2024

Post-Capture Synthesis of Images Using Manipulable Integration Functions

Paul Eberhart
University of Kentucky, pseber2@uky.edu
Author ORCID Identifier:
https://orcid.org/0000-0002-2797-2112
Digital Object Identifier: https://doi.org/10.13023/etd.2024.388

STUDENT AGREEMENT:

I represent that my thesis or dissertation and abstract are my original work. Proper attribution has been given to all outside sources. I understand that I am solely responsible for obtaining any needed copyright permissions. I have obtained needed written permission statement(s) from the owner(s) of each third-party copyrighted matter to be included in my work, allowing electronic distribution (if such use is not permitted by the fair use doctrine) which will be submitted to UKnowledge as Additional File.

I hereby grant to The University of Kentucky and its agents the irrevocable, non-exclusive, and royalty-free license to archive and make accessible my work in whole or in part in all forms of media, now or hereafter known. I agree that the document mentioned above may be made available immediately for worldwide access unless an embargo applies.

I retain all other ownership rights to the copyright of my work. I also retain the right to use in future works (such as articles or books) all or part of my work. I understand that I am free to register the copyright to my work.

REVIEW, APPROVAL AND ACCEPTANCE

The document mentioned above has been reviewed and accepted by the student's advisor, on behalf of the advisory committee, and by the Director of Graduate Studies (DGS), on behalf of the program; we verify that this is the final, approved version of the student's thesis including all changes required by the advisory committee. The undersigned agree to abide by the statements above.

Paul Eberhart, Student

Henry G. Dietz, Major Professor

Simone Sylvestri, Director of Graduate Studies

POST-CAPTURE SYNTHESIS OF IMAGES USING MANIPULABLE INTEGRATION FUNCTIONS

DISSERTATION

A dissertation submitted in partial
fulfillment of the requirements for
the degree of Doctor of Philosophy
in Computer Science in the Stanley
and Karen Pigman College of
Engineering at the University of
Kentucky

By
Paul Selegue Eberhart
Lexington, Kentucky

Director: Dr. Henry G. Dietz, Professor of Electrical and Computer Engineering
Lexington, Kentucky 2024

https://orcid.org/0000-0002-2797-2112

ABSTRACT OF DISSERTATION

POST-CAPTURE SYNTHESIS OF IMAGES USING MANIPULABLE INTEGRATION FUNCTIONS

Traditional photographic practice, as dictated by the properties of photochemical emulsion film, mechanical apparatus, and human operators, largely treats the sensitivity (gain) and integration interval as coarsely parameterized constants for the entire scene, set no later than the time of exposure. This frame-at-a-time capture and processing model permeates digital cameras and computer image processing.

Emerging imaging technologies, such as time domain continuous imaging (TDCI), quanta image sensors (QIS), event cameras, and conventional sensors augmented with computational processing and control, provide opportunities to break out of the frame-oriented paradigm and capture a stream of data describing changes to scene appearance over the capture interval with high temporal precision. Captured scene data can then be computationally post-processed to render images with user control over the time interval being sampled and the gain of integration, not just for each image rendered but for every site in each rendered image, allowing the user to ideally expose each portion of the scene. For example, in a scene that contains a mixture of moving elements some of which are more brightly lit, it becomes possible to render dark and light portions with different gains and potentially overlapping intervals, such that both have good contrast, neither one suffers motion blur, and little to no artifacting occurs at the interfaces.

This dissertation represents a preliminary exploration of the properties, application, and tooling required to capture TDCI streams and render images from them in a paradigm that supports functional post-capture manipulation of time and gain.

KEYWORDS: Computational photography, Exposure, Image capture, Postprocessing, Video Processing

Author's signature: Paul Selegue Eberhart

Date: September 25, 2024

POST-CAPTURE SYNTHESIS OF IMAGES USING MANIPULABLE INTEGRATION FUNCTIONS

By
Paul Selegue Eberhart

Director of Dissertation: Henry G. Dietz

Director of Graduate Studies: Simone Silvestri

Date: September 25, 2024

ACKNOWLEDGMENTS

I would like to acknowledge the many friends, family, and colleges who have been patient about the large chunks of time, energy, and attention I devoted to the work that lead to this dissertation, and also about the large chunks of time devoted to other priorities during the interim.

I would also like to acknowledge my committee members for their time and patience, including two former committee members, Dr. Raphael Finkel and Dr. Nathan Jacobs, who left the University of Kentucky before I finished, but gave me a great deal of excellent advice during their involvement.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1 Introduction

This work re-considers the purpose and mechanism of cameras. In scientific applications, the purpose of a camera often is to count and measure properties of photons. In contrast, classical photography typically regards a camera as a tool to capture an image which reflects the appearance of a scene over a particular exposure interval, as sampled using photons. Generalizing this second view, a camera is a device that creates a model of scene appearance. Under this premise, the pixel value at each point is determined by the arrival rate of photons at that site. Because photon arrival is a discrete random process in which the arrival rate varies over time, scene appearance changes are detected by recording a significant change to the arrival rate. Modeling scene appearance is thus inherently frame-less; the ideal model is composed of a time-varying function for each sensel.

Conventional cameras perform correlated sampling of all sensels over a single fixed exposure interval, resulting in a frame in which each pixel value is determined by the number of photons that arrive during the exposure at the corresponding sensel. This behavior discards potentially useful timing information and reduces the quality of the estimate of scene appearance. For example, as illustrated in figure 1.1, if a particular sensel has no photons arrive during a one second exposure interval, correlated sampling will determine that the value at that site is zero. However, if one photon arrived in the second before and another the second after, the expected value at that site for the exposure interval is an arrival rate of two photons per three seconds. Subsequently computationally rendering frames from a captured per-sensel continuous model produces a more accurate result than the standard correlated sampling of a conventional camera capturing frames.

Because this rendering is after-the-fact, decisions about exposure parameters can be made, and re-made, with information from the scene at the moment of capture, and the exact time interval to be sampled can be arbitrarily adjusted after the fact. The work presented here explores the required changes to cameras to support this mode of capture, and extends the freedom afforded by post-capture rendering, proposing and demonstrating the possibility of composing images not only from fixed intervals, but by integrating the scene model over arbitrary functions in time, space, and gain.

1.1 TDCI

The work described here is built on and advances the new camera model known as **T**ime **D**omain **C**ontinuous **I**maging (TDCI). TDCI treats a digital image sensor as a vast array of millions of individual data channels, one for each sensel, representing

Figure 1.1: Two photon strikes over three time intervals

changes in the incident light at that site over time. For each site, the output of TDCI is a waveform akin to individual audio channels or channels of an oscilloscope.

Instead of attempting to obtain a single, correlated image representing the average value over a period, TDCI stores a compressed form of the entire waveform at each site — a full model of the evolution of the scene over time — and later computationally samples those waveforms to form images. The storage format for a TDCI system is not a series of frames, or even a series of samples, but a description of how the scene content changes over time: an **Im**age **Ev**olution model (IMEV).

The publication record for TDCI technology begins in 2014 [1] with "Frameless, Time Domain Continuous Image Capture" and extends through several publications that provide the context for the work reported here:

1. "ISO-Less?"(2015) [2] established that the gain properties of extant sensors are suitable to support the model

2. "Scene Appearance Change As Framerate Approaches Infinity" (2016) [3] which verifies the relationship between sample rate and scene data required for TDCI systems to operate at reasonable data-rates, further noting that the lighting sets an ultimate limit on how much scene appearance data can be recorded

3. "Shuttering Methods and the Artifacts They Produce"(2019) [4] explores the artifacts and temporal skewing created by various shuttering mechanisms

4. "Temporal Super-Resolution for Time Domain Continuous Imaging" (2017) [5] uses various properties, including rolling shutter timing, to derive more precise temporal data from conventional camera output

Using current sensor technologies, the channels will be wildly under-sampled. The shutter angle of the camera, the ratio of ADCs to sensels, the readout bandwidth from the sensor, and a multitude of other factors make it impossible to truly record the entire wave at each site. The sampling is ideally in uncorrelated content-dependent patterns, so rather than a simple Nyquist or Nyquist-Shannon basis, the generalized limit for uncorrelated sampling later established by Landau [6] applies. In recent decades, this same observation that capturing image data in temporally random and/or uncorrelated ways is desirable has generated a subfield generally referred to as "compressive sensing" [7]. A number of related concepts are discussed in section 1.2.

TDCI hinges on the observation that factors such as scene constancy and photon shot noise limit the information content of those signals. Thus, it is possible to record a significantly reduced amount of *data* and still capture all the *information* provided about the evolution of the scene appearance over time as sampled by the arriving photons.

In TDCI, the captured scene model is *predictive*: a rate of change or expected value of the light at each site is constructed from successive samples, and updated by subsequent sampling. Records — output data — are only generated when the light arriving at a specific sensel differs significantly from the expected behavior. Only a

change to the rate of change, or a second derivative of the incident light, generates a data record in the stored waveform. These records are uncorrelated in time across the sensor; there is no concept of a frame in this model. Ideally, each site is sampled at a rate sufficient to capture the information content of the incident light, and no more. That rate can and will vary across the scene and sensor.

Pixel Value Error Model

The TDCI model relies on an understanding of sampling error in the capture device to create the scene model with a practical sampling and data-rate. This pixel value error model is based largely on the the work described in "Scene Appearance Change As Framerate Approaches Infinity" [3], which specifically investigated the fall-off in additional scene *information* added by increasing the framerate used to sample the scene. This property was demonstrated largely by observing that the compressed size (using both TDCI and conventional H.264 compression) of the recording does not grow linearly nor continuously with the frame rate, but instead appears to approach a fixed bound for any given compression scheme. That bound is determined by the actual content and evolution of the captured scene as made observable by the lighting.

This highlights a critical assumption that underpins the TDCI model. Photography is not about capturing photon properties, but is the practice of statistically sampling scene appearance via photons. Thus, scene changes that happen faster than photons can sample them are inherently unknowable. Unknowable changes are simply assumed not to happen. This assumption is henceforth called scene constancy.

This assumption has important implications, including the observation that statistical variations in photon emission, typically referred to as "photon shot noise," bound the useful temporal resolution with which the scene can be sampled. If the temporal resolution of sampling is increased beyond the fastest statistically-significant change to the photon arrival rate created by the scene and lighting conditions, increasing the sampling rate cannot increase the amount of information captured about the scene. Conversely, the ability to sample high-frequency changes in a scene is bounded by the lighting, as a statistically significant number of photons must be sampled to reliably observe and record any change.

Based largely on the results in this initial work, the TIK tooling [8] which forms the basis for much of the ongoing TDCI research uses a surprisingly straightforward method to generate a suitable error model. A scene which is not changing over time, and is lit consistently over that period, is captured using the camera and configuration for which the error model will be applied. That configuration includes selection of camera parameters, especially ISO sensitivity. The invariant scene capture could be a series of still images, a conventional video, or even an IMEV model.

The goal of the analysis of the static scene capture is to create a **P**probability **D**ensity **F**unction (PDF) that can be used to statistically determine whether a new value for a sensel represents an additional sampling of the same scene content or a sampling of new scene content now in that position. In other words, the PDF attempts to answer the question: is this new sensel reading different from the previous reading by more than sampling noise? If so, a new reading creates a new change record in the

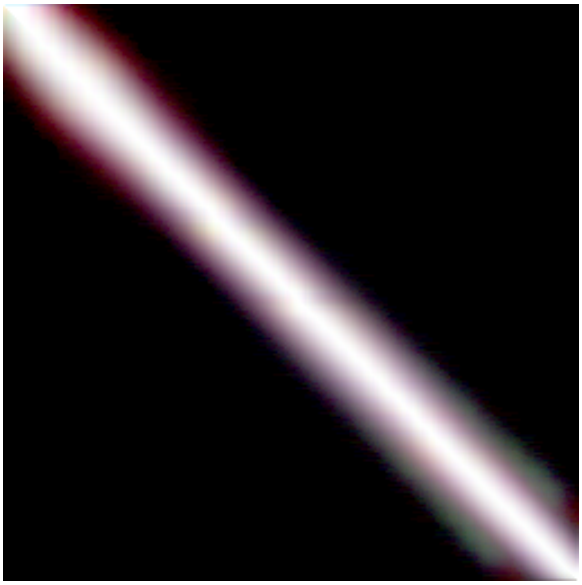


Figure 1.2: RX100IV Noise Model

IMEV. If not, the new reading simply updates the estimate of the sensel value in the still active change record, producing a more accurate value in the record by weighted averaging of the new reading with the value previously in the change record.

In TIK, the PDF is encoded as a square matrix. The Y index represents the ideal value of a sensel — with no noise. The X index represents the value actually read from the sensel. The matrix entry at position X,Y approximates the conditional probability that sampling noise caused the value X to be read when the scene content was actually Y. The probabilities do not just model photon shot noise, but the combined impact of all noise sources on the values read. In most digital cameras, the sensor is enabled to distinguish colors by imposing a **C**olor **F**ilter **A**rray (CFA) that filters the light so individual pixels see only red, green, or blue; this filtering results in potentially different noise characteristics for the different color channels. Thus, the PDF is actually three matrices: one for each of the red, green, and blue color channels. The PDF matrix used for a sensel is the one for the color channel it senses.

These three PDF matrices can be compactly represented, manipulated, and visualized as a single three-color square image. Since the noise model is empirically determined and imperfect, it was found that scaling the PDF matrix to 256×256 (i.e., eight bits per sample) and scaling the probabilities 0..1 to the range 0..255 still produced good results. TIK can compute, store, load, and apply these "pixel value error models" as P6 PPM image files, as described in detail in subsection 1.1. A perfectly noiseless PDF would look like a bright diagonal line from 0,0 to 255,255.

An example of a TIK PDF noise model is shown in figure 1.2. The `tik` tool generates a noise model when called with the `-e` option, as in `tik -e -oErrMod.ppm InputFile.mp4`. This map was generated from a Sony RX100IV camera, shooting 960FPS, with 1/1000 of a second shutter speed, and an unrecorded automatically-selected ISO under mediocre indoor lighting. The resulting video is visibly noisy; an extracted frame is shown in figure 1.3. The relatively large width of the diagonal

Figure 1.3: One frame of the input used to generate the noise model in figure 1.2

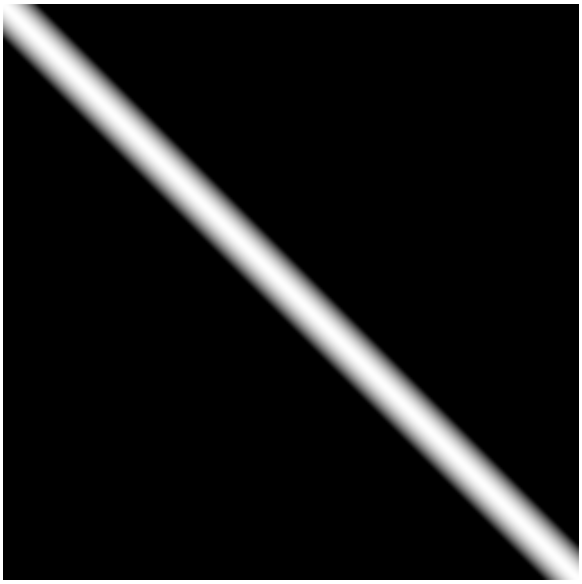


Figure 1.4: TIK's Default Noise Map

is commensurate with observed noise, and the red and green color fringing at either corner indicates difference in the noise properties of the red and green channels. To illustrate the specificity of these noise models, an error map generated from a capture with the same device at the same framerate and under similar lighting conditions, but with a higher shutter angle at 1/250 of a second was included as Figure 2 in earlier work [8], and shows wildly different noise properties.

If TIK is run without a specified noise model, it automatically generates the generic model shown in figure 1.4. This model is not a precise match to any particular camera, but provides an adequate generic approximation for situations where a specific noise model is not readily available. Several of the examples in this document were generated using this default noise model.

TDCI Implementations

There have been a number of publications related to *implementations* of TDCI systems. The most consequential has been a simulation testbed, TIK, slightly unfortu-

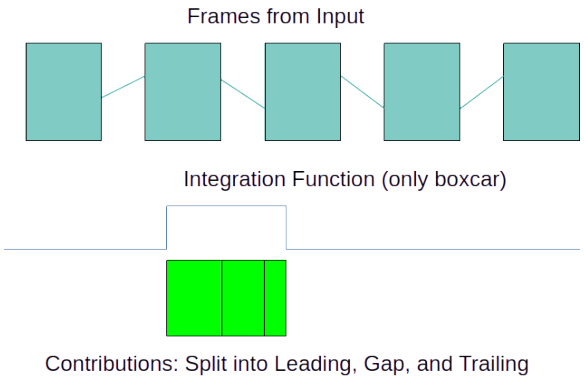


Figure 1.5: TIK Data Model

nately, a dual-use acronym, referring to both the **T**emporal **I**maging from **K**entucky tools, and the **T**emporal **I**mage **K**ontainer format they operate on. The initial development of TIK is documented in "TIK: a time domain continuous imaging testbed using conventional still images and video" [8].

The TIK software and formats create a testbed for performing time domain continuous imaging using conventional still images and/or video captures — and it was a key code infrastructure modified to create prototypes in the current work. The existing TIK tools consist of a set of open-source programs and specifications that allow the generation of noise models, rendering of IMEVs from series of frames, and the rendering of virtual exposures from existing IMEVs.

These three functions are performed offline, using recorded images and taking nontrivial time and computational resources to perform the renderings. Figure 1.5 shows roughly how TIK integrates video data into a frame. Average values for each pixel are known during each frame, as that is the recorded pixel value for that frame. The behavior between frames is approximated as a linear transition from the previous average value to the next. An output image (frame) can be generated from a start time for an interval by summing the duration-weighted estimated values and dividing by the total time represented.

TIK File Formats

The TIK formats extend the NetPBM [9] family of image formats. The NetPBM format suite was developed in the mid 1980s by Jef Poskanzer. NetPBM formats include **P**ortable [**B**it — **G**rey — **P**ix] **M**aps, as well as several less-common extensions, such as the PNM (**P**ortable A**n**y **M**ap) format, which allows for arbitrary pixel encodings. Each standard NetPBM format type can be encoded in two modes. ASCII mode, which initially served as a way to safely transmit image data through plaintext-only channels such as email, and now provides a convenient representation for direct manipulation by a human or primitive text-oriented tools. Binary mode, which is more compact, and also very convenient to directly manipulate when memory-mapped. The NetPBM formats all begin with an ASCII magic number, the meanings of which are described in 1.1. Textual *comments* can also appear in the file header in the form

Type	Magic Number		Extension	Colors
	ASCII	Binary		
Portable BitMap	P1	P4	.pbm	0/1 (White/Black)
Portable GrayMap	P2	P5	.pgm	0-255 (Grays)
Portable PixMap	P3	P6	.ppm	0-255 (RGB)

Table 1.1: NetPBM Magic Numbers

of lines beginning with #.

The TDCI extensions exploit the comment mechanism in the NetPBM format to generate files which appear as valid NetPBM images to tools expecting them, but contain extra data and metadata for TDCI applications. This enables image previews, appropriate MIME behavior, and other user-friendly features with no additional development effort. Some common tools, such as `ffmpeg` or ImageMagick's `display` support concatenate NetPBM files, which can be used for automatic support of uncompressed TDCI data. This header format is so flexible that it can be used to prepend appropriate metadata to formats ingested by the TIK tools which are not able to carry their own, such as folders of frames or conventional video files, by including a .tik containing only metadata.

The TDCI extensions to NetPBM start with a TDCI version header, which must be the first comment in the NetPBM, of the form:
`# TIK V` *version format [parameters]*
where *version* is an eight-digit ISO-style date representing the date on which the TIK tool, and *format* is which kind of encoding is used in this file, as well as any parameters to that format. Fields are delimited by one or more whitespace (space or tab) characters. For example, a file beginning
```
# TIK V 20160712 RGB
```
would signal a file compliant with the 20160712 version of the TIK specification, containing a normal P6 PPM file, a 0 byte, and a stream of RGB-encoded pixel updates. The 20160812 TIK standard has been published as a part of *TIK: a time domain continuous imaging testbed using conventional still images and video* [8].

Another early TDCI implementation was an in-camera capture mechanism described in "A Canon Hack Development Kit implementation of Time Domain Continuous Imaging" (2017) [10]. This implementation builds on CHDK (the **C**annon **H**ack **D**evelopment **K**it) to encode a IMEV directly in a consumer digital camera. The CHDK TIK implementation is necessarily extremely minimal and limited , as the target cameras for this method are, unfortunately, not particularly well suited for TDCI capture. They lack any sort of low-level sensor addressing and are not computationally powerful enough to cover for their deficiencies. The Canon ELPH115 and Canon ELPH160 cameras used for the initial implementation each contain only a pair of ARMv5TE processors at about 83 MIPS, and 36MB of RAM, which supports a memory bandwidth of roughly 59Mb/s for writes and 21Mb/s for reads.

These restrictions limit the TIK captures performed in-camera to the live view resolution of the host camera, 720x240 in the case of the original targets. Operating on the live view stream presents a number of difficulties and advantages. As a substantial benefit, it frees up the RAW buffer, the section of main memory used to store full-resolution, full-depth sensor data before it is converted to JPEG and/or written out. The raw buffer is the bulk of otherwise unallocated main memory in the camera. In the case of the ELPH160, the 20MP sensor at 12bpp consumes roughly 30MB of main memory, leaving only roughly 3MB for user code if it is in use. Other than the obvious extremely restricted resolution, the main disadvantage is that the live view data is delivered at an inconsistent framerate with data delivered in a peculiar packed YUV format which requires decoding, and only provides one U and V value for each group of 4 pixels.

The CHDK TIK capture mechanism is built on top of the motion detect hooks in CHDK, exploiting the similarities between TDCI and various established motion-extraction techniques discussed further in 2.4. The latest 20161130UYVYYY implementation even offers the ability to render frames from specified intervals, albeit in an extremely awkward and limited way, producing images encoded in a format the camera itself is unable to display.

1.2 Related Work

Though the techniques discussed in this section differ in method — as they all build on a presumption of correlated frames as both input and output rather than per-pixel change models — they rely on similar principles or attempt to produce similar results to applications of TDCI suggested in this and prior work.

Video Resampling

The area of research with the largest conceptual overlap may be video resampling — using an input video at one framerate to synthesize a video at another, possibly higher, framerate. For example, it is common that traditional film for cinema projection is shot at 24FPS (Frames per Second). However, in the United States the TV broadcast standard, NTSC (National Television Standards Committee) [11], initially formulated in 1941 and updated for color in 1953, specifies a framerate of 30 FPS for black and white and 29.97FPS (to allow the color data to be inserted in a backward-compatible way [12]) for color at 525 lines of vertical resolution, split into two interlaced fields for 262.5 lines supplied at 60FPS. Other systems assume other frequencies. The PAL (Phase Alternating Line) standard common in European broadcast television specifies a frequency of 25FPS in the form of two alternating, interlaced fields at 50FPS. Converting content filmed in a 24FPS format for broadcast therefore requires significant adjustment to the content.

For PAL conversion from 24 to 25FPS, several simple options exist. One can simply speed up the content by $1/24 \approx 4.1\%$, and hope the viewers don't notice. More commonly, frames are repeated: each 24FPS input frame can be used to supply

two 50i fields, except for once every half second where three fields are derived from the same input frame. This frame repetition can produce a noticeable visual stutter.

NTSC conversion from 24FPS input poses more of a problem. A $30/24 \approx 25\%$ speedup will be extremely evident, so simply adjusting the playback rate as with 24 to 25 conversion is not an option. The traditional method for converting 24FPS input for NTSC $\approx$ 30 FPS playback is the use of "$3:2$ pulldown" creating a 60i format by splitting each frame of 24FPS input into two fields, then duplicating one of fields of each frame in a $2-3-2-3...$ pattern to produce an approximate 60i format, albeit with considerable motion smoothing and frequent stutters.

Modern computer-driven methods of converting between framerates tend to rely on an *optical flow* model [13] — a model of the motion of features and edges in the recorded scene. Instead of cherry-picking frames to down-convert, or duplicating frames to up-convert, the amount of deformation in the scene during the desired target interval can be estimated from analysis of the input frames, and additional frames can be generated to insert at the appropriate intervals using a form of interpolation.

Optical flow models are also used in video *compression* tasks. The H.261 [14] video encoding standard which evolved into MPEG-1 video encoding in 1992 [15] was the first practical, widely deployed, video encoding standard. H.261 relies, as most subsequent compressed video formats do, on a mixture of discrete cosine transforms similar to common single-image compression techniques to compress individual frames and motion compensation for inter-frame compression. General purpose computers at the time of its adoption were incapable of real-time encoding or decoding of compressed video, and lacked the working memory to operate on more than a few frames at a time, so the entire encoding/decoding process was initially performed via offload to dedicated special-purpose hardware which returned frames.

This separation of concerns has continued into the modern era. Notably, in the literature, the motion-aware inter-frame models used for framerate shifting are typically not the optical flow model which is part of in the compressed format the data is stored in, but a separate model constructed from frames rendered from the compressed format then re-analyzed. A number of methods of re-deriving optical flow models from rendered frames are discussed in [16]. More recently, a number of convolutional models [17] have been developed. These tend to be *extremely* computationally expensive, requiring large amounts of memory, time, and often additional complications such as segmentation or reduction from a full 2D model of the frame to a 1D line model in order to resynthesize video of useful resolution. The more complicated models are also prone to "hallucinate" details that did not exist in the source scene, particularly if the model is trained on data other than the content whose framerate is being shifted.

Interpolation based models, even ones that rely on elaborate convolutional interpolation, specifically deal with *video content*: they model the contents of the frames, but do not leverage knowledge of the capture device. For example, there is no modeling of noise nor even of the temporal period represented by each frame (shutter speed or shutter angle). In 2017, a paper "Temporal super-resolution for time domain continuous imaging" [5] demonstrated how the temporal resolution of scene data can be increased by using an awareness of the shuttering behavior and error model of the capture device in combination with sophisticated interpolation. This work was also

rather atypical in that it treated temporal skew due to shuttering and readout as a *feature* to be exploited for temporal information rather than a defect from an assumed ideal of globally correlated sensing.

Even synthesizing a TDCI model from an input video, then re-synthesizing frames from the IMEV model, offers a number of advantages over these techniques. A TDCI model inherently models key properties of the *capture device*, drawing a distinction between scene content and artifacts from the capture device which do not reflect scene content.

In much the same way that convolutional interpolation schemes benefit from including data from a large range of input rather than simply from adjacent frames, a TDCI model-driven re-rendering can leverage captured scene data for extended virtual exposure intervals, even allowing a user to render frames with an effective shutter angle greater than 360°. A TDCI model is also considerably less computationally intensive to derive and generate frames from than convolutional techniques, and operates only on data from the target scene, eliminating the risk of detail not present in the scene being injected during manipulation.

Processing of Multiple Still Images

In addition to video-oriented tools, there are various algorithms designed to process image information from multiple still images of the same scene. Generally, the output of these is a single rendered image.

Shot Selection

Perhaps the simplest multi-shot processing mechanism is Nikon's "Best Shot Selector" (BSS) [18]. It automatically takes a series of images when the operator hits the shutter button, then applies a simple sharpness estimation heuristic to select the "best" image from the set of exposures. This class of techniques leverages the near-zero marginal cost of taking, inspecting, and discarding additional images with a digital camera. It is surprisingly effective in reducing blur from camera shake, largely because camera shake is a stochastic process and some captures will likely hit motion minima.

Buffering

Although it is complicated by requiring user input, it is conceptually even simpler to have the camera capture a sequence of images during a time interval and then have the user select which to retain. This feature is primarily to reduce the probability of missing a shot due to the operator's reflexes or lag in the camera. Camera lag can come from multiple sources, including not only delay in initiating the exposure after press of the shutter button but also lag in presenting an electronic live view to the photographer. Typically, the camera implements this feature using a ring buffer of captured frames. At half press of the shutter button, the buffer begins to fill with captured frames, and captures continue until shortly after the full press of the shutter button. Thus, the buffer contains many images captured around the time at which the full press occurred.

Slight variations on this technique are available under a wide variety of marketing names from every major camera manufacturer: Fujifilm Pre-Shot, Olympus Pro Capture, Nikon Pre-Release Capture, Sony Predictive Capture (in cell phones), and Canon and Panasonic Pre-Burst [19] are all implementations of this mechanic. Again, this feature is enabled by the low cost and high speed of shooting digital, and comes with a large variety of limitations.

The temporal spacing of the shots is limited by the exposure time and the readout bandwidth of the device. Each frame's exposure parameters are set conventionally before they are captured, so pre-exposure buffering does not help to correct exposure parameter related problems — if the frames are not properly exposed, or the relationship between shutter speed and scene motion was not as desired, the result is simply a variety of badly exposed images to choose from. The timing of each frame is also fixed; if the exact moment the operator was trying to capture fell between the burst of exposures, there is no after-the-fact recourse to re-time the capture. Most cameras do not allow for shooting RAW when this feature is in use because they simply do not have enough high-speed buffer to keep multiple RAWs in memory at once. For cameras using mechanical shutters, the rapid repeated action of the shutter can be disturbingly loud while also imparting shake to the camera; thus, these modes are often restricted to electronic shuttering only.

A TDCI system, particularly one enhanced with sophisticated functional control of exposure parameters, makes a better alternative to pre-shot features in a variety of ways. Because each rendered frame in such a system is integrated from an interval selected after the time of capture, it is possible to independently and repeatedly alter the start time and exposure length for a rendered frame with fine-grained control. This ensures that the desired time interval can be represented in an image, whereas that times period might be unsampled using a burst. For example, a 60FPS burst shot with a shutter speed of 1/1000s is missing 94% of the action. Furthermore, because the *gain* of a frame rendered from an IMEV can be controlled during the rendering rather than capture process, the effective "film speed" of the captured frame can be adjusted in concert with the interval adjustments, resulting in better exposed images.

Dark Frame Subtraction

Instead of picking a single image, *Dark Frame Subtraction* combines two images to eliminate fixed pattern noise, excessively bright pixels caused by sensel leakage defects that are sensitive to sensor temperature as well as exposure duration. After shooting the scene normally, a second image is captured with the same exposure settings without opening the shutter. Since the second exposure only has leakage contributing to the image, subtracting that image from the first image tends to cancel out the brightness caused by leakage. Despite the simplicity, this method can yield significant improvements in image quality, particularly for very long exposures under low light conditions, and it is widely used for astrophotography. Dark frame subtraction is easily implemented within a camera; in fact, most modern cameras enable dark frame subtraction by default for exposures over one second in duration. Dark frame subtraction is also frequently referenced under other names based on it's

common application, such as the "Long Exposure Noise Reduction" option in Canon EOS firmware.

Dark frame subtraction can be thought of as an extremely simple sensor noise model. A TDCI system inherently, at the time of encoding, develops are more sophisticated error model which *can* address noise features which vary with time or scene content.

Bracketing and Stacking

Ratcheting up the sophistication of multi-shot features further, modern digital cameras with computer-controlled optical paths often have built-in support to take rapid bursts of exposures while automatically varying exposure settings. These methods are typically referred to as "HDR Bracketing," which allow capture of a High Dynamic Range image by taking exposures with an exposure parameter quickly and automatically varied in steps around an estimated center value. The bracketed exposures are then "stacked" to form a final image combining the correctly-exposed portions of multiple frames to extend the dynamic range of the image. This class of technique offers both dynamic-range enhancement and noise reduction.

These methods were developed quickly after the advent of reasonably high-quality digital image sensors in cameras under computer control, as outlined in the 1997 SIGGRAPH paper "Recovering high dynamic range radiance maps from photographs" [20]. Even this early work addresses one of the weaknesses in bracketing techniques: the images must be precisely aligned, either out of the camera or by later manipulation, in order to properly merge features from the individual frames. Characterizing the response curves of the camera, the relationship of the read-out values in the individual color channels to the incident light, allows computing an approximation to the absolute radiance values in the scene. Thus, manipulation of the tonal range, similar to control of exposure, can be done in postprocessing. "Tone mapping" even allows rendering an image representing a wider range of brightness than the output image can directly represent.

Variations on automatic HDR bracketing have become standard fare in modern digital camera firmware. A community-maintained list of the number, speed, and stride (in EV) of the frames taken by various camera models is maintained at [21].

Many subsequent developments in bracketing largely focus on automatically determining appropriate exposure values for the components [22]. That work examines large sets of photographs taken with different shutter speeds, HDR stacks subsets of them, and analyzes the results to extrapolate an algorithm for estimating optimal spacing and quantity of different shutter speed exposures for an HDR bracket, based on a single auto-exposed image of the scene.

Even more sophisticated work has considered HDR burst design as a multivariate problem [23]. Digital cameras can dynamically vary ISO as well as shutter speed and aperture, and computers can quickly solve (or at least estimate) complex optimization problems to determine the best use of the available parameters and constraints. An example of that design space is "Noise-optimal capture for high dynamic range photography" [23] which applies mixed integer programming techniques for estimating a

time and/or noise optimal capture sequence which can obtain a lower noise, wider dynamic range result and do so in fewer exposures. This class of work demonstrates that careful computational manipulation of a digital camera device in ways that do not map directly to traditional film photography practices can readily produce superior results.

This class of bracket-and-stack technique, however, suffers from several serious limitations. The main limitation is the underlying assumption that the scene appearance is constant. With a perfectly still scene, hand-held shots will require alignment processing even if electronic shuttering, image stabilization, and other methods are used to minimize shake. When motion occurs only in portions of a scene, conventional HDR methods fail for the affected areas. In contrast, the IMEV approach advocated here can make use of the per-pixel periods of stability, obtaining somewhat less dynamic range for things changing quickly, but still obtaining some benefit. Of course, HDR bracketing requires that every part of the scene is properly exposed in at least one of the series of frames, which it cannot easily guarantee; in contrast, TDCI capture can ensure that the full dynamic range that can be sensed by the camera is always available for postprocessing use.

Of course, recent efforts have been made to use convolutional neural networks (CNNs) trained on pairs of LDR and HDR images to estimate a likely HDR image from a single LDR frame by predicting likely values for saturated, underexposed, or otherwise clipped content [24]. Architectures for doing so vary by level of supervision, whether the HDR image is generated directly or by creating a set of estimated exposures and stacking them, and whether the tool primarily fills detail in saturated or underexposed regions [25]. These methods are, as all such approaches, computationally expensive, and prone to produce images which are convincing rather than faithful, potentially containing features not found in the scene data.

Computational Multi-Shot Alignment

The model and processing discussed in the current work uses the evolution of pixel values over time to detect scene changes, but most earlier work treats detection of scene change as an *alignment* problem. The underlying assumption is that the scene appearance itself is constant, but that the camera may be moving with respect to the scene. Aligned images are assumed to be samples of the same unchanging scene, so higher tonal resolution, greater field of view, and even super-resolution can be achieved in postprocessing.

The use of alignment in HDR imaging was discussed in the previous section.

Panorama stitching expands the field of view by aligning multiple differently-positioned exposures and compositing the scene portions covered by all the images. Many stitching tools for this and other purposes have their lineage attached to Helmut Dersch's Panorama Tools (PanoTools) [26] which was first released in 1998. The spread of PanoTools technology was hampered by a 1999 intellectual property dispute with IPIX [27] who asserted that in PanoTools was engaged in copyright infringement related to mechanisms covered by their patent US5185667A "Omniview motionless camera orientation system"[28] — a dispute they later retracted due to evidence of

prior work which may have invalidated their core patents — patents which were subsequently sold to Sony during a bankruptcy proceeding in 2007 [29] and expired in 2011. PanoTools technology was subsequently expanded upon by a community of open-source contributors, and has been incorporated into a wide variety of plugins for popular photo editing software as well as standalone tools like Hugin and PTGui. Panorama stitching involves alignment at its highest level of complexity, since factors like lens distortion must be corrected in order for a significantly-shifted scene view to align. Many consumer cameras and cell phones incorporate panorama stitching modes in which a series of exposures are captured as the user pivots the camera and are then stitched into a single wide image.

Although the IMEV model does not recognize scene change the same way, it can solve a problem with conventional panorama stitching: ensuring appropriate overlap between images to stitch. Often, a user will sweep a camera too fast or too slow, resulting in unstitchable images having too little or too much overlap. In contrast, a TDCI capture can be used to generate ideally-spaced virtual exposures that can be rendered as images and stitched by conventional means.

Alternatively, stitching can be used for *super-resolution rendering* that resolves finer details by recognizing fractional-sensel offsets between captured images. A number of well-known techniques have been established going back at least a decade for spatial super-resolution combining data from multiple frames, generally relying on feature alignment and (weighted) averaging and/or confidence modeling, as in [30] and [31]. Images created by "sub-pixel alignment" generally require contrast enhancement because each sensel in a capture overlaps multiple output pixels. Unfortunately, the IMEV approach here would tend to treat subpixel movements as noise, making this type of processing less effective. However, there is a modest potential improvement in effective resolution from the fact the IMEV tends to reduce noise, thus increasing detail contrast at the level of sensels.

Sophisticated Synthesis

Recently, a variety of highly sophisticated techniques using large numbers of exposures and sequences of computationally expensive algorithms to radically expand the capabilities of compact sensors and optics have become popular, particularly in the cell phone market. As an elaborate end-to-end example of such a system in widespread use, in 2019, a group at Google published "Handheld Mobile Photography in Very Low Light" [32] which describes the methods employed in sophisticated cell phone cameras. Cell phones tend to have rich compute resources as compared to dedicated cameras, so the idea is to leverage that to hide the relatively poor quality of the cell phone camera's tiny sensor. Smaller sensels more quickly reach photon shot noise limits in low lighting, so averaging across a captured and aligned burst is a key way to bring more photons into the computation.

The cited work largely avoids CNN-based techniques as they are too computationally expensive for the mobile environment, and instead combines several of the above techniques, performing automatic estimation of the appropriate set of exposures, including an extra factor of device motion limiting viable exposure times, then

performs content-aware alignment and merging. It also uses pre-trained learning-based models for white balance and tone mapping of the composite image, which are relatively computationally inexpensive and unlikely to generate features not in the scene as compared to methods which use "AI" techniques for frame composition. Rendering from an IMEV inherently uses data from sampling more photons, thus, it can be expected that a similar level of sophistication in processing would yield still better results.

Non Frame-Oriented

Why is IMEV processing less common and less developed than frame-oriented processing? Largely because frames are familiar. Correlated samples for the entire scene or sensor area, i.e. frames, are attractive both because they are readily analogous to the behavior of film, and because computer algorithms for the manipulation of the dense matrix representations frames are most obviously encoded as are well understood. However, there are a number of costs associated with a frame oriented model.

The most apparent cost is that unchanging elements must be resampled and retransmitted, wasting bandwidth and limiting maximum sample rate due to time spent resampling or re-displaying this redundant data.

There is a long (albeit not widely adopted) history of attempting non-frame-oriented incremental updates to *display* devices. As far back as 1994 in "Frameless rendering: double buffering considered harmful" [33], the idea that making correlated updates of an entire image is wasteful and prone to creating visual artifacts was made explicit. In the same computational budget used to perform a double-buffering scheme, a system of computing per-pixel differences then updating those that have changed in random order was show to produce a superior visual effect. Even earlier, some character-based "glass terminal" display systems adopted a similar technique to save bandwidth and update time on slow serial links. Most famously, Gosling Emacs used a `redisplay` mechanism based on a dynamic programming approach to the string-to-string correction problem [34] to construct a minimal set of display commands to update the state of a displayed buffer of text between states. This issue is of ongoing concern, as with recent developments in the Linux display stack [35] and algorithms for operating slow-updating displays like Eink [36].

Clearly, an IMEV representation could be used to directly drive a frameless output device.

1.3 New Work

This work is built on the idea that digital cameras can — and should — be operated in a paradigm in which they capture a scene model and render images from it with functional post-capture manipulation of time and gain. In contrast to previous work, the new contributions unique to the work presented in this document center on the following concepts.

The first and most important claim in this work is that the **tonal properties of a rendered image do not have to be determined by the exposure settings.** In most prior work the exposure settings — set at the time of capture — largely determine the tonal properties of the final image. In this new system, the tonal properties are synthesized from a captured scene model and do not have a direct relationship to the apparent capture and exposure parameters. Earlier systems which manipulate tonal properties explicitly manipulate the tonal properties, a process typically known as remapping, but this system produces computationally rendered virtual exposures with their own resulting tonal properties.

The next claim in this work is that one should **capture a model of evolution of scene appearance over time** rather than attempt to generate exposures directly from the scene. The model used here is the TDCI IMEV discussed in section 1.1; this work was part of and extended the concept and tooling for TDCI. The TDCI IMEV model includes a notion of the error bounds due to sampling. The content of the model can be thought of as a probability density function (PDF) [37] of the value at the site — the value at the site is a time varying value with a time (and content) varying error bound.

This work particularly investigated efficient ways to capture high quality IMEVs by reprogramming extant camera devices. TDCI, as originally envisioned, required a new and unique style of capture device which has not been realized, this work focused on trying to marry TDCI and its potential to conventional imaging sensors. Though much of the experimental tooling ingest video files, video makes a poor IMEV. A video is made up of a series of frames: individual correlated samples of the average value of the scene over a certain period. This has several disadvantages; a video does not make use of a noise model, so there is no distinction between changes due to noise and changes in scene content (though a reasonable noise model for the circumstances of capture can be generated from a video, ideally of a static scene). Video also does not accommodate irregular sampling: the entire scene is updated at some constant frame rate, meaning sections of the scene which are slow moving relative to the frame rate are redundantly sampled and stored, wasting bandwidth and therefore limiting the frequency with which high-frequency content can be updated.

In particular, this work **identified required changes to camera and imaging pipelines to support the capture and rendering process.** While previous TDCI work has contained a variety of proposals on methods for capture and rendering, this work contained a deep investigation of how the structures of conventional cameras can (and cannot) be adjusted to operate in a TDCI-type context. This work operates in a context of capturing and rendering from a model of uncorrelated per-pixel (rather than per-frame) changes. This allows both an elegant mechanism for many operations which are analogous to conventional photographic and image processing tasks — as well as a variety of new manipulations (described below) which are impractical or impossible in the context of correlated frames representing average values over a sampled interval. Much of this work focused on exploring, documenting, and attempting to modify the behavior of "prosumer" grade commercial camera bodies, described extensively in section 3.12.

The connection between capturing a scene model and generating images with tonal properties independent of the capture process is to **render images by computationally sampling the IMEV.** Essentially, rather than integrating the incident light over an interval to produce a frame at the time of capture, the independently sampled incident light at each point can be computationally integrated after the fact, allowing a much higher degree of flexibility and — critically — the ability to repeatedly re-render the same span of scene content with different settings. This allows the user to expose, evaluate, and re-expose all or part of a scene an indefinite number of times, after the time of capture, until they obtain the desired tonal properties.

Although earlier TDCI work specified methods for generating virtual exposures from an interval of an IMEV, this work extended the sampling mechanisms used in the rendering to support arbitrary functions over time and space, which need not be physically realizable by a camera mechanism. Features enabled by functional specification of exposure enable not only behaviors analogous to those achievable with post-processing of one or more conventional exposures but features which are not possible with a conventional system.

On the conventional end, functions describing arbitrary shutter speeds with arbitrary start-times and gains within the interval captured by an IMEV allow the user to tune in the desired instant and exposure of an image. Specifying multiple such exposures for the whole scene creates images much like those produced by conventional multiple exposures or various sorts of merging of independent exposures. Specifying different exposure functions for different spatial areas of the scene allows a super-set of conventional bracket-and-stack processes. Where conventionally the constituent images are restricted to composing a fixed set of non-temporally-overlapping exposures captured in series at the time of capture spatially varying functions give localized choice of effective instant, shutter speed, and film speed for arbitrary portions of the scene over *potentially overlapping* time intervals, and allow the choices of both area and exposure to be set and re-set an indefinite number of times after the fact.

Other features — like negative weights (which *remove* the contributions of incident light during some interval) and time-varying gain (changing the virtual film speed during the interval the virtual shutter is open) — are flexible in ways not closely analogous to photochemical photographic processes, and in ways not even achievable with frame-oriented photo editing software.

Certain benefits are also conveyed by the process of encoding an IMEV; being able to recognize that individual sites appear constant for multiple samples allows for value refinement within a PDF, subsuming many of the de-noising features often applied to frames.

Finally, this work has **demonstrated the feasibility of post-capture time- and space- varying gain manipulation.** Two generations of prototypes implementing the above ideas, discussed in section 3.10 and section 3.13 have validated that it is possible to build an imaging pipeline based on the principles described above, and having done so, that the promised features work as described. A number of sample images rendered from the latter prototype, and the exposure specifications used to generate them, are provided in subsection 3.13 to illustrate both the method's

suitability as an elegant implementation of many existing photographic practices and several new techniques.

During this work, a number of changes to the terminology around TDCI have emerged. Previous work did not employ the Image Evolution Model (IMEV) term, using a variety of phrases like "TDCI stream", "scene model", "scene appearance model", "model of scene evolution", etc. Because much of the work presented here focused on making more sophisticated use of IMEVs, a more descriptive, more compact, and more consistent term was needed, and IMEV was selected as the most appropriate.

Earlier work on TDCI, as in many discussions of camera systems, has referred to the process of converting incident light into an image has been referred to as "integration." This terminology is pleasing for a number of ways; it suggests the mathematical term integration — as in area under a curve — which roughly describes the way in which gathered incident light over a period of time, as determined by the shutter, is converted into an image, modulated by a gain function (colloquially, film speed). Much of the current work documented here has taken the approach of viewing TDCI as a variation on a derivative sensing system; that is, a sensor system that records the derivative of the phenomena being sampled rather than directly capturing the values.

A number of imaging devices which could be described as "derivative cameras" have already emerged, such as event cameras, which are discussed further in subsection 2.1. Specifically, the TDCI sensors which have been proposed or simulated approximately record the *second* derivative of the incident light — each record is an update to the rate of change of the incident light at a particular sensel. Because of the pixel error model central to TDCI, strictly each quantized pixel is value represents a non-uniform quantization of a random process — reasonably approximated by a Poisson process as is frequently used to model photon shot noise [37] — and is therefore more closely the second derivative of a continuous random variable. This choice of terminology and framing was not used in earlier work, but has proven useful useful in thinking about the action of various camera-system components.

Chapter 2 Background

Because this work is largely an exploration into the question "What if we stop treating digital cameras like film cameras?" it is important to establish a baseline model of camera behavior and terminology. The premise of this work is that a number desirable properties achievable in a digital camera system are obscured or obstructed by assumptions carried over from film cameras and affordances for feeble computers in the early years of digital imaging. Such a situation is not surprising; permanent photographic processes emerged from several sources in the 1830s, and the first primitive digital image sensors only appear in 1969 [38]. This hundred and forty year interim provided ample time and market penetration for techniques, terminology, and practices to develop and refine into a powerful cultural inertia.

A digital camera consists of a number photosensitive sensels, an interface for reading out those sensels, and an attached computer system to control the readout and to process and store the resulting data. These sensels and read-out mechanisms may be constructed in a wide variety of ways, but the vast majority of them share a number of properties, and many of those properties are distinct from the behavior of photochemical film. First, sensels are relatively independent. While many cameras share one analog-to-digital converter across a section of sensels, digital cameras are not bound by the restriction that the sampling of the entire frame must be simultaneous, as film cameras are. In fact, excluding a small but growing number of global shutter cameras, the majority of cameras do not have precisely correlated global readout. Furthermore, digital cameras do not require that the readout of the sensels be *correlated*; different sections of the sensor/scene may be sampled not only at different times, but with different parameters to suit the brightness, motion, or other scene properties.

The control mechanism also distinguishes modern cameras from their mechanical, photochemical predecessors. A basic film camera fundamentally has three relatively coarse controls; the sensitivity (film speed) of the installed film, selected from a set of manufacturer options and set long before the time of capture, the shutter speed, and the aperture of the lens. While film cameras picked up various sorts of automation, like several varieties of automatic exposure control, over the decades, the presumption is that those parameters are controlled by a human operator to produce approximately the desired image "through the lens," or automation designed to mimic what a human operator would have done.

Finally, digital cameras do not require that the measurement of incident light and the generation of an image from that sampled data be coupled. In a film camera, those processes are very closely coupled; the photochemical processes in the emulsion at the time of exposure closely bound the resulting image. Some after-the-fact control is possible by manipulating the development process, but the resulting image is largely restricted to relatively small changes effected by relatively blunt instruments like dodging and burning with hand-cut masks.

2.1 The Camera

Because much of this work is focused on new modes of operation enabled by digital cameras, and the persistent assumptions that impede their use, it is important to ground the behavior and terminology around camera systems. A cameras is comprised of a photosensitive element to record incident light, an optical lens to form an image on the photosensitive element, and a shuttering mechanism to control the exposure of the photosensitive element to the image projected by the lens. Particular cameras can vary widely in how these elements are constructed, and which and how parameters of the system can be manipulated by the end user. Relevant details of the construction of these elements are discussed in this section.

Optics of Lenses

While optical lenses themselves are an area of considerable study, for the purposes of imaging work, a small subset of optics is useful in imaging applications, and specifically toward the proposed work. This focuses on the bulk characterization of optical assemblies — primarily the sort sold as photographic lenses, which are somewhat confoundingly also referred to simply as lenses.

Lenses can be characterized in a variety of ways, most straightforwardly by focal length: the distance between the point of convergence and the sensor plane, measured in millimeters. Longer focal lengths result in a more "zoomed in" image for the same other parameters.

Another basic lens property is "lens speed;" the ratio of the system's focal length to the diameter of the entrance pupil — essentially, the amount of light admitted by the lens. Faster lenses represent a relatively larger entrance pupil, and hence admit more light. Lens speed is generally expressed as "f/numbers" or "f-stops," the reciprocal of the relative aperture. This encodes the amount of light admitted into a geometric series of powers of the square root of two.

More formally, optical systems can be characterized by a set of mathematical functions. The image of a point-source passed through an optical system forms the PSF (**P**oint **S**pread **F**unction), essentially the impulse response of the lens. The PSF for a system, or at least a good approximation thereof, is easily obtained by simply imaging an approximate point source with the system in question. Mechanically, an image is the sum of all non-occluded portions of the point spread function, for all points of light in the scene. More easily applicable to imaging is the OTF (**O**ptical **T**ransfer **F**unction), which represents the response of the optical system to sine-wave input; formally, this is the Fourier Transform of the PSF. The OTF is a complex-valued function, but for imaging applications the more convenient real-valued MTF (**M**odulation **T**ransfer **F**unction), formally the absolute value of the OTF, is used, justified by an assumption of (approximate) radial symmetry.

Legacy of Film

Many photographic conventions were set in the pre-digital era, and have been carried into the digital age. Most of these carryover assumptions are reasonable and contribute to the ergonomics of camera use, but many others impede or mislead productive use of modern digital-sensor based cameras. Even years after the transition to more flexible digital sensors, cameras are still designed and operated under assumptions adopted during the film era.

In a film camera, the image is captured by a timed exposure of a surface doped with photosensitive chemicals. The details of the various chemistries used for this purpose are dramatically out-of-scope for this work, a number of the behaviors of this medium, and the parameters used to characterize it, have survived into the digital age. More details of the relevant parameters are detailed below in the discussion of the APEX system.

A number of systems existed for describing the sensitivity of photochemical emulsion film — early systems tended to be specific to a vendor, but eventually standards such as the ASA scale (formally ASA Z38.2.1, later refined into ASA PH2.5 and ANSI PH2.21 for color film) and eventually the ISO 5800 (for color negative film) and related ISO6 and ISO2240 for black-and-white and color negative film respectively. The concept of "film speed" or "ISO" in a digital context is a convenient abstraction for gain, but is chiefly an analogy — a somewhat stretched analogy the rather complicated mathematical details of which are established by ISO 12232 [39] — to allow techniques developed for film exposure to apply to digital cameras.

Another assumption from the pre-computer era which continues to hold sway over camera design is the idea that images, as captured, should be suitable for human consumption. In digital imaging systems, this is often not the case; the consumer of the images is likely to be a computer, which may be programmed to extract specific information from the image, rather than a human who attempting to interpret the capture. Even if the image is ultimately intended for a human viewer, a large degree of computational post-processing is *mandatory* to convert the digitized, color-mosaic image data into an image a human can view, and the collection of parameters and defects which can readily be manipulated in this rendering process are different and much broader than what could be readily manipulated in a chemical exposure process. This leads to unfortunate choices in optical systems; for example, when designing a system intended to capture text for character recognition, designers may choose a well-corrected lens which makes compromises in terms of sharpness in order to produce a more-pleasing image, while the OCR system would be better served by extremely sharp images with situationally harmless but visually displeasing color fringing.

Without this presumption, it starts to become possible to build different kinds of camera, which are better suited for capturing information about a scene, rather than pleasing images. Pleasing images can then be synthesized after the fact to create the desired image or images, with the ability to manipulate exposure parameters in ways that are traditionally set at time of exposure, or not manipulable at all. Sensors could be placed behind lenses designed to perform specific functions (computations) rather than capturing a perceptual model of the scene. For example, a lens could be designed

to project regions of a scene down to a small number of points aligned with the sensels of a very low-resolution sensor. Each sensel can then be read out for changes, resulting in segmented motion detection in the scene without any computational image processing, or even, strictly speaking, imaging.

The most pernicious of these assumptions is that a capture must be a single, uniform integration of the received light across the whole sensor area during a uniform interval. None of the above is true of digital sensors; it is possible and often desirable to expose different sections of a sensor for different intervals, compose an image from multiple samples, integrate after the fact, integrate over different intervals for different parts of the frame, or even weight integration with one or more functions in time over the scene. Occasionally, these assumptions are conditionally violated; HDR (**H**igh **D**ynamic **R**ange) images often use different integration intervals for different parts of the scene, though typically this is performed by taking a sequence of images with different exposure settings and selectively composing them after the fact, rather than by controlling or sampling the sensor in a non-uniform way.

Photographers will sometimes also deviate from simple single-exposure captures of relatively-static images for artistic and/or documentary purposes. Multiple-exposure images were used in early scientific chronophotography to capture motion in a single scene, such as a series of overlaid exposures capturing intervals of an animal's gait. In modern times, a similar effect is sometimes created by after-the-fact layered multiplication of several separate exposures rather than by repeatedly exposing the same photosensitive surface, such that the image remains normally exposed, but changed portions of the image are overlaid, creating a time-lapse effect, and/or allowing the same subject to appear multiple times in the same image. The other common use of multiple-exposure images is for artistic effect; multiple exposures can create translucent portions of images, mask of portions of one exposure by saturating them with another, or other more complicated composition performed by controlling the scenes and photographic parameters at the time of exposure. Typically, these long or multiple exposure images imposed on a single photosensitive surface are not fully separable after the fact like a series of independently exposed frames or TDCI capture would be, as information is lost by saturation or occlusion.

APEX

The APEX system (**A**dditive system of **P**hotographic **EX**posure), designed in 1960 for use with monochrome film and encoded in ASA PH2.5-1960 [40] is still, with minor modifications, the dominant method for discussing exposure parameters. The APEX system is based on the equation

$$\frac{A^2}{T} = \frac{BS_x}{K}$$

where

- A is the f-number, the reciprocal of the relative aperture.
- T is the exposure time, the 'shutter speed', in seconds.

- B is the average scene luminance, the 'brightness', in foot-lamberts.
- S_x is the photographic sensitivity of the medium, the 'film speed', in the ISO system.
- K is the light-meter calibration constant, in cd/m^2

The full equation is rather complicated for manual field use, both due to arithmetic intensity and behavior which will provide several equivalent sets of parameters any of which which may or may not be achievable with any particular camera system. Because of this impractical complexity, some set of simplifications are typically applied in practical applications. One option is to use a mechanical or electronic calculator capable of producing suitable settings with some parameters fixed, as with a particular lens or film speed.

To simplify the arithmetic and range of choices, it is common to collapse the various parameters into a single *exposure value* (E_v) equivalent to the log_2 of either side of the equation,

$$E_v = log_2 \frac{A^2}{T} = log_2 \frac{BS_x}{K}$$

Another simplification, known as the "additive" or "logarithmic" system not only takes the log_2 of either side, but separates the fractions such that the whole calculation can be performed with addition:

$$E_v = A_v + T_v = B_v + S_v$$

where

- E_v is the exposure value, as above.
- A_v is the aperture value; $A_v = log_2(A^2)$
- T_v is the time value; $T_v = log_2(\frac{1}{T})$
- B_v is the speed (or sensitivity) value; $A_v = log_2(NS_x)$
- S_v is the brightness (or luminance) value; $B_v = log_2(\frac{B}{NK})$
  - N is a constant which converts between ASA arithmetic film speed (S_x) and speed value (S_v), $2^{-7/4}$ (approximately 0.30)
  - K is the reflected-light meter calibration constant

Photographic systems with computer control, especially digital cameras, the computation is often further simplified into the APEX96 system, which multiplies standard APEX values by 96 to allow exposure calculations to be easily and accurately performed using only integer math.

The Exif [41] system which is used to include metadata in image files uses the APEX values to encode exposure parameters, albeit with occasional marketing-related inaccuracies in certain values, or strange selections of the K and N constants by some versions of the standard and/or manufacturers.

Sensors

Modern cameras almost exclusively capture with analog sensors whose values are digitized. These sensors are comprised of an array of photosensitive cells (sensels) which convert incoming photons into electrical charge, and ADCs (Analog to Digital Converters) which convert the charges imparted to the sensels into digital values. These basic parts are typically accompanied by a variety of filters and other support elements, forming a sensor stack.

On the sensor, each sensel accumulates charge from photon interactions, which ideally accurately samples a scene. A number of effects can interfere with sampling accuracy, particularly in extreme lighting conditions. If the number of photons striking a sensel is extremely high, a sensel may saturate, reaching the maximum amount of charge it can store, thus clipping signal, or even leak charge into adjacent sensels [42]. If the number of photons striking a sensel is extremely low, the sampling can be overwhelmed with noise from the sensor, such as inaccuracies in the charge handling or ADC, or simply *photon shot noise*, the natural statistical variation in emission rate. The existence of photon shot noise requires that many photons be captured to accurately sample the scene, as the variation may otherwise distort the signal.

There are two major ways in which sensors are constructed; CCD (**C**harge **C**oupled **D**evices) sensors, and CMOS (**C**omplimentary **M**etal **O**xide **S**emi **C**onductor) devices. CMOS sensors are often referred to as "active pixel" sensors by contrast to CCD's "passive pixel" mechanisms, though it is possible to build passive-pixel sensors consisting of only a photodiode and selection logic from CMOS technology.

The basic action of a CCD was designed at Bell Labs in the late 1960s, initially for use as a digital memory device [43], but by 1970 had been adapted for imaging applications there, primarily by the work of Michael Francis Tompsett [44].

A CCD sensor is comprised of one or more rows of PN junctions; silicon structures of p or p++ doped silicon, covered by thin layer of n-doped silicon, separated into individual regions. Each of these regions is exposed to light, and acts as a capacitor or charge-well, converting arriving photons into charge via the photoelectric effect. Between each pair of wells sits an electrode contact, separated from the doped layers by an insulating layer of SiO_2. In order to read the charges collected in each well, the electrodes can be energized in sequence, creating potential wells into which electrons from the un-energized neighboring region will flow, in effect allowing the sensor to be used as an analog shift register. The output of the shift register can then be sampled by a relatively large and sophisticated readout mechanism, in digital sensors this will be an ADC (**A**nalog to **D**igital **C**onverter) possibly in combination with an amplifier.

Many commercial CCDs use a more sophisticated architectures in order to better suit imaging applications. Some CCD sensors are designed with one set of exposed PN junctions to accumulate charge, and a second set of junctions blocked from receiving light, into which the entire set of sampled charges can be shifted in a single operation. These may be constructed in several ways, such as with a contiguous sensing area and storage area of equal size, or with alternating rows of imaging and storage elements. This latter interline approach allows for faster readouts with fewer transfers but reduces the fill factor to around 50%, making such sensors prone to spatial artifacts

or requiring the introduction of microlenses and/or an anti-aliasing filter to redirect the light striking the surface uniformly into the sensitive regions.

CCDs with separate sensing and storage elements offer an advantage in that they can be used without a mechanical shutter, and potentially at very high framerate; the sensor is grounded out, setting the sensor to the dark state, then the exposed junctions are allowed to accumulate charge for an interval. The control electrodes can then be energized to simultaneously shift the entire set of captured charges into the covered row, ending integration.

CCDs have a number of interesting design properties; CCDs are well-suited to ultra high sensitivity applications, as they posses an inherently high quantum efficiency, and the sensels can further be readily modified with features like photo-multipliers in front of the sensing wells, or electron multipliers (essentially avalanche diodes) in the readout path, allowing the reading of charges as small as a single photon. However, CCDs are susceptible 'spillover" effects such as blooming or smearing; if some charge wells become saturated, the electrons will shift into neighboring wells, contaminating the sample. Blooming occurs when the charge escapes to neighboring cells in a direction-independent way, while smearing occurs when the excess charge escapes down the lower-resistance shift path, creating bright line artifacts. The act of introducing the control charges heats the sensor, increasing dark current and hence noise. Thus, CCDs must either be operated with a duty-cycle that allows for cooling intervals, or, for greater complexity and expense but improved performance, be actively cooled.

In contrast to the charge wells employed in CCD sensors, each sensel of an active-pixel CMOS sensor is comprised of a photodiode and some number of control transistors to perform functions such as resetting the charge across the junction, and dumping the charge onto a readout bus. To use the photodiode as a sensor, it is first reverse biased to a known potential using a reset transistor, then exposed. Photons striking the photodiode during exposure will *reduce* the charge on the junction, which can then be sampled to read the amount of light at that location.

Many CMOS sensors are more sophisticated, with the addition of specialized structures such as a pinned photodiode which provides separate sensing and readout junctions, with the charge moved between them via a bias applied by an additional transfer transistor. In addition to providing better isolation, pinned photodiode sensors also have improved noise properties because of their correlated double sampling; readings are taken as the *difference* between the freshly-reset readout diode, and the charge on the readout diode after transferring the sense diode's charge onto it, giving a continuously self-referenced calibration. A common way of classifying CMOS sensors is by the number of transistors per sensel; 3T, 4T and 5T sensors are common, with increasing sophistication, usually leading to a lower fill-factor but better noise properties.

There are two partially-independent size factors in sensors; the overall dimension of the entire light-sensitive area, and the area of each sensel in the area. A small sensor with large sensels will necessarily offer low resolution, but otherwise the two factors are relatively independent, and induce different effects on the properties of

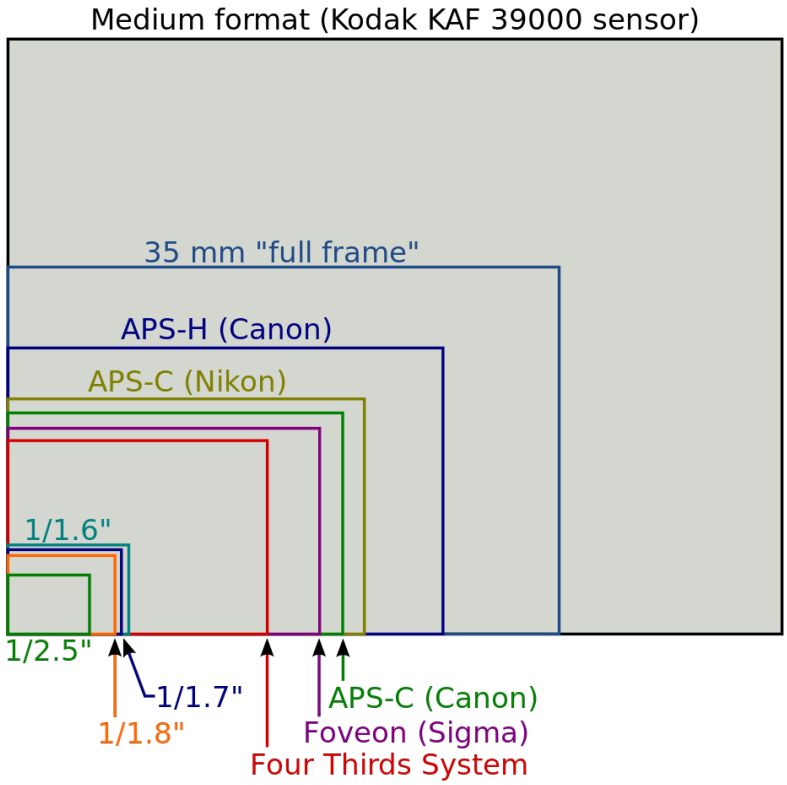


Figure 2.1: Common Sensor Sizes ©Moxfyre/Wikimedia Commons/CC-BY-SA-3.0

the captured image. Larger sensors are obviously physically larger, but also require larger optics to project an image which covers the sensor.

Sensor size is often measured in terms of "crop factor" relative to a "Full Frame" 35mm light-sensitive area of 36mm x 24mm. This convention allows for easier comparison of photographic properties; the effective behavior of a full frame lens attached to a crop sensor body will be multiplied by the crop factor. For example, the common APS-C sensor size of 25.1mm x 16.7mm is said to have a 1.5x crop factor, meaning a 100mm focal length lens for a 35mm camera were attached to a APS-C body, the lens would behave like a $100mm/1.5 = 66.67mm$ lens with the same aperture size. Many cameras and lens systems are marked in 35mm equivalent fields of view (pre-multiplied by the crop factor) rather than actual parameters, which can lead to in comparisons.

Larger sensors will, given the same optics and other parameters, offer a shallower depth of field. This shallower depth of field may be desirable, as in portraits or other tasks with sharply delineated foreground and background, or undesirable, as when capturing landscapes or for gathering the maximum amount of information about the scene.

The measure of resolution itself is more complicated than it first seems. Resolution is, in essence, a measure of the amount of detail in an image. However, resolution can

be measured along a number of dimensions (pixel resolution, spatial resolution, temporal resolution, spectral resolution, etc.), and the methods for measuring resolutions along those various axes may produce wildly different results.

The most obvious measure of resolution for digital imaging systems is pixel resolution; the total number of individually distinguishable light-sensitive dots on a sensor, usually expressed as "pixels." Pixel resolutions are typically expressed either in resolution along the X and Y axis (eg. 640x480, 1920x1080), which also encodes information about aspect-ratio, or simply in terms of the total count, usually expressed as megapixels (MP). While manufacturers often market a camera based on the number of megapixels of the sensor, this metric is easily manipulated, ignores optical effects,and generally does not provide a faithful representation of the qualities of the image produced.

Some of the sensitive area of most sensors is not used for image capture; some pixels around the edge are typically kept dark to provide a black reference, some pixels may be excluded from the processed image due to transformations to correct for lens distortion, and some fraction of pixels will typically be "dead" and not contribute to the gathered data. Furthermore, the stack of filter elements between the lens and sensels (discussed below) will cause some pixels to be ganged together, as with anti-aliasing filters that spread a point of light out across several pixels, or in a color imaging application, the color filters themselves which isolate subsets of pixels to specific narrow frequency bands (colors). In video applications, pixel resolution can be misleading because it is possible not all pixels will be updated in each interval. For example, some encoding schemes *interlace*, updating alternating rows of the image on alternating frames. Pixel resolution can also be deceiving because it is possible to arbitrarily scale digital images with a variety of scaling algorithms which do not add any additional information about the scene (and may create artifacts), but do increase the apparent pixel resolution.

A different method for measuring spatial resolution uses properties of the captured images rather than a characterization of the capture device. For this measure, one determines the smallest features which can be distinguished in a controlled image. The most typical of these measures is line pairs per millimeter (lp/mm); a measure of how many alternating black and white lines can be distinguished at the sensor/film plane. Lp/mm is a very suitable measure for characterizing lenses in isolation, as the test can be performed by simply arranging sample cards with different-density lines and observing the projected image. Unfortunately, lp/mm becomes somewhat ambiguous when used to characterize whole imaging systems, especially digital systems, as the size of the reproduced image will likely be altered from the (possibly unknown) size of the photosensitive element during development in film systems, and will always be altered and resized during reproduction by the display device in the case of digital imaging, making the concept of "per unit length" suspect. Measures of lp/mm in digital systems is further complicated by processing steps in the digital imaging pipeline, which may artificially sharpen or soften the image, experience aliasing at certain spatial frequencies, or otherwise throw the measure. For digital imaging systems, a measure of pixels per inch (ppi) is perhaps the most faithful representation of the informational density of an image. The ppi measure is, in essence, the number

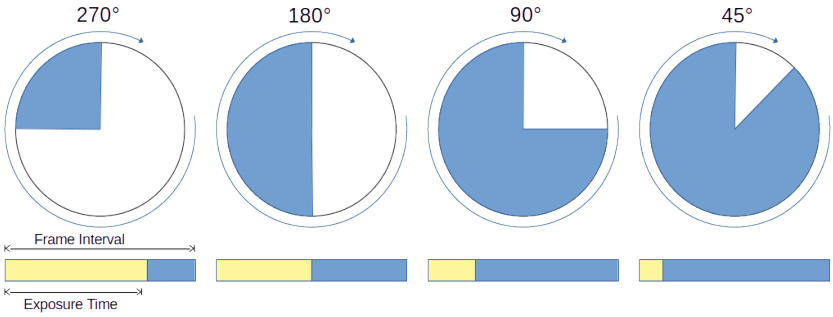


Figure 2.2: The relationship between shutter angle and exposure time.

of *independent* pixel values per unit length. ppi thus encodes the limiting factor on the spatial resolution across the optical elements, sampling device, and processing pipeline.

Other measures of resolution work along other dimensions. Temporal resolution measures the precision of image measurement with respect to time, corresponding roughly with sample rate in other sensing applications. The most common measure of temporal resolution for video capture is Frames per Second (FPS), roughly the number of full-scene updates per second. FPS does not always accurately represent the temporal resolution with which particular details in a scene can be determined. For example, just as for spatial resolution above, some video encodings may not update the whole frame simultaneously. Temporal resolution is also confounded by exposure times. In the common case for a single capture device, frame rate and exposure time are interrelated — the maximum exposure time is the reciprocal of the frame rate, and exposure times shorter than that imply periods in which no information is collected. This relation is described as *shutter angle*. A shutter angle of 360°says that the shutter is open for the entire inter-frame interval, while a 180°shutter angle has the shutter open only for the first half of each interval. Setting shutter angle appropriately is a matter of both the imaging devices' sensitivity to light, and the desired effect. An illustration of shutter angle is shown in figure 2.2.

Some higher-temporal-frequency features may be extracted with lower reliability by analyzing features within a single exposure, most directly through techniques like measuring the spatial dimensions of a motion-blur and dividing by the exposure time. Also like spatial resolution, temporal re-sampling is possible, and unless special efforts are made, can result not only in no additional information, but distracting artifacts. TDCI complicates this measure as the temporal resolution of a TDCI Stream is likely (and preferred) to be non-uniform across the frame, thus defeating many of the uniformity-assuming techniques, and necessitating statistical and/or temporally-integrated measures, which will complicate direct comparisons.

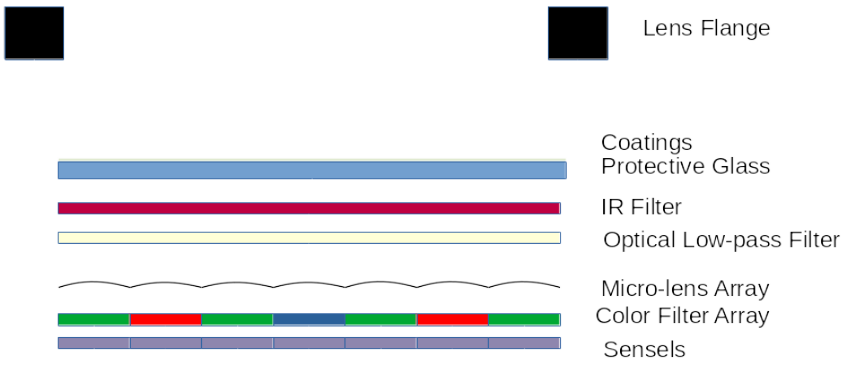


Figure 2.3: A typical sensor stack

The Sensor Stack

Typical imaging sensors are not just the exposed array of sensels, rather, a collection of protective layers, filters, and optical elements are superimposed on the sensor, forming a *sensor stack*. This stack converts the array of photon detectors into an imaging device, and the design decisions in the construction of the sensor stack substantially affect the properties of the data captured by the sensor. Common sensor stack elements in addition to the sensor itself include a protective layer, various blocking filters, a color filter array, a microlens array, and an antialising filter. The specific function of these elements are described in detail in figure 2.3.

The front-most element of most sensor stacks is simply a layer of "clear" glass which acts to protect the other elements of the stack from physical damage. This layer is typically 1-4mm in thickness [45] for commercial cameras.

Another critical layer in the sensor stack are band-pass filters which block or attenuate certain frequency bands of light from reaching the sensels. Infared blocking filters are included in most visible-light cameras to compensate for the natural sensitivity peak of the Silicon they are constructed from in the near infrared (around 750nm) [46] band. This biases the sensor toward human visible portions of the spectrum, which is desirable in sensors capturing image for human viewing. Sensors in applications other than capturing scenes such that the images match human perception, such as astrophotography, low-light imaging ("nightvision"), or in use in FTIR (**F**rustrated **T**otal **I**nternal **R**eflection) touch surfaces often omit this filter, as the infrared sensitivity is desirable in these applications. Other bands may be excluded according to application, either in the sensor stack or, more often, using removable elements mounted to the lens assembly. For example, UV blocking filters may be added when photographing in bright sunlight to lessen the appearance of fringing due to Ultraviolet light refracting spatially away from the well-corrected visible bands. Clever applications of a variety of band-pass filters in conjunction with image processing techniques are important to Multispectral Imaging.

For color imaging, a color filter array is superimposed over the sensor to allow for separation of color data, and hence capture of color images. Each individual sensel in

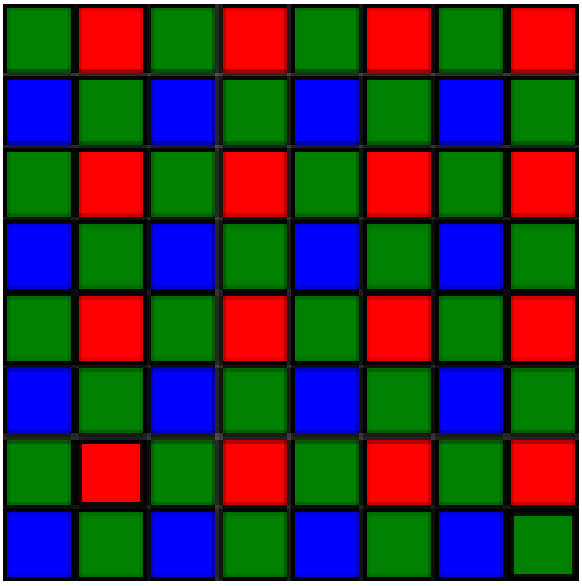


Figure 2.4: Bayer-type CFA pattern

conventional sensors is only an photon collector; it cannot distinguish colors without external filtering. In most camera sensors, a *Bayer filter* [47, p. 3.2.2] is superimposed over the sensor, consisting of alternating stripes of green/red and green/blue alternating color filters. This pattern is chosen largely on human physiology grounds; human eyes are most sensitive to green light, peaking somewhere around 550nm [48], so favoring green in the color filter array maximizes the capture of detail. A typical Bayer pattern is shown in figure 2.4. Other patterns may be chosen to optimize particular spectral properties; some cameras use various RGBW arrangements in which a standard Bayer pattern has some unfiltered senesels interspersed to increase the total amount of light admitted and hence improve overall sensitivity. Similarly, some sensors have been designed with CYYM or CYGM filter arrays filtering for secondary colors in order to, again, increase the total admitted light. Other cameras may have additional filter channels to alter the In many applications, the green channel of a Bayer-filtered sensor is an adequately close approximation of the luminance to be used directly as an approximation. Several well-known image processing tools weight Red 0.3, Green 0.59, and Blue 0.11 by default when performing conversion to grayscale [49].

Various *demosaicing* techniques are used to convert the interlaced color information into a full-color image. These techniques will necessarily cause some aliasing, blurring, or other delocalization of color data and/or a dramatic reduction in image resolution. An example extremely simple demosaicing algorithm would simply assign the value from the nearest neighbor sampling a particular channel to that channel in each pixel, which is straightforward to implement but tends to "smear" colors, particularly where there are sharp gradients between regions. Another simple example method would generate single colored pixel from each 2x2 block of sensels, deriving the red channel from the red-filtered sensel, the blue from the blue-filtered sensel, and the green from the two green-filtered sensels. This method is also conceptually

straightforward, but rarely used because it cuts the image resolution by a factor of four, and is still prone to color corruption. In most applications, more sophisticated demosaicing algorithms are applied, with a minimum ante of bilinear interpolation, and a tendency toward more scene-aware techniques using pixel correlation or grouping to minimize color smearing.

Demosaicing is known to be a dual-problem with *super-resolution* [30], a term covering any technique that improves the effective resolution of an already-captured image via post-processing. Early promising experiments for temporal super-resolution [5] with TDCI streams are likely to also yield effective temporal demosaicing techniques.

Most image sensors include an antialising filter somewhere in the filter stack. This filter, which is sometimes described as an optical low-pass filter, acts to spread each point of light out over a region of the sensor, allowing for more accurate sampling. The primary need for an antialising filter in a camera sensor stack is to prevent moiré patterns; high-frequency repeating patterns in an scene beating with the placement and spacing of sensels in the sensor layer. This spread also helps to compensate for the fill factor of the sensor in another way; by spreading each point of light, features go undetected by "sliping into" the non-photosenstive regions of the sensor. There is a significant tradeoff in the degree of the antialising filter; a too-aggressive AA filter will limit the spatial resolution, visibly blurring the captured image, while a too weak AA filter will still allow certain scene content to produce aliasing artifacts. An ideal antialising filter will eliminate all spatial frequencies above the Nyquist frequency of the imaging system, while faithfully passing all below-Nyquist features. Antialising filters are typically implemented with several layers of birefringent material [50] inserted into the sensor stack. Birefringent materials have a refractive index which is dependent on the direction and polarization of light, splitting the beam into two spatially separated components. Careful selection and arrangement of several birefringent layers produces a filter which splits an incident beam into a desired number of components; typically four to cover a cell of a color filter array; and whose dispersion distance matches the Nyquist frequency of the underlying sensor. In most practical sensors, the antialising filter is implemented in several layers of Quartz or Lithium Niobate [50].

The depth of the optical path in the sensor stack creates a variety of surprising artifacts. These artifacts may compromise image quality in a variety of ways. The physical thickness of the sensor stack begins to have effects even before the light reaches the device. In an increasingly common case, cameras employing an electronic first curtain shutter suffer from gradients in the exposure, as the electronic first curtain is in the film plane, and thus not subject to angular selectivity, while the mechanical second curtain is in front of the film plane, by at least the thickness of the sensor stack and a safety margin, and thus selectively admits off-angle rays. This (and several related) phenomena were documented in a paper I co-authored [4].

Deeper in the sensor stack, in order to compensate for the fill-factor of the sensor and the depth of features above and through the light-sensitive area, the microlens array will attempt to focus light into light-sensitive areas, but irregularities and light leaks in system, as well as intentional cross-contamination from the antialising filter,

creates contamination and dependency between neighboring sensels. These artifacts may bypass the color filter array, if present, further complicating the contamination. Even lower in the stack, physical structures in the manufacturing of sensels may interfere with neighbors; for example, the partially occluded sensels used for phase-detect auto-focus [51] in many sensors have a metal layer on top of the sensitive area to accomplish the occlusion. In addition to receiving less light than regular sensels, requiring the data from these cells be processed differently or discarded and interpolated over, this metal layer can shade or reflect additional light into neighboring sensels, creating line artifacts.

ADCs

Once incident light has been converted to an electrical charge, those charges must be read out as quantized values. This conversion is done in one or more **A**nalog to **D**igital **C**onverters. Typically, the ADC component in a imaging sensor will include some amplification or other prepossessing, but these can safely be regarded as part of the conversion process. There are a number of design decisions around the ADCs in a sensor, setting aside implementation details of the ADC itself, which is out-of-scope for this work but well covered elsewhere [52].

Different sensor designs will include different numbers of ADCs; some sensors, especially CCD sensors, may use a single ADC, shifting rows of (analog) charges 'down' sensel rows, then 'across' a readout row to a single converter. This method is relatively slow, and the extended analog path can introduce noise from the charge transfers, but the use of the same ADC device for each conversion avoids artifacts due to differences between the ADCs, or calibration and post-processing steps to normalize them. Others sensor designs, typically CMOS sensors, are read out with an ADC per-row or per-column. This configuration allows for much faster readouts, as the conversion process is parallelized, at the cost of considerably more ADC hardware, and a slight loss of consistency as different ADCs may exhibit minor performance variations.

Typical sensors are not designed with per-sensel ADCs, as a matter of cost and complexity, though some of the exotic and/or specialized sensor technologies discussed below may approximate independent per-sensel sampling. For example, designed-for-TDCI sensor would not employ any conventional ADCs, but would detect charge threshold crossings at each individual sensel.

There is another design compromise that arises a consequence of the row or column-oriented readout in the vast majority of sensors, between *rolling* or *global* shutters.

In global shutter mode, the entire frame is read out before any sensels are exposed or allowed to begin integrating. In rolling shutter mode, rows/columns that have already been read out are exposed while the readout process continues, such that each sensel is exposed for the same amount of time, but the exposure intervals are staggered across the frame. Rolling shutter allows for higher frame-rates, and a slight reduction in noise due to analog effects on captured charges at rest, but at the cost of potential distortion. The most severe (and intuitive) of these distortions is that

fast moving objects may "smear" in an image captured with a rolling shutter, such that objects moving along the axis of readout will appear elongated or compressed as consecutive intervals sample the object in different positions. Many cameras can be operated in either mode to suit situational needs, or in combination with an external mechanical shutter as in "electronic first curtain".

A final variation in sensor ADC design that bears mention is that sensors may be constructed with the ADC(s) in the same chip and manufacturing process, or with a separately-constructed ADC, coupled to the sensor at the die or package level. Advantages of on-chip ADC designs include better noise properties, as the length and number of junctions in the analog transmission path increases noise, and a potential cost decrease at volume, as it lowers the numbers of components. Advantages of off-chip ADC designs include being able to use tailored fabrication processes for each component, and a potential cost savings because it allows the use of commodity ADC components.

Other Sensor Technologies

Quanta image sensor

A competing design for next-generation image sensors provides interesting contrast to the design decisions in TDCI. Eric Fossum's QIS (**Q**uanta **I**mage **S**ensor) design [53] operates on a basis of photon counting; each QIS sensel (termed "jot" in their writings) is a detector for a single photon over an interval. Each jot in a QIS sensor array is sampled on the order of 1000FPS, and provides a single bit of output at each sample, 1 for a photon interaction, or 0 for none, which are then collated into a bit field, representing a frame. A series of bit fields are collected into a three dimensional bit field in (x,y,t), x and y being the spatial grid of the sensor layout, and t being the time series of samples. Images are synthesized post-capture by sampling into the data, in a process not entirely dissimilar to TDCI. QIS sampling is envisioned as combining along all three axes; pixels are synthesized by combining an (x,y) region of jots summed over an interval. The dimensions of the region and interval can be controlled dynamically, giving desirable options for compression and extended dynamic range which are impractical with conventional sensors, advantages which are shared with TDCI-based imaging.

QIS jots must be much smaller *and more closely spaced* than the diffraction limit of the optical system in front of the sensor, which poses challenges in both fabrication technology and energy density for practical sensors, as compact energy efficient single photon detectors are not yet practically realizable.

QIS differs from TDCI in several key ways. Most importantly, QIS is still frame-based — each QIS sample is an (approximately) simultaneous sample across the whole surface of the sensor while TDCI imaging operates on a series of waveforms updated only when the rate of change for a particular sensel changes. A QIS sensor must, therefore, be able to sustain the data rate of updating the entire sensor capture at the sampling rate. This produces an obvious saturation risk; if the (probable) arrival

and detection rate of photons is high relative to the sampling rate, the sensor will saturate, capturing no meaningful scene information.

QIS is also based in a different set of assumptions — where TDCI is based on the premise that image data is a model of a consistent physical scene, sampled by observing light reflected or emitted from it, QIS imaging assumes the arriving light *is* the model of interest.

Combining the two ideas above from a different perspective, TDCI is based on photon arrival *rate*, while QIS is (at least in principle) in photon arrival *count*. To give an (intentionally oversimplified) example, if a sensel is struck by a single photon between $t = 0$ and $t = 1$, no photons between $t = 1$ and $t = 2$, and two photons between $t = 2$ and $t = 3$, an arrival-rate sensor will regard the value during the interval from $t = 1$ to $t = 2$ as $2/3$. In a count-oriented sensor, the illumination of the point in question during interval $t = 2$ will be considered 0, giving no information about the scene.

Photocell arrays

A possible implementation of a sensor suitable for TDCI, which is sadly outside the scope of the current work, could be constructed from an array of easily fabricated photocells or photo-diodes, with each diode paired with a small computational element. These computing elements could be extremely simple, essentially only needing a few hundreds of transistors per sensel, as proposed in work on Nanocontrollers in the early 2000s [54].

Event Cameras

Event cameras are another recent sensing technology which bears significant similarity to TDCI designs. Like TDCI, event cameras are based on recording only a stream of records for changes in the scene. Unlike TDCI, event cameras' change records are typically only a time, location, and polarity, while TDCI records additionally include a new value — including an initial baseline sample. This lack of a magnitude update (or recorded absolute magnitude) in the update record has several consequences.

First and most problematic is that means an event camera will produce *no* information about static parts of the scene, so no amount of processing will be able to fill in scene features that do not move during the sampling interval without additional out-of-band sensors. Some experimental event cameras have contained a second set of sensels, a second sensor, dual-mode sensels, or other out-of-band system to establish ground truth [55], but this will typically result in problems in the optical path, such as a low fill-factor because the sensels are larger or interspersed, or a complicated optically-degrading beam splitting solution to create two images. More problematically, correlating and integrating the event stream and frame-based capture data is computationally difficult, and, based on recent surveys [55] not even fully solved in general.

Secondly, polarity-only change records (rather obviously) restrict the information in an event; all an update can tell is that that site has changed by more than a threshold.

The particular thresholding configuration may use a fixed threshold level for the sensor, a configurable (by user dictate or feedback) level for the whole sensor, some manner of automatic feedback in individual sensels, or some combination thereof.

Finally, in a problem shared with TDCI systems, event cameras may suffer from saturation. If a large number of changes happen at the same time, event data will be offset or lost as the amount of data being read exceeds the throughput at which records can be stored. This situation would typically be caused by some sort of correlated change in the scene, trivial examples including things like camera motion or sudden lighting changes. This risk of saturation due to correlated events can be mitigated with techniques such as flicker filters (to eliminate predictable, periodic lighting changes, such as power-line-frequency sinusoids), or more sophisticated scene modeling to describe changes correlated due to camera motion or similar systematic behavior in a higher-level representation.

Philosophically, TDCI and event cameras are rather different approaches — TDCI is largely inspired by designing to suit the readily realizable apparatus of cameras and computers, while event cameras are Neuromorphic in inspiration, and tilt their designs toward similarity to biological vision. This does, interestingly, mean that some research directions explored for event cameras and regarded as "less favorable" for neuromorphic work provide interesting information for TDCI work, like some event cameras designed to provide PPM or PWM magnitude representations in [56].

The current primary player in event cameras is inivAtion AG, formed by researchers at ETH Zurich and the University of Zurich who piloted many of the technologies. InivAtion's DVS ("Dynamic Vision Sensor") products represent the commercial state of the art for event camera systems, and provide a good baseline for comparison. In particular, their state-of-the-art commercial product, the DAVIS346 produces a 346x260 grayscale event record, saturates at approximately 12M events per second, and costs 6600CHF (approx $6687 US at time of writing).

Light Field Cameras

Another family of cameras which allow for a significant degree of manipulation of photographic parameters after the fact are light field (or plenoptic) cameras [57]. Unlike conventional cameras, light field cameras attempt to measure not just the magnitude (intensity) and perhaps wavelength (color) of recorded incident light, but a vector quantity also encoding the direction of the incident light. This is desirable because it allows for the depth of scene features to be reconstructed after the fact either to generate 3D images, or to allow optical or computational adjustment of the focus and depth of focus within a range determined by the optical system in conventional images rendered from the recording.

Broadly, Light Field photography is accomplished by imposing a 2D array of micro-lenses somewhere in the optical stack of a camera — typically between the main lens and photosensitive element — to project an array of micro-images. The

"Integral photography" name derives from the fact that that conventional images are rendered from the gathered light field by summing - in modern systems computationally, though early experiments used lens arrays to do so optically - the contribution from the same relative position in the set of micro-images to render a final image.

There are significant disadvantages to light field cameras, the largest being that the resolution of output images is roughly — and with differences between implementations — bounded by the number of micro-lenses in the capture device. There is also a substantial computational requirement to render output images, a user interface problem in designing tools by which an operator can set the range of focus, and the final focus and depth of field of the output image. Also like the work proposed in this dissertation, this is a set of problems which is considerably outside conventional image processing software tools and conventional photographic practices and intuition, which creates a substantial barrier to entry for making a consumer-facing device.

Light field imaging has considerable historical pedigree, going as far back as 1908 when the concept was proposed by Gabriel Lippmann — best known for his work on color photography based on interference which earned him the the 1908 Nobel Prize in Physics — under the name "Integral photography" in the same year. Lippmann's proposal predated the technology to fabricate micro lens arrays (also sometimes termed "lenticular screens") for several decades, though early implementations began to appear by the 1920s. Once the technology to to mechanically construct large, extremely well matched, and very precisely offset lens arrays to render images from the gathered light field is available, it is possible to impose a matching set of lens arrays in a camera and on a print to produce novelty lenticular prints which appear 3D as the viewer's perspective through the lens array shifts. Later, digital cameras and computer technology allowed the rendering process to be computational rather than optical. Critically, in 1992, Adelson and Wang described a mechanically and computationally tractable design for a light field camera from which depth information can be automatically derived by computationally analyzing the correspondence between the micro images[58]. Reversing the depth estimation process — that is altering the stride by which an output image is summed from the micro images — allows for the focus and depth of field of the output to be altered after exposure, in much the same way the work in this dissertation proposes to do for time and gain.

The most visible modern commercial implementation of light-field imaging technology was from Lytro, a company founded in 2006 by Yi-Ren Ng to commercialize light-field camera work from his PhD work at Stanford [59]. Lytro operated until 2018, and produced two models of consumer light field camera [60] as well as the software to manipulate their captures during that time. A decade and $140 Million of investor money later [61], Lytro produced the Illum, a $1,500 one-trick-pony tech demo where both the device and rendering software were brittle, buggy, and slow, and the target audience of trained photographers who might buy such a device were frustrated by the required changes in practice for composition and processing and underwhelmed by the quality of the resulting images [60]. Of note, such a large fraction of the Lytro staff were hired by Google when the company shut down that there were (later proven false) rumors of an acquisition [61] — likely but not verifiably to

work on the computational artificial depth of field features that appeared in Android camera software around the same time. A few scientific imaging companies, most prominently Raytrix GmBH [62], have been offering plenoptic cameras and proprietary software for rendering images and 3D reconstructions from their streams for a more limited, less photographically inclined audience before and after Lytro's brief presence on the market.

2.2 Camera Pipeline

A typical digital camera contains a small computer, which controls the user interface, camera functions and parameters, storage management, and basic image processing. This computer will govern both photographic tasks, such as auto-focus, mechanical stabilization, and metering; and digital transformations, such as demosaicing, encoding into output formats, and potentially more sophisticated transformations including correction for dead pixels, black levels, or even lens distortion. In-body correction for lens artifacts is most common in fixed-lens cameras, but may also be performed by interchangeable lens cameras using data stored in a ROM in the lens. The integration of these controls into a single device also enables a variety of higher-level features, such as automatically focusing on detected faces, automatically shooting suitably-bracketed bursts to process into HDR images, and a wide variety of other conveniences.

This computer is likely a **S**ystem-**o**n-**C**hip (SoC) design, integrating a processor, dedicated image-processing hardware, and I/O controllers for storage and user interface hardware. In the modern era the host processor is typically one or more licensed ARM processor cores. Many camera vendors have long-term ASIC families for these on-board computers, such as Canon's DIGIC, Sony's BIONZ, or Nikon's EXPEED lines. Most vendors also have long-running operating system families shared along their camera line; Canon ran a system built on top of VxWorks until around 2007, and subsequently switched to an in-house operating system known as DRYOS [63], while Sony runs a Linux-based stack [64].

RAW Image capture

Many cameras have an option to capture sensor data with a minimum of processing. This is a typical feature in "professional" cameras, and is often easily added to other models via software hacks [63]. Primarily, RAW capture avoids the lossy compression methods, such as JPEG (described elsewhere), which will remove some information from the capture, and may cause artifacts. Secondarily, raw captures will lack many of the processing steps normally performed in-body; typically, the image will still have the color filter array pattern un-interpolated giving a "mosaiced" view, contain regions of the sensor not typically included in output images such as unlit black-reference areas, and lack any automated lens or sensor correction, etc. Skipping the automatic execution of these steps in the camera body allows the photographer, or a later editor, to manually control the details of the process, apply more computationally-intensive methods, or extract information which might be lost during conventional processing.

A major issue with RAW capture is that RAW formats are not well standardized; because there is very little processing, most cameras have unique or near-unique output, and require a suitable RAW conversion tool to access the generated files. Many commercial image-manipulation tools have support for a wide but not exhaustive selection of camera RAW formats, and Dave Coffins `dcraw` [65] utility aims to offer an open-source decoder for a nearly exhaustive selection of cameras.

There are also semi-RAW formats, such as Adobe's DNG (**D**igital **N**e**g**ative) format, an open lossless container format which bundles a set of metadata for interpreting the data, based on the TIFF image format, which is itself a specialization of the EXIF metadata standard [41], and under consideration for ISO standardization. Few cameras directly export DNG files, though many vendor-specific RAW formats are also based on TIFF.

Compression

Most digital cameras default output format is, in contrast to the RAW image data described above, a compressed, post-processed, image provided with metadata in a standardized format. The vast majority of cameras use the JPEG lossy compression method, and subsequently export images in EXIF format [41], giving an interchange format which bundles the compressed image data with metadata. JPEG compression is based on a number of human-perception optimizations, whose broad strokes are interesting as examples, but whose exact algorithmic details, choice of constants and methods in the color mapping step, basis functions, etc., are specified by a number of mutually-incompatible standards, and not particularly relevant to the project at hand.

JPEG compression is performed in a gamma-adjusted YCbCr color space, which represents colors as luminance (Y) and Cb and Cr color components, which are the blue and red differences from the luma. This expression is useful for compression as it easily allows the prioritization of luminance (discarding of color) data, more or less corresponding to human perceptual sensitivity. JPEG compression also splits each resulting data component into 8×8 blocks, which are converted into a frequency-domain representation via a discrete cosine transform, which essentially transforms the block into a linear combination of 64 basis functions (patterns). This representation can then be quantized, by dividing each coefficient by a constant (typically a different constant for each component) then rounded, which, since we are operating in the frequency domain, essentially removes the high-frequency components to which humans are least sensitive. This quantization step is the only one which is necessarily lossy, though in practice several of the other steps are likely to lose information due to limitations of the implementation of the arithmetic. The resulting simplified coefficients are then (losslessy) entropy encoded, typically via a mixture of run-length and Huffman encoding.

In addition to the desirable effect of providing an interchange format, and compression in the vicinity of 10:1 with minimal visual artifacting, a number of undesirable things happen as a result of JPEG compression. Common artifacts in JPEGs include

blocky high-contrast edges, color corruption where luminances for different colors are similar, and application of a gamma curve reducing accuracy to 8 bits.

2.3 Scene Model

Image capture techniques can be split along a number of conceptual axes. One fundamental distinction among imaging techniques is separated by whether the objective of imaging is conceptualized as recording photons or or creating a scene model. In a photon-recording model, the objective is to faithfully record the light arriving at the sensor during an interval. This view is is appealing in several ways; it is analogous to film, conceptually simple, and requires minimal computational resources. It also has several serious problems, the most fundamental of which have to do with the properties of light itself. The particle nature of light causes photon shot noise (Also sometimes called "photon noise" or "Poisson noise"): a statistical uncertainty [66].

Photon shot noise is worthy of more detailed explanation, as it is Central to the justification for non-photon-recording sensing; the number of photons N measured by a sensel over time t sampling a scene illuminated such that λt is the expected incident photon count is given by the discrete probability distribution

$$Pr(N = k) = \frac{e^{-\lambda t}(\lambda t)^k}{k!}$$

, a standard Poisson distribution. Being a standard Poisson distribution, its variance is equal to its expected value, $E[N] = var[N] = \lambda t$; or more usefully, shot noise (the standard deviation of the signal) grows with the square root of the sampled signal.

Another weakness of photon-recording is that of lighting/color corruption. The lighting in the scene may interfere with the sampling of the scene. Low or narrow-band lighting in particular can easily cause incorrect representation . In the simplest example, a surface which primarily reflects red light, such as a brick, in as scene lit entirely with blue light will most likely appear black in a conventional image, as none of the available light will be reflected from the surface to the sensor. A human observer is likely to recognize the brick remains red, particularly if the scene lighting is varying, due to their expectation of scene and color constancy. The issue of the relationship between color and light discussed further elsewhere in this document, in subsection 2.3.

In contrast, if the objective of image capture is conceived as capturing an accurate model of the scene, a different set of design choices are selected. In the scene-modeling view, arriving photons are viewed as unreliable, stochastic samples of scene properties. Successive samples are used to refine a model of the scene; an area with very few incident photons can be sampled over several intervals to obtain data that trends over the noise level. An area which is saturated during one interval still has information in the scene model from previous samples. TDCI falls firmly into the 'scene model' family.

A different framing which turns out to result in nearly the same division is whether the sensor is interpreted as measuring photonic arrival *rate* or photonic arrival *count*. This is an important but somewhat subtle point about quantizing photonic arrival

energy at sensels. Traditionally, the charges stored in the sensels are interpreted as photon arrival *counts*; the charge accumulated in a sensel during the sampling interval is directly interpreted as the desired output. This provides a film-analog behavior. In digital sensors, it is also possible to interpret photonic arrival in terms of photonic arrival *rates*; the charge accumulated in a sensel for any given interval informs not only the instantaneous value of that sensel, but also the rate-of-change of that value. This rate-based view requires that multiple samples be considered, and leads to other consideration of multiple samples; averages over many samples can be used to derive information in areas where the instantaneous arrival rates are below the noise level. These multi-sample considerations encode an assumption of scene consistency, and lead to essentially the same behaviors as the scene modelling approach above.

Lighting Model and Scene Appearance

The notion of colors as specific frequencies/wavelengths of light does not correspond precisely to human vision. A human viewers' perception of colors is surprisingly subjective, owing to both the physical mechanism by which human eyes sense light, and . Human eyes contain a number of light sensitive cone cells, in three varieties whose sensitivities have respective center frequencies of approximately 420 ("blue"), 530 ("green"), and 560 ("red")nm wavelength [48]. The output signals from these cone cells are then differentiated by opponent process cells, which also come in three varieties; luminance opponent cells, which are stimulated by all three cone cell outputs, C_g opponent cells, which are stimulated by red and blue cone responses and inhibited by green cone responses, giving red-green discrimination, and C_b opponent cells, which are stimulated by red and green cone responses, and inhibited by blue cone responses, giving blue-yellow discrimination.

This subjectivity carries useful advantages and disadvantages, mostly based on the concept of *color constancy*; a viewer's ability to recognize an object maintains its color despite changes in lighting which can alter the wavelength or quantity of reflected light. One such consequence is Metamerism; colors which can be situationally perceived as the same, despite containing dissimilar spectral power distributions. Metamerism is critical to color reproduction; for any given stimulus, a perceptually identical color can be generated by mixing different intensities of three RGB primary-colored light sources, allowing for compact sampling, representation, and reproduction of a large gamut with a simple three-color representation.

2.4 Image Processing and Compression

TDCI technology shares a variety of assumptions and techniques with existing image and video processing techniques. This section discusses a number of adjacent technologies, and their relationship with the concepts and methods of the TDCI approach. Many of these technologies' commonalities are based in their shared notions of scene consistency.

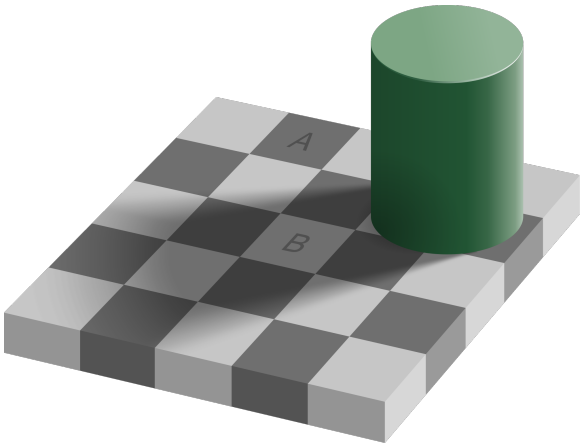


Figure 2.5: Adelson Checkerboard Illusion: Square A is exactly the same shade of grey as square B.

Motion Segmentation

In many computer imaging techniques rely on motion segmentation — the interpretation of sequential scene data into features and trajectories. Motion segmentation is used both for image processing — extracting scene data from images — and video compression, eliminating duplicate information from series of images. The typical form of motion segmentation is *rigid* motion segmentation, the decomposition of a scene into a set of regions or features, and the trajectories of their motion.

Rigid motion segmentation can be implemented in a wide variety of methods, from simple differencing of sequential frames, through statistical classification techniques, wavelet-based methods which distinguish features by scale and frequency, layering techniques, and a variety of other mathematical models. Optical flow, detailed below to provide specific examples, can be used as a form of rigid motion segmentation, typically one which is primarily concerned with the motion extraction function.

Optical Flow

Optical Flow [13] is the study of the apparent motion of objects in a scene, caused by the relative motion of the observer and scene. Optical flow is rooted in psychological, now more trendily referred to as neuromorphic, models to describe motion-sensitive vision in animals by James J. Gibson in the 1940s [67]. After the advent of computer imaging, the optical flow view has become important in its own right, as it allows computers to make inferences about captured image data in ways that are consistent with the physical scene and/or a human observer of the data.

Like the TDCI model, the optical-flow view is deeply rooted in an assumption of scene consistency, and the idea that the sensor is forming a scene model rather than directly capturing reality. For optical flow to be sensible, much less useful, scenes must be comprised of consistent elements, which can be recognized, moving slowly enough that their motion from frame to frame can be computed.

Optical flow can be computed with a large and growing variety of techniques, descending from a wide variety of fundamental approaches including scene differencing, structure matching and tracking, energy or phase models [13]. Many of these techniques extremely well-established, and have been mainstay image processing primitives for decades.

Optical flow is closely related to motion estimation and video compression. In vision-like applications, computing scene and/or camera motion is often the desired data, and other sensing is largely irrelevant. In video compression, optical flow processing is valuable for allowing storage to be traded for computation; it is much smaller to represent a series of frames as a combination of scene elements, some moving along mathematically encoded paths, rather than a series of separately encoded images. These two uses, though rooted in the same mathematical techniques, differ in one crucial aspect; in compression applications it does not matter whether the camera or objects in the scene are moving, while in vision applications distinguishing the two is both desirable and, unfortunately, typically unknowable.

Some sensors are designed specifically for optical flow imaging. By far the most prolific example of such a device is the sensor used in typical optical computer mice. These mouse sensors are extremely low resolution, low depth, but extremely fast downward-facing cameras, coupled with stark controlled lighting, to enable a simple optical flow computation to generate a series of Cartesian offsets, replacing the continuous rotary encoders found in mechanical mice.

TDCI encoding may enable new, convenient optical flow computation techniques owing to the inherent encoding of per pixel time-difference data. Unfortunately the TIK image format family currently in use does not localize this data in ways that obviously lend themselves to inexpensive optical flow calculations.

It would also be desirable that a capture device be able to model camera movement relative to the scene, maintaining the sene model as constant and compensating for the camera motion. Embedding a motion sensor — as is already commonly done many cameras for for orientation detection, for stabilization, etc. — allows a certain amount of automation. Existing optical flow techniques also provide potential options for detecting and compensating for camera motion, but they may or may not be tractable for real-time, in-camera capture, as in-camera compute resources are frequently limited.

Another consistency assumption made in many imaging and sensing applications is that the scene brightness will remain more or less constant for periods much longer than the sampling period. The rise of pulse-width modulation (PWM) controlled LED lighting causes a minor problem for systems that depend on this assumption of brightness consistency, including TDCI. Many modern LED-based lighting systems are controlled by PWM(**P**ulse **W**idth **M**odulation), which produces light-dark cycles at frequencies in the 100-1000Hz range, or with waveforms which are not simple sinusoids or even square waves. As an example, a waveform obtained from a green LED used as a sensor by connecting it across the inputs of a suitably biased Microchip MCP6002 Operational Amplifier, read out with the integrated Oscilloscope of a Digilent ElectronicsExplorer is shown in figure 2.6, illustrating the waveform

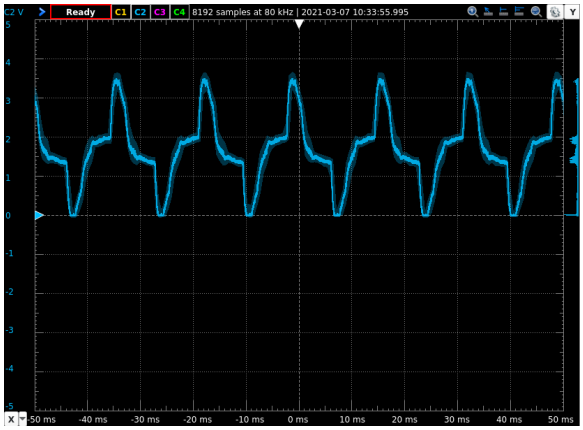


Figure 2.6: Waveform captured from an LED desk-lamp set to less than full brightness. The frequency is around 60Hz, but the shape is decidedly not sinusoidal.

of a TaoTronics TT-DL27 LED desk lamp with controllable brightness and color temperature set to an "intermediate" brightness.

Fortunately, these flickers are are imperceptible to humans due to persistence of vision, appearing as the average value of the lighting, so capturing this flickering is not generally desirable in imaging systems. Unfortunately, whether or not they are a feature of interest, large instantaneous scene lighting changes which, unless expressly detected and discarded, can beat with the sampling frequency and/or dramatically explode the size and frequency of scene-change records in imaging systems. Just as many modern video cameras have flicker elimination settings (sometimes auto-ranging, sometimes specific "50Hz" and "60Hz" settings) for eliminating power-line frequency flicker in various regions as exhibited by florescent lighting, a robust capture system system that samples continuously rather than in a frame-oriented manner will require at least some degree of compensation for these phenomena. The most likely candidate is building a model of the lighting so the data rate does not balloon because the periodic change is accounted for in the encoding. With multiple sources operating at potentially a range of frequencies and waveforms, that may be considerably computationally expensive. A series of brief experiments connecting LEDs with different peak sensitivities to a MCP-6002 Operational Amplifier to observe the waveform of the incident light produced by various common light sources were undertaken, but didn't reveal any patterns worthy of detailed discussion.

Neither TDCI nor IMEV in general manage regular lighting fluctuations in an efficient way. However, all the simple ways to deal with such changes depend on having the scene lighting uniformly following the same cycle, and that is not the case with many forms of LED and other lighting now in use. For example, delying an exposure to hit the peak of a lighting cycle only works if the cycle is simple and repeating. The inefficiency is limited to effective compression achieved in IMEV encoding; the waveform is still encoded and sampling can be manipulated with full after-capture knowledge of the waveform patterns. For example, the work presented in this dissertation would allow virtual exposures to ignore arbitrary portions of the

cycle, constructing exposures using only light from a selected parts of even irregular cycles.

Poisson Image Editing

One of the underpinning assumptions that makes TDCI practical is the notion of scene constancy. Scene constancy is the notion that the space- and time- rates of change in a scene will be slow relative to the timescale and field of view of the image capture. This constancy is already exploited in a number of widely-used techniques. Elsewhere in this document I have discussed the utility of this tendency for video compression, it is also useful for seamless editing. Poisson image editing [68] is a well-established family of image processing techniques which operate based on similar assertions of scene constancy. Poisson editing is somewhat mathematically absurd but readily intuitively motivate-able. Essentially, this process is to guarantee slow, human-imperceptible gradients in intensity [69] on the edges of features, while setting the interior of a region to a specific desired appearance. In the originator's own words, this is accomplished with "Poisson partial differential equation with Dirichlet boundary conditions which specifies the Laplacian of an unknown function over the domain of interest, along with the unknown function values over the boundary of the domain." Fortunately, when discretized (which is the only relevant case) this process reduces to a reasonably manageable quadratic optimization problem.

Chapter 3 Experiments

A number of experiments have been undertaken aiming to explore ways in which modern digital cameras diverge from the abstractions and assumptions under which they are typically operated, and about ways in which those properties can be leveraged to enhance their capabilities. Many of these experiments were published as papers, but collectively they suggest a possibility of re-conceptualizing the use of digital cameras in a way that would allow users to better leverage the properties of the cameras. In particular, changes to the software stack and user interface should allow users to manipulate parameters traditionally considered set at the time the picture is taken post-capture, repeatedly, and with finer granularity than traditional photographic technique.

These experiments are presented in roughly chronological order. Successive experiments informed not only the direction of later experiments, but the overall understanding of the requirements, possibilities, and challenges for a model of photography based on capturing IMEVs and later rendering images from them with arbitrary functional control of the process. These experiments follow two major lines of inquiry. One line was focused on empirically determining the actual low-level behavior of modern camera systems, with an eye toward capturing high-quality IMEVs. The other line was focused on the processing of IMEVs to create images. There was, of course, significant interplay between these two efforts, as findings in one often lead to directions for the other.

3.1 ISOLess

One of the motivations for this work, and the TDCI effort in general, is the observation that many digital cameras appear to be "ISO-Less" (or, more formally, "ISO Invariant") — that is the user-set "film speed" or ISO setting on the camera is partly or entirely implemented as digital gain in the processing pipeline, rather than a change to the gain of the sensor itself. Much of the information about ISO-invariance, especially prior to 2015 when the experiment discussed in this section was performed — came from hobbyist and enthusiast quarters, which tends to be regarded as unreliable in academic circles. This is despite the fact that information from stable pseudonyms on the internet is generally highly reliable, and who has spent a significant amount of time interacting with academic output is acutely aware that results published in academic venues should be considered inflated and untrustworthy until independently verified, as authors respond to the many incentives to hype questionable output. So, the experiment inspired imaging research involving devising and conducting a series of experiments to determine the degree of ISO-Less behavior in a range of consumer digital cameras.

In this series of experiments, 19 different cameras capable of RAW capture released by Canon, Fuji, and Sony between 2000 and 2014 were tested to determine how ISO-invariant their capture behavior actually is. The spread of cameras tested is described in table 3.1 and pictured in figure 3.1. These cameras represent quite a wide spectrum; 11 are CCD based while 8 use CMOS sensors. They operate on RAW formats from

Figure 3.1: The spread of cameras used in ISO Invariance Experiments

Camera Model	Year	Prog	Tech	MP	Min ISO	Max ISO	BPP
Canon G1	2000		CCD	3	50	400	10
Sony F828	2004		CCD	8	64	800	14
Canon A620	2005	CHDK	CCD	7	50	400	10
Canon A640	2006	CHDK	CCD	10	80	800	10
Sony A100	2006		CCD	10	100	1600	12
Canon A590	2008	CHDK	CCD	8	80	1600	10
Canon SD770	2008	CHDK	CCD	10	80	1600	12
Sony A350	2008		CCD	14	100	3200	12
Canon A480	2009	CHDK	CCD	10	80	1600	12
Sony SLT-A55	2010		CMOS	16	100	12800	12
Sony NEX-5	2010		CMOS	14	200	12800	12
Fuji X10	2011		CMOS	12	100	3200	12
Sony NEX-7	2011		CMOS	24	100	16000	12
Canon A4000	2012	CHDK	CCD	16	100	1600	12
Canon EOS-M	2012	ML	CMOS	18	100	12800	14
Canon ELPH115	2013	CHDK	CCD	16	100	1600	12
Canon N	2013	CHDK	CMOS	12	80	6400	12
Sony A7	2013		CMOS	24	50	25600	14
Sony A7 II	2014		CMOS	24	50	25600	14

Table 3.1: Selected properties of cameras used in the ISO-invariance experiments.

10-14BPP, with minimum claimed ISOs from 50-100 and maximums from 400-25600. Their processing pipelines come from different generations and different vendors.

To create an even fairer test, in a subset of cameras where it is reasonably straightforward to do so, images were exposed, digitally boosted, and processed by the same in-camera pipeline as the native-ISO exposures. In the CHDK-enabled Canons, the lua scripting interfaces in [63] expose the controls for the image processing pipeline.

Figure 3.2: JPEG Crops from Canon A4000 at ISO 1600, 16x ISO 100, and 16x ISO 100 Filtered

These facilities include functions to load a RAW into the in-camera JPEG engine with control of the parameters, and a facility for adding or averaging RAW frames, which are all that is required for the experiment. To generate the test sets, exposures were taken at the base ISO, multiplied by successive addition, eg. `i=i+i; i=i+i; i=i+i; i=i+i;` to digitally boost by a factor of 16, then run through the in-camera JPEG encoding pipeline with the parameters set as though it was set to ISO1600. A series of crops from an exposure processed in this manner are shown in figure 3.2. The left-most image is from a native, suitably-exposed ISO1600 exposure, the middle frame is from an exposure shot with all other settings held constant but the gain reduced to ISO100 then the RAW fame multiplied by 16 and rendered through the camera's JPEG engine, and the third crop is the second with a simple de-noise filter applied. The most interesting observation is that the boosted ISO100 image appears to preserve more detail than the native ISO1600 image, but with considerably more noise. If a simple denoise filter is applied, as in the third sample, most of the additional detail is preserved while the majority of noise is eliminated, leaving a image that is not only equivalent to but by most measures superior to the native ISO1600 exposure.

In a digital sensor, the Sv term in the APEX formula should really be split; $Sv = Sv_{analog} + Sv_{digital}$, where Sv_{analog} is a coarse control set at time of exposure by altering the analog gain of the sensels and/or ADC used for readout, and $Sv_{digital}$ is a fine control applied computationally during processing. Traditional exposure and integration are commingled, but to reiterate the point, in a TDCI-type system, they are separated, and the integration step is manipulable in post-processing.

Conceptually, this work exposes that the ISO film-speed analogy is a crutch for simplifying the computations required to generate a good exposure into something a human operator can do on-the-fly, and in particular was designed around traditional sensitized-emulsion with a fixed sensitivity. The typical formula used to determine exposure from ISO film APEX system, described in section 2.1, or one of its integer-math-friendly variants. APEX is an effective way to generate desired exposure for a particular level of incident light, but is effectively a point computation; in scenes

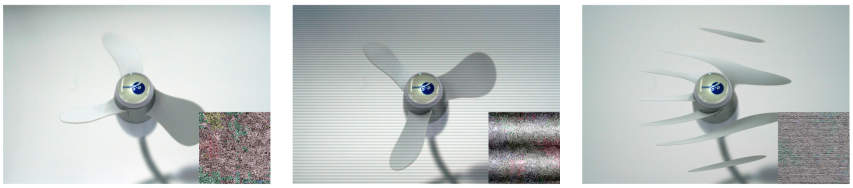


Figure 3.3: A picture of a spinning fan captured by a Sony A7RII under different shuttering modes. Insets show the banding pattern induced by the non-DC lighting interacting with the electronic shutter.

with significant dynamic range, exposing according to APEX requires that the user select the light level at a particular point in the scene or average of an area for correct exposure, and tolerate poor exposure elsewhere — or cover it up later with stitching or manipulation in post-processing.

Some cameras do have controls exposed to manipulate Sv_{analog} at a spatial granularity smaller than the whole sensor at least to a limited degree. In particular, many Canon DSLR models operate on a line-oriented readout, and hackers have determined that the register controlling the analog gain can be altered between line readouts — as in [70] — sacrificing vertical resolution for extended dynamic range. The fact that this method for HDR imaging is effective further demonstrates that the frame-at-a-time APEX style exposure computation inherited from film does not make effective use of digital image sensors.

Contemplating the implications to exposure practices of this work is the genesis for many of the ideas explored in this dissertation.

3.2 Shutter Artifacts

A small related project culminating in a poster at EI2019 explored a number of shuttering artifacts. This paper studied the artifacts created by different kinds of shutters: vignetting from traditional mechanical shutters not being precisely in-plane and having transit times, and non-global electronic shutters' various sorts of temporal and spatial de-synchronization. Some material from this work is integrated into this document in the front matter in subsection 2.1, the full paper is [4].

This work was largely ancillary to the TDCI work, but is relevant in that it was making a close study of the timing behavior of widely-used shuttering mechanisms, an understanding of which is required to properly integrate sensor data into an accurate representation of the scene. It also serves as a further justification for TDCI-style decoupling of capture and integration; constructing a model of the incident light at each point from the available data allows many of these artifacts to be computationally avoided — or conversely, to be synthesized to create the appearance of specific historical capture devices.

A specific set of phenomena investigated in this work is illustrated in figure 3.3, showing a 1/8000s exposure from a Sony A7RII pointed at a rotating fan blade,

Figure 3.4: An image from a Sony A7RII illustrating the effects of the different shutter regimes.

showing both the dislocation of the moving parts of the scene (the distorted fan blades in the third image), and, magnified in the inset, the banding pattern created by the beating of the frequency of the AC light source in the scene and the line-by-line clearing and reading.

Another less well-known phenomena explored in this work is the artifacts produced by operating cameras in electronic first curtain mode. In this mode, the initial zeroing of the sensor is not accomplished by a mechanical shutter, but by electronically dumping charge from the sensels. The sensor is then allowed to gather light for the selected interval, and the process is stopped by a mechanical focal-plane shutter, and the result read out. As shown in figure 3.4, the mechanical focal plane shutter creates shading, as it is not exactly in the image plane (allowing off angle light to encroach), not capable of instantaneous acceleration, and its travel is not correlated with the initial clearing.

Furthermore, this work shows that camera users are already interacting with spatially or temporally non-uniform exposures (discussed in a later section). In fact, it exposes several kinds of currently-tolerated non-uniformity: the incident light that creates a rolling shutter effect is obviously not gathered at the same interval for the entire scene, though in an undesirable way not under user control, rather than configured and leveraged to the user's advantage. These effects are discussed further in the context of desirable non-uniformity.

3.3 Camera Motion

In the introductory discussion about imaging technologies about event cameras, TDCI, and Poisson image editing, problems due to correlated changes are briefly mentioned. These technologies share a reliance on *scene constancy*: the idea that the contents of the scene are unlikely to change extremely quickly relative to the recording mechanism. For the capture devices, these problems take the form of readout bandwidth saturation due to correlated changes, resulting in lost scene data as the update traffic exceeds the read-out bandwidth. These correlated changes can take several forms, such as sudden changes to scene lighting, or motion of the camera itself. These features also cause problems for conventional frame-oriented capture devices, introducing undesired artifacts like flicker or blur. More sophisticated modeling that incorporates lighting or camera motion can reduce the severity of these problems.

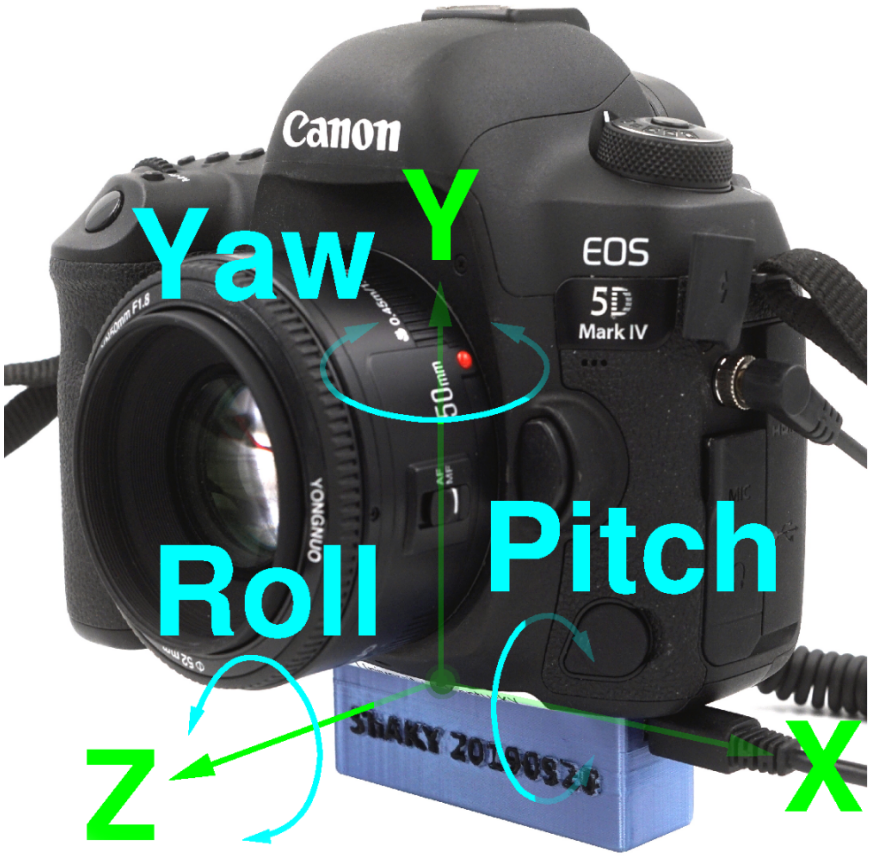


Figure 3.5: Measurement Orientations for ShAKY attached to a Canon 5DIV

As noted in the introductory material, employing some degree of lighting model in the capture device is well established in the form of filtering for line-frequency flicker elimination. More recently, analyzing camera motion in an effort to cancel blur has become common. Many modern digital cameras contain an IMU (Inertial Measurement Unit) tied to a IBIS (In Body Image Stabilization) or OIS (Optical Image Stabilization) mechanism which can move the sensor or lens elements to correct for small amounts of camera motion. The sensors embedded in cameras are not terribly consistent, and the data they generate is not generally stored in a user-accessible way.

Therefore, as a preliminary step to study camera motion, in the summer of 2019 members of the KAOS lab developed a hardware/software system and test protocol to instrument a camera in use, capture detailed motion data synchronized with the exposure along several axes shown in figure 3.5, and convert the data into forms suitable for study.

This system, with the somewhat tortured name "ShAKY" (SHift Angle KentuckY) includes several hardware and software components. On the hardware side, 3D Printed module which attaches to common cameras via a standard $1/4-20$ UNC camera mount thread contains an Arduino Pro Micro ATmega32U4 development board, a MPU-9250 9-axis IMU to read motion, and a 3.5mm TRS connector to interface with the flash sync signal of cameras which support it. An Arduino sketch

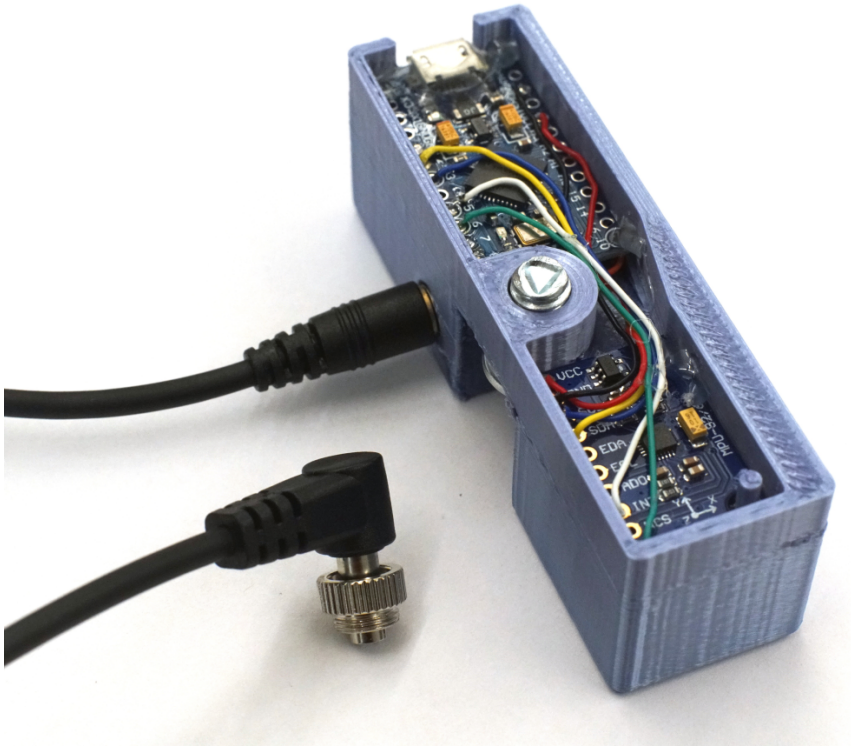


Figure 3.6: A completed ShAKY device, showing the internal electronics

flashed to the Pro Micro to calibrate and read out the IMU and monitor for sync signals, both of which are streamed to a host computer via USB. This module is shown in figure 3.6.

On the host computer, a single-file C program `hostshaky.c`, which shells out to GNUPlot [71] for visualization, performs integration, filtering, and analysis of the raw sensor data into motion data. The processing is performed on a host computer to maximize data-rate; the micro-controller in the camera-attached module is programmed to sample and encode sensor readings at the highest rate sustainable for the device and connection to maximize temporal resolution, while the more computationally intensive analysis happens on the much larger resources of a modern PC.

The experimental procedure is then implemented by a web form displayed on a relatively 4k television set, which presents a questionnaire for relevant details about the camera, lens, human subject under test, and experimental conditions, then encodes them into a QR code integrated into a test pattern for the subject to photograph. This photograph of a QR code is thus inherently tagged by the image content with the experimental data, and by the camera-generated EXIF data with finer details of the exposure for later analysis. An pair of example readouts taken with a Sony A6500 with IBIS and EFC disabled is shown in figure 3.7; the left graph shows the motion with the camera still on a tabletop while the right figure shows motion while hand-held.

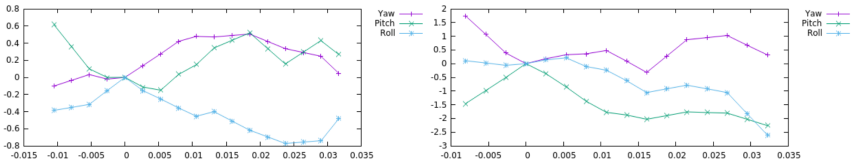


Figure 3.7: ShAKY data comparing still on a tabletop (left) vs. hand-held (right)

Only a small amount of user testing was performed with this device, but the preliminary results do indicate this simple, low-cost, open-source device can provide insights into camera motion. Many of the preliminary findings were not particularly surprising: using a viewfinder (and hence holding the camera closer and making a third point of contact between the operator and camera) roughly halves the amount of measured camera motion. Using electronic first curtain shooting on cameras which support electronic or mechanical shuttering tends to meaningfully reduce shake at the time of exposure.

Other results were more surprising: individual operators are not particularly consistent, the camera itself produces a substantial amount of the observed motion. A particularly surprising finding is that, relative to a single-handed grip, holding the camera with two hands not only does not result in reduced camera motion, but tends to convert less-photographically-problematic x,y,z motion into more problematic pitch and roll.

This work was published as "Characterization of camera shake" at Electronic Imaging 2020 [72].

3.4 Non-Uniform

A major promise of TDCI-like pipelines which has not been previously explored is the ability to produce exposures which are spatially or temporally non-uniform. To define terms, a spatially non-uniform exposure is one where the exposure parameters for each site in the rendered image is not the same — the gain or time interval from which light is integrated can be varied over the area of the rendered image. A temporally non-uniform exposure is one in which the integrated light is not handled consistently across time. Conventional exposure approximately — excluding previously discussed shutter artifacts — treats exposure as a single interval at a constant gain for the entire frame. Temporally non-uniform exposure consists of integrating incident light from multiple different periods, weighting light from different times in the exposure interval differently, or (overlapping with spatial non-uniformity) building the frame from different intervals.

If, for example, the dynamic range of a scene exceeds the dynamic range of the sensor, a traditional integrate-at-exposure camera cannot generate an image which preserves detail in the entire scene. However, if the integration is performed computationally after the fact, there is no restriction that all spatial segments be integrated with the same function, and suitable parameters can be chosen for sections of the image independently.

3.5 Exposure Interval

In a conventional frame-based camera, the photosensitive element — film or digital sensor — is exposed to light for a fixed interval by the opening and closing of a shutter. In the archetypal camera this is accomplished with the use of a focal-plane shutter; a pair of light blocking curtains which slide over the sensor one after the other, with the intervening time comprising the shutter speed.

With a focal plane shutter, and a relatively long exposure, a first curtain opens, allowing light through the lens to reach the sensor, and some time later the second curtain closes, cutting off the exposure. This scenario provides a very good approximation of the whole sensor being exposed for the same time interval, but that is not always desirable. In scenes with very large variations in brightness, exceeding the dynamic range of the sensor, a uniform exposure time will over or under expose parts of the scene. That is, if the bright areas of a scene are properly exposed to capture a maximum amount of detail, the darker areas may be under-exposed and simply appear dark, losing information in that part of the scene. Conversely, if the darker areas of a scene are properly exposed, the brighter areas may be over-exposed and saturate, losing information about that part of the scene.

For very short exposure intervals, a focal plane shutter allows light to strike the sensor by by moving both curtains at the same time, releasing them with an offset less than their individual travel time, thus traversing a slit between the first and second curtain across the sensor. This scenario is a poorer approximation of the entire sensor being exposed for the same interval — while the amount of time each sensel is exposed for approximately the same amount of time, the area along the leading edge of the frame is exposed at an earlier absolute time than the area at the trailing edge of the frame. This offset can result in smearing or other artifacts, particularly if objects are moving in the scene fast enough to create substantial displacements during the shutter interval. Other shuttering methods, such as irising leaf shutters or various electronic dump-and-readout schemes are also employed and create their own distinct artifacts in the resulting image, several of which are detailed in [4].

Some photographers may alter the exposure interval in intentional ways to create specific desired effects. The best-known of these alterations is to shoot multiple exposures on the same photosensitive frame, effectively compositing the incident light during multiple images into a single photo. These techniques, however, are physically complicated to set up, difficult to predict the results of, and require that the setup be executed perfectly at the time of image capture(s). The difficulty of physically realizing complicated exposures, and simplicity of achieving similar effects by computationally compositing images after the fact mean these effects are already most often accomplished by post-processing.

Non frame-based capture schemes like TDCI allow photographers to avoid these problems with exposure by computationally integrating the incident light over one or more interval(s) to create the final image. This means different portions of the scene are not competing for exposure parameters, as they are being sampled independently. Likewise, even the possibility of shuttering artifacts is eliminated, since if the sensels are continuously independently sampled, there are is no correlated scan pattern which

could produce artifacts. Most importantly, computational integration separates the processes of sampling the scene and exposing the image, so once sampled, the same interval of incident light can be exposed over and over to produce images, allowing the photographer to tweak the exposure parameters repeatedly, viewing the resulting image and adjusting until the desired effect is achieved.

3.6 Film Speed

In a conventional fame-based camera, the sensitivity to incident light (gain) is set as a whole-frame parameter, referred to as "Film Speed" for historical reasons. In an actual film camera, this gain is set by the photo-chemical sensitivity of the film being used, measured in modern times with the ISO 5800 system. In a digital cameras, the gain is determined by the "ISO Setting" in the camera. This setting's properties are specified by analogy to the behavior of film in the ISO 12232 [39] standard. In either case, this setting is fixed for the entire frame, for the entire interval of exposure. This whole-scene gain setting is often undesirable as it limits the dynamic range which can be represented in a single capture. Much as for the exposure interval, setting the gain to suit one part of a scene will often leave other parts dramatically under- or over- exposed, losing information about those areas as they saturate or fail to fill in details.

When performing integration computationally after the fact, there is no reason the gain must be uniform for the entire scene. In this scenario, a photographer can specify different gains, or gain functions as above, for different portions of the scene such that each portion of the scene is exposed as desired. There is significant precedent for the desirability of such a feature, as a number of "tricks" allow modern digital cameras to evade whole-scene exposure settings, albeit with significant caveats, discussed in section 1.2.

A second extant technique for cheating whole-sensor gain with modern digital cameras is that the sensor ISO setting is often applied, all or in part, as a digital multiplier in the post-processing [2] after the sensor has been read out, rather than by changing the behavior of the sensels. This property is, lately, referred to as "ISO Invariance" or "ISO-Less shooting". This after-the-fact gain function means information is only gained or lost based on the ISO setting during the image processing pipeline, not during capture. As a result, if images are captured at an ISO invariant camera's base ISO, all the scene information the sensor is capable of capturing will be captured and retained, albeit typically with less-than-pleasing brightness. The image can then be brightened in post-processing, essentially applying digital gain later when the photographer has the advantages of time, multiple tries, and additional compute power to make superior decisions about the gain factor. The gain can also be spatially non-uniform — selectively brightening or darkening parts of the scene is an extremely common post-processing manipulation. If the camera used was ISO Invariant this practice is effectively equivalent to selectively changing the sensitivity of the capture device. This technique, however, does not extend the range of the captured data beyond what the sensor can represent for a fixed interval, and may actually shrink it if the brightening or darkening range-clips any pixels. Employing this technique also

complicates selecting an appropriate shutter speed, since the captured image will be intentionally under-exposed at the time of capture.

By leveraging TDCI encoding, similar effects can be produced while avoiding many of the disadvantages of the methods that rely on traditional photographic modes. Some methods for using TDCI processing to generate images with larger dynamic range and fewer artifacts were previously explored in [73], but the techniques in that work retained the practice of using a single uniform sensitivity and exposure time. Specifically, if the incident light is recorded as a waveform and sampled after the fact, all of the differently-integrated regions can be integrated with time centered at the same instant, avoiding the issue of artifacts due to changes in the scene between sequentially-shot bracketed exposures. Even better, recording the incident light variation rather than a series of exposed images removes the requirement that the parameters for each of the constituent exposures be pre-selected. Since there are no pre-determined exposures under this scheme, there is no danger of clipping regions of the scene due to lack of data or saturation. This allows the selection of the parameters with which regions of the scene are exposed to be done after the fact, as many times as is necessary, until precisely the desired exposure parameters for each part of the scene are found.

3.7 Precedent for Non-Uniform Exposure

There is precedent for the concept of generating variously non-uniform exposures, but existing techniques for generating non-uniformity photochemical or digital photography come with significant limitations which should be improved upon by the techniques proposed in this work.

Many of the closest precursors to arbitrary non-uniformity are darkroom techniques. Dodging and Burning [74], the practice of adding or removing light during the print making process allow a degree of spatial control over effective exposure. Dodging is the process of selectively shielding the print paper from light passing through the film, by physically interposing shaped opaque masks between the film and print over the areas to be affected when it is exposed on an enlarger, casting shadows which effectively lighten the affected region. Burning, conversely, is the practice of selectively allowing additional light to fall on specific regions of the paper to create a darker area, typically by imposing an opaque mask with holes cut in the desired areas part way through the print exposure interval, cutting off the rest of the frame.

Typically, these effects are achieved by manually cutting a mask of the desired shape and size in a piece of opaque non-reflective material- such as dark colored card stock- then hand-holding it in the optical path of the enlarger with wire or line. The support structure can be hidden by keeping it in constant motion, and the edges of the affected area can be modulated by modifying (perhaps over time) the distance between the mask and paper — a mask very close to the paper produces sharp edges, and they edges become larger and more diffuse as it is drawn further away. Adams also suggests techniques like dripping developer solution at different concentration or temperature to effect local changes.

All these techniques are manual, labor intensive, bound to specific photochemical processes, and restricted to locally altering the interpretation the light gathered during the exposure rather than actually altering the interval or gain of that light. Analogous digital practices — selecting areas of an image for selective lightening, darkening, or dynamic range manipulation — are certainly less laborious but only slightly more flexible. The analogy of masks to guide exposure has been extremely valuable in the development of the techniques presented in this work.

Other extant methods of generating non-uniform exposures with existing technologies are based on multiple exposures. These techniques are quite ride ranging; the multiple exposures can be performed on a single piece of film or sensor readout at the time of capture by repeatedly opening and closing the shutter, or by after-the-fact compositing of several single exposures during the development process. In a controlled environment, similar effects can also be accomplished by modulating the light on the scene rather than the shuttering mechanism, as in stroboscopic photography when the shutter is opened for a relatively long interval, and a flash is fired for multiple controlled shorter intervals.

The content and parameters of the separate composited images can vary widely. For artistic purposes, the multiple exposures may be related only by the intent of the photographer. For more consistent compositions of multiple exposures, a common practice "stacking" in combination with "bracketing": taking a series of exposures with different parameters with the possibility of combining the desired features of several to achieve effects not possible through the lens. Exposure can be manipulated by stepping the shutter speed, aperture, or (in a digital camera) sensitivity across a range, then regions of the frames exposed as desired in the constituent frames can be combined. Compositions of multiple differently exposed frames can produce a dynamic range larger than is achievable directly with the same equipment.

In digital camera systems the technique can again be extended in various ways; as digital cameras are not bound by the static sensitivity of a roll of film or the parameter modification speed of a human operator. As a simple example, when bracketing, the onboard computer can vary parameters between successive frames almost as fast as the limit of the desired shutter time. Another even broader example is that where optical techniques are restricted to additive composition of multiple exposures digital setups may use other compositing schemes, such as blending or subtraction. Digital techniques for compositing multiple exposures also afford more flexibly an easily mask out which parts of the final image come from which exposure, including employing various sorts of automation to make the decision without a human operator. Some digital cameras, or cameras integrated into computerized devices even allow some of these techniques to be automatically executed in-body; features with names like "Auto HDR" or the automatic focus stacking features advertised by Olympus [75].

However, the extended digital features do not remove the restriction that the intervals represented by the time individual exposures cannot be overlapping, creating issues for composition, such as discontinuities in the case of moving scene elements. It also does not allow wide after-the-fact adjustment of individual exposures; for example, if a portion of a scene is dramatically over- or under- exposed for every frame

of a bracketed sequence, nothing can be done after the fact to recover information about that part of the scene.

Focus bracketing and stacking bears mention as it continues the analogy of this work to plenoptic cameras's ability to adjust focus post capture; while focus stacking allows a series of frames which were correctly focused, and the focus correctly spaced, at the time of capture to generate extended depth of field, a plenoptic camera can generate extended depth-of-field without relying on a set of step-wise components which were focused correctly at exposure time.

Many of the tools developed in the current work are inspired through a lens of enabling more general, more flexible, and more powerful versions of these existing techniques.

3.8 Non-Uniform Over Time

When performing integration of incident light computationally after capture, there is no restriction that the virtually admitted light must be uniformly "exposed" over a single interval as with a physical shutter. Integration gain functions can be specified which simulate mechanically implausible shutter behaviors with only a small amount of extra difficulty. For example, the gain function can have multiple distinct peaks, producing an effect analogous to multiple exposures. Even less physically realizable, the gain function can slope, vaguely physically analogous to imposing a time-varying neutral density filter over the lens, or (somewhat less precisely, as slope variations will not affect depth of field) irising the aperture during the exposure. It is, of course, entirely possible integrate with gain functions that could not be practically realized by a mechanical means, though the more radical the exposure function, the less unprecedented the effects will be.

In the first Octave-based prototype implementation the integration gain function is represented as a composite spline. Users familiar with image editing will, whether they know it by that name or not, posses at least a passing familiarity with composite Bezier curves, used for drawing arbitrary lines in a wide variety of image editing applications, or (Centripetal) Catmull-Rom splines [76], also used in image editing for specifying color curves in many photographic editing tools, and modeling camera motion in video processing. Initially, Centripetal Catmull-Rom splines seemed particularly appealing because they are straightforward both to visually manipulate, and to compute the value of at arbitrary position, and already widely used in imaging applications. Furthermore, Centripetal Catmull-Rom are inherently smooth and non-looping (mathematically; twice differentiable), making them immune to ambiguities or discontinuities. Unfortunately, a single Centripetal Catmull-Rom spline can represent only a very limited family of functions, which does not include many trivially-interesting cases.

A more general representation of temporal gain functions is therefore required; the method in this phase of experiments used normal cubic splines [77] as a highly-flexible representation which retains the desirable property of always being twice differentiable. Using this scheme, each the integration gain function is specified as a

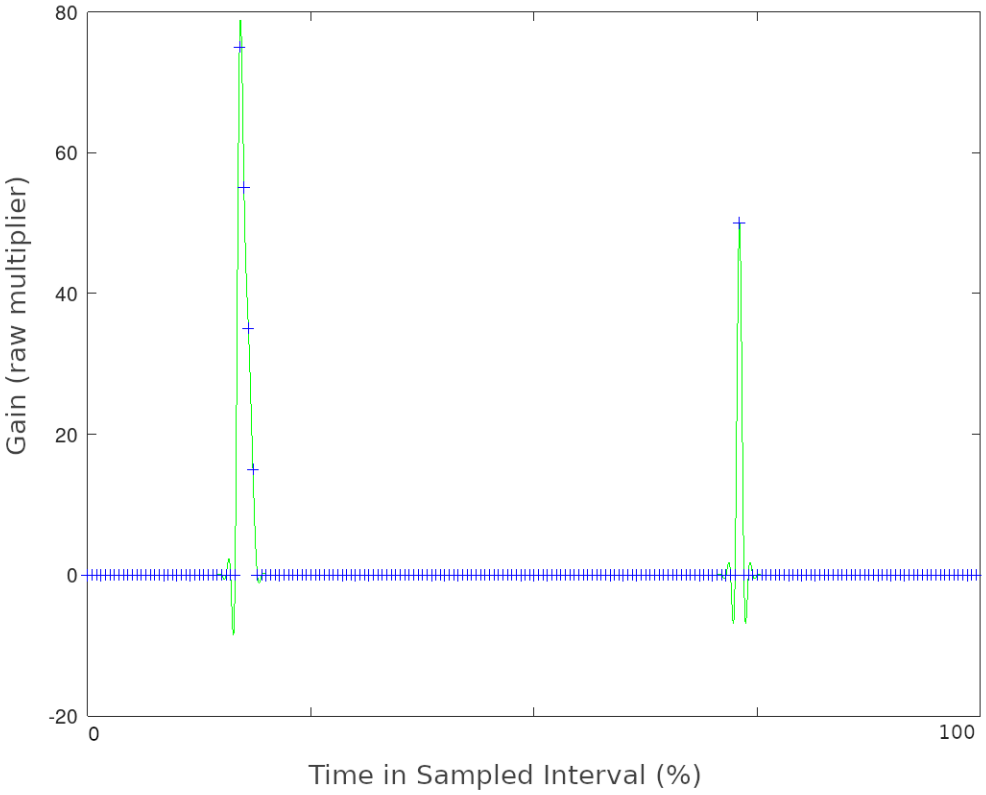


Figure 3.8: Example Integration Function from Octave prototype

series of control points. To use the integration gain function, the domain of the specified control points is mapped to the interval of recorded light (allowing for arbitrary granularity), such that the gain to be applied to the incident light at time $t_{current}$ into the sampled interval is the value of the normal cubic interpolation at $t_{current}/t_{max}$.

While the natural behavior for a computer scientist is to specify the temporal integration gain function as an equation or series of control points, perhaps over a unit interval, for most users this will be extremely awkward. An elegant interface would present the user with an initially horizontal line that they can interactively modify by clicking and dragging to add and modify control points. The integration scale factor for each instant is then the height under the curve at that distance into the interval, which is mathematically straightforward, readily visually representable, and leverages existing mental models likely to be available to those accustomed to image editing tools. The horizontal (x) axis of this function represents time, to be stretched over whatever integration interval is selected. The vertical (y) axis of this function represents the instantaneous gain to be applied to the incident light at time $(t/interval) * fn_{max}$.

Figure 3.8 shows an integration gain function specified with this method representing a double exposure, with the first exposure ramping very quickly to very high gain then somewhat more slowly tapering, and the second very quickly flicking on

Figure 3.9: Capture of riding mower integrated with the function shown in Figure 3.8

and off to a smaller maximum gain. Figure 3.9 shows the result of applying that integration function to a capture of a passing riding lawn mower, rendered from 30FPS video. Note that the resulting image contains two displaced images of the mower. The first mower image corresponds to the first spike in the integration gain function. It is mostly opaque, picks up suddenly with a sharp leading edge, then slowly fades away in a smear as though it were moving quickly relative to the shutter speed. The relative opacity is because the majority of the light energy integrated at those locations in the frame come from the first spike, while the sudden appearance and slow taper are the result of the shape of that first spike. The second mower image, further down the row, corresponds to the second spike. It is relatively sharp and un-smeared because the width of the spike is short compared to the speed the mower was moving. However, it appears relatively translucent because the majority of the light energy integrated at its location was contributed by the background in the earlier portion of the exposure, rather than the time when the mower was at that position.

One interesting detail of arbitrary integration functions is that it is perfectly possible to specify *negative* gain for some portions of the interval. This behavior would be physically analogous to the sensor subtracting the contribution of incident light during portions of the exposure, rather than adding it. This is not something that is physically realizable in a conventional camera, but is very useful creatively for tasks such as subtracting static features from a scene. Partially negative gain functions also provide a ready way to provide even average brightness for differently-

integrated parts of the scene to compensate for intervals with particularly high gain applied.

3.9 Non-Uniform Over Space

Non-uniform integration over space is analogous to creating a scene-specific piece of film whose sensitivity to light varies across it's surface, or the practice or selectively lightening or darkening portions of a scene in post-processing. While applying different gains to different portions of the scene in order to properly expose each is obvious, specifying the spatial regions on which to apply the different gains is somewhat mechanically awkward. Extended discussions on the matter resulted in several unappealing options — specifying by mathematical function (awkward for the user), specifying by rectangular region (restrictive), specifying by arbitrary polygons (complicated and restrictive), or specifying by bucket-fill algorithm (computationally difficult, self feedback problems) - and one promising avenue.

The promising method is to use a mask, drawn as a bitmap of the same resolution as the capture source, colored with a different pixel value for each region to be processed. This covers all the functionality of arbitrary geometry or function-determined regions by offering users a simple, portable, well-known format to generate their complex masks in, while allowing straightforward use cases to simply draw their desires in a basic image editor. A bitmap mask for region definition also allows for straightforward batch processing, either by applying the same generated or otherwise pre-prepared mask for multiple exposures, or enabling the use of an external tool to perform higher-level per-frame functions, such as object tracking, to generate sequential video frames from a TDCI stream with specific exposure properties for different objects in the scene.

The format currently being used to express these spatial exposure masks is a 8-bit P2 PGM with the same spatial size as the TDCI stream to be exposed. Each of the 255 gray levels possible in the format represents a distinct region, and the value of each pixel in the mask PGM specifies which exposure region to apply to the corresponding pixel in the input stream. This way, a simple gray-scale mask can be generated where each pixel in the mask is tagged with the encoded value of the gray level. Each gray level is then assigned a particular integration function. This provides a number of regions far in excess of any easily-conceived practical application, avoids forcing users to deal with any specialty tools or mathematical specifications, and is extremely straightforward for software to both generate and ingest. Each numbered region is then assigned an integration function with which to "expose" that portion of the image.

This technique is quite general. It is possible to not only vary the integration parameters for portions of the scene, analogous to existing HDR techniques, but to directly produce temporally composite exposures. In such an exposure, the output image is generated from sections which spatial sections of the image are are integrated from temporally differently-centered and possibly non-overlapping sub-sections of the sampling interval, opening a wide range of options. One simple application for this combined case might be selecting independent intervals for each face in a group picture

to give each pictured individual open eyes and pleasing expressions, even though though they did not happen at the same time. Another application could be masking a moving object from the background in a scene, and integrating the moving part with a short, sharp-edged, high-gain integration function, and the background with a longer, shallower, lower-gain integration function on the same center, yielding an image with a sharp object with a minimum of motion blur on an apparently well-lit, detailed background.

Including a simple editor which would provide a transparent overlay of a frame preview and some basic drawing tools to generate region mask bitmaps inside the TDCI exposure tool is an obvious nice-to-have, but is not in the critical path for demonstrating the technological features. This is especially reasonable as most users likely to be using complicated exposure behaviors are likely to already have a deep familiarity and established work-flow with their image editing tool of choice, and staying out of their way may even be the better choice in general.

To provide a minimal illustration for the effects possible with this mechanism, figure 3.10 contains a simple top/bottom split mask, specifying the two regions on which to apply the two integration functions in figure 3.11. This mask and function are then applied to a 240FPS video of a foam penguin and foam rock swinging like a pendulum while attached to the same string, resulting in figure 3.12. Note that the penguin, integrated with the function with two wider peaks, appears in two places, with a relatively large amount of motion blur indicating the two longer intervals of integration, while the rock, integrated with the single very narrow peak, appears only once, and relatively sharp-edged, corresponding to the single narrow peak — and that it appears *between* the two images of the penguin though they were moving in concert. Also note that the offsets between the peaks and the appearance of blurring in the image provide a tell that the images of the penguin appear from light contribution during the right-to-left traversal, while the light contribution from which the rock is taken is from the left-to-right return trip, despite appearing "between" the two images of the penguin.

3.10 The Octave Prototype

The proof-of-concept implementation presented in a paper "Non-Uniform Integration of TDCI Captures" at EI2020 [78] is in the form of an Octave script included in appendix A which can ingest a sequence of video frames to simulate continuously sampled incident light, and apply user-specified spatial and temporal non-uniformities to the integration of that light. This proof-of-concept implementation serves primarily as a testbed for algorithms and representations, as well as an easy way to experiment with the effects which can be produced, and is not tuned to be particularly fast or high quality.

In this prototype, spatial non-uniformity is specified by an 8-bit PGM mask as proposed above. Each region to be integrated is assigned a unique gray value, and each gray value is mapped to a corresponding temporal gain function by a simple table of gray value:function index correspondences. Directly encoding the index of the gain function to be used as the gray value was rejected, as numerically adjacent

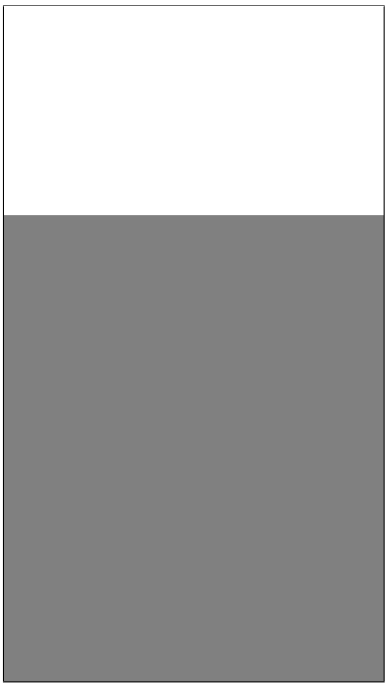


Figure 3.10: Mask specifying two areas of the scene to integrate with different functions.

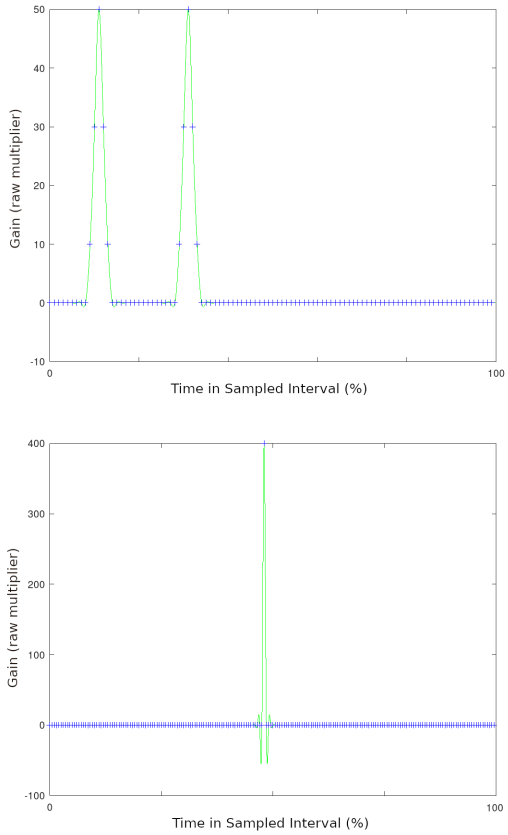


Figure 3.11: Two integration functions

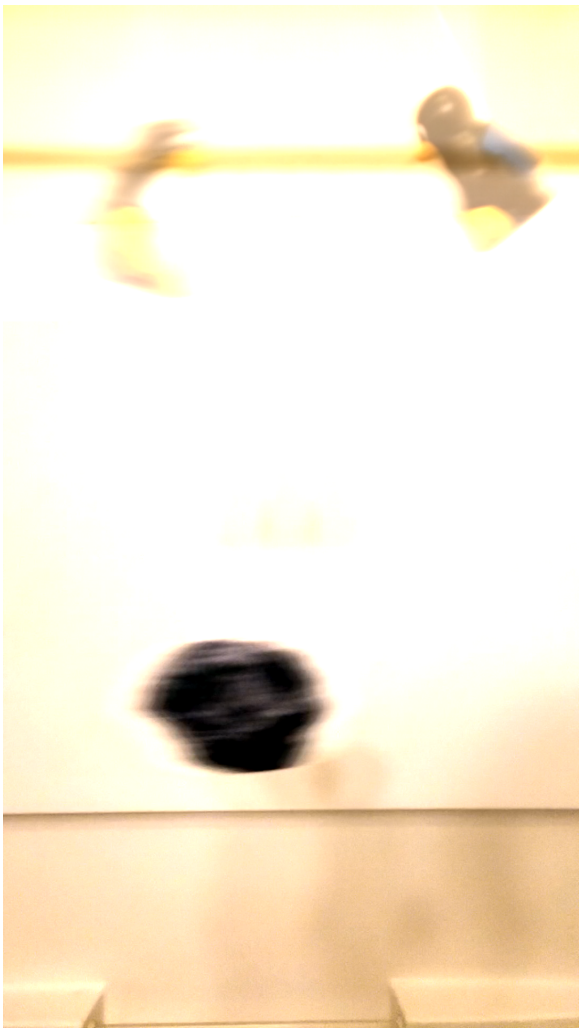


Figure 3.12: Image exposed with the mask and functions from figure 3.11 and 3.10

gray vales are indistinguishable to a human observer, and storing the control point vectors in a sparse representation adds more complexity to the prototype than simply re-mapping the indexing.

Likewise, in this prototype, integration gain functions for temporal non-uniformity are specified by a series of p user-supplied control points per function. These control points are interpolated with a normal cubic spline to give the gain to be applied at time t into a sampled interval of length t_{max} by evaluating the interpolation at $(t/t_{max}) * p$. This means every integration gain function is mapped to the entire input interval, so the granularity of the function can be increased by inputting a larger number of control points, and the position inside the sampled interval can be accomplished by zero padding, without the addition of any other constructs. An arbitrary number of control points can be specified to produce an approximation of any desired integration function, and the multiple functions specified for multiple regions are not required to contain the same number of control points.

The approximation to incident light to be integrated is generated by extracting successive from a video sequence, and treating each as an interval of contributed light for $1/framerate$ seconds. Integration is performed by summing the contribution of each input frame, multiplied by the average value of the mask-specified gain function for the interval represented by the frame, and subsequently dividing this weighted sum image by the number of frames integrated over to normalize the exposure.

This initial prototype does not attempt to interpolate between samples, as was done in the TIK TDCI testbed [8], but a version built on top of TIK has been developed using the algorithms demonstrated in this work. The NUTIK version produces higher-quality output with less dependency on the frame-rate of the video input. It also does so much more quickly.

3.11 Findings from the Octave Prototype

This proof-of-concept implementation has demonstrated the feasibility of using the TDCI paradigm to generate virtual exposures with physically impractical exposure parameters. This mechanism shows promise both for a variety of creative applications, as well as potential for use in scientific applications. One simple example is using this technique to perform HDR stacking with different exposure parameters for different parts of the scene, all centered on the same instant in time allows a photographer to (iteratively) develop the dynamics of an exposure without the common problems of stitching errors from scene motion or incorrectly pre-parameterized exposure settings. In another, the function-driven integration can be used as a more flexible alternative to stroboscopic photography for capturing and analyzing motion, by sampling a scene then imposing a pulse train exposure function until the desired effect is achieved.

Visualizing the effects achieved by these methods, much less their compelling applications, is still rather difficult, since many of them are not achievable with any physically realized camera. So far, the most effective way of visualizing the tool's behavior is to imagine a camera with a focal-plane shutter consisting of an extremely transmissive, extremely fast, LCD of resolution high enough that its dot size is not the limiting factor in the optical path. One can then think of displaying the imposed spatial and temporal functions on this screen in the optical path and — more or less — predict the properties of the resulting image.

This experiment was also valuable in terms of lessons learned while taking an initial foray into the design space. One of the most immediate lessons is that the idea of exposure intervals being continuous smooth functions represented by sophisticated splines was a bad move. Conventional exposures time properties look like slightly slew-limited boxcar functions; roughly 0 except during the exposure interval, which starts and stops abruptly. As indicated by the earlier study of shutter artifacts, they aren't perfectly square, there blades of a mechanical shutter are neither perfectly in the image plane nor capable of traveling infinitely quickly. This secured the idea that the next prototype should make the common case easy; linear interpolation between control points makes it straightforward to represent boxcar type functions,

It also exposed some extra considerations with the concept of negative gains; while negative gain is quite useful for subtracting some incident light for feature isolation, it leaves a risk of negative output, something that no reasonable image format supports.

This is not a completely unprecedented problem; it's perfectly reasonable to read two images in code or an image editor and subtract one from the other to obtain the difference. The matrix-math interpretation is straightforward, and most of the image interpretation follows, but the issue of negative values is already present — what exactly does a red value of -6 represent when the range of no-red to as-red-as-possible is mapped to $0 - 255$?

This is generally treated as a normalization problem; if the output of integration produced negative, saturated, or otherwise unpresentable values, there is a choice on how to handle the case. A later prototype follows the behavior of most image processing software and libraries and explicitly saturates output at 0 and the maximum representable value in its output format as it seemed to produce the least surprising behavior, but is not the only conceptually reasonable option. In this interpretation our -6 red from before simply becomes 0. This saturation method is also the default behavior in many image processing tools, including the OpenCV library [79] used in the construction of later prototypes in this work.

Other options include treating negative values in one color channel as additions to its compliment(s), perhaps scaled by some factor, which is somewhat analogous to color perception models. In this scheme, our -6 red would turn into, say, $+6$ in each of the blue and green channels. This scheme would be both rather computationally inconvenient, and is extremely likely to produce surprising behavior.

Similarly, out of range values could be treated as a tone-mapping problem similar to methods for mapping between image formats with different dynamic range; the output image minimum value could be mapped to the lowest (most negative) value in the integration result, the maximum result of integration to the maximum output value, and the remaining output mapped — presumably linearly though an argument could be made for log or other functions — between those values. In this interpretation, the output mapping of our -6 red depends on the range of result values and would be the position of -6 relative to the least and most result values. This is both more computationally straightforward to implement and holds a better analogy to an existing practice than in inverse-color interpretation, but was also rejected for being likely to produce surprising results.

3.12 Camera Hacking

One of the motivations for this work was the desire for a "real" TDCI camera. Most work on TDCI has focused on the TIK framework discussed in section 1.1, with a few forays into building primitive capture devices, as with the CHDK prototype described in subsection 1.1 — none of which provide a high quality, native TDCI capture device.

Having a high-quality native capture device would offer a number of benefits. Working only on simulated data — as one could obtain from a 3D rendering package, for example — does not provide any assurance that the model will work as desired on natural data. Synthesizing an IMEV from video, as we have with TIK, provides

better assurances that the model is sound with natural data, but doesn't provide much insight on ways to build a practical capture device, and requires multiple, awkward post-processing steps which limit both the usability and the achievable temporal and spatial resolution. Without a capable prototype implementation, it is difficult to prove that the radical departures from established photographic practice required for TDCI are justified — or practical. Designing a ground-up semiconductor design sensor to ideally suit the TDCI model was deemed out-of-scope for the current work, as the necessary tools and expertise were not readily available, and the potential timelines for such an undertaking extend into decades.

To address this issue, one of the main objectives of the work documented here was to explore the internal construction of several camera devices with an eye toward constructing a top-to-bottom TDCI toolchain, which would generate exposures from IMEVs captured directly by a modified prosumer-grade camera. This is hacking, in the older honorable sense of the term, exploring a system to uncover technical details with the goal of using that understanding to improve and extend the abilities of the system.

While building such a device turned out to be impractical due to design features uncovered during reverse-engineering, the exploration reveals a number of interesting insights about the way modern digital cameras are constructed, and in particular the unforced restrictions that result from frame-oriented assumptions in the design of cameras.

Modern digital cameras contain sophisticated computer systems, comprising a CPU, a sensor with typically many ADC channels for readout, a relatively large RAM, one or more storage devices, various special hardware function units such as encoders for specific image formats, and the bussing and DMA (Direct Memory Access) devices to move data between the parts. These cameras, therefore, function as cameras by virtue of the software loaded into them.

A number of options for cameras which could be modified to support in-body TDCI were explored in some depth, and each one hit a showstopping limitation. The technical reasons and apparent motivations for those limitations is an interesting topic in its own right, and will be discussed after the two cases.

Sony a6000

The first and most seemingly promising platform explored was a Sony Alpha mirrorless body. The sensors are excellent, mirrorless cameras are designed to run the sensor more or less continuously exposed to incident light, and because Sony publishes compliance sources for the various open-source software running in their devices [64], we know that devices in that family run a Linux + BusyBox operating system. A number of older Sony Mirrorless bodies also support the PlayMemories Camera Apps [80] ecosystem for loadable software, an Android-like development environment for limited application development of software to run in-body. The PMCA environment is too restricted to even attempt to develop a TDCI-like capture mode in, during the same period of time this work has taken place over, is being discontinued entirely [81].

However, a series of hackers have investigated the firmware update format and PlayMemories API which lead to possibilities to run unrestricted user code in-body. The first of these project was NEX-HACK [82], a project to reverse engineer the Sony Alpha firmware, which resulted in a large amount of approximate documentation, and a tool to unpack, repack, and forge signatures on user-modified firmware updates for Sony Alpha cameras. This initial firmware repacking tool was superseded fwtool.py by ma1co [83], a more complete and polished tool for similar manipulation.

Ma1co also worked from the reverse-engineered documentation to generate a development environment called OpenMemories [84], to enable the development and loading of PlayMemories programs outside of Sony's tools and approval process. One of the reference applications is [85], a suite of tools for hackers to play with the camera, which includes some , and critically, a telnet server allowing a user to connect over WiFi to a root shell and inspect the state of the live system.

Given this access, an exploration of a specific test device, a Sony α6000 (aka ILCE-6000) revealed a number of interesting details, the first of which is that it really is a fairly standard embedded Linux system internally: `uname` returns `Linux localhost 3.0.27_nl-rt106 #1 SMP PREEMPT RT Thu Feb 18 09:45:24 JST 2016 armv7l unknown`, showing the system is based on a customized 3.0 series Linux kernel with Real Time extensions, and basic POSIX interfaces are supplied by a BusyBox binary, as expected from the compliance sources. Exploration with standard Unix utilities reveals 200MB of main memory (there are likely also inaccessible buffers internal to the image capture system), and a four core ARM V7l processor. Inspecting the process table with `ps` shows a Zygote process — the same process manager used by Google's Android operating system, consistent with the PlayMemories interfaces resembling Android. Checking loaded kernel modules with `lsmod` shows a large number of expected mainline features like `mmcio` and `nand` storage interfaces, an 802.11 wireless stack based on the `cfg80211` interface featuring an `ath6kl` chipset attached via sdio, modules to support the built in HDMI with CEC port, and that the kernel is tainted with some proprietary modules. The set of proprietary modules is quite extensive, and includes a set of modules prefixed with the string "osal" (`osal_utm`, `osal_uipc`, `osal_ulogio`) which are not publicly documented but the names suggest an **o**perating **s**ystem **a**bstraction **l**ayer to ease porting of some software components, maybe from another RTOS, and a set of modules `dmac` (speculatively, the interface to a programmable dma controller), `sircs` (whose name strongly suggests it implements a version of the Sony IR Remote Protocol [86] as a bespoke module rather than via the open source LIRC interfaces), a pair of modules `lld` and `ldec` (possibly "load" and "decode"). There is also an module `liro` which is a dependency for a number of the other proprietary modules, which spawns over a hundred kernel threads visible with `ps`.

Going a level deeper, pointing basic binary inspection tools like `objdump`, `strings`, and `binwalk` at some of the proprietary modules — whose interfaces must be at least moderately exposed to allow communication through the kernel module interface boundary — reveals a number of interesting (but not helpful) details. The ideal finding would have been a set of descriptively-named functions to control the sensor readout, but One major observation from comparing the stubs, there is an enormous amount of communication among `liro` and various `osal` modules, as they each make

```
$binwalk liro.ko

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------

0             0x0             ELF, 32-bit LSB relocatable, ARM,
   version 1 (SYSV)
35960         0x8C78          ESP Image (ESP32): segment count: 2,
   flash mode: QUIO, flash speed: 40MHz, flash size: 1MB, entry
   address: 0x0
36191         0x8D5F          mcrypt 2.2 encrypted data, algorithm:
   blowfish-448, mode: CBC, keymode: 8bit
57094         0xDF06          JBOOT STAG header, image id: 0,
   timestamp 0x391C0000, image size: 725352448 bytes, image JBOOT
   checksum: 0x0, header JBOOT checksum: 0x7C1C
```

Figure 3.13: Binwalk results for the liro.ko module on an a6000

large numbers of calls to functions exposed by the others. This observation does not bode well static analysis. Another observation is that the `liro` module appears to embed a partially-encrypted firmware image for a separate computer within the camera; there is a boot header, a small un-encrypted binary, and an encrypted blob, as shown in figure 3.13. This implies that, in addition to many intercommunicating blackbox parts running on the main Linux system, there is a secondary attached processor involved in the device specific low-level functions.

Unfortunately, despite being easy to inspect up to this point, the conclusion from that inspection is that the interface to the sensor and image-handling pipeline are a set of black-box, undocumented, proprietary, binary kernel modules interacting with bespoke hardware and each other.

Reverse engineering those interfaces would be a high-risk project on the same scale as the intended product of this work, have limited reach since no models released after 2017 support PlayMemories, and would be necessary to make an in-camera TDCI implementation superior to the fall-back position of reprocessing captures externally, so this line of inquiry was largely dropped.

Canon

Canon's entire product line shares a number of significant architectural similarities. The native software is built on a Canon developed RTOS (**R**eal **T**ime **O**perating **S**ystem referred to as "DryOS" [87] with various computer platform features and camera-specific functionality implemented on top. DryOS appears to implement the $\mu ITRON$ RTOS kernel specification, and also exposes some POSIX and DOS-like interfaces — for example, when configured to boot from an external storage device, it attempts to launch a file named `autoexec.bin`, a fact which is very important when attempting to load you own code on a camera. DryOS has been used across the Canon line since around 2007. The CHDK (Canon Hack Development Kit) [63]

project targeting Canon's fixed-lens cameras is based around reverse engineering of Canon's interfaces, though CHDK generally limits itself to calling Canon functions.

I have been involved in a number of prior projects implemented by reprogramming Canon compact cameras using the CHDK [63] framework, including a number of multi-camera array designs [88], the ISO-invariance project [2] discussed in section 3.1, providing support for a project which developed a CHDK-based photoplethysmography system [89], and other applications. As described among the earlier work in subsection 1.1, Canon compacts hacked with CHDK were the platform for an early, primitive in-camera TDCI product [10]. Canon's prosumer EOS cameras with interchangeable lenses are (generally) not supported by the CHDK project.

Magic Lantern

However, a sister project, Magic Lantern [90] has developed around Canon EOS bodies, and provides even deeper hooks into the camera. Though the sensors and on-board computational resources are not quite as impressive as those in the Sony mirrorless bodies, this experience makes the Canon EOS prosumer cameras an attractive potential target. For the same designed-for-continuous-exposure reasons as the Sony body selection, the most promising of the Canon options are the mirrorless bodies, so the the exploration on this front took place on a Canon EOS M body. ML focuses on higher-end interchangeable-lens cameras, and directly manipulates hardware configuration registers. In fact, though they are independent projects with entirely separate code bases, Magic Lantern originally derives from the reverse engineered documentation of the CHDK project, and they continue to share information. Magic Lantern operates as a program that runs along side the vendor software; the only modification made to the original system is setting the `BOOTDISK` flag in the onboard Flash, so the camera will attempt to load code from `autoexec.bin` in the root of the CF or SD card.

The Magic Lantern project [90] is a community developed, Open Source (GPL Licensed) project which has developed a development framework and extensive collection of software which is loaded into the camera by the aforementioned process to load user code from attached storage. The Magic Lantern community is chiefly focused on adding video recording features and capabilities to available cameras. Specifically, the project was initiated by Trammell Hudson in 2009 to add extended video features to the 5D Mark II, and management of the project transitioned to A1ex in late 2010. As the understanding of the camera internals, set of supported cameras, and community expanded, it has also extended camera capabilities for general shooting like focus peaking, zebra highlights, and live histograms, added useful shooting modes like intervalometer, motion detect, and automatic exposure bracketing, and a wide variety of other features. Magic Lantern also provides a set of sophisticated developer tools for inspecting camera behavior, and several methods to run custom code on the camera including a scripting interface in the Lua programming language, and a module system to load custom compiled code.

Magic Lantern is academically interesting for several reasons. The first source of academic interest in Magic Lantern is quite similar to the main ML community inter-

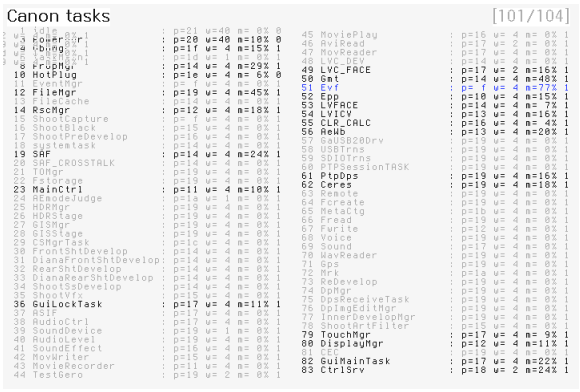


Figure 3.14: Process table of Canon native processes

est; the possibility of modifying a (relatively) capable commercial camera to behave in new ways to construct research prototypes. Because of the exposed internal memory structures and full programming environment, the functional units of the camera can be freely reconfigured up to the limits of the reverse-engineered understanding and/or capabilities of the hardware.

A second interest for research applications is the sophisticated scripting interface exposed by Magic Lantern. The aforementioned Lua scripting and loadable compiled module interfaces allow researchers to both arrange programmatic capture to use the camera as an apparatus for experiments, and also programmatically operate the camera to study its properties. Simple applications of custom code may include programmed timelapses, motion detection, exposure stacking for focus or dynamic range, etc. but as it is a complete programming environment, the possible functionality is limited only by the resources present in the device.

Another other major academic interest for MagicLantern is the insight it provides into the internal design of a (relatively) modern camera system. Camera manufacturers do not generally intentionally provide low-level access to the camera or its' embedded computer system, or even document its features, over concerns about licensing and competition. The reverse engineering efforts required to create community modifications like Magic Lantern expose that information. Close knowledge of the workings of widely available camera systems allow researchers and photographers alike to reason about both the resulting images and the potential capabilities of a camera in ways that a black-box treatment does not, even if they are not aiming to expand or alter the functionality.

The ML software also allows an enormous degree of live instrumentation into the running camera. For example, a built-in feature allows one to view the memory map and utilization, the process tables of both ML and Canon's native software, as in figure 3.14, and a wide variety of other logging and monitoring, several of which are applied later in this work.

In the specific case of the EOS M used as an example in this work, the embedded computer is a Canon DIGIC5 chipset. A rough diagram of the relevant architecture is shown in figure 3.15. On the computer front, two processor cores are ARM5TE 32-bit processors, and total RAM is approximately 256MB. The "Image Preprocess"

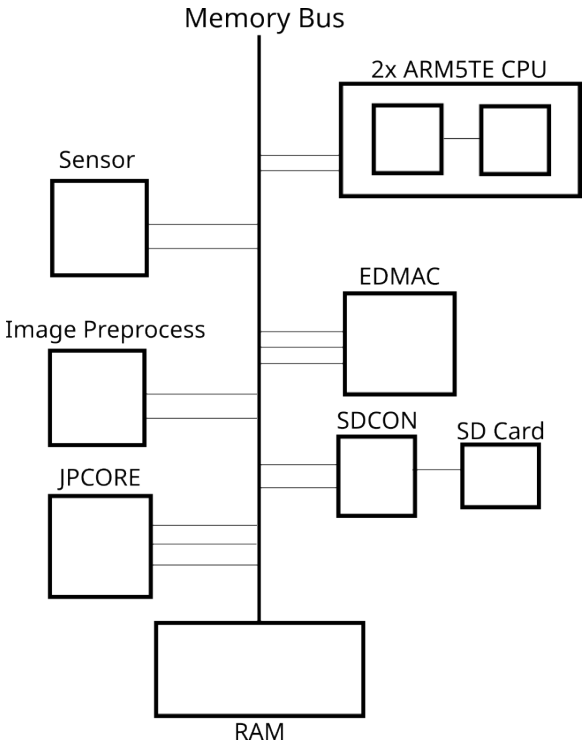


Figure 3.15: Rough layout of DIGIC 5 SoC

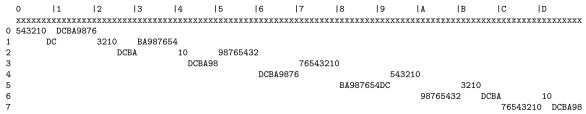


Figure 3.16: The packed layout of Canon 14-bit RAW encoding

and “JPCORE” blocks represent memory-mapped fixed-function hardware for RAW processing and JPEG encoding/decoding (names derived from firmware strings), the SDCON/SD Card blocks represent the interface to the SD card, and the EDMAC is a sophisticated DMA engine which will be discussed in its own section below. Some devices — such as the onboard FlashROM that contains the built-in software, and the various IO devices for reading buttons, blinking LEDs, and interfacing lens mechanics are not included in this diagram, though they are also memory-mapped devices. Strings internal to the firmware refer to those IO devices as the “MPU” and “LPU” for the medium and low speed devices, respectively.

Another relevant detail of the low-level memory behavior of Canon cameras is that the native RAW pixel format for 14- bit Canon cameras is packed in a somewhat convoluted way, shown in figure 3.16. This means efficient processing of the native format involves working on 224-bit (least common multiple of 14 and 32) blocks, necessitating some interesting bit-twiddling and/or specific hardware support, some of which is accomplished by the aforementioned Image Pre-processing hardware, but much of which is managed by the EDMAC discussed in the following section.

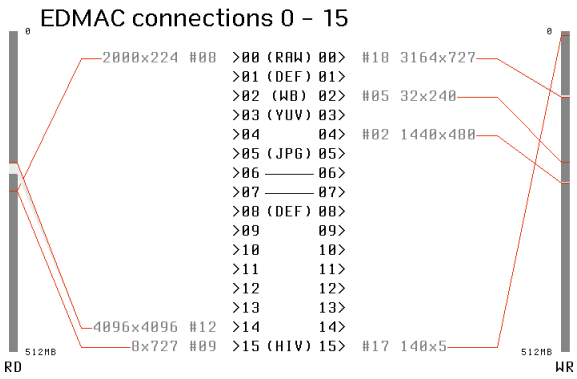


Figure 3.17: First screen of EDMAC live monitoring

EDMAC

One of the major accomplishments of the ML project has been the reverse engineering of the EDMAC **E**ngine **DMA C**ontroller, where DMA in turn means **D**irect **M**emory **A**ccess, device present in Canon Digic SoCs. This functionality was not available for the first several years of the project, but even a partial understanding of it has enabled a great deal of the functionality now supported by ML.

The initial reverse engineering work was performed by a1ex in late 2016 [91]. Shortly after the initial behavior and strings were documented, it was back-matched to a patent, [92] initially filed by Canon in 2003. The EDMAC name and much of the related terminology in the reverse-engineered documentation is derived directly from strings in the firmware, and later more terms were matched to the patent, so unlike many systems whose public understanding is based on reverse engineering, the terminology more-or-less lines up between Canon published documentation and public reverse engineering.

The EDMAC is a sophisticated point-to-point data transfer engine, which can be programmed to move data not only between memory regions, but also to and from various hardware devices. For example, on many camera models EDMAC channel 0 is connected to the raw read-out of the sensor [93], while channel 3 is connected to a hardware JPEG encoder/decoder.

The ML project now contains a loadable module which contains a great deal of live and logging instrumentation for the running EDMAC; it can provide a live view of current EDMAC activity as in figure 3.17, logging at regular intervals to allow study of EDMAC activity during specific operations, and automatically identify unused EDMAC channels which can potentially be repurposed.

The specification of the memory regions to read or write from in the EDMAC is also quite sophisticated; it supports x and y block size, stride, count, and offset arguments, special sizes for last blocks in sequences, and it supports these specifications for both the source and the destination region, allowing it to transfer and transform memory regions of complicated shape and size with minimal CPU involvement. A variety of useful camera behaviors — like cropped sensor readout — are performed using this mechanism. The EDMAC DMA transfer behavior can be described in C as

```
for (int jn = 0; jn <= yn; jn++)
{
    int y     = (jn < yn) ? ya    : yb;
    int off2  = (jn < yn) ? off2a : off2b;
    for (int in = 0; in <= xn; in++)
    {
        int x     = (in < xn) ? xa    : xb;
        int off1  = (in < xn) ? off1a : off1b;
        int off23 = (in < xn) ? off2  : off3;
        for (int j = 0; j <= y; j++)
        {
            int off = (j < y) ? off1 : off23;
            cpu_physical_memory_write(dst, src, x);
            src += x;
            dst += x + off;
        }
    }
}
```

Figure 3.18: EDMAC DMA Transfer Behavior

shown in figure 3.18, taken directly from the QEMU-based development tools built by the ML project.

Visually, this produces access patterns like those in figure 3.19 illustrating an access pattern with $xn = 3$, $yn = 2$, $xb \neq xa$, $yb \neq ya$, and differing positive values for $off1a$ and $off1b$ which affect the stride in the X direction. Negative offsets are also legal; it would cause the tiles to overlap rather than skip.

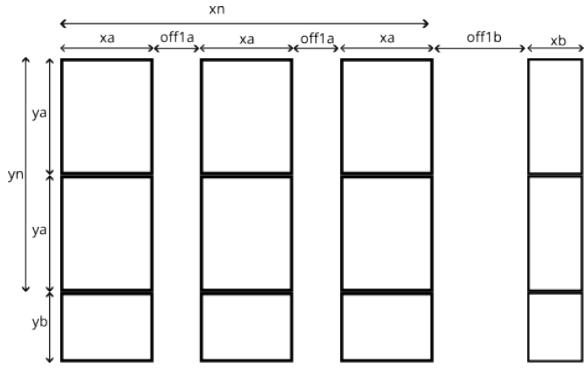


Figure 3.19: EDMAC Memory Layout, with Offsets

Magic Lantern Video

Another relevant accomplishment of the Magic Lantern project has been the creation of the **M**agic **L**antern **V**ideo (MLV) format and associated tooling. The MLV format is, essentially, a container for sequences of raw frame data read directly from the sensor, as well as standardized ways of writing out supporting metadata, and

optionally synchronized audio blocks. The format is specified as a roughly 200-line LGPL licensed header `mlv_structure.h`.

The initial "RAWv1.0" design was chiefly experimental. The more widely used specification, implemented by the `MLV_Rec` and `MLV_Lite` modules and all the major post-processing tooling is considered internally to be "RAWv2.0." Later developments have not altered the on-disc format, but have focused on optimizing the implementation, maximizing utilization of the camera hardware to extract as much data as possible within the available sensor, memory, and (critically) storage bandwidth. Most camera-side development has been focused on the `MLV_Lite` module since it was forked by David Milligan around 2016 [94].

On the EOS M used for testing, the SD bandwidth has an empirical limit of around 60MB/s, which means — best case — full 14-bit RAW MLVs can be captured for longer than the handful of seconds it takes to fill the camera's internal buffers at a resolution of roughly 1728x692 at 30FPS, and then only if one finds an SD card that happens to sustain the maximum observed speed. This resolution is not particularly impressive by modern standards, but a 14-bit RAW video from a ultra-compact body released in 2012 is quite an accomplishment. Unfortunately, the SD standards are not particularly uniformly implemented, so finding a card that will negotiate to the proper mode involves either locating an exact match to a card verified to do so by another user, or extensive trial-and-error.

The chief optimization of the on-camera MLV tooling is that it uses the EDMAC (above) to perform all the data motion; the sensor read-out is DMA on the EDMAC. The SD card write out is DMA on the EDMAC. The cropping is DMA on the EDMAC. The MLV capture tooling also works around various platform limitations, such as the 4GB file size limit on the FAT32 format volumes used on the SD cards in (most) cameras with integrated support for multi-file encoding.

An ecosystem of processing and conversion tools have formed around the MLV specification. Examples include MLVApp [95], a MLV processing tool initiated by ilia3101 (Ilia Sibiryakov), a sophisticated, open-source, community-built tool for processing the resulting MLV files. MLVApp supports the native formats produced by the various ML supported camera models, and allows a user to post-process and manipulate the RAW video in various desirable ways; sophisticated Demosaicing, highly parameterized exposure analysis and manipulation for toning and look, various forms of RAW correction for dead pixels and noise reduction, and conversion to a wide variety of conventional output video formats. It is distributed under a GPL3 license, and is built chiefly in C++ using the Qt toolkit. Many of its features are the result of community members adding specific functionality they desired, and/or hooking code from other open source developments, such as the librtprocess [96] tools derived from the open source RAW still processing software RawTherapee.

Another approach to accessing MLV data is MLVFS [97], also initiated by David Milligan of `MLV_Lite`. It mounts a MLV file as a virtual file system using the FUSE (File System in UserSpace) facility on UNIX-like systems, which allows the individual frames of the MLV stream to be accessed as though they area directory of DNG format RAWs, and consumed by any software which can operate on DNGs.

Limitations

The current list of camera models supported by Magic Lantern are the Canon 5D Mark III, 5D Mark II , 6D , 7D, 60D, 60Da , 50D, 700D / Rebel T5i , 650D / Rebel T4i , 600D / Rebel T3i , 550D / Rebel T2i, 500D / Rebel T1i, 1100D / Rebel T3 and EOS M, as enumerated on the current builds page [98], with a few ports in progress. The newest of these cameras came out in 2012-2013, as platforms after the DIGIC5+ generation have not yet been adequately reverse engineered to build a working port.

Even those cameras which are supported will have features which are not fully exploitable. As an example of attempting to make use of a not-fully-understood feature, the original application that lead to this work was an interest in using the subtraction channel apparently available the EDMAC to perform on-the-fly frame diffing, for use in ongoing research projects. It can be experimentally verified, using the extensive introspection tools included in ML, that the EDMAC has some sort of subtraction mechanism. This mechanism is exposed in the camera UI for "Long Exposure Noise Reduction."

Actually making use of the subtraction mechanism presents two issues: the setup for two-reader one-writer EDMAC operations — like subtract — is not publicly documented, and the location of the subtraction engine in a particular camera is not stubbed into the ML code.

The second problem is relatively straightforward; a camera with ML active can log EDMAC activity at user-controlled intervals, and on the EOS M a comparison of the logs from a series of otherwise identical exposures, with "Long Exposure Noise Reduction" active and without reveals that EDMAC channel 20 is activated only when the subtraction mechanism is active.

The first problem, however, proved to be beyond reasonable effort for the experiment at hand. Though two-reader-one-writer EDMAC functions are visible in the logs, the existing ML code calling EDMAC functions primarily uses it to implement a fast `memcpy()`, or other one-reader-one-writer functions. When an EDMAC operation is configured, there are calls to `StartEDmac(ChanN, 0);` to configure a channel for writing and `StartEDmac(chanN, 2);` to configure a channel for reading. This leads to the natural conclusion that calling `StartEDmac(ChanN, 1);` might plumb a second reader, but making calls of that form doesn't appear to do anything other than hang the camera.

Many of the findings in this section were were presented at Electronic Imgaging 2024 as "Magic Lantern as a Platform for Digital Photography Research" [99], as they are likely to be independently useful for imaging researchers.

Magic Lantern for TDCI

There is a tradeoff in this project between the desire to perform the TDCI encoding as close to the sensor as possible, in order to maximize the number TDCI's attractive features which can be driectly demonstrated, and the extra development challenges as — developing software on a conventional computer over documented interfaces is much easier than working in-camera where computational resources (CPU cycles,

RAM, storage bandwidth) are scarce, interfaces are poorly or un- documeted With this in mind, several approaches to leveraging the access afforded to a prosumer camera by the Magic Lantern project to create TDCI tooling were considered, guiding which details of the Magic Lantern ecosystem were explored in depth.

First and most potentially compelling was building a TDCI capture mechanism by directly in-body by manipulating the EDMAC engine. The attention above to the APIs for sophisticated patterned readout and two-reader-one-writer operations in the EDMAC was in the service of generating a TDCI stream. While the computer in the EOS M is dramatically too slow and memory starved directly maintain a TDCI scene model, leveraging the apparently-present memory region subtraction feature inside the EDMAC to execute on-the-fly frame differencing looked like an avenue to approach in-body TDCI. Because the `mlvlite` code successfully maintains several frames in memory, stealing a few of those frames for manipulation appears potentially feasible. These memory regions could then be used to maintain a current expected value frame, and series of successive differences between the expected and most recent frame, and that diff and the prior diff, to produce an approximation of the discrete time second-derivative for use in a TDCI model. This case is extremely ambitious on both memory use and arithmetic intensity, which gave it the additional downside of likely requiring a small crop area and slow sample rate.

As a fallback, writing out the diff frames rather than the raw data should allow very simple compression schemes — such as run-length encoding, to dramatically improve the speed with which scene data can be written out by eliminating redundancy. This becomes even more effective if the diffs are thresholded based on a noise model of the camera to eliminate changes below significance - the notion of analytically distinguishing scene change from capture noise appears repeatedly in the larger body of TDCI work. This claim is based on the same scene constancy assumptions that underlie most video compression and TDCI bandwidth claims — the differences of adjacent frames should be small compared to the total scene content. Because raw video capture on the target camera is deeply SD write speed limited, this would make an excellent demonstration of the possibility and value of the redundant-data-stripping features of TDCI style capture. However, since the operation of two reader one writer EDMAC operations has proven elusive, and it is even unclear if the subtraction mechanism is truly general or specifically optimized for the sparse case of dead pixel elimination, further exploration along this line was not tractable.

Another possibility offered by EDMAC manipulation is manipulating the readout pattern. Part of the TDCI premise is that more information can be obtained at the same data-rate with non-correlated sampling. The principle is simple: regions of the image which have recently changed at high frequency are likely to continue changing, while regions which have remained relatively constant do not require frequent sampling. Unfortunately, being limited to rectangular regions, having limited control over the relative timing of the memory transfer and readout circuitry, and lack of processing power to re-integrate this data in body would make maintaining the scene model in time to influence the sampling impossible.

Another possible avenue to leverage the Magic Lantern tooling in pursuit of a more sophisticated TDCI implementation is to reprocess an MLV video stream generated

by the camera on a host computer, in the same fashion as previous experiments with TIK. The benefits of such a setup are chiefly that minimally-processed sensor data could be used for the creation of the IMEV, allowing for the error model to be computed earlier in the process, and subsequently potentially to leverage temporal information in other processing steps.

However, the experimental limit on the EOS M under test of RAW video at 1728x692 at 30FPS is not compelling. The limited resolution, particularly temporal resolution, of MLV streams makes the effort-to-reward for pursuing that avenue for TDCI reprocessing less than promising. As a standalone experiment, the MLV container format and reprocessing tools in MLVApp would make a reasonable platform for exploring temporal demosaicing, but a narrow experiment on that front was not compelling given decades of existing work on frame oriented (inter-frame) temporal demosaicing techniques [30].

Other Platforms and Adjacent Experiments

A number of other camera devices came up in adjacent projects, and some of them bear mentioning in the context of the larger work. Several precursor projects relied on Canon point-and-shoot cameras programmed under the CHDK environment, as discussed earlier when introducing the Canon programming environments 3.12 and in discussions of various sub-projects which use them, such as the early in-camera TDCI implementation 1.1 and parts of the IOSLess 3.1 experiments that largely inspired the rest of this work. Unfortunately, the lack of onboard compute resources — both CPU and RAM — on these devices, the lack of access to the sensor data and configuration at levels below the standard Canon firmware APIs exposed by CHDK, and the mediocre sensors do not make them suitable targets for further development.

Yet a different group of related projects were built using devices which are more embedded development platforms with small image sensors attached, rather than dedicated camera systems. Most prominently the AI-Thinker ESP32Cam product [100]. This board combines an ESP32 microcontrolller development system, and an Omnivisioin ov2640 camera module. The camera captures up to 2MP (1632x1232) color images at up to 15FPS, and is connected over the SCCB bus allowing a great degree of low-level control. The ESP32 module provides a dual core 32Bit Xtensa LX7 processor clocked at around 240MHz — a relatively capable processor among the camera devices explored — but only 520kb of SRAM and 8MB of PSRAM which is extraordinarily restrictive in an imaging device. It also provides a capable wireless stack, and, perhaps most importantly, a vendor-supported development environment intentionally design designed for user reprogramming, unlike the hostile development environments afforded by nearly other every device in the space.

One project involved a much slower scene-constancy and non-uniform sampling based incremental imaging device called Lafodis160, described in “An Ultra-Low-Cost Large-Format Wireless IoT Camera” [101]. This project essentially constructs a 2D polar robot, shown in figure 3.20 that can locate the sensor of an ESP32 anywhere within the image circle of a large-format lens, allowing one to (extremely slowly) programmatically sample any or all of the 160mm diameter image circle projected by

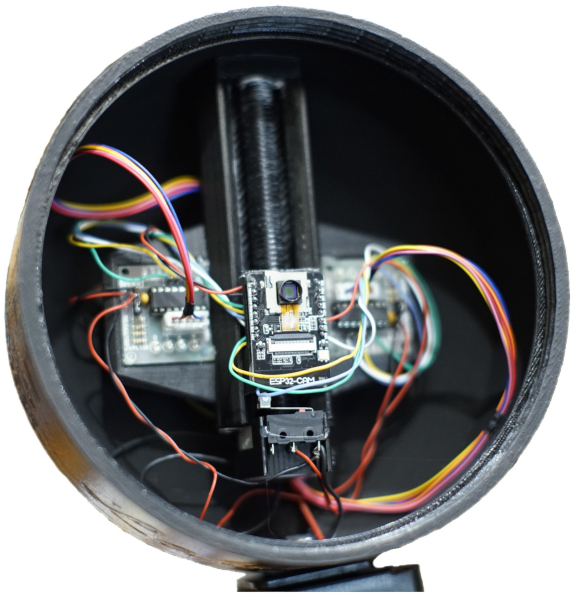


Figure 3.20: The LAFODIS160 polar slow-scan camera

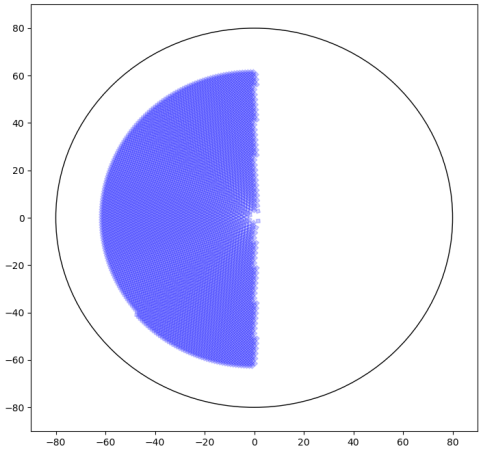


Figure 3.21: An in-progress full-coverage scan for LAFODIS160

the lens, by physically placing the ESP32-CAM's sensor at the location — sampling a square 1.534mm on a side into 1600x1200 pixels of input at a time. My work on the LAFODIS touched on the electronics and motion system, but owing to pandemic-era difficulties in collaborating on hardware, was largely in generating control software to plan and visualize motion patterns; the relation to the larger body largely being that LAFODIS allows (and requires) the image to be constructed by a series of potentially-uncorrelated samples of sections of the scene - the output of a script which generates both a visualization of the sensor pattern (blue square per exposure) and the commands for the motion system is shown in the midst of a naive full-coverage incremental sweep in figure 3.21. This bears significant commonality to one of the arguments for non-unifom capture behavior; that exposure parameters can be set appropriately for the area of the scene, and areas of the scene with a history of faster-changing content to be recorded more frequently as a way to reduce data rate while maintaining a relatively sound scene model.

Much of the ESP32-CAM work was in collaboration with a series of undergraduate researchers, which resulted in, among other products, a publication "ESP32-CAM as a programmable camera research platform" [102].

Unfortunately, restrictions like the limited memory and slow low-resolution sensor make the ESP32-CAM module entirely unsuitable for anything to do with implementing TDCI or other decoupled capture and integration imaging systems, but they do enable a wide variety of interesting experimentation on camera systems.

3.13 NUTIK

Based on the relative promise of the early non-uniform model and the greater-than-anticipated distance to an end-to-end testbed, the next research objective turned to integrating the lessons from the Octave-based non-uniform integration prototype discussed in section 3.10 into the TIK tooling discussed in section 1.1. This effort is an opportunity to refine ideas developed, improve both the quality and performance of the synthesis process, and generally offer a closer approximation to the proposed functionality.

To this end, a more sophisticated prototype of a non-uniform exposure system was constructed: **N**on-**U**niform **T**IK (NUTIK). This version was constructed by first significantly updating, and then extending, TIK. The updates, mainly in 2022, centered on converting TIK to use the OpenCV library for image input and output; however, there were also a number of bug fixes applied. The extensions added support for reading and applying functions and masks to enable non-uniform integration for rendered images.

NUTIK adds command line arguments specifying masks to separate regions for different integration functions, function specification files to provide lists of integration functions mapped to the mask values. A mask file can be specified to `nutik` with the command line option `-mMaskFile.pgm` and Function files are specified to nutik with the command line option `-kFnFile.fn`, in edits to the front-end code shown in appendix B.4.

A mask file is an 8-bit PGM image (as PGM reading has been deferred to libraries, ASCII encoded P2 or a binary encoded P5 are both accepted). By associating gray level with a mask, a single PGM image can specify up to 256 non-overlapping regions, the union of which is the entire image. Each mask is associated with a different exposure processing function.

This mask must be of of the same spatial resolution as the IMEV, and thus also of the image to rendered, so that each site has a clearly defined gain. The mask can be created by drawing an image with the same spatial resolution as an exposed frame in a conventional image editor. Conceptually, it is difficult to reconcile mask image construction with the fact that an IMEV is not an image. However, one or more NUTIK-rendered frames from the IMEV can be used to to isolate desired correctly exposed features. In general, this is a very powerful way to apply existing image-based analysis to IMEV streams. For example, even image-oriented artificial intelligence methods can be applied to help select regions or objects — as is done in subsection 3.13. Whatever method is used to identify which pixels belong to each mask, the PGM mask image has all pixels belonging to a particular selection assigned the same gray level.

Compared to the Octave prototype, NUTIK uses a different scheme for representing functions and a different syntax for encoding them. Experimentation with the original Octave prototype revealed that, for most purposes, the functions used do not require smooth curves. The simplest, and quite common, case is camera-like exposures specified by boxcar functions whose length in X is the duration of the exposure and height in Y is the gain. The decision to use sophisticated interpolation through a series of control points in the Octave version makes representing boxcar functions difficult, as near-vertical features in higher-order functions or splines tend to produce overshoot, undershoot, or rolled over corners. Interpolation also significantly complicates the internal processing required, slowing rendering. Specifying functions as piecewise linear makes the processing faster and handles boxcars without interpolation artifacts while still allowing good approximations to arbitrary smooth functions. If desired, a separate software tool could be developed to automatically construct piecewise linear specifications from arbitrarily complex smooth functions.

The encoding used in NUTIK specifies one function per line, in the format: `M`*nn*`[`*t0*`:`*g0*`],[`*t1*`:`*g1*`],...,[`*tn*`:`*gn*`]` where *nn* is a value 0-255 for the corresponding mask value, each *tn* is a time in nanoseconds from the start of the capture, and each *gn* is a gain to be applied at that point. Lines not starting with an `M` are discarded as comments.

The points *must* be specified in increasing order by time, such that they describe a function. Consideration was given to sorting times in software, but early experiments made it clear that humans are prone to making errors when the sequence is not written in time order. For example, accidental time gaps and overlaps easily result from minor typographical errors — with often bizzarre results — while time order specifications make such errors more apparent. One convenience feature which was added to the NUTIK implementation is that times before the first or after the last control point are assumed to have the value of the first or last control point. This is again an optimization for the common case in which there are only contributions integrated

```
Digit ::= '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'
Uint ::= Digit+
Int ::= '-' Uint | Uint
Float ::= Int | Int '.' Uint
Tuple :: = '[' Float ',' Float ']'
Fn ::= 'M' Int '{' Tuple+ '}'
Specfile ::= Fn NEWLINE +
```

Figure 3.22: EBNF grammar of the language for specifying exposure functions

from relatively brief windows, surrounded by extended regions of no contribution before and after.

An EBNF grammar for the specification format is supplied in listing 3.22.

This simple textual format is relatively easy to manually write and edit as well as being simple to parse. It also is relatively straightforward to programmatically generate, with an eye toward using it as an interface for potential future interaction with higher level software.

There are two implementations of this exposure function specification file format. One is integrated into the NUTIK codebase for generating exposures as `maskgain.[h,cpp]`, shown in appendices B.1 and B.2. The other is in a support tool for plotting functions.

`FnPlotter.py`, listed in appendix B.5, is a relatively simple Python script which plots all the functions specified in the file as a set of stacked graphs. This is accomplished by parsing exposure function specification files into a data structure compatible with the MatPlotLib [103] plotting library, then generating a subplot for each specified function. This support tool allows for easy visualization of described functions in a human-readable form. In practice, this tool was found to be absolutely necessary as a debugging aid.

Another feature considered but rejected for this prototype is the design of an integrated format that would allow a single file to contain an exposure mask, the function specifications, and perhaps even the IMEV data. The idea of an integrated format seems appealing as an interchange format and for operator convenience, but these components are different enough that there is little practical benefit in combining them. Using plain text and PGM images enables easy manipulation of the files with existing tools.

The bulk of the development in this prototype is dedicated to adjusting the rendering code to follow the function specifications.

NUTIK uses the existing TIK infrastructure to render a `.tik` file from an input video stream, then performs function-controlled rendering of output frames from that `.tik` file. A higher the frame rate for the input video stream generally results in a more precise the temporal resolution for the IMEV. This is a command-line conversion tool, called as `./tik` *Clipname.mp4 ClipName*`.tik` to generate a TIK encoded stream *ClipName*`.tik` from an input video in any of a wide variety of OpenCV-supported video formats. For metadata inconsistency reasons, the frame-rate and shutter angle of the input can be explicitly specified, as in `./tik -f120 -a180` *Clip-*

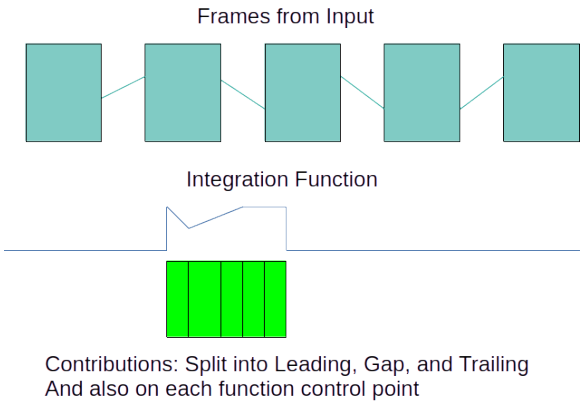


Figure 3.23: NUTIK Data Model

Name.mp4 *ClipName*`.tik`, specifying a frame-rate of $120fps$ (`-f120`) and a shutter angle of 180° (`-f180`).

The tik tool is then called again with exposure parameters to generate a frame, eg. `./tik -mMask.pgm -kFnFile.fn ClipName.tik`. `-mMask.pgm` designates a mask file as described above to spatially map exposure functions, and `-kFnFile.fn` designates an exposure function specification file to temporally map exposure behavior. If the `-m` and `-k` options are supplied, NUTIK defaults to only integrating from the input stream for the interval defined by the control point before and after the first non-zero control point in any function in the supplied exposure function spec file.

During this non-uniform integration step, the TIK rendering mechanism implemented in `render.cpp`, shown in appendix B.3 adds a number of calls to functions in `maskgain.[h,cpp]` to determine the appropriate weight to be applied to each output site from the input samples encoded in the tik file.

A TIK-format IMEV consists of change records for each pixel. Although different pixels do not necessarily have their values change at the same time, for any given pixel, there is a sequence of change records that effectively defines the times at which that pixel's value changed in the scene. Similarly, a piecewise linear integration function implies a sequence of points in time where the integration weighting changes. Any time at which either a change record or a linear integration segment endpoint occurs thus defines an edge at which the weighted value changes. The computation of an output pixel's value is thus a summation over intervals between edges contained within the duration of the integration function. This summation is done using `double` arithmetic, to ensure sufficient accuracy after normalization. The normalization to produce final pixel values simply divides the sum by the total duration of the integration function having a non-zero weight. Thanks to masking, there may be as many as 256 different integration functions being applied to different subsets of the pixels.

A diagram of this data model is shown in figure 3.23.

Because non-uniform exposure is not a concept to which a great deal of existing intuition applies, designing input functions and anticipating results is challenging. A great deal of diagnostic output, (enabled with the pre-processor directive `#define CHATTY` is present in the non-uniform exposure code to inspect the behavior and trace

the exposure at a specified site in the output frame. This extra output is largely to verify that surprising results are the result of intended behavior rather than any sort of software malfunction.

Through these computations, NUTIK can, with extreme flexibility, expose a frame with up to 256 spatial regions and temporally with a function specified in 256 control points per region from an input scene model captured with a conventional camera. Admittedly, nearly all aspects of the user interface to NUTIK are not particularly intuitive nor easy to use, but this is a reasonable target for a system providing a user-friendly interface. NUTIK execution time for rendering an image is generally measured in seconds and scales with the resolution of the image and the duration of the live interval. Typical performance would be around 6 seconds to render a 640×480 image covering a few tens of input samples. However, NUTIK does not currently use any parallel execution, and most of the algorithms and data structures are amenable to massively-parallel execution.

Samples from NUTIK

With the NUTIK tool in hand, we present a number of examples to illustrate both the functionality of the tooling and new abilities presented by the post-capture synthesis of images using manipulable integration functions.

Shared Center for Sharp Margins

Consider photographing a dance performance. The subject of the photograph is a dancer on a stage doing a solo, with other dancers still moving in minor roles on stage behind them. The subject is spot-lit, and the rest of the stage is relatively dark. This situation is absolutely pathological for existing photographic practices.

Taking a short-interval, moderate-gain image to properly expose the soloist and get a sharp image with little motion blur/artifact, the background will be *irrecoverably* dark and likely noisy. Even if you're willing to digitally manipulate in post, some areas in the background will have received an amount of light below the noise level of the sensor, so no real detail can be recovered. Taking a longer-interval exposure to get detail in the background is problematic because the dancers are all moving and will blur. Taking a higher-gain exposure to get detail in the background is problematic because the soloist will be *irrecoverably* blown out. In many circumstances it's now possible to "cheat" with burst shooting and stacking — but the moving figures will create artifacts when composited, because the center time of the different frames will be non-overlapping. The proposed non-uniform integration method suggests sampling the scene for a period around the desired image, then computationally integrating different areas *independently* with different time and gain to allow features. Because the intervals are set computationally after the fact, they can overlap, preventing edge artifacts. Each area of the image (as dictated by a user-generated map) can be exposed optimally, rather than having to select a compromise parameter for the whole scene or stitch from a range of pre-set guesses.

```
M0{[0:0], [1904999999:0], [1902000000:4],
    [1912000000:4], [1922000001:0], [3000000001:0]}
M128{[0:0], [1899999999:0], [1900000000:1],
    [1960000000:1], [1960000001:0], [3000000001:0]}
```

Figure 3.24: HalfSharpFn2.fn

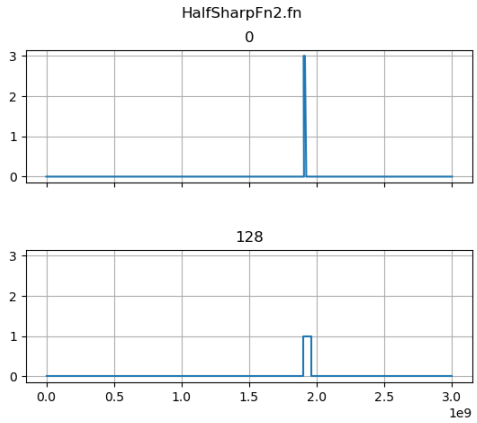


Figure 3.25: Visualization of HalfSharpFn2.fn

Not having the ability to host a dance recital in our lab, we substituted a brightly-colored plush dinosaur being shaken violently in front of a machine room window. The scene was captured using the 120FPS high speed video of a Nikon Z6 III, then cropped to 640x480 to center the subject before tik encoding. Tik encoding via `./tik -f120 -oorange.tik OrangeDinoCropTrim.mp4`, then exposure via `./tik -mHalf.pgm -kHalfSharpFn2.fn orange.tik`

The supplied exposure function specification file `HalfSharpFn2.fn` is shown in figure figure 3.24, also shown graphically in figure 3.25. Note that both of these functions are boxcars, they are overlapping, and are roughly reciprocal: one is three times as high, and one is three times as wide. The two defined regions of the image to be exposed with the specified functions are defined by figure 3.26.

The resulting exposed frame is shown in figure 3.27. The shorter, higher gain function produces a sharper image, as expected because there is less scene motion during the integrated interval, and the longer, lower gain function produces a comparable exposure, but with much more blur as the scene moved more during the integration interval. Because the shorter interval is contained in the longer interval — something which cannot be accomplished with a single camera bracket-and-stack scheme — the sharp edges of the moving (dinosaur) feature of the shorter exposure are inside the blur radius of the longer exposure.

A Negative Gain

As an example of negative gain's utility for feature extraction, consider an exposure rendered from the same input as the previous exposure, using only a single function

Figure 3.26: Half Mask

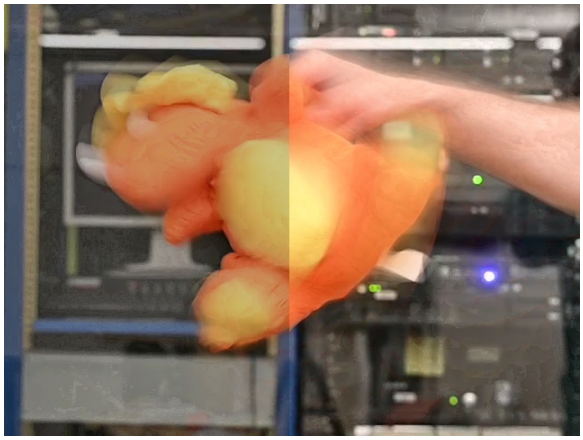


Figure 3.27: Rendered Frame

which makes a positive $1/30s$ exposure with gain 1 surrounded by $1/60s$ exposure with gain -0.5 to either side, all occurring 1.5 seconds into the sampled interval. This exposure is described by figure 3.28, which is visualized as a graph in figure 3.29. This single function is tagged 0, so a black frame of matching resolution is supplied as the mask, to indicate it should be applied to the entire scene. The negative intervals to either side of the positive interval *remove* the light contributions from those intervals, magnifying the specific differences at the exposed instant from its surroundings.

This function is then exposed by calling NUTIK as follows: `./tik -m../AllZero.pgm -k../FeatureExtraction.fn OrangeFast.tik` , resulting in the "diff" frame shown in figure 3.30. While the total subtracted interval is less than the added interval, the gain is slightly lower, leaving a de-saturated image of the static features, but strongly exaggerating the changed regions.

A Time-Varying Gain

As a minimally weird demonstration of a behavior which is wildly unrealizable in conventional cameras, consider a frame in which the top and bottom halves have

```
M0{[0:0.0],[1500000000:0.0], [1500000001:-0.5],
    [1516666000:-0.5], [1533332000:1],
    [1566665000:1],[1566665001:-0.5],
    [1583331000:-0.5], [1583331001:0.0],
    [3000000000:0.0]}
```

Figure 3.28: FeatureExtraction.fn

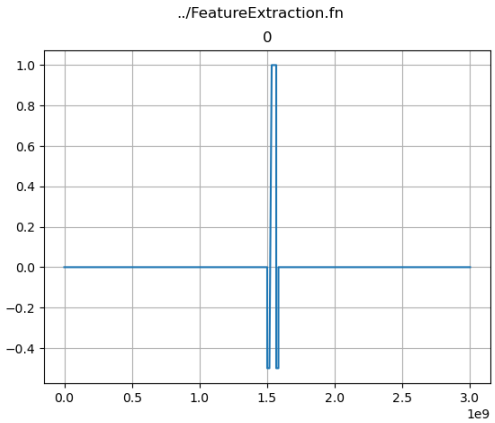


Figure 3.29: Visualization of FeatureExtraction.fn

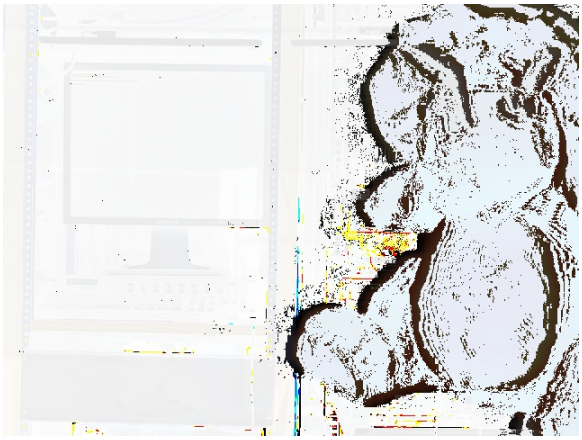


Figure 3.30: Rendered Frame showing exaggerated differences

```
M0{[4000000000:0], [4033333332:0],
    [4033333333:-0.75], [4099999999:2.5],
    [4100000000:0], [5000000000:0]}
M128{[4000000000:0], [4033333332:0],
    [4033333333:2.5], [4099999999:-0.75],
    [4100000000:0], [4500000000:0]}
```

Figure 3.31: RampFn2.fn

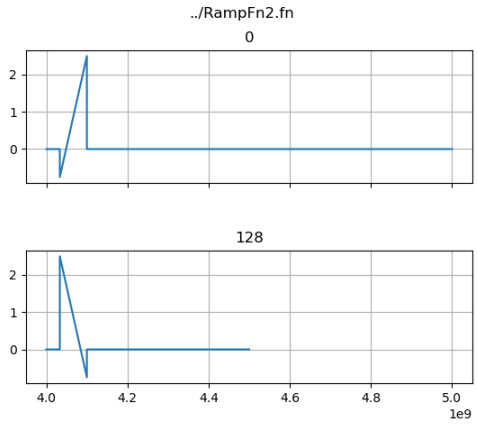


Figure 3.32: Visualization of RampFn2.fn

their gain ramped in opposite directions during the exposure interval. The function in figure 3.31 describes a 2/30 second exposure starting four and one third seconds into the interval. For the top half of the frame, gain is ramped from -0.75 to 2.5, while the bottom half ramps from 2.5 to -0.75 over the same interval. The function is shown graphically in figure 3.32, the split halves in figure 3.33. NUTIK is called with `./tik -m../VHalf.pgm -k../RampFn2.fn OrangeFast.tik` to render the image in figure 3.34.

This essentially describes splitting the sensed area in half and dynamically changing the ISO setting for the two halves during the period of exposure. In particular, the difference is evident in the appearance of the dinosaur's right foot. It is wildly unlikely this exact behavior would ever be desired, but it serves to demonstrate the generality of the method.

Video Resampling

Video resampling for frame rate shifting, as discussed in section 1.2, has been previously noted as a useful application of TDCI models and TIK-style tools that synthesize a scene model from an input video. This set of example exposures illustrates some advantages of adding non-uniform exposure to that formula.

Figure 3.35 shows a frame grabbed from the 960FPS input video of a lurid pink dinosaur being shaken in front of a calibration target, used to generate the IMEV from which the later frames are rendered, representing the 0.00104 seconds of scene

Figure 3.33: Vertical Half Mask

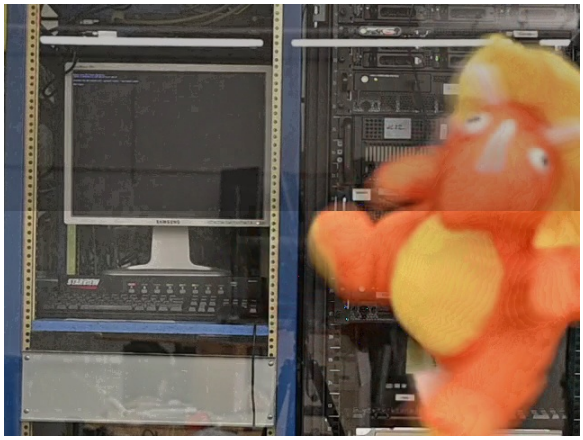


Figure 3.34: Rendered Frame showing the effect of ramp functions

Figure 3.35: Frame grabbed from the input video at 0.325 seconds.

data centered at 0.325 seconds. Under common frame rate shifting methods, this frame would be extracted and displayed as the frame for the surrounding interval.

Figure 3.36 is a virtual exposure from the IMEV, rendered from 0.3 to 0.35 seconds with a boxcar function at gain 1, as rendered in figure 3.37. This provides a more faithful representation of the scene content over the longer interval — observe the motion blur appropriate to the exposure time and scene motion. This rendering is

Figure 3.36: Virtual exposure over boxcar 0.3s to 0.35s

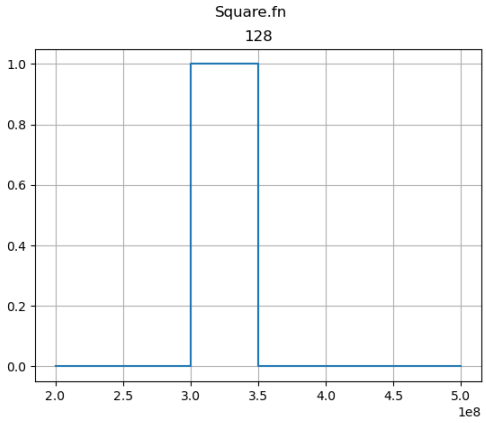


Figure 3.37: Exposure function used to generate figure 3.36

still a bit more sophisticated than simple interpolation over the frames in the interval, because it is aware of the shutter angle and noise properties. Note that, unlike most of the examples here, this is a rendering that the original TIK software could implement.

Finally, figure 3.38 shows a virtual exposure rendered over the same 0.3s to 0.35s interval, but with a triangle function with peak gain 2 centered on 0.325s rather than a simple boxcar. This rendition smoothly incorporates data from the entire relevant interval, but provides sharper edges to features by emphasizing the state at the center time, creating a frame which correctly represents the state of the scene during the interval in question — preventing discontinuities between frames — but also preserves sharp edges of moving features. This rendition is a unique ability of the new work presented here; dynamically varying the gain of scene contribution over time is not a practice common to any well-know photographic practice. The closest physical analog to this behavior is likely a leaf shutter which admits more light during the center of the time interval than during the temporal edges when the leaves are moving. The leaf shutter, however, also changes the effective aperture during the exposure, which produces significant side effects (i.e., the changing of depth of field).

The desirable effect produced by a ramped exposure method is apparent when comparing the black squares in the background in the upper-right-hand corner of the three frames. In the grabbed frame the squares are sharp but at the position in the scene at exactly 0.235s. In the boxcar exposure frame, features have hard-edged

Figure 3.38: Virtual exposure over triangle function from 0.3s to 0.35s

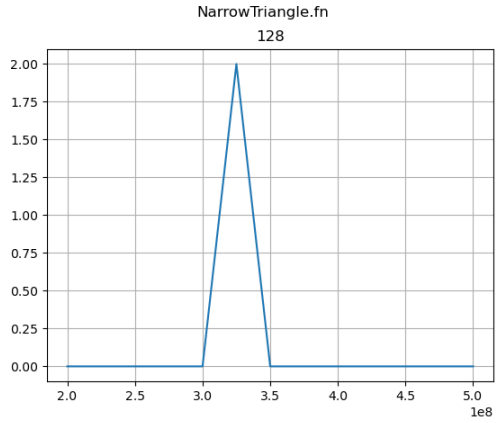


Figure 3.39: Exposure function used to generate figure 3.38

motion trails, producing a larger image of each square. The triangle exposure frame provides a relatively sharp individual frame, but with a softer edge allowing moving objects to appear to be moving in a more fluid way. This effect is also evident when looking at the dinosaur overall; the triangle exposure seems to be simultaneously sharper and with softer, more natural, edge blur.

A Motion Study

To provide an example of the utility of a tool such as NUTIK for representing and studying time-varying phenomena, a scene was contrived in which a UK-branded basketball rolls down an inclined track while being filmed at 960FPS with a Sony RX100IV. The resulting video was then trimmed to length and converted to an IMEV with NUTIK, `./tik -f960 -a200 -oBallRoll.tik BallRollCrop.mp4`.

A common attempt to illustrate this sort of motion might consist of a multiple exposure or stroboscopic photograph. Such an image is straightforward to generate with NUTIK; exposing a function specifying a series of boxcars of desired width — say 1/100 of a second — at the desired times, chosen here to be t=0s, t=0.4s, t=0.8s, and t=1s. An appropriate function can be written as shown in figure 3.40 and plotted in figure 3.41. The result produces is the image figure 3.42. This processing is much easier to implement and more flexible than attempting to synchronize a shutter or

```
M128{[0:1], [10000000:1], [10000001:0],
    [399999999:0], [400000000:1], [410000000:1],
    [410000001:0], [799999999:0], [800000000:1],
    [810000000:1], [810000001:0], [999999999:0],
    [1000000000:1], [1010000000:1], [1010000001:0]}
```

Figure 3.40: BallMultiple.fn

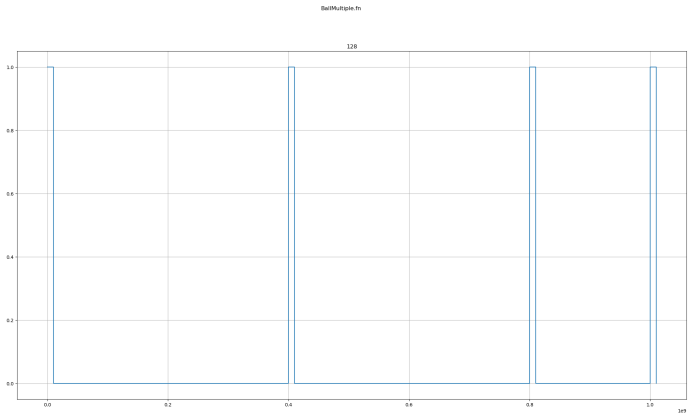


Figure 3.41: Plot of the function generating a virtual multiple exposure image

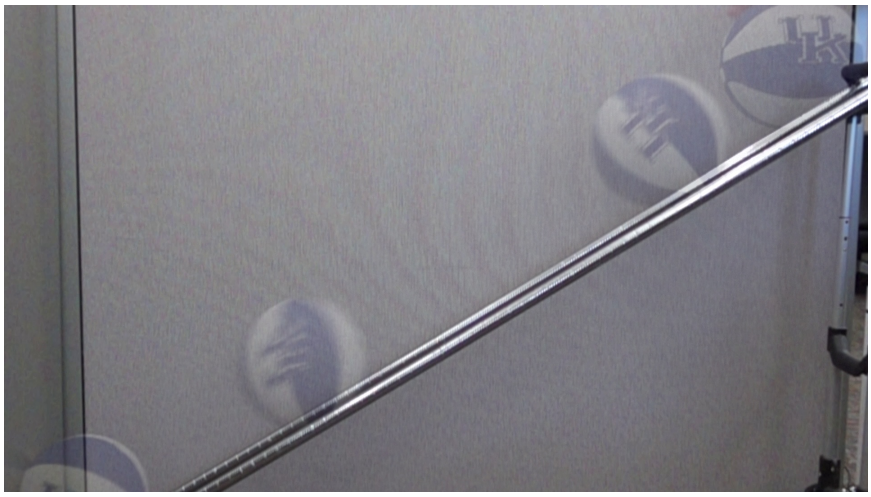


Figure 3.42: A virtual multiple exposure of a ball accelerating down a ramp.

strobe with the motion of the ball at the time of capture. While this is a reasonable rendition, the individual exposures do not produce particularly well-defined images due to the background bleeding through from the other constituent exposures, and there is not much in the way of blur to suggest motion.

The NUTIK tool can instead be used to produce much more sophisticated motion study images which more intuitively represent the behavior of the accelerating ball, showing its state at several sub-intervals while also correctly rendering the continuous

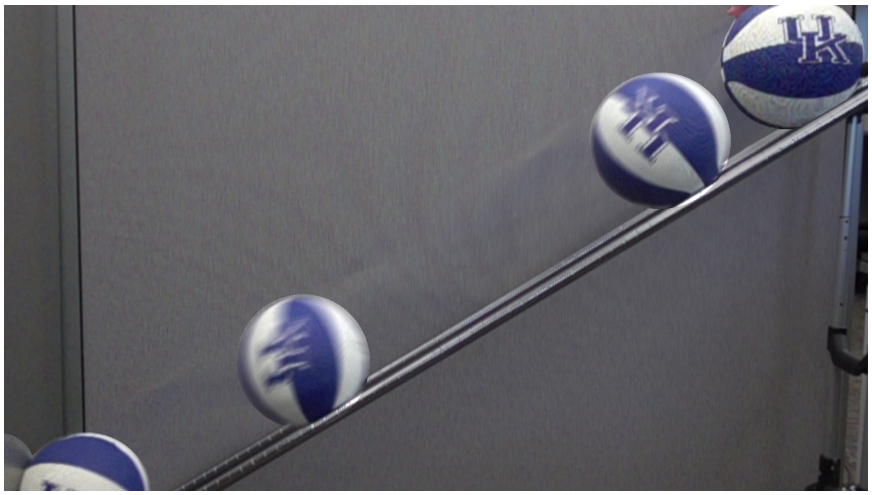
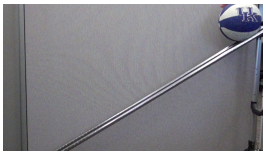
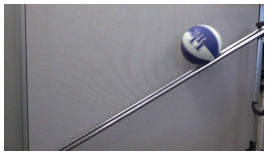


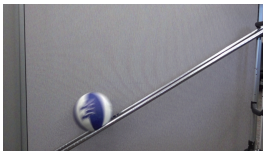
Figure 3.43: A motion study of a basketball accelerating down a ramp



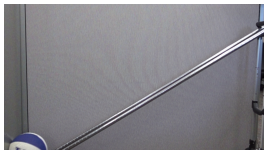
(a) 1/100s at 0s



(b) 1/100s at 0.4s



(c) 1/100s at 0.8s



(d) 1/100s at 1s

Figure 3.44: Virtual exposures rendered in the process of designing this motion study

motion blur for the entire second from the IMEV. This final image is shown in figure 3.43.

To compose this exposure, a series of exposure functions each of which exposes for 1/100 of a second — an interval not corresponding to a source frame — with weight 1 were used to generate frames from the model starting at t=0s, t=0.4s, t=0.8s, and t=1s respectively. These functions exposed the full frame by use of a solid mask, eg. `./tik -m1080128.pgm -kBall2.fn BallRoll.tik`. These start times were adjusted interactively; the first attempt spaced them at 0.2 second intervals, but the ball at 0.2s was found to overlap the ball at 0s, which detracted from the desired effect, so the spacing was adjusted and re-exposed from the model until the short exposures rendered the ball in the desired locations. This series of frames is shown in figure 3.44

Owing to the short virtual shutter speed, each of these frames produces a reasonably sharp rendition of the ball in the position it occupied at the time of the virtual exposure. However, the first and last are entirely crisp as the ball was nearly still during those intervals, while the second shows slight motion blur and the third even

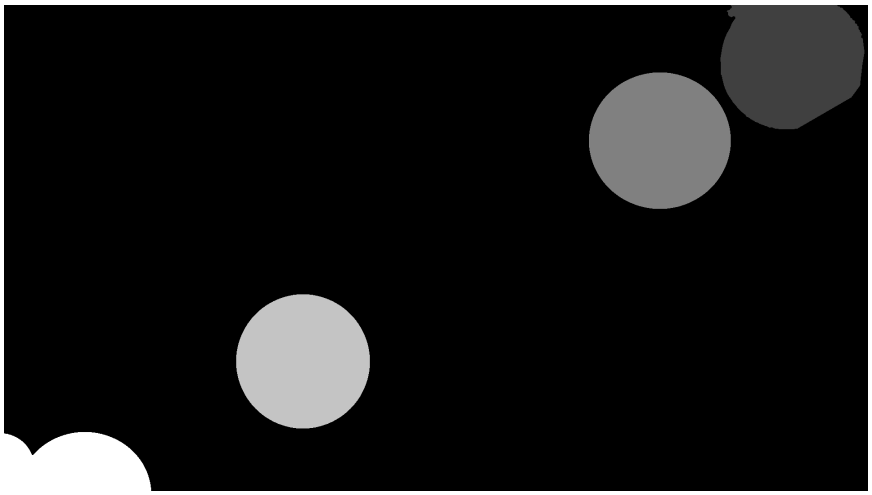


Figure 3.45: The mask used to generate the accelerating ball motion study

more for the same exposure interval. This corresponds with expectation as the ball was accelerating down the ramp during the captured interval.

Once the four individual frames were exposed, they were loaded into an image editor, and the area occupied by the ball in each frame was selected with standard selection tools. Those selected areas were then transferred to a mask with a background of value 0, and bucket filled with tag values of 64, 128, 196, and 255 respectively to create BallMask.pgm shown in figure 3.45

An additional frame was exposed for the entire interval from t=0 to t=1s with weight 1, to render a photographically-correct motion blur of the ball, with the streak of the ball becoming increasingly less saturated as it accelerated and hence occupied each position for less of the interval. The motion trail in this 1s exposure was deemed "too subtle" for the desired effect, so a 1/100 second exposure after the ball had come to rest with negative weight was added, to reduce the contribution of the static scene content and exaggerate the moving element (the streak of the ball). After several iterations, weight -25 was found to nicely accentuate the desired effect, resulting in the image in figure 3.46 Note that there is a small patch of corruption due to an un-representable negative integration result near the bottom-left-hand corner of this frame; this will not be an issue in the final rendering as that area is masked to be exposed with a different function.

The individual exposure functions were then combined into a single function specification, `BallComposite.fn`, shown in figure 3.47, a plot of which generated with `FnPlotter.py BallComposite.fn` is shown in figure 3.48.

Finally, NUTIK is run one more time, as `./tik -mBallMask.pgm -kBallComposite.fn BallRoll.tik` to create the final exposure shown above in figure 3.43, rendering a final image from the scene model which selectively composites the fading motion trail of the exaggerated long exposure with shorter exposures from several short subintervals, all of which are of time and duration chosen interactively after the time of capture, to create the final motion study.

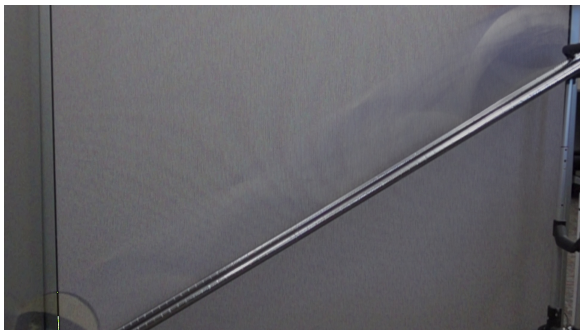


Figure 3.46: A full 1s virtual exposure of a basketball accelerating down a ramp

```
M0{[0:0], [1:1], [1000000000:1], [1000000002:-25],
    [1010000000:-25], [1010000001:0]}
M64{[0:1], [10000000:1], [10000001:0],
    [1000000000:0]}
M128{[0:0], [399999999:0], [400000000:1],
    [410000000:1], [410000001:0], [1000000000:0]}
M196{[0:0], [799999999:0], [800000000:1],
    [810000000:1], [810000001:0], [1000000000:0]}
M255{[0:0], [999999999:0], [1000000000:1],
    [1010000000:1], [1010000001:0]}
```

Figure 3.47: BallComposite.fn

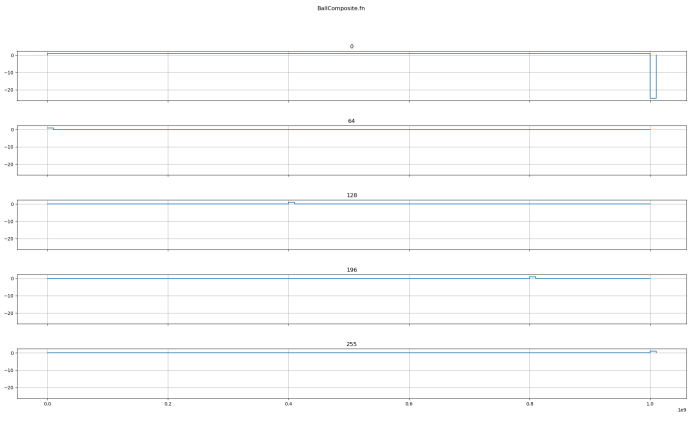


Figure 3.48: Plot of the function specification for the motion study

```
M0{[0:0], [600000000:0], [800000000:2],
    [800000001:0], [1000000000:0]}
M64{[0:1], [10000000:1], [10000001:0],
    [1000000000:0]}
M128{[0:0], [399999999:0], [400000000:1],
    [410000000:1], [410000001:0], [1000000000:0]}
M196{[0:0], [799999999:0], [800000000:1],
    [810000000:1], [810000001:0], [1000000000:0]}
M255{[0:0], [999999999:0], [1000000000:1],
    [1010000000:1], [1010000001:0]}
```

Figure 3.49: BallCompositeOneBlur.fn

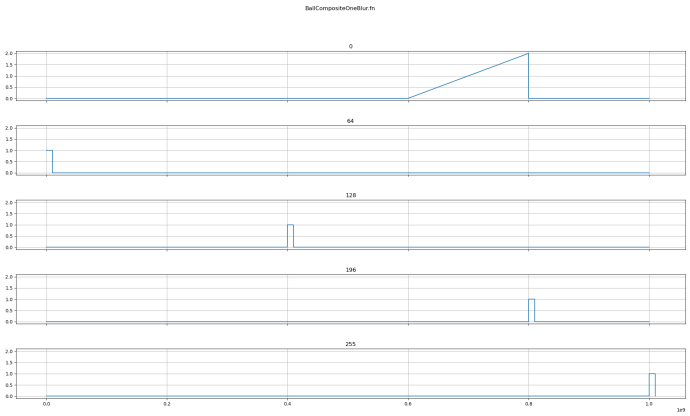


Figure 3.50: Plot of functions to produce blur only leading to one ball position

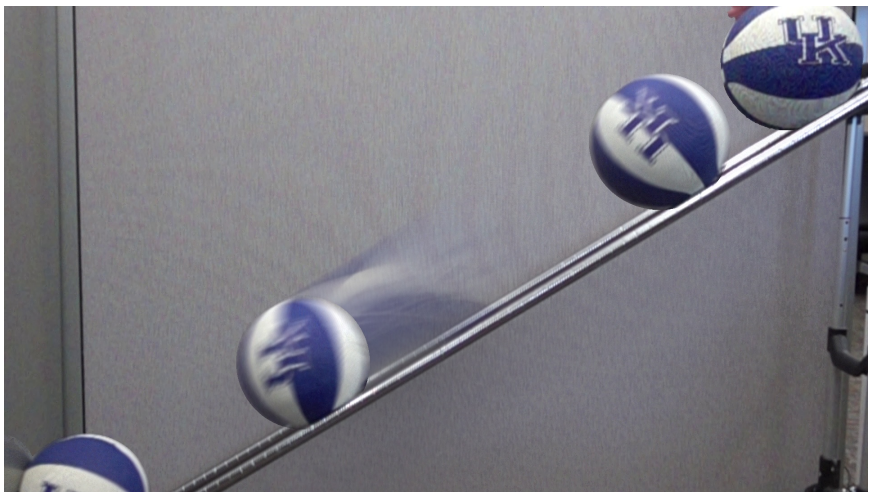


Figure 3.51: Final rendered motion study with blur only leading to one position

The authors is aware of no other tool which could directly create such an image, and particularly not from a single camera or without direct control of the scene. Multiple exposures from a single camera would require synchronization at the time of capture to capture the ball at each desired time, and would not provide the motion blur path. Additional camera(s) could be used to capture the long and short exposure(s), which could then be composited in post, but that would require perspective correction and alignment to synchronize the exposures. With complete control of the scene lighting, a similar effect could be produced from a single long exposure with a strobe synchronized to dramatically increase the lighting for the shorter intervals, though adjusting the respective light levels to create such an image would likely require extended experimentation. This means such a process could not be used on a scene where the lighting was uncontrolled, the event could not be precisely repeated and/or measured, and the subject was sensitive to lights — all of which would be the case if one were trying to generate such a motion study of, for example, an athlete.

To contrive an interesting derived case which is even more physically unrealizable, if one wanted to specifically emphasize the motion blur as the ball approached one specific "frozen" location, it is relatively straightforward to modify the exposure function for the background to specifically emphasize the motion blur leading up to one of the short exposures. In this case, the function is modified to be a ramp from 0.6s to 0.8s with maximum height 2 to fade in the motion blur leading up to the area the ball occupies during the exposure beginning at 0.8s. This is performed with the function spec file shown in figure 3.49, plotted in figure 3.50 resulting in the image shown in figure 3.51.

Selective Exposure with Moving Objects

As an example of the temporal flexibility afforded by this system, a short 120FPS video of two of the author's cats playing with a wand toy was converted into a IMEV.

The rendered frame, shown in figure 3.52 aims to properly expose the scene and both cats, one at the exact instant it connects with the toy, and the other when it is facing the former with its eyes open, and despite the fact that it is in shadow.

To compose this image, frames were rendered from the IMEV at varying center times until the desired instant where the leaping cat connected with the puff was identified. The width of the interval around this center time was then adjusted to produce a desired exposure for the background content.

A mask, shown in figure 3.53, was then generated by drawing over the newly created background frame at the desired instant to isolate the areas occupied by the cats, tagging the background with one value (0), and each of the two cats with unique values (64 and 128) to allow the exposure on the background and each cat to be adjusted independently. The length and gain of the functions exposing the cats was then adjusted for intervals *overlapping* the interval used to expose the background, such that each cat was independently situated and exposed as desired. The selected functions are shown in figure 3.54; producing the desired image entailed exposing the orange cat, tagged 64, for a shorter interval toward the end of the interval used for the background at higher gain, to cut off head movement early in the interval,

Figure 3.52: Rendered frame from intervals independently selected after exposure

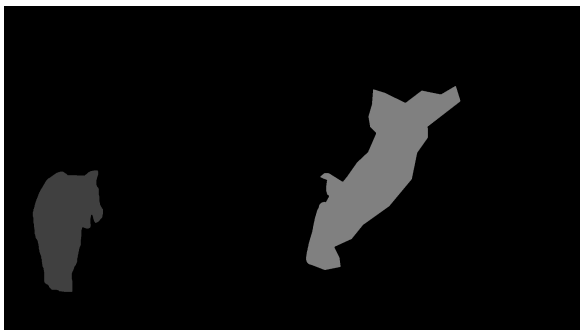


Figure 3.53: Mask used to select regions of the frame to separately expose

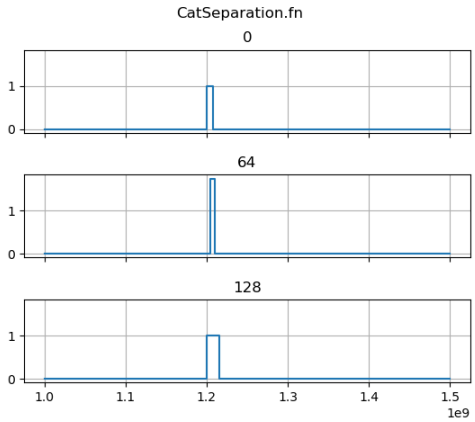


Figure 3.54: Functions used to expose the different sections of the cat frame

providing sharp features at consistent overall brightness. The black cat, tagged with 128, was exposed for a longer period extended around the interval used to expose the background to provide more motion blur.

This example was chosen to illustrate that several of those steps described are difficult or impossible with a conventional imaging process, even when each constituent exposure is a relatively conventional boxcar function. Selecting the exact instant and interval over which a frame will be exposed *after the time of capture* can only be approximated by a conventional frame-capture process. Typically, capturing an exact instant relies on the reflexes of the camera operator, and may be mechanically

assisted by shooting several closely spaced frames around the time the operator triggers 1.2, but that only improves the situation if the exposure is correct and one of the captured frames happens to line up with the desired instant. The very basic TDCI operation of selecting when the exposure interval is after the fact is unusual.

Furthermore, adjusting the exposure *interval* after the fact is essentially impossible. Exposed frames give only the average value for the scene over the exposure interval, so directly generating a shorter or longer interval from rendered frames — especially from frames of similar shutter speed and spacing to the desired result — is at best a matter of artificial blur or de-blur around moving elements rather than a genuine change of interval, as the fine-grained temporal information has already been lost. Next, while many stacking methods allow for mask-driven composition, getting to change the intervals independently and *after* the mask is created from scene data is not an option when stacking frames. In addition to the freedom of exposure intervals being varied independently after the fact, this process also illustrates that the intervals can be *overlapping* which is impossible from an image composited from captured frames; each captured frame in a burst from the same camera must be from a separate non-overlapping interval (and shooting from multiple cameras requires setting up and controlling multiple cameras followed by perspective correction).

Improvements for NUTIK

Quite a number of improvements to further enhance the NUTIK prototype present themselves.

The major category is helper utilities to ease the creation of suitable masks and exposure functions. These tasks are currently quite laborious. Exposure functions in particular are also rather mentally taxing to design, as they represent a degree of freedom for which there is no analog in other practices.

The addition of helper utilities to allow graphical click-and-drag creation of function specification files would improve the user experience and lower the barrier to entry to setting up a development. Supplanting a visual tool with a library of examples to help tie functions with useful — or at least aesthetically interesting — effects would further improve the utility of visual representations of functions.

Helpers for generating useful masks could also be constructed relying on contemporary imaging technology. The same foreground-isolation methods used for features that typically carry names like “blur background” or “portrait mode” could instead be used to generate a mask separating foreground and background features, allowing selected foreground objects to be easily exposed differently than their surroundings.

On the programming front, the NUTIK code was developed to with the principle of not introducing any inessential complexity to the prototype, and this carries a variety of potential performance implications.

Determination of the weighting of each sample contribution involves a chain of dependent memory lookups; the gain depends on the ending control point in the function, which depends on the beginning control point point in the function, which depends on the mask value for the location. Dependent memory fetches are potentially extremely slow on most computers, but the size of the data structures in question is

such that architectural caches should largely hide the problem. Smaller less sophisticated host processors as found in embedded systems tend to feature memories which are faster *relative to the processor*, so these concerns would not be exaggerated in embedded implementations, as onboard a camera. Adjacent output locations tend to be exposed by the same function, so adding simple caching to the `weightherebetween` and/or `gainherenow` may offer some low-hanging performance gains. Each of the lookups above are also currently implemented as forward linear searches; switching to a more efficient search algorithm is, in principle, a drop-in local change with no inter-function implications.

Fortunately, empirically the non-uniform exposure changes add less than 2% to the execution time of earlier TIK versions supporting only uniform exposure, making these performance concerns not a pressing issue.

Improving the performance of the base TDCI processing in TIK should also be relatively straightforward; like most image processing problems, rendering an output image from input samples is embarrassingly parallel across the spatial dimensions and color channels of the output image, and easily decoupled across time dimensions. This means that, if someone were foolish enough to try, it should be as straightforward as any parallelization ever is to map the loops that walk the image locations and color channels to parallel-for type constructs which would execute efficiently in stripes or patches across a large number of independent cores or even — as the operations for each site or channel chiefly only differ by data — SIMD or GPU hardware.

Temporal resolution of the NUTIK tool is dependent on the maximum video framerate supported by the device used to capture the input video, but could be improved beyond the rate provided directly by the capture device by incorporating the temporal super-resolution from shutter behavior techniques proposed in 2017 in "Temporal super-resolution for time domain continuous imaging" [5].

Chapter 4 Discussion

Much of this work has been an exercise in exploring the degree to which digital cameras are *simulacra* of film cameras. Though the super-majority of the photographic [104] and cinematographic [105] market has shifted from film to digital sensors, storage, and processing, the devices, practices, and tools have remained firmly anchored in practices derived from film cameras. Rather than being superseded by digital cameras designed *as digital cameras*, film cameras have been supplanted by digital cameras and tools that go out of their way to simulate their film predecessors.

4.1 A Better Way To Use a Digital Camera

Crutches like the "Sunny 16 Rule" — an alliteratively convenient aphorism whose origins appear lost to time stating that "On a sunny day set aperture to f/16 and shutter speed to the reciprocal of the film speed for a subject in direct sunlight." — are effective for allowing a human to quickly estimate a reasonable exposure given a few fixed parameters for an entire scene at time of capture, but that is not the circumstance under which modern photography occurs.

For much of their history cameras have been increasingly automated. Photoelectric light meters integrated into the camera body and coupled to the exposure mechanism became common by the 1930s, with cameras like the Contaflex (1935) and Super Kodak Six-20 (1938). These devices are simple averaged-intensity meters with simple couplings to the operation of the camera, but allowed the operator to at least partially defer to automation. Microprocessors appeared in consumer cameras long before digital sensors — in the mid 1970s products like the Canon AE-1 (1976) [106] consolidated some of the profusion analog electronic and clockwork mechanisms for camera automation into a single IC as one of the early applications of microcontrollers. Within a few years, the now familiar so-called "PASM" (Program, Aperture Priority, Shutter Priority, Manual) [107] control scheme for expressing user priorities to a computer agent emerge with the Canon A-1 (1978) and have remained more or less remained a constant in the "more serious" camera market segments since.

Not only are most cameras equipped with some form of integrated sensors and computer controls which combine the sensor data with some indication of user intent to actually drive the camera mechanism, modern digital photography almost always has an embarrassingly powerful computer in the loop with the main sensor data which can easily make specific, localized decisions about exposure. In particular, modern mirrorless cameras are continuously sampling and processing the output of the sensor -the screen or EVF (**E**lectronic **V**iew **F**inder) is sampled, processed, automatically exposure controlled stream. Furthermore, the gain of the captured image is not a static feature. Where with photosensitive emulsion film the gain is set for the entire scene long before the exposure when the film is loaded into the camera, in a digital camera the gain can not only be altered on the fly, it can be altered — at least to a degree — *after the fact* with no apparent loss of image quality, and in principle set differently for different parts of the frame, though few cameras expose much control on that front. Thus a key lesson from this work is that photographers should consider

a different paradigm for setting exposure, aiming to expose to *capture the maximum amount of secene data* rather than to capture a pleasing image, then generate the desired integrated frame after the fact by applying digital processing to the gathered scene data. The feature to optimize on in this paradigm is to minimize the number of sensels which are out of range in the scene — the scene information gathered from a sensel which is (within noise level of) saturated or (within noise level of) black is much lower than information from a sensel which is anywhere within its range, no matter how badly exposed.

4.2 Capture, Then Integrate

A major suggestion from this work, like previous studies into TDCI, is that there should be a decoupling of capture and integration. In film photography, the gathering of incident light and because the process by which the scene information is absorbed is the same photochemical reaction that renders at least an intermediate image (a film negative). Digital cameras are currently operated under the same regime, but the capture and integration processes are not inherently linked.

The main challenge to separating the capture of a scene model or record of incident light for each point in the scene is managing the data-rate. More sophisticated sensor readout schemes with faster, more numerous ADCs to read out the sensor, and faster, larger memories in cameras all contribute to the feasibility of such a system, but do not solve the problem entirely. Emerging alternative read-out regimes, like most event cameras, tend to suffer from information loss due to "swamping" in the event of correlated motion producing events exceeding their readout bandwidth. The TDCI method of recording second derivatives — changes to the rate of change — of incident light, clipped by a noise-model-aware threshold is extremely promising, but still far from credible hardware implementations. Fortunately, the specific details of how the data-rate to support capturing a continuous scene model are independent of the idea that that separation brings benefits.

The benefits of this separation are substantial. Recording a scene model defers the decision about exposure parameters until after the time of capture. Many photographs are spoiled by failing to correctly estimate the correct timing, duration, and gain for exposure under particular lighting conditions. In a simple example, trying to photograph a young child running presents the photographer and/or their computer agent in the camera with an array of parameters that separate a delightful image from something unusable. A small change to exposure time separates a delightful picture of a child running from a tragic picture of the child falling to the ground. A small change to exposure interval or sensitivity separates clear rendering from a blown-out or dark and grainy frame. Conventionally, all these decisions must be estimated at least approximately correctly before the instant to be photographed.

Deferring those choices allows the photographer to not only carefully select parameters at their leisure, it allows them to virtually re-expose the same scene an arbitrary number of times and examine the results until the desired result or results is obtained. Furthermore, the gain and interval of integration is not constrained to a single value for the entire frame; portions of the frame can be arbitrarily exposed

with different times and effective sensitivity to obtain desired results which may be out of reach of the dynamic range of the recording sensor device.

As an example a scene with a dim field and a fast-moving but more brightly lit subject presents a problem for traditional photographic practice. This is not an uncommon situation; for example, a moving performer under a spotlight presents exactly this situation. ISO and exposure interval set to properly expose the subject will not only underexpose the background, there will be visible noise in the below-noise exposure. ISO and exposure interval set to avoid a dark, grainy background will leave an excessively bright subject with significant motion blur.

In a decoupled, non-uniform exposure scene, instead of attempting to produce an appropriate image on the fly, the photographer instead captures the incident light during the desired interval. Later, they can use the record of incident light to assemble — for example setting the interval of integration such that the subject is properly exposed with the desired amount of motion blur. They can also *independently* determine a set of exposure parameters which will expose the background properly with a minimum of noise. Finally, they can mask off the regions of the image and generate an exposure using the two functions for different areas, but using the records of incident light centered on the same instant.

A similar trick can almost be performed to a degree with burst shooting, but that doesn't allow the intervals for the differently integrated regions to overlap in time — a serious problem if, for example, the subject is moving.

Even stranger things are possible. The gain doesn't have to be positive — it's perfectly possible to subtract the light gathered during an interval from the light gathered in another. Subtraction is useful for isolating features — particularly differences from the average of a period.

4.3 Violating Assumptions Makes Everything Harder

This work violates a number of deeply-held assumptions across a number of concepts, and many of the challenges derive from the tendency that the more foreign something is, the more difficult it is to integrate with existing technology. The *variety* and *magnitude* of those violation was not clear until a deeper investigation, and a direct discussion of some of those assumptions and how they became rooted is worth having.

The first, most obvious, and deepest violation is that the vast majority of other imaging technologies are frame-oriented, while this work treats time (and gain) as continuous dimensions. In the discussion of mechanical shutters and emulsion photography, it's clear where photographers obtained that assumption: the use of APEX shutter and film speed graduated by doubling and halving in order to make exposure calculations tractable for a human operator impose a consistent, discrete scale on both, and that assumption is deeply ingrained into professional practitioners. Likewise, mechanical movie cameras operate on series of frames of film exposed for consistent pre-determined intervals — the entire concept of shutter angle is premised on a mechanical method for discretizing frame rate and exposure intervals. That analogy constrains shutter angle to be less than 360°, but there is no such constraint on shutter angle in the current work.

Early digital photography carries over many of those assumptions simply as a matter of practice and technique, and those assumptions were baked into software tools for manipulation of digital images. Interestingly, digital *video* encoding tends to have a continuous-time component; motion compensation is applied as a form of compression to allow many frames to be derived from a single complete reference/key frame by recording changes from the reference rather than complete frames, an offshoot of ideas about Scene Constancy and Optical Flow discussed earlier 2.4. Given that almost all video formats operate in the optical-flow domain for storage and transmission, it would be nice to edit directly in that context instead of by decompressing a section into a frame then re-compressing.

In an even deeper, broader form, humans — at least culturally — don't tend to think of time as a continuous dimension. We talk about discretized time — seconds, nanoseconds, intervals — but don't even have language to discuss time as a continuous phenomena the way we discuss spectra for frequency ombré for color.

4.4 Changes in Cameras

Much of the content earlier in this dissertation has been devoted to discussing changes in camera technology, and how various assumptions have persisted despite the decline of the technologies they were grounded in. Using the same lens, it makes sense to look at current trends in camera design, both to see which assumptions they might violate or reinforce, and to see how they might interact with the proposed design of cameras that operate on non-frame-based capture and post-capture synthesis.

The Rise of MILCs

One change we are toward the end of which seems to be informed by an effort to transition from a film-oriented to sensor-and-computer oriented camera design is the gradual transition from **S**ingle **L**ens **R**eflex (SLR) digital cameras to **M**irrorless **I**nterchangeable **L**ens **C**amera (MILC) designs.

The point of an SLR was to be able to compose and focus your image by directly looking through the taking lens. Early electronic sensor designs could not provide a reasonable quality live view, so the industry made **D**igital **SLR**s (DSLRs) that differed from film SLRs only in that an electronic sensor was placed where the film would have been. As sensor technology progressed, it became feasible to obtain a usable live view, but DSLRs designs continued because the phase-detection autofocus mechanism that evolved in film SLRs could not be implemented using the main sensor; a mirror of some sort was needed to give a separate phase-detection sensor access to the image projected by the lens. Eventually, methods for including phase-detection mechanisms on the main sensor (either by masked pixels or dual pixels) removed the last obstacle to simplifying the camera design to have a single, mirrorless, optical path. Thus, the camera market has dramatically shifted from SLRs to MILCs [108].

Obviously, a camera capable of high-quality live view also can shoot video. Especially with MILC cameras, there has been a fusion of the still and video camera markets. It is now expected that a high-end camera can automatically focus and

shoot quickly and even record high-quality video. In other words, new capabilities have been embraced, but they began as almost unplanned side-effects of sensor evolution.

What has not happened is a full re-examination of why cameras are designed as they are and what new capabilities they should aspire to have. The current work suggests that frameless treatment of time as a full dimension should be a key aspirational goal.

Upcoming Changes in Cameras

Although the frameless handling of time advocated here is an outlier, many other research efforts and some commercial products in development also focus on more flexible handling of temporal properties of cameras.

There are a number of exotic new camera sensor technologies that employ a non-frame-oriented sample-then-integrate paradigm. In the preliminary research for this work, QIS sensors were discussed as a related work to precursor TDCI designs. In the interim, several developments have made QIS and IMEV even better suited to each other. Experimental QIS sensors have now scaled out to a 163 megapixel active pixel QIS sensor design [109].

QIS operates on exposure times of around 100µS to 1mS. This extremely short exposure is because the cells have a quantum efficiency of around 70% in the visible band, and each cell saturates in around 5000 $e-$, making it saturate very quickly in bright light. Bandwidth is also problematic, even with 66 400MSPS LVDS lanes to transfer data off-chip, the readout system in the cited design limits the sensor to around 7.5FPS at full resolution. That is far too low a frame rate for video, let alone for processing with a tool like NUTIK.

An IMEV model, however, has several things to offer to this line of development. Using TDCI to replace the already sophisticated on-chip readout system could squash redundant samples unchanged from exposure to exposure, allowing for more efficient use of the off-chip bandwidth, and thus increase frame-rate in a frame-oriented readout mode. Because their readout scheme is already ramp-function driven and their saturation intervals are short, it could also be well suited for even deeper application of a TDCI model with an uncorrelated read out mode that natively generated an IMEV.

The function driven integration that is the focus of this dissertation would also allow such a sensor to simulate exposures longer than saturation limits. Integrating at low gain would allow the rendering of long virtual exposures with characteristic visual effects suggesting motion from combinations of the very short saturation intervals, and doing so selectively would allow absurd linear dynamic range of hundreds of thousands to one.

The quest for higher readout speeds in more conventional imaging sensors also fits well with the idea of providing a native IMEV output. Using the Sony A9 III [110] as an example, the addition of ever-faster readout schemes with more independent ADCs and larger high-speed on-package memory are certainly advantageous to the construction of IMEV-capture cameras. The larger arrays of more ADCs should be

helpful in implementing TDCI-like readout schemes because more ADCs could allow a better approximation to non-correlated readout. At this time, there is no indication that the camera firmware would support this type of use.

For large, fast buffers, the A9 III in particular claims "1.6 seconds of 120fps photography at 14-bit Raw quality" and those are 24MP frames, implying approximately 7.5GBytes of buffer which can be filled as fast as it can be exposed and read out. This large buffer would provide plenty of room to store the several frame-sized intermediate components of a TDCI model, avoiding many of the memory-pressure concerns that pervaded the extensively studied cameras, and in particular the EOS M.

Another emerging feature typified by the A9 III is the addition of faster rolling shutters or even global shutters in cameras with competitive image quality has mixed implications for TDCI-like technology. These shutters produce image captures in which sensels are sampled much closer to simultaneously — fully correlated sampling. However, global shuttering prevents the sort of structured shutter estimation that allows post-processed TDCI schemes to resolve finer time-granularity [5]. On the other hand, global electronic shutter implies the ability to dump the entire sensor in parallel, apparently in analog form into a second diode used exclusively as a storage element [111]. That same hardware feature logically could enable implementation of a TDCI-type continuous readout mode.

Of course, any improvement in capture framerate improves the quality and abilities using a tool like NUTIK to reprocess video, burst still captures, or even "open gate" full-resolution raw video captures.

Most Cameras are Phones

Although the market for dedicated cameras has not disappeared, the huge growth market for cameras has been in cell phones — and secondarily in **I**nternet **o**f **T**hings (IOT) devices. Both those classes of devices generally have small sensors and thus limited image quality. However, cell phones have significant compute power onboard, and both cell phones and IoT have access to huge compute facilities as "edge computing" devices.

To make these devices more effective, the market is becoming quite comfortable with cameras that do a considerable amount of processing to render an image. Many cell phones now support features like "portrait mode" which uses selective computationally applied blur to simulate shallower depth of field than is achievable with little to no aperture control. Likewise, many consumer devices now synthesize their output from several cameras. The Light L16 [112] was an exercise in taking this technology to the illogical extreme, cramming 16 camera modules with different sensors and small lenses into a single device, then controlling them in concert and merging the output to allow computational alterations to zoom and exposure. Though the Light L16 failed on the market, variations on the technology are present in most high end cellular phones for years, typically operating on a cluster of 2-4 cameras with optics of different focal length.

Manufacturers have even argued that there are "No Real Pictures" — as Samsung's Patrick Chomet recently claimed [113] in a press release, some time after

Samsung was caught injecting artificial detail from training images into pictures of the moon [114]. Thus, many user-facing cameras are no longer operating in a genuinely frame-oriented mode, synthesizing the image from assembled scene data across multiple sensors, times, and possibly just making-up probable or desirable content from image data in the tools' training set. Use of these methods is a large part of why cell phones have grown such powerful computing resources. Perhaps these same compute resources could be better applied to create and render from IMEV forms? IMEV would provide significant advantages, including noise reduction, without the possibility of making pretty hallucinations.

In the Hands of Users?

As quickly became apparent in the course of this research, thinking about IMEV rendering is neither a small nor incremental change to the standard ways of thinking about photography. Even for the examples presented in this dissertation, it was hard to visualize the impact of various integration functions, let alone to design ones to obtain a particular desired effect. Software tools can help with this, as demonstrated by `FnPlotter.py`, but tools that facilitate use of a model do not necessarily make people comfortable thinking about the model in the first place.

As was discussed in section 1.2, there have been a number of algorithms implemented in cell phones and other imaging systems that perform some operations that approximate portions of what IMEV processing can do. A good example would be the multi-shot low-light modes in many cameras. This is how IMEV processing is expected to first reach users: as an improved version of some "magical" feature already in their cameras. The motivation for cameras using IMEV is not likely to be interest in the new model, but recognition of the fact that it can create "better" images without hallucination.

Once the infrastructure starts to exist in consumer-level imaging systems, it is likely that the user would initially see it only as semi-automatic controls. A crude mock-up of such an interface is shown in figure 4.1. In that interface, the "time" slider would control the center of the desired exposure interval to be extracted from an IMEV. Neither control would affect the brightness, color, focus, etc.; those would be preserved. The sharpness would implicitly control the combination of virtual shutter speed and integration function shape. A lower sharpness would imply a longer virtual shutter speed with a more gently ramped integration function. A higher sharpness would approach a short-duration boxcar integration. Certainly, more sophisticated algorithms could control the internal parameters and more controls could be presented to the user to manipulate the integration function, to implement common types of multiple exposure, object-selection for masking, etc.

It is plausible that we will eventually get fully-realized digital-native camera systems which operate more or less like the IMEV systems proposed here: sampling light then computationally constructing images from the samples with adjustable timing and gain under full user control. However, that requires a big change in perspective. Ren Ng's refocus-after-capture plenoptic digital camera technology, alluded to elsewhere in this document, can be considered a similarly sweeping change of perspec-

Figure 4.1: A mockup of a limited UI for re-timing a frame from a scene model

tive. He founded a company, Lytro, to commercialize the work in his dissertation. Despite the advantage of a century of theoretical precedent and 20 years of more closely aligned prior work, and with $140M of Venture Capital money, the technique still has not found wide acceptance. Two commercial products were produced, but photographers simply were not comfortable with them, and the company eventually failed. Unlike plenotpic cameras, the TDCI-derived decoupled IMEV capture and integration methods discussed in this work do **not** inherently ask the user to accept a loss of spatial resolution in return for the ability to repeatedly manipulate parameters post-capture. However, that does not imply photographers will be any more comfortable with this approach.

Chapter 5 Conclusion

This work suggests that, by decoupling the process of capturing a scene appearance model from rendering an output frame, a number of desirable properties and new degrees of freedom can be introduced to the photographic process. Computationally rendering frames from an **Im**age **Ev**olution (IMEV) model, as defined in the current work, affords a photo editor the freedom to adjust the tonal properties by independently and continuously controlling the effective time, shutter speed, and gain for all pixel sites in a rendered frame. These adjustments are not made at the time of capture, but after the fact from the captured model. Additionally, frames can be rendered for a multiplicity of parameter choices from a single captured model, allowing rendering of frame sequences (video) or interactive tuning of parameters to render a single frame. These potential advantages were explored and confirmed by constructing prototype software including NUTIK: a tool which implements the new frame rendering process.

This work extends earlier work on **T**ime **D**omain **C**ontinuous **I**maging (TDCI). The concept of TDCI is rooted in a new way to have image sensors sample scene appearance. TDCI post-processing centered on approximating what a conventional camera might have been able to do, but with various decisions deferred to after capture. More precisely, TDCI processing allowed the interval from which a frame is rendered to be selected arbitrarily after the time of capture independent of the timing properties of the capture. The current work has extended these concepts significantly. While earlier work presented and used various device-specific formats to record TDCI data, the current work defines the more abstract concept of an IMEV model not tied to any particular capture mechanism. The current work also explored how images can be rendered from an IMEV model in ways that are not just delaying decisions, but treating time and space as manipulable dimensions that can be used to produce images that are unrealizable using conventional photographic processes. Novel processing concepts defined and demonstrated in the current work include:

- The ability to have each rendered exposure represent not just any desired time interval within the capture, but also to vary the weight of contribution within a time interval. An example of this is shown in figures 3.31 and 3.34.

- A rendered frame need not represent a contiguous time interval, but can represent any user-selected weighted combination of the time-variant scene content over any set of arbitrary intervals. This aspect can be considered a generalization and extension of the concepts of multiple exposures and stacking functions. Multiple examples of this are presented in section 3.13.

- The ability to have negative weights for contributions. This provides functionality similar to that of an event camera, allowing removal of scene content that was unchanged over selected time intervals. It also allows much more general use of temporal differencing, which is impossible to accomplish by conventional means. Because such manipulations are an entirely new capability, the results are difficult to anticipate. However, with NUTIK the user can explore the wide

range of effects that can be achieved. Figures 3.28 through 3.30 demonstrate a use of negative weights.

- Through the use of masking, the virtual exposure parameters can be applied selectively and independently to arbitrary regions, or objects, within the scene. It can be difficult to know a priori where an object mask should be placed, but that problem can be solved by creating the object mask using normal "smart selection" software applied to a rendering produced as an intermediate step. For example, to create a sharp image of an object moving across the scene, the region it occupied during a desired interval could be exposed with a short virtual exposure. The rest of the scene can be exposed over a longer, potentially overlapping, interval to capture motion blur and/or dark details. An extended example of options afforded along these lines are illustrated by figures 3.40 through 3.51.

The most convenient representation of a time-varying scene model is not a sequence of frames, but an IMEV data structure that exclusively encodes records describing when and how the expected value for each pixel changes. As was discovered in previous work developing TDCI, recognizing when the expected value has significantly changed requires a statistical understanding of the noise in pixel value measurements. The current work leverages the same computations used in TDCI to determine and apply these noise statistics.

It is clear that commodity digital cameras are generally designed to behave superficially like simulations of film cameras. However, the assumptions associated with that simulation were found to permeate the internal operation of the cameras examined in this research. The frame orientation is surprisingly evident in every layer of processing from sensor readout through image storage. Despite this, it is possible within that framework to somewhat awkwardly implement IMEV capture. The work reported here strongly suggests that at least the Sony A6000 and Canon EOS M mirrorless cameras can be reprogrammed for IMEV capture without adding or modifying hardware. The main issues encountered in the commodity cameras studied were:

1. Although random access readout of sensels on a CMOS sensor is theoretically possible, the particular sensors and controllers used support only readout of a single, potentially strided, rectangular ROI (region of interest) between sensor resets. Thus, more data must be read from than sensor than is theoretically necessary for IMEV capture.

2. To detect which sensel values have changed from their predicted values, the entire sensor must be read out. The cameras studied did not have sufficient main memory working space to perform the conversion to an IMEV form at full resolution. This situation has steadily improved in subsequent camera generations; the available main memory has grown faster than the size of a full captured frame.

3. Even if there was sufficient memory space, the processors in the cameras studied are not fast enough to maintain a high sample rate at full resolution. The Sony

A6000 has a quad-core ARM7 and the Canon EOS M has a dual-core ARM5. Neither is fast enough to perform even conventional frame capture processing without the assistance of special-purpose hardware for JPEG compression, etc. It is possible that some of that hardware could be adapted or enhanced to speed up IMEV capture processing; our study was not able to reverse-engineer a way for software to make effective use of the existing function units.

It seems that the vast majority of implementation choices were made the simulation of film-like behavior was found to be far more than skin deep. After several years exploring how the internals of these cameras could be repurposed to more directly produce IMEV recordings, no clear path was found. Too many camera hardware and software interfaces are tuned to work on frames, and the excess computational capacity in the cameras was not sufficient to successfully layer IMEV capture on top of that infrastructure. The recent emphasis on the fusion of video and still photography, combined with the ever-reducing costs of putting additional computing resources in the camera, should make IMEV capture easier in future systems, as discussed in Section 4.4. Additionally, one can hope that the abilities associated with IMEV processing, as identified here, will inspire the introduction of hooks making it easier to experiment with IMEV capture in future camera models.

5.1 Future Work

As much as conventional cameras may be evolving toward designs that can effectively simulate IMEV capture, the ideal IMEV capture device would be a TDCI camera. These have proven difficult to develop, but the current work should make the development effort easier to justify.

In a completely different direction than this work took, observations made while refining the TDCI model suggest a possible mathematical formalization of the mechanism. An IMEV model should only emit a record when the predicted value of a sensel has changed. The simplest possible prediction would be that a sensel's value is constant, in which case the IMEV emits records corresponding to the first derivative of the function which is that sensel's value over time. Thus far, all the TIK file formats effectively encode first derivatives. Greater compression should be attainable by using a more sophisticated prediction. For example, if the sensel's rate of change is modeled as a simple slope, an IMEV record could be emitted only when sampling of the sensel indicates a change to the slope, effectively recording the second derivative. IMEV coding is fundamentally a one-dimensional signal compression problem, and a wide range of compression techniques that have been applied to audio channels might be applicable. Of course, any changes need to be measured taking a noise model into account. The empirically-derived probability density function maps used in this work — and in earlier TDCI work — appear to be effective and robust enough to be used for determining when statistical changes are significant even for more advanced compression schemes.

For the NUTIK prototype and its related tooling, there are a number of obvious next steps.

- The NUTIK code base is full of unutilized parallelism. All of the per-pixel and per-color-channel operations in the rendering loop in particular are pleasantly parallel. Utilizing multiple cores or even a GPU to significantly speed processing could probably be achieved with the addition of a relatively small number of OpenMP directives.

- NUTIK contains a number of places where it performs series of linear searches of relatively small data structures to compute gains. Fancier data structures would speed-up these searches if the integration functions being applied were exceptionally complex. However, linear search of data structures that fit in level 1 cache is not a significant source of delay.

- The user interface to NUTIK is a command line structure that gives great flexibility and zero intuition. In large part, the awkwardness of the command line is due to NUTIK being a single tool with multiple functions, including construction of an error (noise) model, conversion of video or still images into a TIK-format IMEV, and rendering of images from an IMEV. Separating these functions would simplify the command line for each, and a graphical user interface could make parameter setting more intuitive.

- The exposure function specification file format used by NUTIK is editable as a text file, but is ponderous and highly error-prone. A graphical user interface for drawing or otherwise interactively shaping exposure functions could dramatically reduce the difficulty for people to begin playing with IMEV processing concepts. An oversimplified interface mockup appears in Figure 4.1.

- Mask generation also would benefit from being integrated in the graphical user interface for exposure function specification. Image editors can easily create and manipulate masks, but here all the masks being used by an exposure function are encoded as a single image, not as separate 1-bit mask layers – merging layer masks adds some steps to the workflow. Beyond that, it is common that a mask will be created from a rendering of the same IMEV with a simple interval, and that does not happen within a separate image editor. It would be highly desirable for the graphical user interface for exposure function specification be able to directly render temporary images and use them to create and edit masks. Ideally, that tool could also incorporate some templates for common applications and intelligent default handling potentially using artificial intelligence to recognize background and foreground elements, etc.

- In decoupling exposure parameters from how image data is acquired, the basic concept of what constitutes a correct exposure becomes lost. Instead, IMEV rendering has a normalization problem. One could think in terms of resolving this by an auto-levels type of mechanism, or by high dynamic range tone mapping, but the IMEV data provides a great deal of flexibility that should be accessible to the user. NUTIK has been through a number of different normalization schemes during its development. Perhaps most intuitive to photographers would be normalization by the total area under the curve for each

function, but this rule behaves strangely when a negative gain is applied. Many developmental versions of NUTIK normalized the entire output frame to the longest live interval. This practice made it convenient to design the individual functions, but very difficult to work out sets of functions which produce roughly even overall brightness, as the user has to design functions with approximately the same area under the curve to maintain reciprocity. The current version normalizes to the live time of the function used to generate each site — that is the total summed value of scene data contributions at each point is divided by the total time the integration for that point was non-zero. This convention makes it straightforward to expose images with relatively uniform brightness from multiple different functions applied to different regions, and multiple non-zero segments applied to the same region, both of which are otherwise somewhat difficult to compute. However, this causes weights to work in a somewhat counter-intuitive way; the gains specified in the functions are the *fraction of the scene appearance taken from that portion of the interval* so exposures of different length with the same specified gain will produce images of the same brightness. A user interface that can quickly preview normalization choices should make the process of adjusting tonality more intuitive.

NUTIK and `FnPlotter.py` were created to be freely available as an open source tools. Although neither of these tools is tuned for end-user experience, they do serve as a working prototype implementation of the new core technologies for IMEV rendering. They are intended to encourage future research, experimentation, construction of user-friendly tools, and eventually widespread application of the principles presented here.

Appendices

Appendix A Non-Uniform Proof Of Concept: Matlab Prototype

Listing A.1: NonUniformPoC.m

```
# A simple tool for applying non-uniform gain to integrating a
    series of image data
# Makes a normal cubic spline based on user-supplied control points
    to model the gain function for each region
# Applies the average gain for the frame interval sampled from that
    integration function
# Selects gain functions based on a set of regions specfied as an
    input-sized pgm with one gray per region/function.

#### This part is setup ####
#Image Input Setup
fr=(1/240); #framerate of input sequence, in seconds (not sensed,
    video package broken on 5.x)
path="20190930_205109.mp4"; #Path to source file

#Read in an appropriate user-created mask
mask=imread('PenguinRockMask.pgm');
mask=cast(mask,'double');
#Mask values defined in the mask
# would be better to get from mask automatically,
# but I keep seeing border pixels w/ garbage
gray=[255,128];
fns=length(unique(gray));

#Integration Functions Setup
pts= [100, 200];
x=0:(max(pts)-1);
y=zeros(fns,max(pts));
y(1,1:pts(1))=y=[0 0 0 0 0 0 0 0 0 10 30 50 30 10 0 0 0 0 0 0 0 0 0
    0 0 0 0 0 0 10 30 50 30 10 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
     0 0 0 0 0 0 0 0 0 0 0 0 0 ];
y(2,1:pts(2))=[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
     0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
     0 0 0 0 400 0 0 0 0 0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0 0
     0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
     0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
     0 0 0 0 0 0 0 0 0 0 0 0 0 0];

#### End of configuration ####

#changem cribbed from,
# https://stackoverflow.com/questions/11952037/replace-values-in-
    matrix-with-other-values
# because I'm not using a commercial toolbox for this
function mapout = changem(Z, newcode, oldcode)
% Identical to the Mapping Toolbox's changem
```

```
% Note the weird order: newcode, oldcode. I left it unchanged from
    Matlab.
    if numel(newcode) ~= numel(oldcode)
        error('newcode␣and␣oldcode␣must␣be␣equal␣length');
    end

    mapout = Z;

    for ii = 1:numel(oldcode)
        mapout(Z == oldcode(ii)) = newcode(ii);
    end
end

#Show the integration functions (comment to hide)
for dfs=1:fns
  figure(dfs);
  xspline=0:0.01:pts(dfs);
  yspline=spline(x(1:pts(dfs)),y(dfs,1:pts(dfs)),xspline);
  ylabel("gain");
  xlabel("fractional␣time");
  plot(xspline,yspline,"g-",x(1:pts(dfs)),y(dfs,1:pts(dfs)),"b+");
endfor


#Have ffmpeg process video for frames
# In a better world, I'd use the video package, but it's currently
    broken on 5.x
# https://savannah.gnu.org/bugs/?51344
# That would give automatic frame-rate extraction, but nooo
##ffmpegline=["ffmpeg -i ", path ," -vcodec ppm img%03d.ppm"]
##system(ffmpegline);

t=0; #time
fc=0; #number of frames ingested, maintained seperately because I'm
    lazy
sumimg=0;

#for each input image
files = dir('img*.ppm');
numimg = length(files);
texp = fr*numimg;
for file = files'
  #read in image
  img=imread(file.name);
  #Promote for range
  img= im2double(img);
  #compute gains for that frame
  for gfs=1:fns
    #Still don't quite touch the last control point
    gain(gfs)=( ...
          interp1(x(1:pts(gfs)),y(gfs,1:pts(gfs)),(t/texp)*(pts(gfs)
              -1.01),"spline") + ...
          interp1(x(1:pts(gfs)),y(gfs,1:pts(gfs)),((t+fr)/texp)*(pts
              (gfs)-1.01),"spline"))/2;
```

```
    #printf("Frame %d gain %d\n", fc, gain);
  endfor
  #apply to frame


  gainmask=changem(mask,gain,gray);
  imgmod=img.*gainmask;
  #add frame to sum
  sumimg=imgmod+sumimg;
  fc=fc+1; #incriment frame count
  t=t+fr; #incriment time
endfor


#Divide by number of frames
finalimg=sumimg/fc;
#cleanup
##delete img*.ppm;
#save and display rendered image
imwrite(finalimg,"result.ppm");
figure(gfs+1);
imshow(finalimg);
```

Appendix B NUTIK

B.1 NUTIK Mask Gain Header

Listing B.1: MaskGain.h

```
/* Header for adding temporally and spatially non uniform
    integration support to tik*/

#include <limits>

#define MAXLINE 1000
#define MAXPTS 20

typedef struct{
  double time;
  double gain;
}point;


extern point fns[256][MAXPTS]; //Points per function
extern int fnc[256];
extern Mat mask;



//Declare the functions

int readMask(char * maskfile);
int readGainFns(char * gainFnFile);

void dumpGainFns();

double nextControlTime(uint32_t where, double now);
double gainherenow(int where, double now);
double weightherebetween(uint32_t where, double start, double end);
double FirstLive();
double LastLive();
double LiveTime();
double LiveTimeFor(uint32_t where);
```

B.2 NUTIK Mask Gain Implementation

Listing B.2: MaskGain.h

```
/*Awful hacks to add approximate nonuniform integration to tik*/

#include "tik.h"

#define CHATTY 0

// These will usually be sparse, but they are small anyway
point fns[256][MAXPTS]; //Points per function
int fnc[256]={0};

Mat mask;

double gainherenow(int maskval, double now)
{
  int before = -1;
  int after = -1;

  if(fnc[maskval] == 0){
    fprintf(stderr, "No function defined for value %d\n",maskval);
    return 0;
  }

  //Find nearest points
  //Range check, take the end value if out of range
  if(now <= fns[maskval][0].time){
    //printf("Time %lu before first specified time %lu\n",now,fns[
       maskval][0].time);
    before = 0;
    after = 0;
  }
  if(now >= fns[maskval][fnc[maskval]-1].time)
  {
    //printf("Time %lf after last specified time %lf\n",now,fns[
       maskval][fnc[maskval]-1].time);
    before = fnc[maskval]-1;
    after = fnc[maskval]-1;
  }

  int i = 0;
  while(((before == -1) || (after == -1)) && (i < fnc[maskval]-1))
  {
    if(fns[maskval][i].time <= now && fns[maskval][i+1].time >now){
       //Find closest point before
      before = i;
    }

    if(fns[maskval][i].time < now && fns[maskval][i+1].time >= now){
        //Find closest point after
      after = i+1;
    }
```

```
    i++;
  }
#if CHATTY
  printf("Matched␣time␣%lf␣in␣function␣%d␣with␣before␣=␣%d␣(%lf:%lf)
     ␣after␣=␣%d␣(%lf:%lf)␣\n",now, maskval, before, fns[maskval][
     before].time, fns[maskval][before].gain,after, fns[maskval][
     after].time, fns[maskval][after].gain);
#endif
  // Compute gain at point
  if(before == after){ // We're on a defined point
    return fns[maskval][before].gain;
  }
  else{
  //Proportional linear interpolate between
    return (fns[maskval][before].gain*(fns[maskval][after].time -
       now)/(fns[maskval][after].time - fns[maskval][before].time))
    + (fns[maskval][after].gain*(now - fns[maskval][before].time)/(
       fns[maskval][after].time - fns[maskval][before].time));
  }
}


//Find the gain for a contribution over an interval at a point
//Tik keeps times in doubles, so I guess this takes times as doubles
   ?
// Weight for an interval contribution is the length times the
   average gain for that interval
// Makes gains as continuous as the input data...
double weightherebetween(uint32_t where, double start, double end){
  double len = end-start; // How long is the interval
  unsigned long center = (start+end)/2; // result to unsigned long
     ns for lookup
  //printf("Center Time is %lu\n",center);
  //Need to turn a location into a maskval; tik is full of raw
     pointers.
  // I think I can take the where from tik, and split on row size?
  uint32_t r = where/mask.cols;
  uint32_t c = where%mask.cols;
  if((r<0 || r>mask.rows)||(c<0 || c>mask.cols)){
    fprintf(stderr,"Attempted␣invalid␣lookup␣at␣(%d,%d)\n",r,c);
  }
  double centergain = gainherenow(mask.at<uchar>(r, c),center);
#if CHATTY
  //if(where == 90000)
  printf("Center␣Gain␣is␣%lf,␣interval␣is␣%lf␣\n", centergain, len);
#endif
  return centergain * len;
}

//Read a grayscale PGM Image as the mask
int readMask(char * maskfile)
{
  printf("Reading␣mask␣from␣%s\n",maskfile);
  mask = imread(maskfile , IMREAD_GRAYSCALE );
```

```
  if ( !mask.data )
  {
    printf("No image data in mask\n");
    return -1;
  }
  return 0;
}

//A helper to print out all the currently loaded gain functions in
    the same format they are accepted
void dumpGainFns()
{
  printf("Read functions: \n");
  for(int i=0;i<256;i++)
  {
    if(fnc[i] != 0)
    {
      printf("M%d{",i);
      for(int j=0;j<fnc[i];j++)
      {
        printf("[%lf,%lf]",fns[i][j].time, fns[i][j].gain);
        if(j==fnc[i]-1){printf("}\n");}
        else { printf(","); }
      }
    }
  }
}

int readGainFns(char * gainFnFile)
{
  printf("Reading gain functions from %s\n",gainFnFile);
  FILE *ffp;
  ffp = fopen(gainFnFile, "r");
  if(ffp == NULL)
  {
    fprintf(stderr,"Could Not Open Gain Function File %s",
       gainFnFile);
    exit(1);
  }

  //Read a set of exposure functions
  char line [MAXLINE];
  char *tok;
  char *tup;
  int maskval;

  double gn;
  double tn;

  while(fgets(line, sizeof line , ffp) != NULL) //Strip Lines
  {
      tok = strtok(line, "{"); // Split header from list
      if(tok != NULL){
```

```
        //if(tok[0] != 'M'){continue;} // Skip lines not shaped like
            functions.
        sscanf(tok," M %d ",&maskval);
        if( maskval >= 0 && maskval < 256){ //Check if it's a valid
           mask value
          //printf("Found function for Mask Value %d", maskval);
          if(fnc[maskval]!=0){
            fprintf(stderr,"Warning: Mask value %d multiply defined\
               n", maskval);
          }
          int pt = 0;
          tok = strtok(NULL, "{"); //Get the list
          tup = strtok(tok,","); //Split Tuples
          while(tup != NULL){
            sscanf(tup," [ %lf : %lf ] ",&tn, &gn);
            fns[maskval][pt].time = tn;
            fns[maskval][pt].gain = gn;
            pt++;
            if(pt > MAXPTS) {
              fprintf(stderr,"Too many points defined for mask value
                 %d, max of %d\n", maskval, MAXPTS);
              return (-1);
            }
            tup = strtok(NULL,",");
          }
            fnc[maskval]=pt;//Store the number of points for this
               value
        }
        else{//Maskval out of range
          fprintf(stderr,"Mask value %d out of range (only 0-255
             supported)\n", maskval);
          return (-1);
        }
      }
  }
  printf("Read mask of size %d x %d\n",mask.cols, mask.rows);
  return 0;
}

//Returns the time (in ns) of the NEXT control point specified in
   the input for the current mask
//  To use in partitioning
// Special case 0 to mean "no relevant time partition?" for times
   after the last control point?
double nextControlTime(uint32_t where, double now){

  //printf("Entering NextControlPoint with time %lf\n",now);
  double after = -1;

  //Coordinate fix
  uint32_t r = where/mask.cols;
  uint32_t c = where%mask.cols;
  if((r<0 || r>mask.rows)||(c<0 || c>mask.cols)){
    fprintf(stderr,"Attempted invalid lookup at (%d,%d)\n",r,c);
```

```
  }
  int maskval = mask.at<uchar>(r, c);

  if(fnc[maskval] == 0){
    fprintf(stderr, "No function defined for value %d\n",maskval);
    return 0;
  }

  //now before first control point
  if(now < fns[maskval][0].time){
    return fns[maskval][0].time;
    //return 0;
  }
  //now after last control point
  if(now >= fns[maskval][fnc[maskval]-1].time)
  {
    return numeric_limits<double>::infinity(); //We read off the end
        of the control interval
  }
  // Have to search
  //printf("Last Control Point %d:[%lf,%lf]\n",fnc[maskval]-1,fns[
      maskval][fnc[maskval]-1].time,fns[maskval][fnc[maskval]-1].gain
      );
  int k = 0;
  while(k < (fnc[maskval]-1))
  {
    // printf(" Checking %d,%lf\n",k,fns[maskval][k].time);

    // if(now > fns[maskval][k].time) {printf(" %d Before\n",k);}
    // if(now == fns[maskval][k].time) {printf(" %d At\n",k);}
    // if(now < fns[maskval][k].time) {printf(" %d After\n",k);}

    if((fns[maskval][k].time<=now) && (fns[maskval][k+1].time>now))
    {   //Find closest point after
      //after= fns[maskval][k+1].time;
      //printf("nextControlTime for function %d time %lf: Found %lf
          at position %d\n", maskval, now, after,k+1);
      return fns[maskval][k+1].time;
    }
    k++;
  }
  fflush(stdout);
  fprintf(stderr, "Exited nextControlTime for function %d time %lf
      with no time found!\n", maskval, now );
  exit(1);
  return 0;
}

/*Locate the control point _before_ the first non-zero gain in the
    function spec*/
// One before because interpolation includes the prior point
double FirstLive()
{
  double first = numeric_limits<double>::infinity();
```

```
  for(int i=0;i<256;i++)
  {
    if(fnc[i] != 0)
    {
      for(int j=0;j<fnc[i];j++)
      {
        //The next point has non-zero gain and
        // the current point has an earlier time than previously
           seen
        // Update the earliest
        if((fns[i][j+1].gain != 0) && (fns[i][j].time<first))
        {
          first = fns[i][j].time;
        }
      }
    }
  }
#if CHATTY
  printf("First live control time is %lf\n",first);
#endif
  return first;
}

/*Locate the control point after the last non-zero gain in the
   function spec*/
// One past because interpolation includes the next point
double LastLive()
{
  double last = 0;
  for(int i=0;i<256;i++)
  {
    if(fnc[i] != 0)
    {
      for(int j=0;j<fnc[i];j++)
      {
        //The next point has non-zero gain and
        // the current point has a later time than previously seen
        // Update the earliest
        if((fns[i][j].gain != 0) && (fns[i][j+1].time>last))
        {
          last = fns[i][j+1].time;
        }
      }
    }
  }
#if CHATTY
  printf("Last live control time is %lf\n",last);
#endif
  return last;
}

//Return the total amount of time there are non-zero contributions
// If there are multiple functions, return the longest live time
//  A little sloppy because of slopes, will tend to "underexpose"
```

```
double LiveTime()
{
  double live = 0;
  double fnlive =0;
  double start = 0;
  double stop = 0;
  for(int i=0;i<256;i++) //For each function
  {
    if(fnc[i] != 0) //Which has control points
    {
      for(int j=0;j<fnc[i];j++) //Scan the control points
      {
        //The current control point has 0 gain and next point has
           non-zero gain
        if((fns[i][j].gain == 0) && (fns[i][j+1].gain != 0))
        {
          start = fns[i][j].time;
          //printf("Additional start time %lf\n",start);
        }
        //The current point has non-zero and the next has 0 gain
        if((fns[i][j].gain != 0) && (fns[i][j+1].gain == 0))
        {
          //printf("Additional stop time %lf, adds %lf\n",fns[i][j
             +1].time,fns[i][j+1].time - start);
          fnlive += fns[i][j+1].time - start;
        }
      }
    }
    //Update longest found
    if(fnlive > live)
    {
      live = fnlive;
    }
    fnlive=0;
  }
//#if CHATTY
  printf("Longest␣live␣control␣time␣is␣%lf\n",live);
//#endif
  return live;
}

//Return the total amount of time there are non-zero contributions
   for a specific function
double LiveTimeFor(uint32_t where)
{
  double fnlive =0;
  double start = 0;
  double stop = 0;
  double deadtime = 0;

  //Coordinate fix
  uint32_t r = where/mask.cols;
  uint32_t c = where%mask.cols;
  if((r<0 || r>mask.rows)||(c<0 || c>mask.cols)){
```

```
    fprintf(stderr,"Attempted␣invalid␣lookup␣at␣(%d,%d)\n",r,c);
  }
  int maskval = mask.at<uchar>(r, c);


  if(fnc[maskval] == 0){
    fprintf(stderr, "No␣function␣defined␣for␣value␣%d\n",maskval);
    return 0;
  }
    if(fnc[maskval] != 0) //Function has control points
    {
      for(int j=0;j<fnc[maskval];j++) //Scan the control points
      {
        //The current control point has 0 gain and next point has
           non-zero gain
        if((fns[maskval][j].gain == 0) && (fns[maskval][j+1].gain !=
            0))
        {
          start = fns[maskval][j].time;
        }
        //The current point has non-zero and the next has 0 gain
        if((fns[maskval][j].gain != 0) && (fns[maskval][j+1].gain ==
            0))
        {
          fnlive += fns[maskval][j+1].time - start;
        }
      }
    }
  //printf("LiveTime of function %d = %lf\n",maskval,fnlive);
  return fnlive;
}
```

B.3 NUTIK Render Implementation

Listing B.3: MaskGain.h

```
/*  render.cpp
*/

#include "tik.h"

typedef struct {
  double  start, stop;  /* Exposure interval */
  double  qual;   /* Samples used to make image */
  double  divby;    /* For scaling */
  double  *image;
  int more;   /* More pixels not yet done? */
} interval_t;

static  interval_t  render[MAXRENDER];
static  int   rendersp = 0;
static  char    *infile = 0;
static  int   dottik = 0;
static  double    minstart = 0;



#define FOR_RENDER  for (r=0; r<rendersp; ++r)
#define START   (render[r].start)
#define STOP    (render[r].stop)
#define QUAL    (render[r].qual)
#define DIVBY   (render[r].divby)
#define IMAGE   (render[r].image)
#define MORE    (render[r].more)
#define SUM(X)    *(IMAGE + (X))

//#define WATCHSPOT 90000
#define WATCHSPOT 273600
// Broken in old version 91058

static inline unsigned int
xinu(unsigned int x)
{
  unsigned int y;

  *(((char *) &y) + 0) = *(((char *) &x) + 3);
  *(((char *) &y) + 1) = *(((char *) &x) + 2);
  *(((char *) &y) + 2) = *(((char *) &x) + 1);
  *(((char *) &y) + 3) = *(((char *) &x) + 0);
  return(y);
}

static void
TIKrenderWriteFile(register int r)
{
  char outfile[4*1024];
```

```
  char *s;

  sprintf(outfile, e_outfile, r);
  INFO("writing␣\"%s\"␣as␣frame␣%d␣(%1.3fs␣@␣%1.3fs)␣from␣\"%s\"␣
     with␣quality␣%1.3f\n",
       outfile,
       r,
       (STOP-START) / 1000000000.0,
       START / 1000000000.0,
       infile,
       QUAL);

  printf("Watched␣pixel␣has␣value␣[R,G,B]=[%lf,%lf,%lf].\n",IMAGE[
     WATCHSPOT],IMAGE[WATCHSPOT+1],IMAGE[WATCHSPOT+2]);

  if ((s = tdci2jpeg(outfile, IMAGE)) != 0) {
    ERROR(ERROR_WRITE,
          "%s␣%s\n",
          outfile,
          s);
  }

  MORE = 0;
  free(IMAGE);
  IMAGE = ((double *) 0);
}

void
badformat()
{
  ERROR(ERROR_FORMAT, "bad␣format␣in␣TIK␣RGB␣file\n");
}

void
TIKrenderSimul(void)
{
  //printf("Beginning TIKrenderSimul\n");
  register uint8 *p;
  register int r;

  double fnfirstlive=FirstLive();
  double fnlastlive =LastLive();

  printf("Control␣Function␣live␣range␣from␣[%lf,%lf]\n",fnfirstlive,
     fnlastlive);
  //printf(" expose -b%lf -t%lf \n",fnfirstlive/1e9,(fnlastlive-
     fnfirstlive)/1e9);

  /* Anything requested? */
  if (rendersp < 1) {
    ERROR(ERROR_EXPOSE,
          "no␣exposures␣requested\n");
  }
```

```
/* Open the input file */
if (dottik) {
  if (0 > (PNMfd = open(infile, O_RDONLY))) {
    ERROR(ERROR_OPEN,
          "could␣not␣open␣TIK␣file␣%s\n",
          infile);
  }
} else {
  start_opencv(infile);
}

/* Initialize wave stuff */
memcpy(&wavework, &waveref, sizeof(waveref));
if (waveref.x > 0) {
  /* Non-zero X means this is next frame */
  wavework.b += waveref.f;
}
wavep = &wavework;
//printf("Before setting up wave stats\n");
/* Check we have an image */
if (p = PNMreader()) {
  double gamma[256];
  register double invgamma;
  register uint32 xy;
  register uint32 xyc;
  register double *from;
  register uint8 *ref;
  register double wf;
  register double wt;
  register double now;
  register uint32 where;
  register int i, off, n, r;
  //printf("Found an input\n");
  memcpy(&waveref, &wavework, sizeof(waveref));
  xy = wavex * wavey;
  xyc = xy * ((wavetype == wavePNM) ? 3 : wavec);
  from = ((double *) calloc(xy, sizeof(double)));

  /* Mark all frames as no pixels rendered */
  FOR_RENDER MORE = xy;

  ref = p; /* Set initial TDCI values */
  wf = ((wavetype == wavePNM) ? (1000000000.0 / 24) : wavef);
  wt = (((wavet < 1) || (wavet > wavef)) ? wavef : wavet);
  now = (waveb / 1000000000.0) + wf;
  n = 0;
  //printf("After setting up wave stats\n");
  /* Create gamma mapping info */
  if (e_gamma == 0) {
    e_gamma = ((waveg == 0) ? 1.0 : (waveg / 1000000.0));
  }
  for (i=0; i<256; ++i) {
    gamma[i] = ((e_gamma == 1) ?
          ((double) i) :
```

```
            pow(((double) i), e_gamma));
    }
    invgamma = 1.0 / e_gamma;
    //printf("Before selecting wavetype\n");
    /* Stuff specific to different formats */
    switch (wavetype) {
    case waveRGB:
      //printf("Entered WaveRGB at time %lf\n",now);
      /* Should be 0 separator; skip over it */
      if (PNMnextc()) badformat();
      PNMnextc();

      /* Process TDCI until a little past stop...
         past stop because we need frame after so we can
         interpolate the slope in the gap between samples
         Initialize where to inc past end of first frame
      */
      where = xy - 1;
      while (!PNMtimedout) {
        register uint32 shift = 0;
        register int c;
        uint8 v[3 + 1]; /* +1 to allow PNMnextbn hack below */
        register int natstart = n;
        register int left = 0;

        /* Skip to next changed pixel */
        ++where;
        do {
          /* 7 bits of skip at a time, low first */
          where += (((c = PNMc) & 0x7f) << shift);
          shift += 7;
          while (where >= xy) {
            /* In another frame... */
            where -= xy;
            now += wf;
            ++n;
            INFO("Frame␣%d:␣%1.3fs␣␣␣␣␣␣␣␣\r",
                 n,
                 (now / 1000000000.0));
          }
          PNMnextc();
          if (PNMtimedout) badformat();
        } while (c & 0x80);

        /* Where are we? */
        off = where * wavec;

        /* Get value of this sample */
#ifdef  NOTNOW
        for (i=0; i<wavec; ++i) {
          v[i] = PNMc;
          PNMnextc();
          if (PNMtimedout) badformat();
        }
```

```
#else
        v[0] = PNMc;
        PNMnextbn(&(v[1]), wavec);
        if (PNMtimedout) badformat();
        PNMc = v[wavec];
#endif

        /* Add contribution from previous sample to sum */
        if (now > minstart) {
          FOR_RENDER {
            //printf("Rendering frame START:%lf, STOP:%lf\n",START,
                STOP);
            if ((MORE > 0) && (now > START)) {
              double f = from[where];
              double g = now - (wf - wt); /* start of gap between
                  samples */
              double fstart = ((START > f) ? START : f);
              double fstop = ((g > STOP) ? STOP : g);
              double gstart = ((START > g) ? START : g);
              double gstop = ((now > STOP) ? STOP : now);
              uint32 off = where * wavec;

              /* Newly rendering this frame? */
              if (IMAGE == ((double *) 0)) {
                if (!(IMAGE = ((double *) calloc(rendersp, xyc *
                    sizeof(double))))) {
                  ERROR(ERROR_EXPOSE,
                        "cannot␣allocate␣memory␣for␣TIK␣virtual␣
                            exposure");
                }
                printf("Starting␣frame␣in␣WaveRGB\n");
              }

              /* Add portion of previous sample */
              if (fstart < fstop) {
                double sfstop = fstop; //Assume it's the end for now
                double sfstart = fstart; //Assume it's the beginning
                    for now
                double nextcp=nextControlTime(where, sfstart);
                //Split on control points
                while(nextcp< fstop){
                  sfstop = nextcp;
                  if(where == WATCHSPOT){
                    printf("Cracking␣leading␣sample␣[%lf,%lf]␣on␣
                        control␣point␣%lf\n",sfstart,sfstop,nextcp);
                  }
                  double w = sfstop - fstart;
                  double weight = weightherebetween(where, sfstart,
                      sfstop);
                  if(where == WATCHSPOT){
                    printf("Including␣leading␣sample␣from␣[%lf,%lf],
                        ␣weight␣%lf\n",sfstart,sfstop,weight);
                  }
```

```
    for (i=0; i<wavec; ++i) SUM(off+i) += (gamma[ref[
        off+i]] * weight* DIVBY);
    /* Add-in fraction from real data frames */
    QUAL += (1.0 + (n - natstart)) * (w / (now - f));
    //Advance the interval
    sfstart=sfstop+1;
    nextcp=nextControlTime(where, sfstart);
  }
  // Do the remainder after the last control point
    double w = fstop - sfstart;
    double weight = weightherebetween(where, sfstart,
        fstop);

  if(where == WATCHSPOT){
  printf("Including␣end␣of␣leading␣sample␣interval␣
     from␣[%lf,%lf],␣weight␣%lf\n", sfstart,fstop,
     weight);
  }
    for (i=0; i<wavec; ++i) SUM(off+i) += (gamma[ref[
        off+i]] * weight * DIVBY);
    /* Add-in fraction from real data frames */
    QUAL += (1.0 + (n - natstart)) * (w / (now - f));
}/* End Previous Sample Portion*/
//printf("Finished Previous Sample\n");

/* Add portion of gap as a slope */
/* This code is only called if the input shutter angle
    is expressly specified*/
if (gstart < gstop) {
  double wstart = (gstart - g) / (now - g); //(start-
     beginning of gap)/(now-beginning of gap)
  double wstop = (gstop - g) / (now - g);
  double wgap = gstop - gstart;

  double nextcp=nextControlTime(where, gstart);
  double sgstart = gstart;
  double sgstop = gstop;
  //Need to chop the gap not the end model
  // double swstart = wstart;
  // double swstop = wstop;
  double swgap = gstop - gstart;

  double weight;

  if(where == WATCHSPOT){printf("Handling␣gap␣from␣[%
     lf,%lf]\n",gstart,gstop);}
  while(nextcp < gstop)//Input function changes during
      gap
  {
    //Cut a partition
    if(where == WATCHSPOT){printf("Split␣gap␣on␣%lf\n"
        ,nextcp);}
    sgstop=nextcp;
```

```
    weight = weightherebetween(where, sgstart, sgstop)
        ;
    if(where == WATCHSPOT){printf("Processed␣gap␣from␣
        [%lf,%lf]␣with␣weight␣%lf\n",sgstart,sgstop,
        weight);}
    for (i=0; i<wavec; ++i) {
      /* Compute gap end values, then average * wgap
         to get area */
      double vstart = (ref[off+i] * (1.0 - wstart)) +
         (gamma[v[i]] * wstart);
      double vstop = (ref[off+i] * (1.0 - wstop)) + (
         gamma[v[i]] * wstop);
      SUM(off+i) += (weight * ((vstart + vstop) / 2.0)
         * DIVBY);
      }
  /* Add-in gap fraction as if it is from one frame */
    QUAL += (wgap / (now - f));
    sgstart=sgstop+1;
    nextcp=nextControlTime(where, sgstart);
  }
  //Otherwise just do the remainder in one shot
  weight = weightherebetween(where, sgstart, sgstop);
  if(where == WATCHSPOT){printf("Processed␣trailing␣
     gap␣from␣[%lf,%lf]␣with␣weight␣%lf\n",sgstart,
     sgstop,weight);}
  for (i=0; i<wavec; ++i) {
    /* Compute gap end values, then average * wgap to
       get area */
    double vstart = (ref[off+i] * (1.0 - wstart)) + (
       gamma[v[i]] * wstart);
    double vstop = (ref[off+i] * (1.0 - wstop)) + (
       gamma[v[i]] * wstop);
    SUM(off+i) += (weight * ((vstart + vstop) / 2.0)*
       DIVBY);
  }
  /* Add-in gap fraction as if it is from one frame */
  QUAL += (wgap / (now - f));
}/* End Gap Portion */

if (now > STOP+wf) {
  printf("Entering␣trailing␣code\n");
  /* Add-in data for other pixels */
  int pos;
  int off = 0;
  for (pos=0; pos<xy; ++pos) {
    double f = from[pos];
    double localstart = ((f < START) ? START : f);
    double w = STOP - localstart;
    double weight = weightherebetween(pos,localstart,
       STOP);
    double nextcp = nextControlTime(pos,localstart);

    if(pos == WATCHSPOT){
```

```
  printf("Handling␣trailing␣sample␣from␣[%lf,%lf],
     ␣next␣control␣point␣=%lf\n",localstart,STOP,
     nextcp);
}
double splitstart = localstart;
double splitstop = ((nextcp < STOP)?nextcp:STOP);
   // whichever comes first
double subw = splitstop - splitstart;

while(splitstop < STOP){
  // Get weight for sub-interval
  weight = weightherebetween(pos,splitstart,
     splitstop);

  if(pos == WATCHSPOT){
  printf("Handling␣trailing␣sub-sample␣from␣[%lf,%
     lf],␣weight=%lf\n",splitstart,splitstop,
     weight);
  }
  //Make Contribution
  if (subw < 0) {
    subw = 0;
    weight = 0;
    if(pos == WATCHSPOT){
    printf("Squashed␣negative␣weight\n");
    }
  }
  off=pos*wavec;
  for (i=0; i<wavec; ++i) {
    SUM(off) = SUM(off) + ((gamma[ref[off]] *
       weight)* DIVBY);
    ++off;
    ++QUAL;
  }

  //Update Control region
  splitstart = splitstop;
  nextcp = nextControlTime(pos,splitstart);
  splitstop = ((nextcp < STOP)?nextcp:STOP);
}
//Remainder of last split to stop
weight=weightherebetween(pos,splitstart,splitstop)
   ;
subw = splitstop - splitstart;
if(pos == WATCHSPOT){
printf("Handling␣trailing␣final␣sub-sample␣from␣[%
   lf,%lf],␣weight=%lf\n",splitstart,splitstop,
   weight);
}
//Make Contribution
if (subw < 0) {
  subw = 0;
  weight = 0;
  if(pos == WATCHSPOT){
```

```
              printf("Squashed negative weight\n");
              }
            }
            off=pos*wavec;
            for (i=0; i<wavec; ++i) {
              SUM(off) = SUM(off) + ((gamma[ref[off]] * weight
                 )* DIVBY);
              if (e_gamma != 1) {
                SUM(off) = pow(SUM(off), invgamma);
              }
              ++off;
              ++QUAL;
            }
          }


          printf("Re-normalizing Areas\n");
          //Re-normalize for individual functions
          off=0;
          for(where = 0; where<xy; ++where){
            for(i=0;i<wavec;++i){
              IMAGE[off]*=DIVBY/LiveTimeFor(where);
              ++off;
            }
          }
          //printf("Correcting exposure quality\n");
          /* Correct exposure quality */
          QUAL /= xy;
          /* Done with this virtual exposure; write the file
             */
          TIKrenderWriteFile(r);
        }/*End Trailing Sample Portion*/
      }

      /* Still any images left incomplete? */
      left += (MORE > 0);
    }

    /* Check to see if any frames not yet rendered */
    if (left < 1) goto noneleft;
  }

  /* Record this sample */
  from[where] = now;
  for (i=0; i<wavec; ++i) ref[off+i] = v[i];
}

/* Update image with last samples...
   which we'll assume persist until stop,
   because we can't know otherwise  ;-)
   Also divide by time to get average values.
*/
//PSE: What is this? I thought we were already to the end?
FOR_RENDER
```

```
  if (MORE > 0) {
    char outfile[4*1024];
    char *s;

    off = 0;
    for (where=0; where<xy; ++where) {
      double f = from[where];
      double w = STOP - ((f < START) ? START : f);

      //Does this potentially need to be cracked on a control
         point?
      double weight = weightherebetween(where, f, w);
      if(where == WATCHSPOT){
        printf("Processing␣residual␣from␣[%lf,%lf],␣weight␣%lf␣\
           n",f,w,weight);
      }
      if (w < 0) {
        w = 0;
        weight = 0;
        if(where == WATCHSPOT){
        printf("Squashed␣negative␣interval␣segment␣in␣residual\n
           ");
        }
      }
      for (i=0; i<wavec; ++i) {
        SUM(off) = (SUM(off) + (gamma[ref[off]] * weight)) *
           DIVBY;
        if (e_gamma != 1) {
          SUM(off) = pow(SUM(off), invgamma);
        }
        ++off;

        /* Add-in fraction from real data */
        ++QUAL;
      }
    }

    printf("Re-normalizing␣Areas\n");
    //Re-normalize for individual functions
    for(where = 0; where<xy; ++where){
      for(i=0;i<wavec;i++){
        IMAGE[where+i]*=DIVBY/LiveTimeFor(where);
      }
    }

    /* Correct exposure quality */
    QUAL /= xy;

    /* Done with this virtual exposure; write the file */
    TIKrenderWriteFile(r);
  }
  break;

case waveUYVYYYold:
```

```
/* Should be a P5 header next...
   we know how to read that!
*/
for (where=0; where<xyc; where+=6) {
  ref[where] += 128;
  ref[where+2] += 128;
}
p = P5reader();
while (p) {
  register double f = now - wf;
  register double g = now - (wf - wt); /* start of gap between
      samples */
  register int left = 0;

  FOR_RENDER {
    if ((MORE > 0) && (now > START)) {
      register double fstart = ((START > f) ? START : f);
      register double fstop = ((g > STOP) ? STOP : g);
      register double gstart = ((START > g) ? START : g);
      register double gstop = ((now > STOP) ? STOP : now);
      register double wstart = (gstart - g) / (now - g);
      register double wstop = (gstop - g) / (now - g);
      register double wgap = gstop - gstart;

      /* Newly rendering this frame? */
      if (IMAGE == ((double *) 0)) {
        if (!(IMAGE = ((double *) calloc(rendersp, xyc *
            sizeof(double))))) {
          ERROR(ERROR_EXPOSE,
                "cannot␣allocate␣memory␣for␣UYVYYY␣virtual␣
                   exposure");
        }
      }

      /* Update quality */
      ++QUAL;

      for (where=0; where<xyc; where+=6) {
        p[where] += 128;
        p[where+2] += 128;
      }
      for (where=0; where<xyc; ++where) {
        /* Add contribution from previous sample to sum */
        if (now > START) {
          /* Add portion of previous sample */
          if (fstart < fstop) {
            register double w = fstop - fstart;
            SUM(where) += (gamma[ref[where]] * w);
          }

          /* Add portion of gap as a slope */
          if (gstart < gstop) {
            /* Compute gap end values, then average * wgap to
               get area */
```

```
            register double vstart = (ref[where] * (1.0 -
               wstart)) + (gamma[p[where]] * wstart);
            register double vstop = (ref[where] * (1.0 - wstop
               )) + (gamma[p[where]] * wstop);
            SUM(where) += (wgap * ((vstart + vstop) / 2.0));
          }
        }
      }

      /* Update more and write file */
      if (now >= STOP) {
        /* Done with this virtual exposure; write the file */
        TIKrenderWriteFile(r);
      }
    }

    /* Still any images left incomplete? */
    left += (MORE > 0);
  }

  /* Check to see if any frames not yet rendered */
  if (left < 1) goto noneleft;

  /* Next image */
  now += wf;
  free(ref);
  ref = p;
  p = P5reader();
}

/* Did we get another? If so, discard it. */
if (p) free(p);

/* Update image with last samples...
   which we'll assume persist until stop,
   because we can't know otherwise  ;-)
   Also divide by time to get average values.
*/
FOR_RENDER
if (MORE > 0) {
  /* Newly rendering this frame? */
  if (IMAGE == ((double *) 0)) {
    if (!(IMAGE = ((double *) calloc(rendersp, xyc * sizeof(
       double))))) {
      ERROR(ERROR_EXPOSE,
            "cannot␣allocate␣memory␣for␣UYVYYY␣virtual␣
               exposure");
    }
  }

  /* Update quality */
  ++QUAL;

  for (where=0; where<xyc; ++where) {
```

```
      register double f = now - wf;
      register double w = STOP - ((f < START) ? START : f);

      if (w < 0) w = 0;
      SUM(where) = (SUM(where) + (gamma[ref[where]] * w)) *
         DIVBY;
      if (e_gamma != 1) {
        SUM(where) = pow(SUM(where), invgamma);
      }
    }

    /* Done with this virtual exposure; write the file */
    TIKrenderWriteFile(r);
  }
  break;

case waveUYVYYY:
  /* Make all values signed */
  for (where=0; where<xyc; where+=6) {
    ref[where] += 128;
    ref[where+2] += 128;
  }

  /* Resize from[] */
  free(from);
  from = ((double *) calloc(xyc, sizeof(double)));

  /* Process TDCI until a little past stop...
     past stop because we need frame after so we can
     interpolate the slope in the gap between samples
     Initialize where to inc past end of first frame
  */
  PNMnextc();
  where = 0;
  while ((!PNMtimedout) && (PNMnextw() != 1)) {
    register uint32 change = PNMw;
    register int left = 0;

    if (change & 1) {
      /* This is a span record */
      int skip = 0;
      where += ((change + change) & ~3);

      /* Skip complete frames */
      while (where >= xyc) {
        now += wf;
        ++skip;
        where -= xyc;
      }

      if (skip > 0) printf("now␣=␣%gs\n", now/1000000000.0);
    } else {
      /* Correct sample signedness */
      for (i=0; i<4; ++i) {
```

```
  switch ((where+i) % 6) {
  case 0:
  case 2:
    *(((uint8 *) &PNMw) + i) ^= 128;
  }
}

/* Add contribution from previous sample to sum */
if (now > minstart) {
  double refd[4], newd[4];

  /* Map the old and new values */
  for (i=0; i<4; ++i) {
    refd[i] = ref[where+i];
    newd[i] = *(((uint8 *) &PNMw) + i);
  }

  FOR_RENDER {
    if ((MORE > 0) && (now > START)) {
      register double f = from[where];
      register double g = now - (wf - wt); /* start of gap
          between samples */
      register double fstart = ((START > f) ? START : f);
      register double fstop = ((g > STOP) ? STOP : g);
      register double gstart = ((START > g) ? START : g);
      register double gstop = ((now > STOP) ? STOP : now);

      /* Newly rendering this frame? */
      if (IMAGE == ((double *) 0)) {
        if (!(IMAGE = ((double *) calloc(rendersp, xyc *
            sizeof(double))))) {
          ERROR(ERROR_EXPOSE,
                "cannot␣allocate␣memory␣for␣TIK␣virtual␣
                   exposure");
        }
      }

      /* Add portion of previous sample */
      if (fstart < fstop) {
        register double w = fstop - fstart;

        for (i=0; i<4; ++i) {
          SUM(where+i) += refd[i] * w;
        }

        /* Add-in fraction from real data frames */
        QUAL += (w / (now - f));
      }

      /* Add portion of gap as a slope */
      if (gstart < gstop) {
        register double wstart = (gstart - g) / (now - g);
        register double wstop = (gstop - g) / (now - g);
        register double wgap = gstop - gstart;
```

```
        for (i=0; i<4; ++i) {
          /* Compute gap end values, then average * wgap
             to get area */
          register double vstart = (refd[i] * (1.0 -
             wstart)) + (newd[i] * wstart);
          register double vstop = (refd[i] * (1.0 - wstop)
             ) + (newd[i] * wstop);
          SUM(off+i) += (wgap * ((vstart + vstop) / 2.0));
        }

        /* Add-in gap fraction as if it is from one frame
           */
        QUAL += (wgap / (now - f));
      }

      /* Update more */
      if (now > STOP+wf) {
        /* Add-in data for other pixels */
        register int pos;

        for (pos=0; pos<xyc; ++pos) {
          register double f = from[pos];
          register double w = STOP - ((f < START) ? START
             : f);
          if (w < 0) w = 0;
          SUM(pos) = (SUM(pos) + (ref[pos] * w)) * DIVBY;
          ++QUAL;
        }

        /* Correct exposure quality */
        QUAL /= (xyc / 4);

        /* Done with this virtual exposure; write the file
           */
        TIKrenderWriteFile(r);
      }
    }

    /* Still any images left incomplete? */
    left += (MORE > 0);
  }

  /* Check to see if any frames not yet rendered */
  if (left < 1) goto noneleft;
}

/* Update from pixel data */
for (i=0; i<4; ++i) {
  ref[where+i] = *(((uint8 *) &PNMw) + i);
  from[where+i] = now;
}
where += 4;
while (where >= xyc) {
```

```
        now += wf;
        where -= xyc;
      }
    }
  }

  /* Update image with last samples...
     which we'll assume persist until stop,
     because we can't know otherwise  ;-)
     Also divide by time to get average values.
  */
  FOR_RENDER
  if (MORE > 0) {
    /* Newly rendering this frame? */
    if (IMAGE == ((double *) 0)) {
      if (!(IMAGE = ((double *) calloc(rendersp, xyc * sizeof(
          double))))) {
        ERROR(ERROR_EXPOSE,
              "cannot␣allocate␣memory␣for␣UYVYYY␣virtual␣
                 exposure");
      }
    }

    /* Update quality */
    ++QUAL;

    for (where=0; where<xyc; ++where) {
      register double f = now - wf;
      register double w = STOP - ((f < START) ? START : f);

      if (w < 0) w = 0;
      SUM(where) = (SUM(where) + (ref[where] * w)) * DIVBY;
    }

    /* Done with this virtual exposure; write the file */
    TIKrenderWriteFile(r);
  }
  break;

case wavePNM:
  /* Should be a P6 header next...
     we know how to read that!
  */
  printf("PSE:␣Entered␣WavePNM␣Case\n");
  p = P6reader();
  while (p) {
    register double f = now - wf;
    register double g = now - (wf - wt); /* start of gap between
        samples */
    register int left = 0;

    FOR_RENDER { //For every output frame
      if ((MORE > 0) && (now > START)) { //that still needs work
```

```
register double fstart = ((START > f) ? START : f); //
    Start as far into the current sample this output
    frame starts at
register double fstop = ((g > STOP) ? STOP : g); // If
    the end of this input frame is after the end of the
    output frame, stop short
register double gstart = ((START > g) ? START : g); //
    This is the beginning of the part interpolated from
    the gap between samples
register double gstop = ((now > STOP) ? STOP : now);//
    End of the part of the output interval in the gap
register double wstart = (gstart - g) / (now - g); //
    This is the next gap interval, that gets constrained
    by the vstart/vstop values on the fly below?
register double wstop = (gstop - g) / (now - g);
register double wgap = gstop - gstart;

/* Newly rendering this frame? */
if (IMAGE == ((double *) 0)) {
  if (!(IMAGE = ((double *) calloc(rendersp, xyc *
      sizeof(double))))) {
    ERROR(ERROR_EXPOSE,
          "cannot␣allocate␣memory␣for␣PNM␣virtual␣
             exposure");
  }
}

/* Update quality by denoting another sample has been
    included*/
++QUAL;

for (where=0; where<xyc; ++where) {
  /* Add portion of previous sample */
  if (fstart < fstop) {
    register double w = fstop - fstart; //How long is
        this input sample contributing to the output
        frame
    SUM(where) += (gamma[ref[where]] * w); //gamma for
        correction, w is weighting by the amount of time
        it is contributing
    printf("PSE:␣Contribution␣Added␣in␣WavePNM\n");
  }

  /* Add portion of gap as a slope */
  //Wstart wstop are the weighting for the gap length
  if (gstart < gstop) {
    /* Compute gap end values, then average * wgap to
        get area */
    register double vstart = (ref[where] * (1.0 - wstart
        )) + (gamma[p[where]] * wstart);
    register double vstop = (ref[where] * (1.0 - wstop))
         + (gamma[p[where]] * wstop);
    SUM(where) += (wgap * ((vstart + vstop) / 2.0));
  }
```

```
      }

      /* Update more and write file */
      if (now > STOP+wf) {
        for (where=0; where<xyc; ++where) {
          SUM(where) = (SUM(where) * DIVBY);
          if (e_gamma != 1) {
            SUM(where) = pow(SUM(where), invgamma);
          }
        }

        /* Done with this virtual exposure; write the file */
        TIKrenderWriteFile(r);
      }
    }

    /* Still any images left incomplete? */
    left += (MORE > 0);
  }

  /* Check to see if any frames not yet rendered */
  if (left < 1) goto noneleft;

  /* Next image */
  now += wf;
  free(ref);
  ref = p;
  p = P6reader();
}

/* Did we get another? If so, discard it. */
if (p) free(p);

/* Update image with last samples...
   which we'll assume persist until stop,
   because we can't know otherwise  ;-)
   Also divide by time to get average values.
*/
FOR_RENDER
if (MORE > 0) {
  /* Newly rendering this frame? */
  if (IMAGE == ((double *) 0)) {
    if (!(IMAGE = ((double *) calloc(rendersp, xyc * sizeof(
        double))))) {
      ERROR(ERROR_EXPOSE,
            "cannot␣allocate␣memory␣for␣PNM␣virtual␣exposure")
               ;
    }
  }

  /* Update quality */
  ++QUAL;

  for (where=0; where<xyc; ++where) {
```

```
            register double f = now - wf;
            register double w = STOP - ((f < START) ? START : f);

            if (w < 0) w = 0;
            SUM(where) = (SUM(where) + (gamma[ref[where]] * w)) *
                DIVBY; // Add in some guesses for data after the last
                sample
            if (e_gamma != 1) {
              SUM(where) = pow(SUM(where), invgamma);
            }
          }

          /* Done with this virtual exposure; write the file */
          TIKrenderWriteFile(r);
        }

        break;

      default:
        badformat();
      }

      /* Clean-up */
noneleft:
      free(ref);
      free(from);

      /* Close the input file */
      if (dottik) {
        close(PNMfd);
      }
    }
}

void
TIKrenderFile(int d, char *filename)
{
  dottik = d;
  infile = filename;
  rendersp = 0;
}

void
TIKrender(double start, double stop)
{
  /* Normalize start, stop and convert to nanoseconds */
  start *= 1000000000.0;
  stop *= 1000000000.0;
  if (stop < start) { double t = start; start = stop; stop = t; }

  /* When does the first frame start? */
  if ((rendersp == 0) || (start < minstart)){
    minstart = start;
    //printf("Minstart updated to %lf\n",minstart);
```

```
  }
  /* Queue it up */
  render[rendersp].start = start;
  render[rendersp].stop = stop;
  render[rendersp].qual = 0;
  // Least-nonsense to normalize by how long functions are non-zero?
  // Still produces "surprising" behaviors.
  //render[rendersp].divby = 1.0 / (stop - start);
  //render[rendersp].divby = 1.0 / LiveTime();
  //Set divby to 1 and do local normalization at the end.
  render[rendersp].divby = 1.0 ;
  render[rendersp].image = ((double *) 0);
  ++rendersp;
}
```

B.4 NUTIK Frontend Implementation

Listing B.4: MaskGain.h

```
/*  tik.cpp

  Temporal Image Kentucky/Kontainer

  Original by Henry Dietz, June 2016

  See tik.h for notices.
*/

#include "tik.h"


char  *myname;  /* For error messages, name of this program */

FILE  *TDCIfile = 0;

double  e_begin = 0;
char  *e_noisefile = 0;
double  e_fps = 0;    /* Frames Per Second */
double  e_gamma = 0;    /* Encoding gamma (0 means not set) */
int e_n = 0;
char  *e_outfile = 0;
char  *e_outtik = ((char *) "pnm.tik");
char  *e_outimage = ((char *) "tik%05d.jpg");
double  e_percent = 4.55;
double  e_quality = 75;
double  e_t = 0;    /* Shutter time, Tv in seconds */
int e_what = WHAT_UNSPEC;
int e_interactive = 0;  /* Interactive mode? */
time_t  e_update;
int e_maketype = MAKE_UNSPEC;

//PSE: Probably need to made a default mode that always returns 1?
char *e_maskfile = 0;
char *e_fnfile = 0;


double
atofrac(char *s)
{
  /* Atof, but allowing fractions with 1/ */
  if ((s[0] == '1') && (s[1] == '/')) {
    return(1.0 / atof(&(s[2])));
  }
  return(atof(s));
}

int
main(register int argc,
register char **argv)
```

```
{
  register char *s;
  register int i, n;
  register char *fname = 0;
  register double now;
  time_t real_time = (e_update = time(0));

  /* Set interactive mode based on stderr */
  e_interactive = isatty(2);

  /* Process command line... */
  myname = argv[0];
  if (argc < 2) {
usage:
    fprintf(stderr,
      "Usage: %s {exposure_settings}\n"
      "-a#       Set Tv by shutter angle; Tv=(angle/360)/FPS\n"
      "-b#       Begin exposure time in seconds (default 0); needs .
         tik input\n"
      "-eNAME    Error model is NAME; blank creates from video of a
         constant scene\n"
      "-f#       Frames per second, FPS (default 24)\n"
      "-g#       Gamma of encoded data (1.0 linear default; 2.2
         typical JPEG)\n"
      "-i#       Interactive progess update seconds-1 (default %d, 0
         means never, -i toggles)\n"
      "-kFNFILE  Specify a function spec file for exposure functions\
         n"
      "-mMASKFILE Specify a (PGM) mask file to define areas for
         exposure\n"
      "-n#       Number of frames to encode/decode (default 1 output
         , all input)\n"
      "-oNAME    Output filename is NAME (default %s)\n"
      "-p#       Minimum %% probability same to factor (default %0.2
         f; std. dev. are 32, 5, 0.3)\n"
      "-q#       Quality %% (default %0.0f; for JPEG output or TIK
         encoding)\n"
      "-t#       Shutter speed in seconds, Tv (default to 1/FPS or
         1/60s)\n"
      "-v        Input is a video to be converted to .tik TDCI
         output\n"
      "FILENAME  Process file as specified by earlier options, .tik
         assumed to be TDCI\n",
      myname,
      e_interactive,
      e_outtik,
      e_percent,
      e_quality);
    exit(ERROR_USAGE);
  }

  for (i=1; i<argc; ++i) {
    if (*argv[i] == '-') {
      switch (*(argv[i]+1)) {
```

```
case 'a':
  e_t = atofrac(argv[i]+2);
  if (e_fps == 0) {
    ERROR(ERROR_ARG,
          "cannot␣set␣shutter␣angle␣before␣-f␣to␣set␣FPS\n");
  }
  if (e_t <= 0) {
    ERROR(ERROR_ARG,
          "shutter␣angle␣must␣be␣greater␣than␣0,␣not␣%s\n",
          (argv[i]+2));
  }
  e_t = (e_t / (e_fps * 360.0));
  break;
case 'b':
  e_begin = atofrac(argv[i]+2);
  e_what |= WHAT_TIK;
  break;
case 'e':
  e_noisefile = (argv[i]+2);
  if (*e_noisefile) {
    /* Read error model file */
    uint8 *p;

    if ((0 > (PNMfd = open(e_noisefile, O_RDONLY))) ||
        (0 == (p = PNMreader())) ||
        (PNMx != 256) ||
        (PNMy != 256) ||
        (PNMmaxval > 255)) {
      ERROR(ERROR_READ,
            "could␣not␣read␣error␣model␣file␣%s\n",
            e_noisefile);
    }
    if (wavetype != waveNOISE) {
      WARN("file␣%s␣used,␣but␣not␣marked␣as␣TIK␣NOISE\n",
           e_noisefile);
    }

    memcpy(&(PNMerr[0][0][0]), p, (256*256*3));
    close(PNMfd);
    free(p);

    /* Only makes sense for encoding */
    e_what |= WHAT_VIDEO;
  } else {
    /* Force create error model file */
    e_what = WHAT_ERRMOD;
  }
  break;
case 'f':
  e_fps = atofrac(argv[i]+2);
  break;
case 'g':
  e_gamma = atofrac(argv[i]+2);
  break;
```

```
case 'i':
  if (*(argv[i]+2) == 0) {
    e_interactive = !e_interactive;
  } else {
    e_interactive = atoi(argv[i]+2);
    if (e_interactive < 1) e_interactive = 1;
  }
  break;

//PSE20240626: Add k and m cases for NU
case 'k':
  e_fnfile = argv[i]+2;
  readGainFns(e_fnfile);
  dumpGainFns();
  break;
case 'm':
  e_maskfile = argv[i]+2;
  readMask(e_maskfile);
  break;

case 'n':
  if (*(argv[i]+2) == 0) {
    e_n = 0;
  } else {
    e_n = atoi(argv[i]+2);
    if (e_n < 1) e_n = 1;
  }
  break;
case 'o':
  e_outfile = (argv[i]+2);

  /* Guess type based on end of filename... */
  if (e_maketype == MAKE_UNSPEC) {
    register char *p = argv[i] + 2;
    register int len = strlen(p);

    switch (len) {
    case 1:
      if (*p != '-') break;
      /* Fall through... */
    case 0:
      /* PPM stream to stdout for "-o-" or "-o" */
      e_outfile = ((char *) "-");
      e_maketype = MAKE_PPM;
      if (e_what != WHAT_ERRMOD) e_what |= WHAT_TIK;
      break;
    case 2:
    case 3:
      /* Too short to say... */
      break;
    default:
      /* Does it end in something we know? */
      p += (len - 4);
      if (p[0] == '.') {
```

```
            if (((p[1] == 'J') || (p[1] == 'j')) &&
                ((p[2] == 'P') || (p[2] == 'p')) &&
                ((p[3] == 'G') || (p[3] == 'g'))) {
              e_maketype = MAKE_JPEG;
              e_what |= WHAT_TIK;
            }
            if (((p[1] == 'P') || (p[1] == 'p')) &&
                ((p[2] == 'P') || (p[2] == 'p') || (p[2] == 'N')
                   || (p[2] == 'n')) &&
                ((p[3] == 'M') || (p[3] == 'm'))) {
              e_maketype = MAKE_PPM;
              if (e_what != WHAT_ERRMOD) e_what |= WHAT_TIK;
            }
          }
        }
      }
      break;
    case 'p':
      e_percent = atofrac(argv[i]+2);
      if ((e_percent < 0) || (e_percent > 100)) {
        ERROR(ERROR_ARG,
              "%s␣is␣not␣valid;␣must␣be␣between␣-p0␣and␣-p100\n",
              argv[i]);
      }
      e_what |= WHAT_VIDEO;
      break;
    case 'q':
      e_quality = atofrac(argv[i]+2);
      if ((e_quality < 0) || (e_quality > 100)) {
        ERROR(ERROR_ARG,
              "%s␣is␣not␣valid;␣must␣be␣between␣-q0␣and␣-q100\n",
              argv[i]);
      }
      break;
    case 't':
      e_t = atofrac(argv[i]+2);
      break;
    case 'v':
      e_what |= WHAT_VIDEO;
      break;
    default:
      goto usage;
    }
  } else {
    /* What are we supposed to do with this file?
       Can't be nothing or more than one thing.
    */
    register char *p = argv[i];
    register int dottik = 0;

    /* Guess based on end of filename... */
    while (*p) ++p;
    p -= 4;
    if ((p[0] == '.') &&
```

```
        ((p[1] == 'T') || (p[1] == 't')) &&
        ((p[2] == 'I') || (p[2] == 'i')) &&
        ((p[3] == 'K') || (p[3] == 'k'))) {
      /* Force encoding exposures */
      e_what = WHAT_TIK;
      dottik = 1;
    } else {
      if (e_what == WHAT_UNSPEC) e_what = WHAT_VIDEO;
    }

//      if (e_t && (e_fps == 0)) e_fps = 1.0 / e_t;
//      if (e_fps && (e_t == 0)) e_t = 1.0 / e_fps;
    switch (e_what) {
    case WHAT_ERRMOD:
      if (e_n == 0) e_n = E_N_MAX;
      if (e_outfile == 0) e_outfile = e_outtik;
      INFO("creating error model \"%s\" from \"%s\"\n",
           e_outfile,
           argv[i]);
      start_opencv(argv[i]);
      PNMnoise(e_outfile);
      break;
    case WHAT_VIDEO:
      if (e_n == 0) e_n = E_N_MAX;
      if (e_outfile == 0) e_outfile = e_outtik;
      if (e_noisefile == 0) {
        /* No error model specified; make one */
        register int x, y;
        for (y=0; y<256; ++y) {
          for (x=0; x<256; ++x) {
            int z = 255 - ((x > y) ? (x-y) : (y-x));
            if (z < 0) z = 0;
            z -= ((255 - z) * (255 - z));
            if (z < 0) z = 0;
            PNMerr[x][y][0] = z;
            PNMerr[x][y][1] = z;
            PNMerr[x][y][2] = z;
          }
        }
      }
      start_opencv(argv[i]);
      PNMtdci(e_outfile);
      break;
    case WHAT_TIK:
      if (e_outfile == 0) e_outfile = e_outimage;
      now = ((e_begin < 0) ? 0 : e_begin);
      if (e_n <= 1) {
        /* One frame only */
        e_n = 1;
        if (e_t == 0) e_t = ((e_fps > 0) ? (1.0/e_fps) : (1.0/60))
           ;
        if (e_fps == 0) e_fps = e_t;
      } else {
        /* Video sequence */
```

```
          if (e_fps == 0) e_fps = 24;
          if (e_t == 0) e_t = 1.0 / e_fps;
        }

        /* Create exposures */
        /*PSE: Hack this to default to the functions in -k if
           supplied
         if a spec file is loaded,
           FirstLive() and LastLive() can replace now and now+e_t
        ... But don't do it until any problems and edge cases are
           worked out
         */
        //if(e_fnfile){TIKrender(FirstLive(), LastLive());}

        TIKrenderFile(dottik, argv[i]);
        for (n=0; n<e_n; ++n) {
          TIKrender(now, now+e_t);
          now += (1.0 / e_fps);
        }

        /* Simultaneously render all frames */
        TIKrenderSimul();

        /* Done with this file */
        break;
      default:
        ERROR(ERROR_ARG,
              "what to do with %s? %s%s%s\n",
              argv[i],
              ((e_what & WHAT_ERRMOD) ? " Make error model?" : ""),
              ((e_what & WHAT_VIDEO) ? " Encode as TIK?" : ""),
              ((e_what & WHAT_TIK) ? " Make exposures?" : ""));
      }
    }
  }

  INFO("completed in %d seconds\n", ((int)(time(0) - real_time)));
  return(0);
}
```

B.5 Function Plotter Script

Listing B.5: MaskGain.h

```
#!/usr/bin/python3
# Plots a gain function file for human viewing
import sys
import matplotlib.pyplot as plt
import parse as ps

if len(sys.argv) != 2:
  print ("Usage: FnPlotter FnFile.fn")

fns=[];

# Line format is M$n{$t0:$g0],...,[$tm:$gm]}
with open(sys.argv[1],"r") as fnfile:
  for line in fnfile:
    # Echo it back for testing
    #print(line.strip())
    ef={}
    if line.startswith('M'):
      tokens=line.split("{")
      #print(tokens)
      fnum=int(tokens[0][1:])
      #print("fnum="  + str(fnum) +"\n")
      ef["maskval"]=fnum
      #Strip the trailing curly and newline
      tuples = tokens[1][:-2].split(",")
      times=[]
      gains=[]
      for pts in tuples:
          #print(pts)
          vals=ps.parse("[{}:{}]",pts)
          #print("Time: " + vals[0] + " Gain: " + vals[1])
          times.append(float(vals[0]))
          gains.append(float(vals[1]))
    ef["times"]=times
    ef["gains"]=gains
    fns.append(ef)


#Print the data structure serialization if it looks like the sketchy
    parser is broken
#print(fns)

# Special case the single function in a file case because matplotlib
    is a little dumb
if len(fns) == 1:
  fig, ax = plt.subplots()
  fig.suptitle(sys.argv[1])
  ax.plot(ef["times"],ef["gains"])
  ax.set(title=ef["maskval"])
  #ax[i].set(xlabel='time (ns)', ylabel='gain',title=ef["maskval"])
```

```
  ax.grid()

else:
  fig, ax = plt.subplots(len(fns),sharex=True,sharey=True,
     gridspec_kw={'hspace': 0.505})
  fig.suptitle(sys.argv[1])
  i=0
  for ef in fns:
    ax[i].plot(ef["times"],ef["gains"])
    ax[i].set(title=ef["maskval"])
    #ax[i].set(xlabel='time (ns)', ylabel='gain',title=ef["maskval
       "])
    ax[i].grid()
    i=i+1

# fig.savefig("test.png")
plt.show()
```

Bibliography

[1] H. G. Dietz, "Frameless, time domain continuous image capture," in *Electronic Imaging 2014*, vol. 9022, 2014, pp. 7–12. DOI: 10.1117/12.2040016. [Online]. Available: http://dx.doi.org/10.1117/12.2040016.

[2] H. G. Dietz and P. S. Eberhart, "Iso-less?" In *Electronic Imaging 2014*, vol. 9404, 2015, pp. 94040L-94040L–14. DOI: 10.1117/12.2080168. [Online]. Available: http://dx.doi.org/10.1117/12.2080168.

[3] H. Dietz, Z. Snyder, J. Fike, and P. Quevedo, "Scene appearance change as framerate approaches infinity," *Electronic Imaging*, vol. 2016, no. 18, pp. 1–7, 2016, ISSN: 2470-1173. DOI: doi:10.2352/ISSN.2470-1173.2016.18.DPMI-259. [Online]. Available: http://www.ingentaconnect.com/content/ist/ei/2016/00002016/00000018/art00012.

[4] H. Dietz and P. Eberhart, "Shuttering methods and the artifacts they produce," *Electronic Imaging*, vol. 2019, no. 4, pp. 590-1-590–7, 2019, ISSN: 2470-1173. [Online]. Available: https://www.ingentaconnect.com/content/ist/ei/2019/00002019/00000004/art00010.

[5] H. Dietz, P. Eberhart, J. Fike, K. Long, and C. Demaree, "Temporal superresolution for time domain continuous imaging," *Electronic Imaging*, vol. 2017, no. 15, 2017.

[6] H. J. Landau, "Necessary density conditions for sampling and interpolation of certain entire functions," *Acta Mathematica*, vol. 117, no. none, pp. 37–52, 1967. DOI: 10.1007/BF02395039. [Online]. Available: https://doi.org/10.1007/BF02395039.

[7] E. J. Candès *et al.*, "Compressive sampling," in *Proceedings of the international congress of mathematicians*, Madrid, Spain, vol. 3, 2006, pp. 1433–1452.

[8] H. Dietz, P. Eberhart, J. Fike, K. Long, C. Demaree, and J. Wu, "Tik: A time domain continuous imaging testbed using conventional still images and video," *Electronic Imaging*, vol. 2017, no. 15, pp. 58–65, 2017, ISSN: 2470-1173. DOI: doi:10.2352/ISSN.2470-1173.2017.15.DPMI-081. [Online]. Available: http://www.ingentaconnect.com/content/ist/ei/2017/00002017/00000015/art00010.

[9] J. Poskanzer, B. Henderson, and C. netpbm, *Netpbm*, Jan. 2014. [Online]. Available: http://netpbm.sourceforge.net/doc/.

[10] K. Long, H. Dietz, and C. Demaree, "A canon hack development kit implementation of time domain continuous imaging," *Electronic Imaging*, vol. 2017, no. 15, pp. 66–72, 2017, ISSN: 2470-1173. DOI: doi:10.2352/ISSN.2470-1173.2017.15.DPMI-075. [Online]. Available: http://www.ingentaconnect.com/content/ist/ei/2017/00002017/00000015/art00011.

[11] N. T. S. C. (1951-1953), *Report and Reports of Panel No. 11, 11-A, 12-19: With Some Supplementary References Cited in the Reports, and the Petition for Adoption of Transmission Standards for Color Television Before the Federal Communications Commission*. 1953. [Online]. Available: https://lccn.loc.gov/54021386.

[12] I. C. Abrahams, "Choice of chrominance subcarrier frequency in the ntsc standards," *Proceedings of the IRE*, vol. 42, no. 1, pp. 79–80, 1954. DOI: 10.1109/JRPROC.1954.274612.

[13] J. L. Barron, D. J. Fleet, and S. S. Beauchemin, "Performance of optical flow techniques," *International Journal of Computer Vision*, vol. 12, no. 1, pp. 43–77, Feb. 1994, ISSN: 0920-5691. DOI: 10.1007/BF01420984. [Online]. Available: http://dx.doi.org/10.1007/BF01420984.

[14] THE INTERNATIONAL TELEGRAPH AND TELEPHONE CONSULTATIVE COMMITTEE, "Codec for audiovisual services at n x 384 kbit/s," INTERNATIONAL TELECOMMUNICATION UNION, Standard, 1988.

[15] ISO/IEC JTC 1/SC 29, "Iso/iec 1172: Coding of moving pictures and associated audio for digital storage media at up to about 1.5 mbit/s," International Organization for Standardization, Geneva, CH, Standard, 1993.

[16] S. Baker, D. Scharstein, J. Lewis, S. Roth, M. Black, and R. Szeliski, "A database and evaluation methodology for optical flow," *International Journal of Computer Vision*, vol. 92, pp. 1–31, Jan. 2007. DOI: 10.1007/s11263-010-0390-2.

[17] S. Niklaus, L. Mai, and F. Liu, "Video frame interpolation via adaptive separable convolution," in *2017 IEEE International Conference on Computer Vision (ICCV)*, Los Alamitos, CA, USA: IEEE Computer Society, Oct. 2017, pp. 261–270. DOI: 10.1109/ICCV.2017.37. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/ICCV.2017.37.

[18] I. Nikon. "Photography terms and video terminology - bss (best shot selector)." (), [Online]. Available: https://www.nikonusa.com/learn-and-explore/photography-glossary#term-bss-best-shot-selector-.

[19] "These cameras can capture images before youve even finished clicking." (Nov. 11, 2022), [Online]. Available: https://www.digitec.ch/en/page/these-cameras-can-capture-images-before-youve-even-finished-clicking-25486.

[20] P. E. Debevec and J. Malik, "Recovering high dynamic range radiance maps from photographs," in *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '97, USA: ACM Press/Addison-Wesley Publishing Co., 1997, pp. 369–378, ISBN: 0897918967. DOI: 10.1145/258734.258884. [Online]. Available: https://doi.org/10.1145/258734.258884.

[21] "Auto exposure bracketing settings by camera model." (Jul. 19, 2024), [Online]. Available: https://www.hdrsoft.com/resources/aeb.html.

[22] R. Pourreza-Shahri and N. Kehtarnavaz, "Exposure bracketing via automatic exposure selection," in *2015 IEEE International Conference on Image Processing (ICIP)*, 2015, pp. 320–323. DOI: 10.1109/ICIP.2015.7350812.

[23] S. W. Hasinoff, F. Durand, and W. T. Freeman, "Noise-optimal capture for high dynamic range photography," in *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2010, pp. 553–560. DOI: 10.1109/CVPR.2010.5540167.

[24] G. Eilertsen, J. Kronander, G. Denes, R. K. Mantiuk, and J. Unger, "Hdr image reconstruction from a single exposure using deep cnns," *ACM Transactions on Graphics*, vol. 36, no. 6, pp. 1–15, Nov. 2017, ISSN: 1557-7368. DOI: 10.1145/3130800.3130816. [Online]. Available: http://dx.doi.org/10.1145/3130800.3130816.

[25] D. Marnerides, T. Bashford-Rogers, J. Hatchett, and K. Debattista, "Expandnet: A deep convolutional neural network for high dynamic range expansion from low dynamic range content," *Computer Graphics Forum*, vol. 37, 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:3732673.

[26] T. Helmut Dersch. "Panorama tools." (), [Online]. Available: https://www.panotools.org/ (visited on 08/05/2024).

[27] V. Bushnell. "Patents on graphical software." (), [Online]. Available: http://www.cptech.org/ip/business/software/graphics.html#ipix (visited on 08/05/2024).

[28] S. D. Zimmermann, *Omniview motionless camera orientation system*, Feb. 1993. [Online]. Available: https://patents.google.com/patent/US5185667A/en.

[29] K. Hart, "Sony wins bid for ipix camera technology patents," *The Washington Post*, Feb. 5, 2007. [Online]. Available: https://www.washingtonpost.com/wp-dyn/content/article/2007/02/04/AR2007020401109.html (visited on 08/05/2024).

[30] S. Farsiu, M. Elad, and P. Milanfar, "Multiframe demosaicing and super-resolution of color images," *IEEE Transactions on Image Processing*, vol. 15, no. 1, pp. 141–159, Jan. 2006, ISSN: 1057-7149. DOI: doi:10.1109.

[31] B. K. Karch and R. C. Hardie, "Robust super-resolution by fusion of interpolated frames for color and grayscale images," *Frontiers in Physics*, vol. 3, 2015, ISSN: 2296-424X. DOI: 10.3389/fphy.2015.00028. [Online]. Available: https://www.frontiersin.org/articles/10.3389/fphy.2015.00028.

[32] O. Liba, K. Murthy, Y.-T. Tsai, *et al.*, "Handheld mobile photography in very low light," *ACM Trans. Graph.*, vol. 38, no. 6, Nov. 2019, ISSN: 0730-0301. DOI: 10.1145/3355089.3356508. [Online]. Available: https://doi.org/10.1145/3355089.3356508.

[33] G. Bishop, H. Fuchs, L. McMillan, and E. J. S. Zagier, "Frameless rendering: Double buffering considered harmful," in *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '94, New York, NY, USA: Association for Computing Machinery, 1994, pp. 175–176, ISBN: 0897916670. DOI: 10.1145/192161.192195. [Online]. Available: https://doi.org/10.1145/192161.192195.

[34] J. Gosling, "A redisplay algorithm," in *Proceedings of the ACM SIGPLAN SIGOA Symposium on Text Manipulation*, Portland, Oregon, USA: Association for Computing Machinery, 1981, pp. 123–129, ISBN: 0897910508. DOI: 10.1145/800209.806463. [Online]. Available: https://doi.org/10.1145/800209.806463.

[35] X. Hugl. "Fixing kwin's performance on old hardware." (Jun. 26, 2024), [Online]. Available: https://zamundaaa.github.io/wayland/2024/06/25/fixing-kwin-perf-on-old-hardware.html (visited on 06/25/2024).

[36] C. Su. "Enhancing the user experience of e-paper with fast (and partial) updates." (Jun. 30, 2023), [Online]. Available: https://www.pervasivedisplays.com/fast-partial-updates-provide-an-improved-user-experience-when-using-e-paper-displays/ (visited on 08/11/2024).

[37] K. Shanmugan and A. Breipohl, *Random Signals: Detection Estimation and Data Analysis* (Detection Estimation and Data Analysis). Wiley, 1988, ISBN: 9780471612742. [Online]. Available: https://books.google.com/books?id=5A37AAAACAAJ.

[38] J. R. Janesick, "Scientific charge-coupled devices," in Bellingham: SPIE, 2001, ch. 1:History, Operation, Performance, Design, Fabrication and Theory, ISBN: 9780819480392. DOI: 10.1117/3.374903. [Online]. Available: https://www.spiedigitallibrary.org/ebooks/PM/Scientific-Charge-Coupled-Devices/.

[39] ISO, *INTERNATIONAL STANDARD ISO12232-2006 - Photography - Digital still cameras - Determination of exposure index, ISO speed ratings, standard output sensitivity, and recommended exposure index*, 2006.

[40] D. Kerr, "Apex - the additive system of photographic exposure," Tech. Rep., 2007. [Online]. Available: http://dougkerr.net/Pumpkin/articles/APEX.pdf.

[41] CIPA Standardization Committee, *Cipa dc-008-2016: Exchangeable image file format for digital still cameras: Exif version 2.31*, Jul. 2016. [Online]. Available: http://www.cipa.jp/std/documents/e/DC-008-Translation-2016-E.pdf.

[42] H. G. Dietz, "Fujifilm x10 white orbs and deorbit," *SPIE Conference Proceedings*, vol. 8660, pp. 8660–8675, 2013. DOI: 10.1117/12.2004411. [Online]. Available: https://doi.org/10.1117/12.2004411.

[43] M. F. Tompsett, G. F. Amelio, and G. E. Smith, "Charge coupled 8-bit shift register," *Applied Physics Letters*, vol. 17, no. 3, pp. 111–115, 1970. DOI: 10.1063/1.1653327. [Online]. Available: https://doi.org/10.1063/1.1653327.

[44] N. Tompsett Michael Francis (Summit, *Charge transfer imaging devices*, Apr. 1978. [Online]. Available: http://www.freepatentsonline.com/4085456.html.

[45] R. Cicala, *Sensor stack thickness: When does it matter?* L. R. Blog, Ed., Jun. 2014. [Online]. Available: https://www.lensrentals.com/blog/2014/06/sensor-stack-thickness-when-does-it-matter/.

[46] A. Darmont, "Spectral response of silicon image sensors," Aphesa, Tech. Rep., Apr. 2009. [Online]. Available: http://www.aphesa.com/downloads/download2.php?id=1.

[47] J. Nakamura, *Image Sensors and Signal Processing for Digital Still Cameras*. Boca Raton, FL, USA: CRC Press, Inc., 2005, ISBN: 0849335450.

[48] "Trichromatic colour reproduction and the additive principle," in *The Reproduction of Colour*. Wiley-Blackwell, 2005, ch. 2, pp. 9–17, ISBN: 9780470024270. DOI: 10.1002/0470024275.ch2. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/0470024275.ch2. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/0470024275.ch2.

[49] Eric R. Jeschke, *Converting color images to b&w*, I'm sure there's a more academic source for the 30/59/11 RGB ratio, but this is what I know it from, 2002. [Online]. Available: https://www.gimp.org/tutorials/Color2BW/.

[50] A. Davies and P. Fennessy, *Digital Imaging for Photographers with CD (Audio)*, 4th. Newton, MA, USA: Butterworth-Heinemann, 2001, Accessed via Google Books, ISBN: 0240515900. [Online]. Available: https://books.google.com/books?vid=ISBN0240515900.

[51] H. G. Dietz, "Credible repair of sony main-sensor pdaf striping artifacts," *Electronic Imaging*, vol. 2019, no. 4, pp. 585-1-585–7, 2019, ISSN: 2470-1173. [Online]. Available: https://www.ingentaconnect.com/content/ist/ei/2019/00002019/00000004/art00006.

[52] Á. R.-V. Juan A. Leñero-Bardallo Jorge Fernández-berni, "Review of adcs for imaging," vol. 9022, 2014, pp. 9022 - 9022 –6. DOI: 10.1117/12.2041682. [Online]. Available: https://doi.org/10.1117/12.2041682.

[53] E. R. Fossum, J. Ma, S. Masoodian, L. Anzagira, and R. Zizza, "The Quanta Image Sensor: Every Photon Counts," *Sensors (Basel)*, vol. 16, no. 8, Aug. 2016.

[54] H. G. Dietz, "Programmable nanocontrollers for nanodevices," University of Kentucky, Tech. Rep., 2003. [Online]. Available: http://aggregate.org/KYARCH/cases2003.pdf.

[55] G. Gallego, T. Delbrück, G. Orchard, *et al.*, "Event-based vision: A survey," *CoRR*, vol. abs/1904.08405, 2019. arXiv: 1904.08405. [Online]. Available: http://arxiv.org/abs/1904.08405.

[56] C. Posch, D. Matolin, and R. Wohlgenannt, "A qvga 143 db dynamic range frame-free pwm image sensor with lossless pixel-level video compression and time-domain cds," *IEEE Journal of Solid-State Circuits*, vol. 46, no. 1, pp. 259–275, Jan. 2011. DOI: 10.1109/JSSC.2010.2085952.

[57] C. Hahne. "The standard plenoptic camera." (), [Online]. Available: http://www.plenoptic.info/index.html (visited on 06/07/2024).

[58] E. Adelson and J. Wang, "Single lens stereo with a plenoptic camera," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 14, no. 2, pp. 99–106, 1992. DOI: 10.1109/34.121783.

[59] R. Ng, "Digital light field photography," Ph.D. dissertation, Stanford University, 2006.

[60] D. Pierce, "Lytro illum review: This is the camera of the future," *The Verge*, Jul. 30, 2014. [Online]. Available: https://www.theverge.com/2014/7/30/5949913/lytro-illum-review (visited on 06/07/2024).

[61] M. Kosoff, "Lytro, a 'magic' camera startup that raised $140 million, announces layoffs and pivots to virtual reality," *Business Insider*, Feb. 26, 2015. [Online]. Available: https://www.businessinsider.in/lytro-a-magic-camera-startup-that-raised-140-million-announces-layoffs-and-pivots-to-virtual-reality/articleshow/46387133.cms.

[62] M. Zhang, "The first plenoptic camera on the market," *petapixel*, Sep. 23, 2010. [Online]. Available: https://petapixel.com/2010/09/23/the-first-plenoptic-camera-on-the-market/.

[63] *Canon hack development kit*. [Online]. Available: http://chdk.wikia.com/wiki/CHDK.

[64] Sony Group Corporation. "Sony global - source code distribution service." (), [Online]. Available: https://oss.sony.net/Products/Linux/common/search.html (visited on 03/14/2024).

[65] D. Coffin, *Decoding raw digital photos in linux*. [Online]. Available: https://www.cybercom.net/~dcoffin/dcraw/.

[66] S. W. Hasinoff, "Photon , poisson noise (preprint)," in *Computer Vision; A Reference Guide*, K. Ikeuchi, Ed., Oxford: Springer US, 2014, pp. 608–610. [Online]. Available: https://people.csail.mit.edu/hasinoff/pubs/hasinoff-photon-2012-preprint.pdf.

[67] J. J. Gibson, *The perception of the visual world / James J. Gibson*, English. Hooughton Mifflin Boston, 1950, xii, 235 p. :

[68] P. Pérez, M. Gangnet, and A. Blake, "Poisson image editing," *ACM Transactions on Graphics*, vol. 22, no. 3, pp. 313–318, Jul. 2003, ISSN: 0730-0301. DOI: 10.1145/882262.882269. [Online]. Available: http://doi.acm.org/10.1145/882262.882269.

[69] E. H. Land and J. J. McCann, "Lightness and retinex theory," *Journal of the Optical Society of America*, vol. 61, no. 1, pp. 1–11, Jan. 1971. DOI: 10.1364/JOSA.61.000001. [Online]. Available: http://www.osapublishing.org/abstract.cfm?URI=josa-61-1-1.

[70] a1ex. "Dynamic range improvement for some canon dslrs by alternating iso during sensor readout." (), [Online]. Available: http://phd-sid.ethz.ch/debian/magiclantern/dual_iso.pdf (visited on 07/14/2014).

[71] T. Williams, C. Kelley, and many others, *Gnuplot 4.6: An interactive plotting program*, http://gnuplot.sourceforge.net/, Apr. 2013.

[72] H. Dietz, W. Davis, and P. Eberhart, "Characterization of camera shake," *Electronic Imaging*, vol. 32, no. 7, pp. 228-1–228-1, 2020. DOI: 10.2352/ISSN.2470-1173.2020.7.ISS-228. [Online]. Available: https://library.imaging.org/ei/articles/32/7/art00011.

[73] H. Dietz, P. Eberhart, and C. Demaree, "Multispectral, high dynamic range, time domain continuous imaging," *Electronic Imaging*, vol. 2018, no. 5, pp. 409-1-409–9, Jan. 28, 2018, ISSN: 2470-1173. DOI: doi:10.2352/ISSN.2470-1173.2018.05.PMII-409. [Online]. Available: https://www.ingentaconnect.com/content/ist/ei/2018/00002018/00000005/art00012.

[74] A. A. with Robert Baker, "The print," in 3. 1271 Avenue of the Americas, New York, NY 10020: Little, Brown and Company, 1995, vol. Ansel Adams Photography, ch. The Fine Print: Control of Values, pp. 89–127, ISBN: 0-8212-2187-6.

[75] P. Baumgarten, "Focus stacking & bracketing with om-d," OM Digital Solutions Corporation, Tech. Rep., 2016. [Online]. Available: https://learnandsupport.getolympus.com/learn-center/photography-tips/macro/focus-stacking-bracketing-with-om-d.

[76] E. E. Catmull and R. Rom, "A class of local interpolating splines," *Computer Aided Geometric Design*, pp. 317–326, 1974. [Online]. Available: https://api.semanticscholar.org/CorpusID:118383557.

[77] R. Burden and J. Faires, "Numerical analysis," in Cengage Learning, 2010, ch. 3, pp. 144–153, ISBN: 9780538733519. [Online]. Available: https://books.google.com/books?id=zXnSxY9G2JgC.

[78] P. Eberhart and H. G. Dietz, "Non-uniform integration of tdci captures," *Electronic Imaging*, vol. 32, no. 7, pp. 330-1–330-1, 2020. DOI: 10.2352/ISSN.2470-1173.2020.7.ISS-330. [Online]. Available: https://library.imaging.org/ei/articles/32/7/art00017.

[79] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[80] *Sony playmemories camera apps, a camera application download service*. [Online]. Available: https://www.playmemoriescameraapps.com.

[81] Sony Group Corporation. "Playmemories camera apps ending." (), [Online]. Available: https://www.sony.com/electronics/support/articles/00289030 (visited on 03/14/2024).

[82] k. oz_paulb kenan, *Fwtool*, 2012. [Online]. Available: http://www.personal-view.com/faqs/sony-hack/fwtool.

[83] ma1co, *Fwtool.py*, 2018. [Online]. Available: http://www.personal-view.com/faqs/sony-hack/fwtool.

[84] ma1co, *Sony-pmca-re*, 2018. [Online]. Available: https://github.com/ma1co/Sony-PMCA-RE.

[85] ma1co, *Openmemories:tweak*, 2018. [Online]. Available: https://github.com/ma1co/OpenMemories-Tweak.

[86] K. Shirriff. "Understanding sony ir remote codes, lirc files, and the arduino library." (Mar. 21, 2010), [Online]. Available: https://www.righto.com/2010/03/understanding-sony-ir-remote-codes-lirc.html (visited on 03/21/2024).

[87] Canon Inc. "Canon technology — dryos (archived)." (), [Online]. Available: https://web.archive.org/web/20080116050120/http://www.canon.com/technology/canon_tech/explanation/dryos.html (visited on 11/20/2008).

[88] H. Dietz, C. Demaree, P. Eberhart, C. Kuball, and J. Y. Wu, "Lessons from design, construction, and use of various multicameras," *Electronic Imaging*, vol. 30, no. 5, pp. 182-1–182-1, 2018. DOI: 10.2352/ISSN.2470-1173.2018.05.PMII-182. [Online]. Available: https://library.imaging.org/ei/articles/30/5/art00004.

[89] H. Dietz, C. Parrish, and K. D. Donohue, "Self-contained, passive, non-contact, photoplethysmography: Real-time extraction of heart rates from live view within a canon powershot," *Electronic Imaging*, vol. 31, no. 13, pp. 146-1–146-1, 2019. DOI: 10.2352/ISSN.2470-1173.2019.13.COIMG-146. [Online]. Available: https://library.imaging.org/ei/articles/31/13/art00012.

[90] *The magic lantern project*. [Online]. Available: https://www.magiclantern.fm/.

[91] a1ex et.al. "Magic lantern forum: Topic: Edmac internals." (), [Online]. Available: https://www.magiclantern.fm/forum/index.php?topic=18315.msg245609 (visited on 12/20/2023).

[92] K. Ushida, Y. Naoi, Y. Katahira, and K. Morishita, "Image processing apparatus and image processing method," US7817297B2, Dec. 2010. [Online]. Available: https://patents.google.com/patent/US7817297.

[93] The Magic Lantern Community. "Magic lantern firmware wiki: Register map." (), [Online]. Available: https://magiclantern.fandom.com/wiki/Register_Map (visited on 12/20/2023).

[94] e. David Milligan. "Magic lantern forum: Topic: Mlv lite." (), [Online]. Available: https://www.magiclantern.fm/forum/index.php?topic=16650.0 (visited on 12/20/2023).

[95] I. Sibiryakov. "Mlv-app: All in one mlv processing app." (), [Online]. Available: https://github.com/ilia3101/MLV-App (visited on 12/20/2023).

[96] CarVac. "Librtprocess: A project to make rawtherapee's processing algorithms more readily available." (), [Online]. Available: https://github.com/CarVac/librtprocess (visited on 12/20/2023).

[97] D. Milligan. "Mlvfs: Mlv (magic lantern raw video format) "converter" that uses fuse to allow "mounting" of mlv files as filesystems." (), [Online]. Available: https://bitbucket.org/dmilligan/mlvfs/src/master/ (visited on 12/20/2023).

[98] Magic Lantern Development Team. "Magic lantern main builds." (), [Online]. Available: https://builds.magiclantern.fm/.

[99] P. S. Eberhart and H. Dietz, "Magic lantern as a platform for digital photography research," *Electronic Imaging*, 2024.

[100] Ai-Thinker Technology. "Esp32-cam camera development board." (), [Online]. Available: https://docs.ai-thinker.com/en/esp32-cam (visited on 05/06/2024).

[101] H. Dietz and P. Eberhart, "An ultra-low-cost large-format wireless iot camera," *Electronic Imaging*, vol. 33, no. 7, pp. 70-1–70-1, 2021. DOI: 10.2352/ISSN.2470-1173.2021.7.ISS-070. [Online]. Available: https://library.imaging.org/ei/articles/33/7/art00005.

[102] H. Dietz, D. Abney, P. Eberhart, *et al.*, "Esp32-cam as a programmable camera research platform," *Electronic Imaging*, vol. 34, no. 7, pp. 232-1–232-1, 2022. DOI: 10.2352/EI.2022.34.7.ISS-232. [Online]. Available: https://library.imaging.org/ei/articles/34/7/ISS-232.

[103] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007. DOI: 10.1109/MCSE.2007.55.

[104] N. Sekar. "Intersection of s-curves and disruption: Digital photography vs. film photography." (2024), [Online]. Available: https://medium.com/@nareshnavinash/intersection-of-s-curves-and-disruption-digital-photography-vs-film-photography-0084192f4a63 (visited on 07/09/2024).

[105] S. Follows. "When and how the film business went digital." (2017), [Online]. Available: https://stephenfollows.com/film-business-became-digital/ (visited on 07/09/2024).

[106] C. Odenbach. "The canon ae-1: A new kind of slr." (2016), [Online]. Available: https://www.678vintagecameras.ca/blog/the-canon-ae-1-a-new-kind-of-slr (visited on 06/09/2024).

[107] M. Johnston. "A (very) brief and somewhat lighthearted history of modes." (2014), [Online]. Available: https://theonlinephotographer.typepad.com/the_online_photographer/2014/04/a-very-brief-history-of-modes.html (visited on 06/09/2024).

[108] MPB, Inc. "Timeline: The rise of the mirrorless camera." (2023), [Online]. Available: https://www.mpb.com/en-us/content/latest-from-mpb/timeline-the-rise-of-the-mirrorless-camera (visited on 07/09/2024).

[109] J. Ma, D. Zhang, D. Robledo, L. Anzagira, and S. Masoodian, "Ultra-high-resolution quanta image sensor with reliable photon-number-resolving and high dynamic range capabilities," en, *Sci. Rep.*, vol. 12, no. 1, p. 13 869, Aug. 2022. [Online]. Available: https://www.nature.com/articles/s41598-022-17952-z.

[110] J. Fisher, "Hands on: Sony's speedy a9 iii has a global shutter, 120fps burst rate," *PCMag*, Nov. 9, 2023. [Online]. Available: https://www.pcmag.com/news/hands-on-sonys-speedy-a9-iii-has-a-global-shutter-120fps-burst-rate.

[111] R. Butler, "Sony a9 iii: Global shutter comes with an image quality cost," *dpreview*, Jan. 4, 2024. [Online]. Available: https://www.dpreview.com/articles/6717086661/sony-a9-iii-image-quality-dynamic-range-analysis.

[112] A. Lee, "Review: The light l16 is brilliant... and braindead," *petapixel*, Dec. 8, 2017. [Online]. Available: https://petapixel.com/2017/12/08/review-light-l16-brilliant-braindead/.

[113] J. Schneider, "Samsung argues that there is 'no such thing as a real picture'," *petapixel*, Feb. 5, 2024. [Online]. Available: https://petapixel.com/2024/02/05/samsung-argues-that-there-is-no-such-thing-as-a-real-picture/.

[114] R. Amadeo, "Moon-gate: Samsung fans are mad about ai-processed photos of the moon," *Ars Technica*, Mar. 16, 2023. [Online]. Available: https://arstechnica.com/gadgets/2023/03/samsung-says-it-adds-fake-detail-to-moon-photos-via-reference-photos/.

[115] A. Y. Khaled, N. Ekramirad, C. A. Parrish, *et al.*, "Non-destructive detection of codling moth infestation in apples using acoustic impulse response signals," *Biosystems Engineering*, vol. 224, pp. 68–79, 2022, ISSN: 1537-5110. DOI: https://doi.org/10.1016/j.biosystemseng.2022.10.001. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1537511022002239.

[116] H. Dietz, P. Eberhart, and A. Rule, "Basic operations and structure of an fpga accelerator for parallel bit pattern computation," in *2021 International Conference on Rebooting Computing (ICRC)*, 2021, pp. 129–133. DOI: 10.1109/ICRC53822.2021.00029.

[117] H. Dietz and P. Eberhart, "Senscape: Modeling and presentation of uncertainty in fused sensor data live image streams," *Electronic Imaging*, vol. 32, no. 14, pp. 392-1–392-1, 2020. DOI: 10.2352/ISSN.2470-1173.2020.14.COIMG-392. [Online]. Available: https://library.imaging.org/ei/articles/32/14/art00024.

[118] S. M. Sridhar, H. G. Dietz, and P. S. Eberhart, "Use of flawed and ideal image pairs to drive filter creation by genetic programming," *Electronic Imaging*, vol. 28, no. 18, pp. 1–1, 2016. DOI: 10.2352/ISSN.2470-1173.2016.18.DPMI-016. [Online]. Available: https://library.imaging.org/ei/articles/28/18/art00023.

[119] P. Eberhart and H. Dietz, "A compiler target model for line associative registers," in *15th Workshop on Compilers for Parallel Computing*, (Vienna University of Technology, Vienna, Austria), Jul. 2010.

Vita

Paul Selegue Eberhart of Lexington, KY earned Bachelors of Science in Electrical Engineering, Computer Engineering, and Computer Science (2009), and a Masters of Science in Electrical Engineering (2019) at the University of Kentucky. He has worked in a series of instructional roles for the department of Electrical and Computer Engineering at the University of Kentucky beginning in 2011.
His publications include:

- P. S. Eberhart and H. Dietz, "Magic lantern as a platform for digital photography research," *Electronic Imaging*, 2024
- H. Dietz, D. Abney, P. Eberhart, *et al.*, "Esp32-cam as a programmable camera research platform," *Electronic Imaging*, vol. 34, no. 7, pp. 232-1–232-1, 2022
- A. Y. Khaled, N. Ekramirad, C. A. Parrish, *et al.*, "Non-destructive detection of codling moth infestation in apples using acoustic impulse response signals," *Biosystems Engineering*, vol. 224, pp. 68–79, 2022, ISSN: 1537-5110
- H. Dietz, P. Eberhart, and A. Rule, "Basic operations and structure of an fpga accelerator for parallel bit pattern computation," in *2021 International Conference on Rebooting Computing (ICRC)*, 2021, pp. 129–133
- H. Dietz and P. Eberhart, "An ultra-low-cost large-format wireless iot camera," *Electronic Imaging*, vol. 33, no. 7, pp. 70-1–70-1, 2021
- P. Eberhart and H. G. Dietz, "Non-uniform integration of tdci captures," *Electronic Imaging*, vol. 32, no. 7, pp. 330-1–330-1, 2020
- H. Dietz, W. Davis, and P. Eberhart, "Characterization of camera shake," *Electronic Imaging*, vol. 32, no. 7, pp. 228-1–228-1, 2020
- H. Dietz and P. Eberhart, "Senscape: Modeling and presentation of uncertainty in fused sensor data live image streams," *Electronic Imaging*, vol. 32, no. 14, pp. 392-1–392-1, 2020
- H. Dietz and P. Eberhart, "Shuttering methods and the artifacts they produce," *Electronic Imaging*, vol. 2019, no. 4, pp. 590-1-590–7, 2019, ISSN: 2470-1173
- H. Dietz, P. Eberhart, and C. Demaree, "Multispectral, high dynamic range, time domain continuous imaging," *Electronic Imaging*, vol. 2018, no. 5, pp. 409-1-409–9, Jan. 28, 2018, ISSN: 2470-1173
- H. Dietz, C. Demaree, P. Eberhart, *et al.*, "Lessons from design, construction, and use of various multicameras," *Electronic Imaging*, vol. 30, no. 5, pp. 182-1–182-1, 2018
- H. Dietz, P. Eberhart, J. Fike, *et al.*, "Temporal super-resolution for time domain continuous imaging," *Electronic Imaging*, vol. 2017, no. 15, 2017

- H. Dietz, P. Eberhart, J. Fike, *et al.*, "Tik: A time domain continuous imaging testbed using conventional still images and video," *Electronic Imaging*, vol. 2017, no. 15, pp. 58–65, 2017, ISSN: 2470-1173
- S. M. Sridhar, H. G. Dietz, and P. S. Eberhart, "Use of flawed and ideal image pairs to drive filter creation by genetic programming," *Electronic Imaging*, vol. 28, no. 18, pp. 1–1, 2016
- H. G. Dietz and P. S. Eberhart, "Iso-less?" In *Electronic Imaging 2014*, vol. 9404, 2015, pp. 94040L-94040L–14
- P. Eberhart and H. Dietz, "A compiler target model for line associative registers," in *15th Workshop on Compilers for Parallel Computing*, (Vienna University of Technology, Vienna, Austria), Jul. 2010