in Table 5.1 while the non-uniform samplings are shown in Figure 5.8. All videos are 720x480, except for "Reefer Madness" which is 320x240. The output durations were chosen to create compelling time-lapse results that were too short to be handled properly by traditional time-lapse methods (*i.e.*, they exhibited undesirable temporal aliasing artifacts).

Non-Uniform Sampling Results:

- **Burning Candle** (*58 Minutes* $\rightarrow$ *13 Seconds*)

  This uniform 262x time-lapse of a candle burning misses the events of wax dripping and the candle breaking. To make a video with more of the overall duration showing the dripping wax I primarily used the min-error metric for the candle's wax (masking the flame, which was not of interest), thus making the sampler approximate the wax's changes. A small amount of uniformity was used to prevent the wax movement at the end from monopolizing the samples.

- **Cookie Baking** (*9 Minutes* $\rightarrow$ *8 Seconds*)

  This uniform 65x time-lapse of a cookie baking is not temporally aliased, but the camera was slightly unsteady due to nearby vibration from footsteps. The stabilized output sampling used the min-change metric along with $\Upsilon(i,j)$ to identify similar frame pairs with a mostly-uniform sampling. Thus, the frames where the camera moved were avoided.

- **"Reefer Madness"** (*68 Minutes* $\rightarrow$ *20 Seconds*)

  For this 1936 film, I used the non-uniform sampler with the min-change metric. Playing the video uniformly >200x results in incomprehensible scene flashes. The non-uniform sampler chose a few frames from longer scenes that appeared similar, thus keeping the time-lapse within each scene long enough to allow identification. Although many scenes are skipped, the scenes that are included are recognizable. Note, this summary could

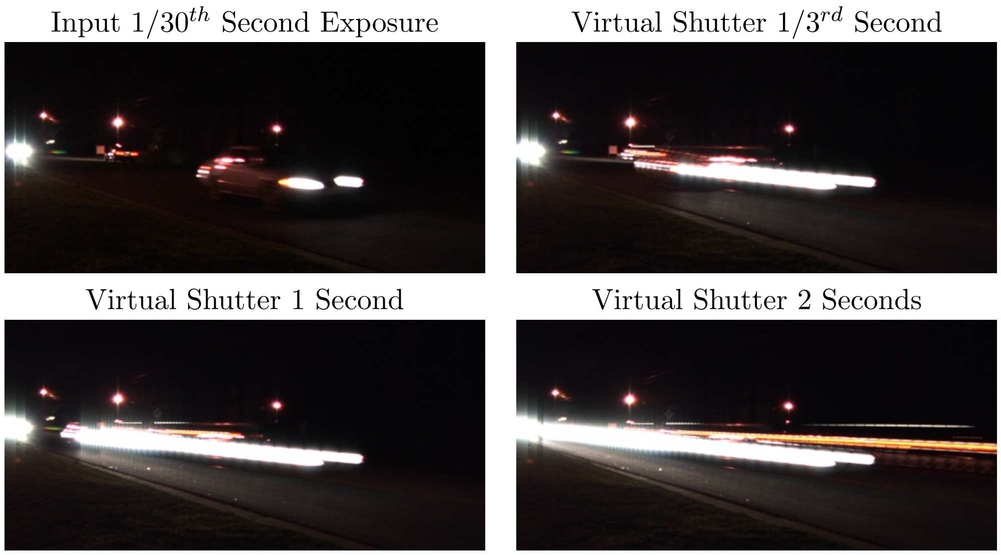
Input $1/30^{th}$ Second Exposure
Virtual Shutter $1/3^{rd}$ Second
Virtual Shutter 1 Second
Virtual Shutter 2 Seconds

Figure 5.6: An input frame from the nighttime car headlights sequence and three extended exposures using the maximum virtual shutter to mimic film saturation. Although the longest sampling interval for a 10x speedup is $1/3^{rd}$ of a second, the virtual shutter simulates longer exposures from video-rate footage.

be used as an index into the full-length video as long as the v is preserved to provide reverse lookup from the time-lapse to the source.

Virtual Shutter Results:

- **Building Front** (*34 Seconds* $\rightarrow$ *5 Seconds*)

  This short video of walking students shows the benefits of motion tails to convey motion, even with 7x uniform sampling. Low-pass filtering makes the students nearly disappear but motion tails of either $\Psi = 4$ or 8 make the motion contiguous and fluid (Figure 5.9).

- **Nighttime Car Headlights** (*60 Seconds* $\rightarrow$ *6 Seconds*)

  A popular time-lapse effect seen on Internet video sites captures car headlights at night. Because the headlights saturate the sensor, long exposures appear as uniform streaks. The streak lengths are limited by the sampling interval, because the shutter must close before the next exposure. Here, the maximum virtual shutter is used to create exposures 3 and 6 times times longer than the maximum sampling interval (Figure 5.6).

- **Car Defrosting** (*29 Minutes → 10 Seconds*)

  This time-lapse of a car's windshield defrosting allows for several alternate interpretations: minimizing the aliased motion and also depicting all the motion. To minimize aliasing, both the median virtual shutter with a wide Ψ neighborhood and low-pass filtering are shown (Figure 5.7). To depict all motion, motion tails were used to show the cars and other street activity that were badly aliased.

Combined Results:

- **Street Corner** (*12 Minutes → 7 Seconds*)

  This 120x uniform time-lapse (Figure 5.1) depicts a street corner with sporadic traffic that contains frames where cars pop in and out. Using the min-error metric, I generated a non-uniform sampling that slows down when cars drive by. Aliasing was then further reduced with motion tails. Alternatively, I solved with the min-change metric (with a small λ uniformity term), creating a sampling with smooth cloud motion and no cars. Because some blowing leaves remained, a short median virtual shutter was used to remove them.

- **Crowded Sidewalk** (*17 Minutes → 15 Seconds*)

  A uniform 70x time-lapse video shows a sidewalk before, during, and after a class change that exhibits severe "popping" aliasing artifacts. Initially, only a few students walk by, then traffic picks up, then dies down again. I improved the video with a min-error non-uniform sampling (and a tiny λ uniformity mix), which better approximated the motion by devoting more samples to frames with the most activity, in this case during the class change (resulting in a uniformly busy sidewalk). Motion tails were also added to provide visual cues of the students' paths (Figure 5.9).

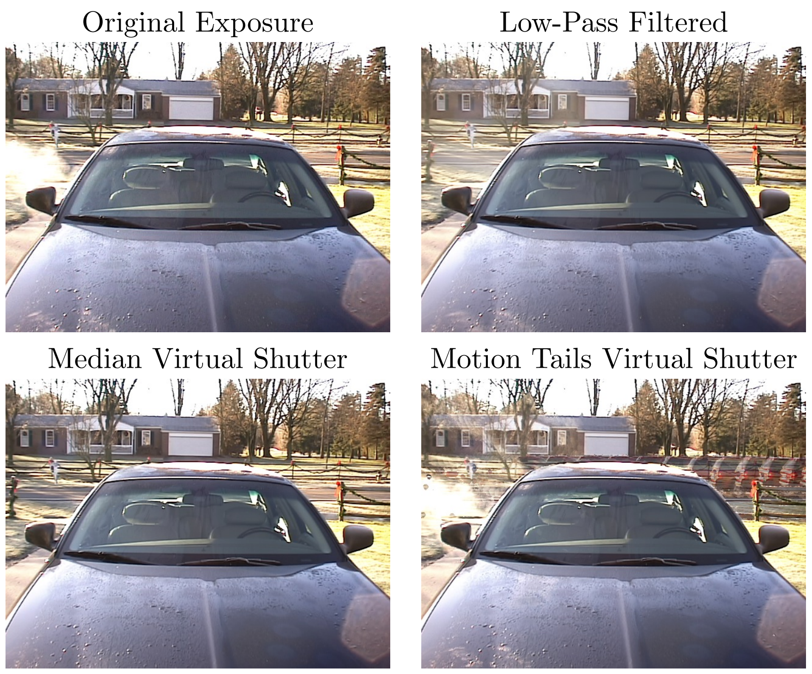

Figure 5.7: An input frame from the car defrosting sequence and three different output exposures from the virtual shutter with varying characteristics. The low-pass results washout regions where the car's exhaust appeared, while the median virtual shutter result removes the exhaust. The motion tails output shows the combined exhaust of multiple frames plus evidence of an SUV driving by.

5.5 Summary

I have presented methods for creating computational time-lapse videos which provide superior sampling characteristics over traditional time-lapse methods. Based on user-specified characteristics, time-lapse videos were generated with non-linear spatio-temporal filtering and non-uniform sampling, both not possible in traditional cameras. The non-uniform sampling optimally chooses the constituent frames of the video using knowledge of the entire video's samples and motion. The virtual shutter then combines neighborhoods of multiple video-rate exposures into longer exposures with reduced aliasing artifacts. These techniques were then used, both independently and together, to process a variety of typical and novel time-lapse videos.

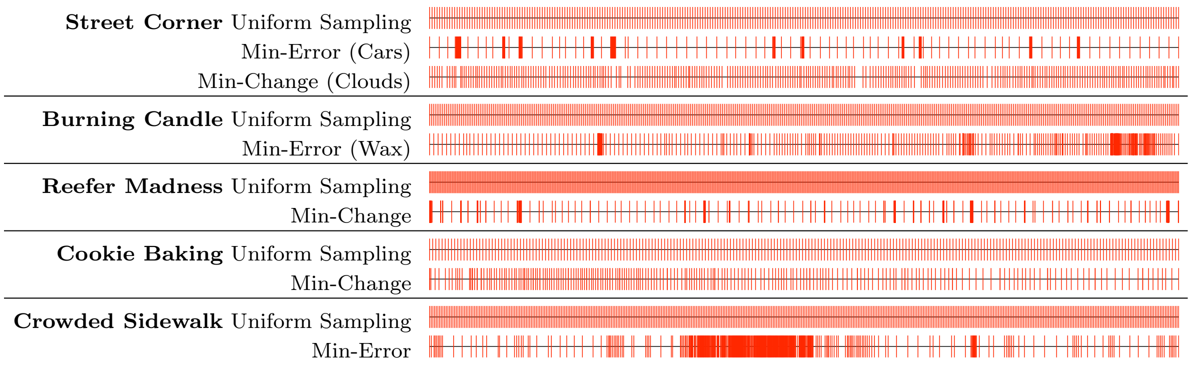

Figure 5.8: Visualization of the sampling results, where each vertical line represents a sampled frame in v. Samplings using the min-error metric choose the majority of their frames from within periods of change and motion to best approximate the video. Alternatively, the samplings using the min-change metric avoid frames dissimilar to other frames.

	Duration (Frames)			**Non-Uniform Error Metric**				**Uniformity**	**Virtual Shutter**			**Run Time**
Title	**Input**	**Output**	q	**Sampling**	α	**log** β	**log** γ	λ	**Type**	Ψ	ζ	**hours:minutes**
Burning Candle	104849	400	600	Min-Error	1.09	-6.11	4.87	.05	-	-	-	7:20
Cookie Baking	15757	240	200	Min-Change	1.00	-5.24	0	.04	-	-	-	:05
"Reefer Madness"	40975	600	500	Min-Change	1.00	-7.08	5.89	-	-	-	-	:15
Building Front	1025	150	-	Uniform	-	-	-	-	Tails	4, 8	.96	:15
Car Headlights	3096	300	-	Uniform	-	-	-	-	Max.	6	-	:10
Car Defrosting	52544	300	-	Uniform	-	-	-	-	Median	10	-	:30
Car Defrosting	52544	300	-	Uniform	-	-	-	-	Tails	4	.93	:45
Street Corner - Cars	25483	210	500	Min-Error	1.00	0	0	-	Tails	3	.95	5:15
Street Corner - Clouds	25483	210	500	Min-Change	.65	-7.37	5.38	.15	Median	2	-	:45
Crowded Sidewalk	31128	450	500	Min-Error	1.00	-9.49	0	.01	Tails	4	.92	3:15

Table 5.1: Parameters used to generate the video results. These were selected manually by the user. Cells marked as '-' indicate variables not used in the result. A value in the λ category indicates the combined uniformity metric $\Upsilon(i,j)$ was used along with the listed error metric. Runtimes reflect all processing (error calculation, sampling, and virtual shutter) on a single core of a 2 GHz Intel Core Duo. Most of the running time is spent on error metric calculations, which are a function of the metric, duration, q, and resolution.

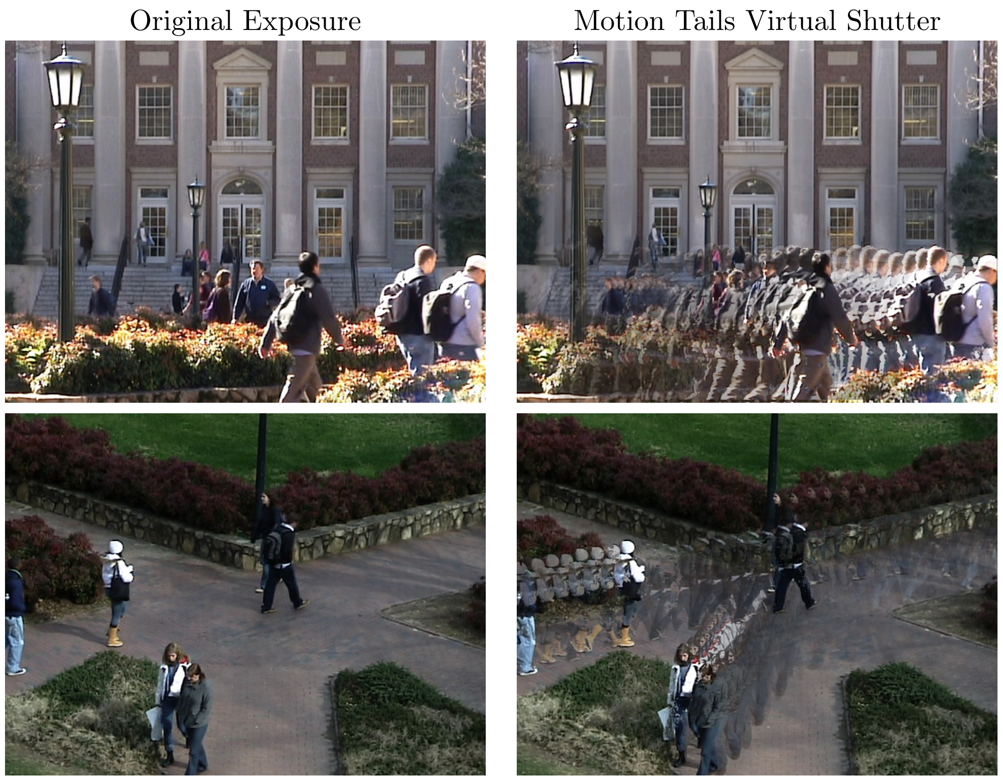

Figure 5.9: Two examples of using motion tails to depict dense motion paths between sampled time-lapse frames. The building front result (above) uses uniform sampling, while the crowded sidewalk (below) is non-uniformly sampled.

CHAPTER 6

CONCLUSIONS

In the preceding chapters, both the foundation and applications of computational video were explored to address the claims made in the thesis statement:

> Computational video enables a new class of processing tools for enhancing and improving video capture quality by leveraging information found across many frames.

Chapter 3 addressed the underlying issues of how computational video can be conceptualized and efficiently processed. Spatio-temporal volumes, and their sheared variants, were presented to allow information from adjacent pixels and frames to be readily accessed. Efficiency was addressed through the Proscenium spatio-temporal video editing framework which showed how bi-directional filter graphs can reduce memory and computational requirements. To demonstrate how volumes could be interactively used for computational video, a prototype video editing and enhancement system was also presented.

Having proven its viability, I then dedicated the remainder of the dissertation to addressing the claim that computational video methods can enhance and improve captured video by using information found throughout the video.

First, the quality of low-light video was enhanced to resemble well-exposed video. By using additional information from the local neighborhood of poorly-exposed samples, their true underlying value was determined. This process involved extending the exposure time of those samples by integrating many temporally and spatially-adjacent samples. How much integration was necessary was determined by tone mapping, while the integration itself was determined through adaptive non-linear filtering. This resulted in spatially-varying exposure times for each pixel, not possible in a traditional imager. This concept was carried further by introducing neighborhoods of information that spanned multiple spectra by using non-visible IR video. Dual bilateral filtering demonstrated that multispectral enhancement was possible without introducing non-visible elements into the result.

As an alternate type of computational video enhancement, the resampling of videos was also discussed. In addition to the spatial resampling of videos (seen as shearing of spatio-temporal volumes), resampling can change the temporal properties of video. Time-lapse videos, being significantly shorter than video-rate footage of the same capture duration, exhibit an extreme class of resampling. This was performed by learning about the video in its entirety and finding the optimal temporal resampling. Finally, spatio-temporal filtering was again considered to achieve non-linear multi-frame filtering effects.

In conclusion, computational video techniques have been shown to be viable and capable of a wide range of enhancements including the improvement of captured video quality. These improvements included effects not possible in-camera, including optimal resamplings, overlapping exposures, object removal, non-linear filtering, and expanded dynamic range. Thus, computational video enables a wide range of desirable video characteristics that increase the overall quality of video.

6.1 Directions for Future Research

Computational video has recently established itself as an active research area in computer graphics as witnessed by the large number of works involving computational video mentioned in the Previous Work. As a result, there has been a great deal of concurrent research with the work described in this dissertation. Thus, there is continuing interest in the general area of finding better ways to enhance video, and thus there will be future research in the field of computational video. I now discuss such future research in the areas of computational video's technical underpinnings, its existing applications, and new computational video applications.

- One of the core problems of computational video is that it has very high memory requirements to enable random video access. This was overcome in the later chapters by processing in a FIFO queue of frames where only a reasonable window (often a few seconds) of a much longer clip was decompressed and stored in RAM at any given time. Finding a compression and decompression scheme that allows much finer random access to pixels (as opposed to decompressing whole frames or large blocks of spatial pixels) would then require only the compressed video to be stored in memory.

- For real-time video processing, the ideal goal is to process video at or faster than 30 fps, as opposed to the current offline model, allowing both the LDR enhancement and time-lapse algorithms to be performed in-camera. Reducing noise and improving dynamic range prior to compression might also lead to reduced bit rates and support for compression schemes that incorporate foreground and background models. For time-lapse, the reduction of the number of frames stored would be drastically reduced. However, the current offline model is useful because it allows large non-causal kernels, non-destructive processing, and solutions optimal to the entire video to be found.

- Throughout this dissertation, the algorithm implementations were entirely uni-processor based. Thus, an obvious extension to this work would involve finding efficient multiprocessor implementations. The filter graph nature of Proscenium makes for easy separation of processing components, but algorithms such as ASTA are more difficult to distribute, as processing time is related to the overall brightness and noise level of each pixel, which changes over time. For this reason, a research group is already using the ASTA algorithm as an example of modern adaptive image processing algorithms to study multiprocessor scheduling (Block et al., 2007). Another direction is to implement such algorithms on the GPU (Graphics Processing Unit) which can compute many per-pixel filters in parallel.

- Many of the algorithms described here involved shearing to remove camera motion, which in turn requires stabilization. Making this more general (*i.e.*, allowing spatially varying transforms more complex than the current planar projective transforms) would allow improved processing of transforming 3D shapes. However, there are difficulties involved due to tracking and stabilization which are often confounded by poor imaging conditions. In the case of low-light enhancement, the tracking requires cleaner images (less noise, sharper edges, etc.), but the algorithms doing that enhancement require better tracking. An iterative back-and-forth approach or possibly a combined method may be an improved way to tackle this problem.

- There still remains much work in developing techniques for tone mapping LDR videos to screen dynamic ranges or even HDR dynamic ranges. The approach presented in Section 4.2.2 is a good starting place, as it tone maps while masking high-frequency noise in dark areas. However, the tone mapping of videos with "peaky" histograms (those with an uneven distribution of dynamic range) are still very difficult without washing out the bright elements at the expense of amplifying dark areas.

- There are also areas for further development in computational time-lapse video. The min-change metric $\delta(i, j)$ could be altered to compare neighborhoods of frames as opposed to single frames. As in "Video Textures" (Schödl et al., 2000), velocity would be preserved in addition to visual similarity. Also, repetitive motions could be cleverly resampled to give the illusion they were occurring at video-rate speeds in the time-lapse result (thus, a desirable form of aliasing exploiting beat frequencies). Although camera motion between frames was not considered because time-lapse cameras are typically fixed, an error metric could be constructed to optimize for a constant camera velocity. A shortcoming of the time-lapse solver's dynamic programming approach is that errors must be defined pairwise between i and j. Because objectives such as maximizing the smoothness of the derivative of intervals in v are not possible, other solution techniques may be worth considering.

- Computational color processing is another area that warrants consideration. Chrominance could be enhanced robustly using multispectral edge detection or multiframe blending. Low-level color handling, such as for multi-frame color demosaicing, is already an interesting area of ongoing and future work (Bennett et al., 2006).

- Finally, many more applications exist that would be benefitted by combining information imaged throughout a video. These could range from generating HDR imagery from well-exposed 8-bit video captured with automatic gain control to automated wire removal using backgrounds propagated from surrounding samples. The application domain of video restoration of missing or corrupted samples offers a problem also well suited to this type of enhancement.

6.2 Closing Remarks

I have presented a wide variety of uses of computational video that have applicability to both professional and amateur users. Arguably, computational video has the most potential to help less-experienced users, as it can help bring a professional look to less than optimally captured video. Therefore, as computational video algorithms receive more research and solve new and exciting problems, it is a reasonable hope that these techniques will make their way to a community outside of pure research. Empowering the amateur filmmaker to create amazing video is a lofty but attainable goal that we all should aspire to.

APPENDIX A

PROSCENIUM PFILTER SPECIFICATIONS

This appendix provides the full specifications of how PFilters are implemented for use in Proscenium filter graphs (Section 3.2.3) along with expanded descriptions of the example PFilters of Section 3.2.4.

A.1 PFilter Specification

Bi-directional data flow through PFilters is enforced by having each fulfill the requirements of a basic interface shown in Figure A.1. They must also be able to describe their size in width, height, and number of frames. They must internally know if they are a discrete or continuous filter. Most importantly, they must be able to handle reading and writing of individual pixels, which is done by connecting the input filters and defining the filter's functions. If not overridden, the defaults pass along the values obtained from their input PFilters.

The `pixelWidth` and `pixelHeight` are the measurements in pixels of the viewable area of each frame. Proscenium assumes that these are constant across all frames of a sequence, so smaller images must be padded with *empty* pixels. The number of frames, `numFrames`, is a discrete quantity that makes the assumption that the frame rate is constant, but variable rates (i.e. resampling) can be simulated through intermediary PFilters.

Pixel values are queried with the discrete `getActualPixel(int x,y,t)` function or the continuous `getPixel(float x,y,t)`. These functions only know the requested pixel coordinate and return the color at that pixel; nothing else. Unless these functions are overridden they pass along the query to the private functions `runDiscreteFilter(int x,y,t)` and `runContinuousFilter(float x,y,t)` after providing bounds checking. These are the most commonly overridden functions when creating a new filter. If a PFilter is defined as discrete (by Boolean parameter `isDiscrete`), a call to `getPixel()` defaults to trilinear interpolation.

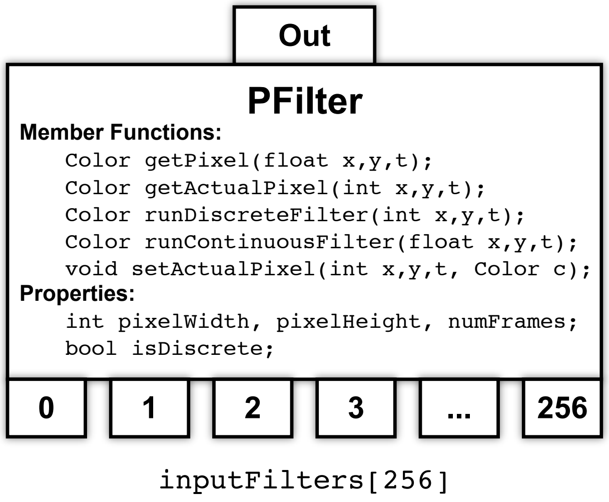
Out
PFilter
Member Functions:
Color getPixel(float x,y,t);
Color getActualPixel(int x,y,t);
Color runDiscreteFilter(int x,y,t);
Color runContinuousFilter(float x,y,t);
void setActualPixel(int x,y,t, Color c);
Properties:
int pixelWidth, pixelHeight, numFrames;
bool isDiscrete;
0
1
2
3
...
256
inputFilters[256]

Figure A.1: Each PFilter object supports multiple input ports and a single output. By combining the results from different input filters, video sequences can be modulated and combined. The methods shown comprise the standard interface for all PFilters.

Inside each PFilter is an array called `inputFilters[]` which associates a PFilter pointer with an integer index. The filter graph is constructed by setting these values. By convention, the PFilter associated with the number 0 is its primary data path, carrying the video output of the filters below, while the others are designed for specific purposes depending on the filter type. For data flow reasons PFilters only know what PFilters are their inputs and do not know their outputs. Many PFilters may share the output of a PFilter, but only one PFilter can be assigned to each input of a PFilter.

A.2 Example PFilter Implementations

Now, the full details of the PFilters described in Section 3.2.4 are given to allow the reader to implement PFilters similar to those used in Proscenium.

A.2.1 Simple Color Correction

To demonstrate a simple discrete filter, PCorrect adjusts the blue values of its input pixel and sends it to the output with the red, green, and α unchanged. The `isDiscrete` parameter is set to true so that all processing will pass through the `runDiscreteFilter()` virtual function:

```
class PCorrect : public PFilter
{
public:
     PCorrect() { isDiscrete = true; blueAdjust = 20; }

     Color runDiscreteFilter(int x, int y, int t)
     {
          Color inColor = inputFilters[0]->getActualPixel(x,y,t);
          int tempBlue = inColor.B + blueAdjust;
          if(tempBlue > 255)tempBlue = 255;
          if(tempBlue < 0)tempBlue = 0;
          return Color::FromArgb(inColor.A,
               inColor.R,inColor.G,tempBlue);
     }
     int blueAdjust = 0;
};
```

This PFilter has no internal storage, so any user requested color changes must be propagated down to its input. In the process, the color correction must be run in reverse, so that when it progresses through the filter in the forward direction at a later time, it will be filtered, and the desired color will result again (assuming no bracketing at 0 and 255 occurs). This is easily accomplished by overriding another function:

```
void setActualPixel(int x, int y, int t, Color newColor)
{
     int tempBlue = newColor.B - blueAdjust;
     if(tempBlue > 255) tempBlue = 255;
     if(tempBlue < 0) tempBlue = 0;
     Color alteredColor = Color::FromArgb(newColor.A, newColor.R,
          newColor.G, tempBlue);
     inputFilters[0]->setActualPixel(x, y, t, alteredColor);
}
```

A.2.2 Video Framing

With PFrame, the true extents of a movie can be hidden by manipulating the PFilter's `pixelWidth`, `pixelHeight`, and `numFrames` properties. This PFilter serves two purposes. First, it acts as a zoom. Videos are sampled during interaction, meaning that removing data on the edges allows greater detail to be shown for the remaining portion. If the PFrame is left in place it will continue to act as a crop, but if it is removed all of the occluded data on the sides is still present.

This effect is achieved by substituting new values for `pixelWidth`, `pixelHeight`, and `numFrames`. The filter then becomes responsible for handling the fact that the origin may no longer be at (0,0,0). Finally, it must reverse this operation when a pixel is written so that the write it transmits to its input PFilter will be the original coordinate and not the offset coordinate.

To simplify this example, only the x-dimension will be framed, but the same technique applies to all three dimensions. `myOffset` is the horizontal distance from the origin that the frame begins, and `myWidth` is its horizontal size:

```
Color runDiscreteFilter(int x, int y, int t)
{
     if ((x >= 0) && (x < myWidth))
          return inputFilters[0] > getActualPixel(x + offX, y, t);
     else return Color::FromArgb(0,0,0,0); // Empty Pixel
}

void setActualPixel(int x, int y, int t, Color newColor)
{
     inputFilters[0]->setActualPixel(x + offX, y, t, newColor);
}
```

The use of the *empty* pixel is important, as it indicates the presence of an area outside the video. Bounds checking is explicitly done here because it is crucial to make sure occluded pixels cannot pass through.

A.2.3 Background Restoration

The PBackground filter returns a color based on a function of the matching (x, y) coordinates in every frame. Therefore, if this function were solved for every (x, y) pair in the video a new image of the background would result. Here, the median of each color channel is combined into a new color to estimate the background color. Also, those pixels with an α below some threshold are not considered. Alternatively, background filters are also possible that select the modes of the color component distributions. Iterative background filling approaches also exist, such as those by Bertalmio et al (2000).

The `runDiscreteFilter()` function disregards the t value, and takes the median of all the pixels with the matching (x, y) and returns that color. No `setActualPixel()` method is provided because this output is not directly related to any single frame's pixel.

```
Color runDiscreteFilter(int x, int y, int t)
{
     Color tempColor, finalColor;
     for(int i=0;i<inputFilters[0]->numFrames;i++)
     {
          tempColor = inputFilters[0]->getActualPixel(x,y,i);
          if(tempColor.A == 255)
               Sort the Red, Green, and Blue values into buckets
     }
     return Color::FromArgb(255, Median of Red, Blue, and Green);
}
```

I designed another background restoration filter called the *edge filling filter* which returns the temporally nearest opaque pixel that is spatially aligned to a transparent pixel. This technique is particularly useful for videos with translating or rotating cameras when building panoramas. This filter also keeps track of auxiliary information in order to speed up its execution. When a pixel is first accessed, it starts at $t = 0$ and tries every pixel at that (x, y) in temporal order until a sufficient α is found. The same process is repeated in reverse from the end of the movie. It then stores the frame numbers of these first and last frames. On subsequent requests to any t at that (x, y), a comparison determines if the location falls within this usable range. If so, that pixel's color is returned, else it uses the color of the nearest pre-determined temporal edge.

A.2.4 Caching

Performing the operations of a large number of interconnected PFilters can become very compute intensive. At some point caching becomes a convenient method to speed up operations. The PCache is a configurable cache that complements Proscenium's model of lazy evaluation. When a PCache is added into a filter graph it does not immediately cache all the pixels. Instead, it waits to be queried about a pixel before retrieving a value from its input PFilter. This accelerates subsequent accesses, but not the initial access.

A PCache exists as a block of RGBα or monochrome pixels exactly the same size as the PCache's input video source. Each frame is separately stored as a bitmap, and pointers to each frame are stored in an array for quick access. This allows frames to be deleted or inserted without regenerating the entire data structure.

The cache is initially filled with an arbitrary reserved value, which is referred to as the *unsolved* color. When the cache receives a `getActualPixel()` it checks its personal data structure, and upon finding the *unsolved* value at that coordinate, it queries its input. It takes the return of that function, updates its own data structure, and then returns it back up the filter graph. As a special case of a PCache, a *locked* PCache is one that will never perform a lookup even upon finding the unsolved color.

The PCache's behavior exhibits the following qualities. First, changes made through calls to `setActualPixel()` affect the data in the PCache, but are not propagated to its input terminals. Therefore, when a PCache receives an incoming `setActualPixel()` request, its internal data structure is modified so that subsequent calls to the PCache return the new value. Enforcing this creates a difficulty in alerting PCaches that they are invalidated by upstream changes to their inputs. Because all requests for pixel data flow in the opposite direction, there is no direct way for a PFilter to notify later PCaches that its data changed. They will either go on unaware of the inconsistency or rely on the application to invalidate a portion of the PCache back to its unsolved state, which is the solution used in this work.

A.2.5 Video Files

To process video, the filter graph must at some point contain the raw source video footage. The raw data is provided as yet another PFilter derived class. The source data class, called PMovie, is derived from the PCache class. It is essentially an unchanged PCache that defaults to being in the *locked* state, and therefore causes no input data lookups. It also adds member functions to load video and bitmap files into its frames.

For output to video files on disk, the mechanism is a PFilter called PAVIOut. It is placed on the end of the filter graph where the application would normally make its queries so that it uses the exact same data that the user sees on screen.

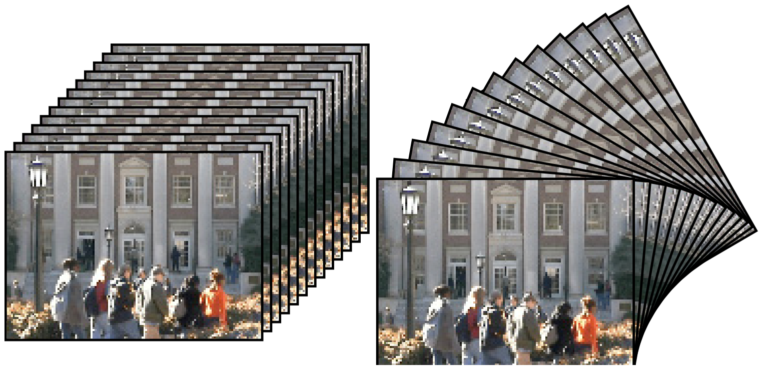

Figure A.2: Proscenium supports arbitrary projective shears of the volume. This facility enables objects moving within the field of view to be spatially stabilized across all frames.

A.2.6 Video Shearing

Proscenium's virtual shearing is implemented as a filter that can be added and, if desired, removed from the filter graph. The filter handles all bi-directional data flow in the filter graph for reading from and writing to the underlying, un-sheared volume. It is important to again note that shearing is performed as a filter to both eliminate the need to keep a second copy in memory and to allow edits to be propagated to the original video. The specifics of determining the underlying projective transform matrices for shearing are discussed in Section 3.2.2.

Once the transform matrices are known, they are loaded into the PShear filter. It then solves to find the coordinates of the four corners of each frame. The minima and maxima of these values determine the rectangular bounding box of the new volume. To enforce the convention that the upper-left hand corner of each frame is always at $(0, 0)$, a translation is used to reorient PShear's transformed coordinate space to the origin. The `pixelWidth` and `pixelHeight` of the PFilter are substituted with the sizes of the new extents.

PShear handles `getActualPixel()` requests by returning discrete pixel data without complex interpolation or blending. It receives a discrete pixel coordinate which it multiplies by the transform matrix for that frame, and then rounds to an integer value and returns that color or *empty* if it fell outside the volume. Because only one sample is taken, it results in nearest-neighbor-style interpolation. As before, more complex interpolations are possible.

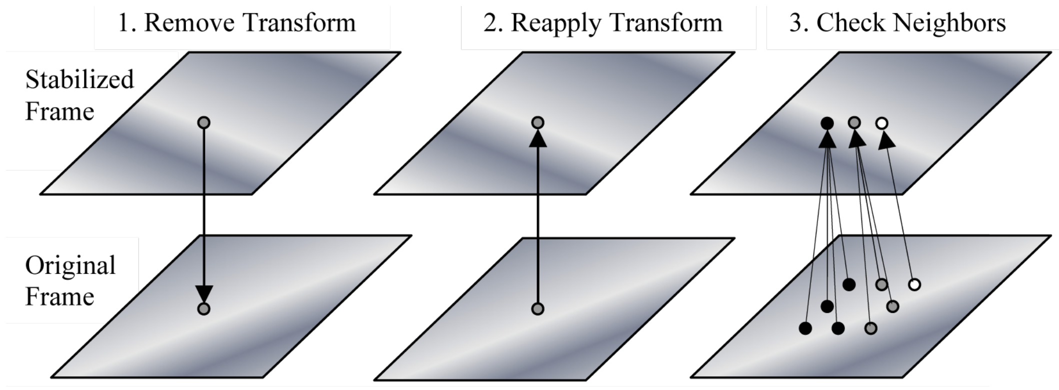

Figure A.3: Multistep process to ensure all pixels that round to a stabilized pixel are modified to reflect an edit.

`setActualPixel()` is implemented in a similar fashion, but it cannot operate using the same nearest-neighbor approach. A naïve implementation would take the target coordinate and multiply it by the transform matrix and then set the pixel color in the source material at that coordinate. However, this often does not result in the expected effect. This is because although each discrete coordinate input to `getActualPixel()` corresponds to just a single source pixel, `setActualPixel()` can have a one-to-many, one-to-one, or even a one-to-none relationship, which can occur when large minimizations result because of stabilizations.

If the user wants to change the color of an entire region, as opposed to a single pixel, a useful solution involves taking the corners of the bounding polygon in the transformed coordinate space and warping them back to the original coordinate space. Performing the fill operation in the new polygon in the source materials coordinate space will then be sure to change all pixels that fall in the transformed boundary.

However, this strategy only works for large areas, and a more precise methodology is needed for performing per-pixel edits in transformed space (Figure A.3). The following methodology implements such an approach:

> To handle the per-pixel case, the transformed pixel coordinate is backwards-mapped to the source material and rounded to the nearest pixel's "center". This slightly-adjusted coordinate is mapped back to the transformed space and rounded to the nearest pixel. If after these two phases of rounding and transformation the

pixel coordinate is different from the one that it began upon, it is labeled as a one-to-none relationship and no change is made.

If the original and modified coordinates in the transformed space are the same, then the change is made in the source material, as if it were a one-to-one mapping. Now, each of the adjacent pixels in the source material are mapped to the transformed space and rounded. If any of these map to the original transformed coordinate, they are also changed, signifying a one-to-many relationship. Every time a match is found, the search window is expanded by one pixel. It is assumed that all possible stabilizations will have all affected source pixels adjacent to each other (thus, no “bow-tie” projective transforms are allowed).

BIBLIOGRAPHY

Acosta-Serafini, P. M., Masaki, I., and Sodini, C. G. (2004). Predictive multiple sampling algorithm with overlapping integration intervals for linear wide dynamic range integrating image sensors. *IEEE Transactions on Intelligent Transportation Systems*, 5(1):33–41.

Agarwala, A., Dontcheva, M., Agrawala, M., Drucker, S., Colburn, A., Curless, B., Salesin, D. H., and Cohen, M. (2004). Interactive digital photomontage. *ACM Transactions on Graphics*, pages 294–302.

Agarwala, A., Zheng, K. C., Pal, C., Agrawala, M., Cohen, M., Curless, B., Salesin, D., and Szeliski, R. (2005). Panoramic video textures. *ACM Transactions on Graphics*, 24(3):821–827.

Alter, F., Matsushita, Y., and Tang, X. (2006). An intensity similarity measure in low-light conditions. In *Proceedings of the European Conference on Computer Vision*, pages 267–280.

Apple Inc. (2007). QuickTime 7.1.

Barash, D. (2002). A fundamental relationship between bilateral filtering, adaptive smoothing, and the nonlinear diffusion equation. *IEEE Transactions on Pattern Recognition and Machine Intelligence*, 24(6):844–847.

Bennett, E. P., Uyttendaele, M., Zitnick, C. L., Szeliski, R., and Kang, S. B. (2006). Video and image Bayesian demosaicing with a two color image prior. In *Proceedings of European Conference on Computer Vision*, volume 1, pages 508–521.

Bertalmio, M., Sapiro, G., Caselles, V., and Ballester, C. (2000). Image inpainting. In *Proceedings of ACM SIGGRAPH*, pages 417–424.

Bhat, K. S., Seitz, S. M., Hodgins, J. K., and Khosla, P. K. (2004). Flow-based video synthesis and editing. *ACM Transactions on Graphics*, 23(3):360–363.

Bidermann, W., Gamal, A. E., Ewedemi, S., Reyneri, J., Tian, H., Wile, D., and Yang, D. (2003). A .18 micrometer high dynamic range NTSC/PAL imaging system-on-chip with embedded DRAM frame buffer. In *Proceedings of the IEEE International Solid-State Circuits Conference*, pages 212–213.

Block, A., Anderson, J., and Devi, U. (2007). Task reweighting under global scheduling on multiprocessors. *Real-Time Systems*.

Bolles, R. C., Baker, H. H., and Marimont, D. H. (1987). Epipolar-plane image analysis: An approach to determining structure from motion. *International Journal of Computer Vision*, 1(1):7–55.

Boomgaard, R. and Weijer, J. (2002). On the equivalence of local-mode finding, robust estimation and mean shift analysis as used in early vision tasks. In *IEEE International Conference on Pattern Recognition*, pages 927–930.

Bouguet, J.-Y. (2000). Pyramidal implementation of the Lucas Kanade feature tracker. Technical report, Intel Corporation, Microprocessor Research Labs.

Bovik, A. C. (2000). *Handbook of Image and Video Processing.* Academic Press.

Boykov, Y., Veksler, O., and Zabih, R. (1999). Fast approximate energy minimization via graph cuts. In *Proceedings of the International Conference on Computer Vision*, pages 377–384.

Braun, M. (1995). *Picturing Time: The Work of Etienne-Jules Marey.* University of Chicago Press.

Brostow, G. J. and Essa, I. (2001). Image-based motion blur for stop motion animation. In *SIGGRAPH 2001, Computer Graphics Proceedings*, pages 561–566.

Buades, A., Coll, B., and Morel, J. M. (2005). Denoising image sequences does not require motion estimation. In *IEEE Conference on Advanced Video and Signal Based Surveillance*, pages 70–74.

Buehler, C., Bosse, M., and McMillan, L. (2001). Non-metric image-based rendering for video stabilization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 609–614.

Cheung, V., Frey, B. J., and Jojic, N. (2005). Video epitomes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 42–49.

Choudhury, P. and Tumblin, J. (2003). The trilateral filter for high contrast images and meshes. In *Proceedings of the Eurographics Symposium on Rendering 2003*, pages 1–11.

Chuang, Y.-Y., Agarwala, A., Curless, B., Salesis, D., and Szeliski, R. (2002). Video matting of complex scenes. *ACM Transactions on Graphics*, 21(3):243–248.

Cohen, M. F., Colburn, R. A., and Drucker, S. (2003). Image stacks. Technical Report MSR-TR-2003-40, Microsoft Research.

Debevec, P. E. and Malik, J. (1997). Recovering high dynamic range radiance maps from photographs. In *Proceedings of ACM SIGGRAPH*, pages 369–378. Addison Wesley, Computer Graphics Proceedings, Annual Conference Series.

DeMenthon, D., Kobla, V., and Doermann, D. (1998). Video summarization by curve simplification. In *Proceedings of ACM Multimedia*, pages 211–218.

Divakaran, A., Peker, K. A., Radhakrishnan, R., Xiong, Z., and Cabasson, R. (2003). Video summarization using MPEG-7 motion activity and audio descriptors. Technical Report TR-2003-34, Mitsubishi Electric Research Laboratory.

Douglas, D. and Peucker, T. (1973). Algorithm for the reduction of the number of points required to represent a line or its caricature. *The Canadian Cartographer*, 10(2):112–122.

Drago, F., Myszkowski, K., Annen, T., and Chiba, N. (2003). Adaptive logarithmic mapping for displaying high contrast scenes. In *Proceedings of EUROGRAPHICS*, pages 419–426.

Dubois, E. and Sabri, S. (1984). Noise reduction in image sequences using motion-compensated temporal filtering. *IEEE Transactions on Communications*, 32(7):826–831.

Durand, F. and Dorsey, J. (2002). Fast bilateral filtering for the display of high-dynamic range images. *ACM Transactions on Graphics*, 21(3):257–266.

Edgerton, H. E. and Killian, J. R. (1979). *Moments of Vision*. MIT Press.

Eisemann, E. and Durand, F. (2004). Flash photography enhancement via intrinsic relighting. *ACM Transactions on Graphics*, 21(3):670–675.

Fattal, R., Lischinski, D., and Werman, M. (2002). Gradient domain high dynamic range compression. *ACM Transactions on Graphics*, 21(3):249–256.

Fay, D., Waxman, A., Aguilar, M., Ireland, D., Racamato, J., Ross, W., Streilein, W., and Braun, M. (2000). Fusion of multi-sensor imagery for night vision: Color visualization, target learning and search. In *Proceedings of the International Conference on Information Fusion*.

Fels, S., Lee, E., and Mase, K. (2000). Techniques for interactive video cubism. In *Proceedings of ACM Multimedia*, pages 368–370.

Francis, J. J. and Jager, G. D. (2003). The bilateral median filter. In *Proceedings of the 14th Symposium of the Pattern Recognition Association of South Africa*.

Freeman, W. T. and Zhang, H. (2003). Shape-time photography. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 264–271.

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison Wesley, Boston, MA.

Garnett, R., Huegerich, T., Chui, C., and He, W. (2005). A universal noise removal algorithm with an impulse detector. *IEEE Transactions on Image Processing*, 14(11):1747–1754.

Grossberg, M. and Nayar, S. (2004). Modeling the space of camera response functions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(10):1272–1282.

Hua, X.-S., Lu, L., and Zhang, H.-J. (2003). AVE: Automated home video editing. In *Proceedings of ACM Multimedia*, pages 490–497.

Jobson, D. J., Rahman, Z.-U., and Woodell, G. A. (1997). A multiscale retinex for bridging the gap between color images and the human observation of scenes. *IEEE Transactions on Image Processing*, 6(7):965–976.

Jostschulte, K., Amer, A., Schu, M., and Schroder, H. (1998). Perception adaptive temporal TV-noise reduction using contour preserving prefilter techniques. *IEEE Transaction on Consumer Electronics*, 44(3):1091–1096.

Kang, S. B., Uyttendaele, M., Winder, S., and Szeliski, R. (2003). High dynamic range video. *ACM Transactions on Graphics*, 22(3):319–325.

Kinsman, E. (2006). The time-lapse photography FAQ. *http://www.sciencephotography.com.*

Klein, A., Sloan, P. P., Finkelstein, A., and Cohen, M. F. (2001). Video cubism. Technical Report MSR-TR-2001-45, Microsoft Research.

Klein, A. W., Sloan, P.-P. J., Finkelstein, A., and Cohen, M. F. (2002). Stylized video cubes. In *Proceedings of ACM Symposium on Computer Animation*, pages 15–22.

Kwatra, V., Schödl, A., Essa, I., Turk, G., and Bobick, A. (2003). Graphcut textures: Image and video synthesis using graph cuts. *ACM Transactions on Graphics*, 22(3):277–286.

Lee, J.-S. (1983). Digital image smoothing and the sigma filter. *Computer Vision, Graphics, and Image Processing*, 24:255–269.

Lee, S. H. and Kang, M. G. (1998). Spatio-temporal video filtering algorithm based on 3-D anisotropic diffusion equation. In *Proceedings of the International Conference on Image Processing*, pages 447–450.

Li, H., Manjunath, B. S., and Mitra, S. K. (1994). Multi-sensor image fusion using the wavelet transform. In *Proceedings of the International Conference on Image Processing*, pages 51–55.

Liu, C., Torralba, A., Freeman, W. T., Durand, F., and Adelson, E. H. (2005). Motion magnification. *ACM Transactions on Graphics*, 24(3):517–526.

Liu, X. and El Gamal, A. (2003). Synthesis of high dynamic range motion blur free image from multiple captures. *IEEE Transactions on Circuits and Systems, Fundamental Theory and Applications*, 50(4):530–539.

Liu, Z., Shan, Y., and Zhang, Z. (2001). Expressive expression mapping with ratio images. *ACM Transactions on Graphics*, 20(3):271–276.

Mayer-Patel, K. and Rowe, L. A. (1997). Design and performance of the Berkeley continuous media toolkit. *Multimedia Computing and Networking*, SPIE 3020:194–206.

McGuire, M., Matusik, W., Pfister, H., Hughes, J., and Durand, F. (2005). Defocus video matting. *ACM Transactions on Graphics*, 24(3):567–576.

Microsoft Corporation (2007). DirectShow (DirectX 9.0c).

Muybridge, E. (1955). *The Human Figure In Motion*. Dover Publications.

Nandhakumar, N. and Aggarwal, J. (1997). Physics-based integration of multiple sensing modalities for scene interpretation. *Proceedings of the IEEE*, 85(1):147–163.

Nayar, S. and Branzoi, V. (2003). Adaptive dynamic range imaging: Optical control of pixel exposures over space and time. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1–8.

Nayar, S. and Branzoi, V. (2004). Programmable imaging using a digital micromirror array. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 436–443.

Pattanaik, S. N., Tumblin, J., Yee, H., and Greenberg, D. (2000). Time dependent visual adaptation for fast realistic image display. In *Proceedings of ACM SIGGRAPH*, pages 47–54. Addison Wesley, Computer Graphics Proceedings, Annual Conference Series.

Peli, T. and Lim, J. S. (1982). Adaptive filtering for image enhancement. *Optical Engineering*, 21(1):108–112.

Perez, J.-C. and Vidal, E. (1994). Optimum polygonal approximation of digitized curves. *Pattern Recognition Letters*, 15:743–750.

Perez, R., Gangnet, M., and Blake, A. (2003). Poisson image editing. *ACM Transactions on Graphics*, 22(3):313–318.

Perona, P. and Malik, J. (1990). Scale-space and edge detection using anisotropic diffusion. *IEEE Transactions of Pattern Matching and Machine Intelligence*, 12(7):629–639.

Petschnigg, G., Agrawala, M., Hoppe, H., Szeliski, R., Cohen, M. F., and Toyama, K. (2004). Digital photography with flash and no-flash pairs. *ACM Transactions on Graphics*, 23(3):661–669.

Pohl, C. and Genderen, J. V. (1998). Multisensor image fusion in remote sensing: Concepts, methods, and applications. *International Journal of Remote Sensing*, 19(5):823–854.

Porter, T. and Duff, T. (1984). Compositing digital images. *Computer Graphics*, 19(3):253–259.

Pratt, W. (1997). *Developing Visual Applications; XIL: An Imaging Foundation Library*, volume ISBN 0-13-461948-X. Prentice Hall.

Press, W. H., Teukolsky, S. A., Vetterling, W. T., and Flannery, B. P. (1992). *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, Cambridge.

Raskar, R., Ilie, A., and Yu, J. (2004). Image fusion for context enhancement and video surrealism. In *Proceedings of the International Symposium on Non-Photorealistic Animation and Rendering*, pages 85–94.

Rav-Acha, A., Pritch, Y., Lischinski, D., and Peleg, S. (2005a). Dynamosaics: Video mosaics with non-chronological time. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 58–65.

Rav-Acha, A., Pritch, Y., Lischinski, D., and Peleg, S. (2005b). Evolving time fronts: Spatio-temporal video warping. Technical Report HUJI-CSE-LTR-2005-10, The Hebrew University of Jerusalem.

Rav-Acha, A., Pritch, Y., and Peleg, S. (2006). Making a long video short: Dynamic video synopsis. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 435–441.

Reibel, Y., Jung, M., Bouhifd, M., Cunin, B., and Draman, C. (2003). CCD or CMOS camera noise characteristics. *European Physical Journal of Applied Physics*, pages 75–80.

Riddle, P. N. (1979). *Time-Lapse Cinemicroscopy*. Academic Press, New York, New York.

Sammon, C. (1969). A nonlinear mapping algorithm for data structure analysis. *IEEE Transactions on Computers*, C-18(5):401–409.

Sand, P. and Teller, S. (2004). Video matching. *ACM Transactions on Graphics*, 23(3):592–599.

Schödl, A., Szeliski, R., Salesin, D. H., and Essa, I. (2000). Video textures. In *Proceedings of ACM SIGGRAPH*, pages 489–498.

Shectman, E., Caspi, Y., and Irani, M. (2005). Space-time super-resolution. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(4):531–545.

Shi, J. and Tomasi, C. (1994). Good features to track. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 593–600.

Smith, M. A. and Kanade, T. (1997). Video skimming and characterization through the combination of image and language understanding techniques. Technical Report CMU-CS-97-111, Carnegie Mellon University.

Smith, S. M. and Brady, J. M. (1997). SUSAN - A new approach to low level image processing. *International Journal of Computer Vision*, 23(1):45–78.

Stockham, T. G. (1972). Image processing in the context of a visual model. *Proceedings of the IEEE*, 60:828–842.

Terry, M., Brostow, G. J., Ou, G., Tyman, J., and Gromala, D. (2004). Making space for time in time-lapse photography. In *Proceedings of ACM SIGGRAPH Technical Sketches.*

Therrien, C., Scrofani, J., and Krebs, W. (1997). An adaptive technique for the enhanced fusion of low-light visible with uncooled thermal infrared imagery. In *Proceedings of the International Conference on Image Processing*, pages 405–408.

Toet, A. (1990). Hierarchical image fusion. *Machine Vision and Applications*, 3(1):1–11.

Toet, A. (2005). Colorizing single band intensified nightvision images. *Displays*, 26(1):15–26.

Tomasi, C. and Manduchi, R. (1998). Bilateral filtering for gray and color images. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 836–846.

Tsin, Y., Ramesh, V., and Kanade, T. (2001). Statistical calibration of CCD imaging process. In *IEEE International Conference on Computer Vision*, pages 480–487.

Tukey, J. W. (1971). *Exploratory Data Analysis.* Addison-Wesley, Menlo Park, CA.

Tumblin, J. and Rushmeier, H. E. (1993). Tone reproduction for realistic images. *IEEE Computer Graphics and Applications*, 13(6):42–48.

Tumblin, J. and Turk, G. (1999). Lcis: A boundary hierarchy for detail preserving contrast reduction. In *Proceedings of ACM SIGGRAPH*, pages 82–90. Addison Wesley, Computer Graphics Proceedings, Annual Conference Series.

Wang, H., Raskar, R., Xu, N., and Ahuja, N. (2007). Videoshop: A new framework for spatio-temporal video editing in gradient domain. *Graphical Models*, 69(1):57–70.

Wang, J., Bhat, P., Colburn, R. A., Agrawala, M., and Cohen, M. F. (2005). Interactive video cutout. *ACM Transactions on Graphics*, pages 585–594.

Wang, J. Y. A. and Adelson, E. H. (1994). Representing moving images with layers. *Transactions on Image Processing*, 3(5):625–638.

Ward, G. (1991). Real pixels. In *Graphics Gems II*, pages 80–83. Academic Press.

Weiss, B. (2006). Fast median and bilateral filtering. *ACM Transactions on Graphics*, 25(3):519–526.

Welsh, T., Ashikhmin, M., and Mueller, K. (2002). Transferring color to greyscale images. *ACM Transactions on Graphics*, 21(3):277–280.

Wexler, Y. and Simakov, D. (2005). Space-time scene manifolds. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 858–863.

Wildemuth, B. M., Marchionini, G., Yang, M., Geisler, G., Wilkens, T., Hughes, A., and Gruss, R. (2003). How fast is too fast? Evaluating fast forward surrogates for digital video. In *ACM/IEEE Joint Conference on Digital Libraries*, pages 221–230.

Wolberg, G. (1995). *Digital Image Warping*, volume ISBN 0-8186-8944-7. Wiley-IEEE Computer Society Press.

Yee, H., Pattanaik, S. N., and Greenberg, D. P. (2001). Spatio-temporal sensitivity and visual attention for efficient rendering of dynamic environments. *ACM Transactions on Graphics*, 20(1):39–65.

Zomet, A., Feldman, D., Peleg, S., and Weinshall, D. (2003). Mosaicing new views: The crossed-slits projection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(6):741–754.

Zwicker, M., Pfister, H., Baar, J. V., and Gross, M. (2002). Pointshop 3D: An interactive system for point-based surface editing. *ACM Transactions on Graphics*, 21(3):322–329.