TIK: a time domain continuous imaging testbed using conventional still images and video

Henry Dietz, Paul Eberhart, John Fike, Katie Long, Clark Demaree, and Jong Wu;
Department of Electrical and Computer Engineering, University of Kentucky; Lexington, Kentucky

Abstract

Time domain continuous imaging (TDCI) centers on the capture and representation of time-varying image data not as a series of frames, but as a compressed continuous waveform per pixel. A high-dynamic-range (HDR) image can be computationally synthesized from TDCI data to represent any virtual exposure interval covered by the waveforms, thus allowing both exposure start time and shutter speed to be selected arbitrarily after capture. This also enables extraction of video with arbitrary framerate and shutter angle. This paper presents the design, and discusses performance, of the first complete, fully open source, infrastructure supporting experimental use of TDCI: TIK (Temporal Imaging from Kentucky or Temporal Image Kontainer). The system not only provides for processing TDCI `.tik` *files, but also allows conventional video files and still image sequences to be converted into TDCI* `.tik` *files.*

Introduction

For as long as there have been cameras, there has been the notion that the unit of capture is an image. The shutter is opened, light energy is collected, the shutter is closed, and the energy collected is processed to create an image. Even movies and video are nothing more than a sequence of images. Time Domain Continuous Imaging (TDCI)[1] offers an alternative model in which time is a fully-formed dimension. The value of each pixel is treated as a continuous function varying over time, and an image is nothing more than the integral of each pixel's waveform over a designated exposure interval.

The philosophy of TDCI

Fundamentally, TDCI is based on the assumption that scene appearance usually changes as a continuous function. Further, the scene usually changes slower than sources of sampling noise, such a photon arrival rate – and if it doesn't, it is impossible to know that it didn't. This is fundamentally different from how the photographic and image-processing communities normally think about image capture, which is as recording light levels rather than constructing a model of time-varying scene appearance. Two simple thought experiments help to illustrate the difference.

Suppose that you wish to create an image representing the appearance of a scene in the time interval 1..2. In a relatively dark region of the image, there is a particular pixel that receives not a single photon during that period. However, imagine that the time period for which the camera was active was actually 0..3, and that two photons were sensed by that pixel during that interval. One photon arrived at time 0.999 and the second at time 2.001. What is the best estimate of the value of that pixel worth of scene content for the time interval 1..2? Classically, one would say 0 because no photons arrived during that interval. However, TDCI would say that the best estimate is 2/3 of a photon. Superficially, the concept of having 2/3 of a photon seems a violation of quantum mechanics, but what it is really saying is that the portion of the scene sampled by the pixel probably had the same appearance throughout the 0..3 interval, with an expected value of 2/3. In essence, noting arrival of a single photon at a particular time is measuring photon shot noise at least as much as it is measuring scene appearance. If the scene content isn't changing, then averaging over more samples (i.e., a longer time interval) provides a statistically more accurate estimate of scene appearance.

For the second thought experiment, imagine that you have a blue sheet of paper. The paper appears blue in a wide range of lighting conditions. However, you decide to take that sheet of paper into the darkest cave and turn off all lights. What color is the paper then? Classical photography and image processing say it is black, because there is no light observed from it. However, we suggest that it is probably still blue – it just cannot be proven that it is still blue without using light to sample it. Lacking evidence that the sheet's color has changed, we argue that it is most reasonable to assume that the sheet is still blue, but very dark because it is so poorly lit.

Put simply, the goal of TDCI is construction of a model of the inherent, persistent, material properties of scene appearance over time as revealed by a lighting model, but free of sampling noise. Thus, TDCI not only allows after-capture specification of the time interval represented by a rendered image, but also can produce a result with low noise – often less noise than the photon shot noise that was present during the actual interval.

The images in Figure 1 clearly demonstrate this improvement. These four images all start one second into a four-second 960FPS high-speed video of a static scene captured using a Sony RX100 IV. More precisely, the camera was set to 1/1000s shutter speed and ISO 6400, and the capture was made at 959.04FPS using 1136x384 resolution, which it then upscaled and compressed as a 1920x1080 MP4 video. The first image is a single video frame extracted using `ffmpeg`. The next three images are all virtual exposures computationally rendered using `tik`. `tik` was used to create a noise model, then that noise model was applied to create a TDCI stream in `tik` format from the MP4 video, and finally `tik` was used to computationally render images for time intervals starting at the same time as the frame grab. The three rendered images represent virtual shutter speeds of 1/960s, 1/24s, and 1s (the last two are shown smaller here to save space).

Figure 1. *960FPS frame grab from Sony RX100 IV video and TDCI virtual exposures with shutter 1/960s, 1/24s, and 1s*

Although no spatial noise reduction nor post-processing has been applied to the `tik`-rendered images, the improvement in image quality over the video frame is obvious. Very conservatively using the `tik`-generated error model (i.e., requiring high confidence to treat a slightly unexpected value as noise), the average pixel's value in this video was found to be stable for approximately 7 frames. Thus, although the analysis is done on a per-pixel basis, the 1/960s virtual exposure is obtaining benefits very similar to those that would be obtained by averaging over 7 stacked[2, 4] frames. Of course, if there were significant changes in appearance of any portions of the scene, the pixel values in those portions were not stacked by `tik`. The quality of the longer virtual exposures is even higher because they are in effect stacking respectively about 47 and 967 frames each.

To approximately quantify the improvement, a simple signal-to-noise ratio (SNR) can be computed taking the 1s virtual exposure as the reference signal. Computing directly on the pixel values encoded in the JPEG images, the average pixel SNR is roughly 13 for the frame grab, 20 for the 1/960s virtual exposure, and 26 for the 1/24s virtual exposure. However, converting the JPEG pixel values to a roughly linear gamma, the average pixel SNR more than doubles going from the frame grab to the 1/960s virtual exposure and nearly doubles again going to the 1/24s virtual exposure. These improvements are less than one might expect if conventional stacking were used, but that is a direct consequence of the probability threshold having been set to very conservatively treat much of the sampling noise as actual changes in the scene appearance. In other words, unlike conventional stacking, even scene appearance changes that were well below the average noise level predicted by the noise model can be preserved.

Implementation of TDCI

There are many benefits to TDCI capture and one giant disadvantage: no TDCI cameras yet exist. Construction of a true TDCI capture camera requires new sensor technology. However, as was suggested earlier[3], many of the benefits of TDCI come from the representation of, and computation on, image data in a TDCI format. These benefits can be realized without the need to construct a new type of imaging sensor:

- Because the waveform is a smooth curve, it essentially interpolates higher-precision values between and across photonic samples; thus, it could provide lower noise and a higher usable dynamic range
- The virtual exposure interval to be represented by an image can be arbitrarily selected after capture as any portion of the waveform, without concern for the usual APEX exposure constraints[5]
- Movies can be rendered at any framerate without stuttering artifacts by simply selecting an appropriate virtual exposure interval for each frame
- The shutter angle can be set arbitrarily; the choice of shutter speed is decoupled from the choice of framerate

Having laid that theoretical groundwork, the remaining big question about use of TDCI with conventional cameras was simply: do all the details fit? Is it really practical to implement TDCI using data from conventional cameras? Thus, the goal of this paper is completely pragmatic: to explore the details of a complete software environment for TDCI.

TIK is an *open-source* environment and file format specification intended to support not only use by the research group developing TIK, but also real experiments with TDCI by both other researchers and ordinary users of video and still images. The name TIK is a somewhat strained mnemonic referencing Temporal Imaging from Kentucky or Temporal Image Kontainer (with container deliberately misspelled). The name also suggests passage of time in small units, which is precisely how this system organizes time-varying image data. As suggested above, TIK is both the name of a tool suite and of set of a file formats, which would normally be stored in files ending with the `.tik` suffix.

TIK, the file format(s)

One might ask why there should be a new file format associated with TDCI data; can't one of the multitude of existing file formats be used? The answer is that the new `.tik` file formats are precisely that: simple extensions of existing image file formats. There are two different sets of file formats proposed for TIK data. Work on specifying an "advanced" TIK format, extending the DNG file format[6] to represent raw TDCI data, is not yet completed. The "basic" TIK format is geared toward representation of data that has already been processed into a conventional colorspace, and it is used for all the work reported in this article.

A basic `.tik` file is essentially an extension of the widely-supported formats associated with the `Netpbm`[7] family of tools. For example, those formats are the preferred output of the `dcraw`[8] raw image converter.

In particular, the Portable Gray Map (PGM) and Portable Pixel Map (PPM) formats are extended for TDCI using TIK. Binary PGM files begin with a magic number which is the ASCII character sequence `P5`, and PPM files begin with `P6`. The header structure is very simple and also formatted as ASCII characters, allowing simple one-line comments starting with the `#` character. Hiding the necessary additional fields in comments that begin with `# TIK` provides the obvious benefit that these the extra fields are harmlessly ignored (yet faithfully preserved) by most existing tools. However, unlike nearly all other image file formats, because the header is completely expressed as ASCII characters, an ordinary text editor can be used to create and edit headers. In contrast, DNG files extend the TIFF standard, which directly provides mechanisms for adding arbitrarily complex fields – but it does not provide any simple way to manage encoding and decoding the fields that are not normally processed by the particular tool being used to operate on the file.

Why is textual encoding of the header so important for TIK? Sometimes, the header is the entire `.tik` file. For example, when processing a conventional video file as input, it is still necessary for TIK tools to know information including the framerate, shutter speed, and even time sequencing of a *rolling shutter*. Rather than devising ways to modify every existing video file format to embed that information, our approach is to simply allow a purely textual `.tik` file to provide the needed additional information about the contents of one or more files in standard video or still image formats.

For `.tik` files that directly contain any image data, the data immediately following the header is always formatted so that it begins with an image that will be recognized by any tool that understands conventional PGM/PPM files. In this way, loading a `.tik` file as a PGM/PPM will always provide a preview image. Some tools, such as `ffmpeg` and ImageMagick `display`, allow multiple PGM/PPM files (including their textual headers) to simply be catenated into a single file; our `.tik` file format allows such data to be handled directly. The TDCI compressed formats stored in `.tik` files all start *immediately after any such conventional image data*, using a single 0-valued byte as a separator to render the TDCI data invisible to tools expecting catenated PGM/PPM data.

TIK file formats

All of the information about the encoded data in a basic `.tik`-format file is specified using structured comments that each begin with "`#`", one or more spaces or tabs, the word "`TIK`", one or more spaces or tabs, and then a series of space or tab separated words. Each word is either a decimal integer ASCII number or a keyword that does not start with a digit nor negative sign. There may be a variable number of words in a structured comment, but the first word defines how the other words will be treated, and the sequence ends at the the end of the line.

The very first structured comment in a `.tik` file, normally the second line of the file, must be a version comment of the form:

`# TIK V` *version format ...*

The *version* is an 8-digit number specifying the standard compliance date of the TIK encoding. For example, `20160712` would mean that the file is formatted as specified by the TIK standard that was in effect on July 12, 2016. The *format* specifies which of different types of encoding is used in the file, and some of those types require additional parameters. At this writing, there are five different formats.

The first two formats do not actually contain any image data, but are simple textual headers describing ordinary files containing stills or videos:

- `20160721 CONVERT` *pattern numBegin numEnd* – this file specifies a *pattern* for naming one or more files, each of which holds image pixel data in any still image format that the ImageMagick `convert` tool can transform into a P6 PPM file. The pattern is taken as a format string which is used with `sprintf` and one integer parameter to produce each file name. The integer parameter first has the value *numBegin*, and is incremented by 1 each time a file is processed, ending with the last value not greater than *numEnd*. For example, "`IMG%05u.JPG 1 4`" would attempt to process the sequence of images `IMG00001.JPG`, `IMG00002.JPG`, `IMG00003.JPG`, and `IMG00004.JPG`; if any image cannot be opened, it will be skipped, but still counted against the framerate. For example, if there was no `IMG00002.JPG`, but the framerate was set as 1FPS, then the three other frames would be interpreted as spanning time intervals from 0..1s, 2..3s, and 3..4s (this behavior can be useful for processing timestamped surveillance still captures).
- `20160721 FFMPEG` *filename* – this file specifies the *filename* of a video file from which frames can be extracted using `ffmpeg`. If *filename* is omitted, the video filename is assumed to be the next argument on the `tik` command line. In this way, information about a video can be specified without needing to incorporate the information in each video file. The output from `ffmpeg` is a stream of P6 PPM files, one per frame.

The next two formats are representations of TDCI streams:

- `20160712 RGB` – the header and initial image are processed like a normal P6 PPM file. However, after a single 0 byte, the rest of the file consists of a stream of records. Each record encodes the number of pixels whose values are unchanged (within the noise model) from their previously-recorded values as a spatio-temporal span distance and a tuple of red, green, and blue (RGB) color channel values for the unexpected pixel value. The pixels within a frame are scanned either in the usual nest of increasing Y wrapping increasing X coordinates, or in a sequence determined by the timing of a rolling shutter. A span distance greater than the number of pixels in an image represents entire frames in which all pixels had expected values. If the span is 0-127, it is output as a single byte; otherwise, the 7 least-significant bits of span are output in a byte ORed with 0x80, the remaining span is shifted right by 7 bits, and the process repeated until no 1 bits remain in span. For example, a span of 257 pixels followed by an RGB value of 0x11, 0x22, 0x33 would be encoded as the five bytes: 0x02, 0x01, 0x11, 0x22, 0x33.
- `20160712 UYVYYY` – most Canon PowerShot cameras are not capable of high-framerate video capture, but using the Canon Hack Development Kit (CHDK)[9] to reprogram the camera, it is possible to record live view data at a relatively high framerate. To minimize in-camera processing, the image data is treated as a sequence of P5 PGMs with 6/4 the actual X resolution, thus allowing the native UYVYYY representation of four pixels to be used directly. This encoding is further complicated by the fact that the YUVYYY values used inside CHDK are unsigned for Y, but signed for U and V, and this is maintained in the TIK file. The U and V color component values are shared by a group of four pixels; RGB color is computed as:

```
R = min(max(((Y«12)+(V*5743)+2048)»12),0),255)
G = min(max(((Y«12)+(U*1411)+(V*2925)+2048)»12),0),255)
B = min(max(((Y«12)+(U*7258)+2048)»12),0),255)
```

Finally, the fifth format encodes a statistical model of "error" in pixel values:

- `20160804 NOISE` – an ordinary P6 PPM file in which the image encodes a noise model (see below).

TIK file metadata

The `V` structured comment is generally followed by a sequence of other structured comments that describe important attributes of the image data. All number attributes are expressed as integers to avoid roundoff issues. These include:

- `B` *number* – the time delay, in nanoseconds, to when the first image capture began. This is used to correct for synchronization delays between multiple cameras or captures.
- `E` *number* – an exposure value (EV) that can be used for approximately scaling to known luminances. This is intended to be used to find mappings of equivalent pixel values between images taken by different cameras, but is not currently precise enough to be useful for that purpose.
- `F` *number* – the frame time in nanoseconds; 1,000,000,000/FPS. Note that this is not necessarily the same as the shutter open time, but is often a somewhat longer interval.
- `G` *number* – the 1000000 * gamma by which the pixel values should be decoded. The intent is to approximately linearize values, but in practice tonal non-linearities are often more complex than can be described by a single gamma value, so this mapping may change in the future.
- `R` *number numberXdiv numberYdiv* – a method for specifying the scan order and timing of a rolling shutter. The *number* is the number of microseconds it takes for the rolling shutter to traverse the sensor, while the other values specify the order. The pixel at coordinates X,Y is sampled at a time that is *X*number/numberXdiv + Y*number/numberYdiv*. If either of *numberXdiv* or *numberYdiv* is zero, it means that dimension suffers no delays; if negative, it means the corresponding axis is scanned in reverse order (large to small).
- `T` *number* – the shutter open time in nanoseconds. The sum of this time and the last pixel's rolling shutter delay is assumed to always be no longer than the time per frame. However, the shutter open time is commonly less than the time per frame.
- `X` *number* – the X dimension (width) of the images. The units are those of pixel data blocks. For `RGB` data, each tuple counts as one unit; for `UYVYYY` data, each tuple of six values counts as one unit.
- `Y` *number* – the Y dimension (height) of the images.
- `Z` *number* – the maximum value of a color channel; the white point.

Although it would be easy to add other structured comments to provide additional metadata, our goal is to keep the basic `.tik` format as simple as possible to encourage experimentation with it. There are literally hundreds of different (and potentially useful) metadata attributes used by various cameras, but we feel that complexity is best left to definition of the "advanced" TIK format, which we anticipate being a extension of DNG.

TIK, the program

There are three fundamentally distinct operations that the `tik` program can perform:

- Creation of a `.tik` file representing the the noise model for a TDCI stream. The combination of all input files is assumed to represent a completely static scene, from which a model describing capture noise is created. The TDCI stream input may be given as any combination of movies in conventional video file formats, sequences of individual still image files, and even `.tik` files.
- Creation of a `.tik` file representing a TDCI stream. The combination of all input files is encoded in a way that allows efficient processing, but also provides significant data compression in the time domain. The TDCI stream input may be given as any combination of movies in conventional video file formats, sequences of individual still image files, and even `.tik` files.
- Rendering of virtual exposures as images or movie frames. The TDCI stream input may be given in the form of one or more `.tik` files, but the tool also will accept movies in conventional video file formats and sequences of individual still image files.

Although these functions are currently integrated into a single application because they share the code infrastructure for handling PPM files, it is easier to consider them as separate programs. They may in fact become separate programs when support for an advanced file format is added. Thus, each of these three functions is considered separately in the subsections which follow.

Creation of an error model (noise map)

In a paper at Electronic Imaging 2016[10], it was observed that increasing framerate results in a decreasing amount of additional scene appearance data when an appropriate noise model is applied to the images. However, a poor model of noise-induced value errors does not have this happy effect. Overestimating noise removes subtle shading gradations and causes temporal errors by delaying recognition of scene changes. Underestimating noise provides little compression and also fails to enhance the signal to noise ratio (a secondary benefit which comes from averaging over more samples). Unfortunately, obtaining an accurate noise model is difficult because there are many interacting factors.

The method used by `tik` to create an error model is remarkably straightforward, but can account for even relatively subtle error sources and interactions. A set of time-sequenced images, and/or video(s), is captured of a completely static scene under as similar as possible conditions to those that will be used for capture of the TDCI source material. This static scene data is essentially histogrammed to produce a set of transition probabilities for each value in each color channel.

Figure 2 is the error model that `tik` created for the same 960FPS Sony RX100 IV video that was sampled in Figure 1. In a noise map image, the X axis position is the previous pixel value and the Y axis position is the current value. Each pixel in the error model image is given RGB coloring that reflects the probability of that value transition in each color channel. The pixel value ranges and probabilities are scaled to the range 0..255, so the complete noise map is a single 256x256 P6 PPM-formatted image with three bytes per pixel. In any noise map, there generally should be a bright line from the upper left to the lower right; this simply represents the previous and current pixel values being the same. However, the noise model is made monotonically non-decreasing as the difference between values approaches zero, essentially thickening that line, although different noise in different color channels can impose color tints in the map and "smearing" of the line is not always symmetric. In the example in Figure 2,

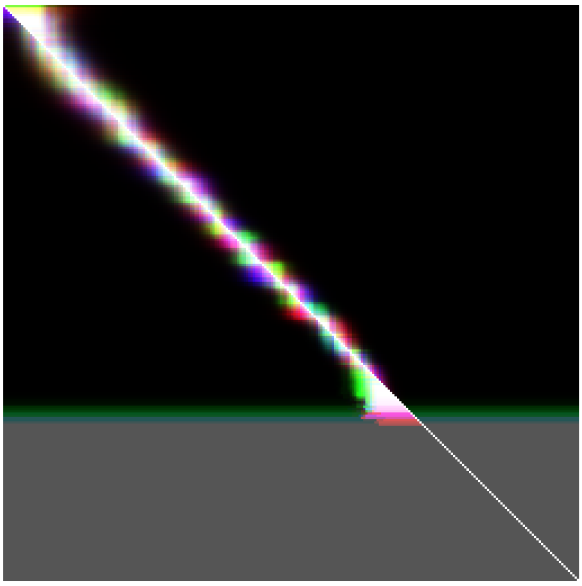

Figure 2. *Noise map computed from 960FPS Sony RX100 IV video*

we also see that the scene never actually made use of the brightest possible pixel values... in other words, the video was significantly underexposed.

It is useful to note that, although the `tik`-constructed noise model is marked as a `20160804 NOISE` file, it is actually an ordinary P6 PPM file that may be manipulated using conventional image editing software. It is even easy to construct a noise model directly if the capture noise characteristics are well understood. This simplicity and flexibility was judged to outweigh the fact that various types of correlated noise, such as an increase in one color channel's noise when another color channel has a higher value, cannot be directly represented in this type of map.

Creation of a TIK TDCI representation

Given input that is not already a TDCI format (e.g., not a `RGB` nor `UYVYYY .tik` file), the synthesis of a continuous waveform for the value of each pixel over time is relatively straightforward. However, if there are multiple image data sources, all sources are assumed to be spatially aligned. For example, two videos can only be merged by `tik` if there is a direct correspondence between the area covered by every pixel X,Y from each source. All images and video must be made to share the same point of view and pixel resolution before they can be processed by `tik`.

It should be noted that the images and/or video sequences being used to generate a single TDCI `.tik` compressed file need not have regular timing nor any other particular temporal properties. The timing-annotated pixel values extracted from still images and video frames form time intervals during which the average pixel value is known. Gaps between these sampled intervals must be interpolated in the temporal domain, but the interpolation need not be directly encoded in the `.tik` file, because it will be performed at the time that virtual exposures are extracted. However, sample overlaps (i.e., from multiple cameras sampling the same pixel worth of scene content) must be resolved into the appropriate number of pixel change records.

When `tik` must determine if a pixel's value has changed enough to warrant creation of a new value change record, it uses the specified noise model file (or a built-in generic one) to look up the conditional probability that the apparent change that occurred does not reflect anything more than the variation due to noise. The user may specify the lowest probability that should be considered noise. Only lower-probability combinations create change records. The analogy is somewhat imprecise, but the usual approximate standard deviation threshold values of 32%, 5%, and 0.3% can be applied with similar meaning.

However, `tik` also computes a type of confidence metric for each pixel value. Some devices have worse noise than others, so their noise models will lead `tik` to trust their values less than those of temporally-overlapping less-noisy sources for the same pixel value. Beyond that, even within a single video or still sequence, some pixel values represent stable averages over a large temporal sequence of samples, and those values are trusted more than ones computed from single samples. At this writing, we are still tuning the confidence metric computation.

At this writing, the current version of `tik` is 20160804, so the TDCI stream output is a `20160804 RGB` file (which is compatible with the `20160712 RGB` standard still in effect at that time). Despite that file format having been optimized for fast processing rather than maximum compression, significant compression is usually obtained. For example, the original four-second 960FPS video from the Sony RX100 IV is a 1,071,963,827 byte MP4 file. However, the `.tik` file is just 155,511,438 bytes, just 14.5% of the MP4 size – for a compression factor of nearly 6.9X. Of course, increasing the noise probability threshold generally will increase compression significantly, but with the potential side-effect of averaging-out small changes in scene appearance.

Rendering of virtual exposures

The rendering of images by `tik` from a TDCI stream (e.g., one or possibly more `.tik` files) is simply controlled by specifying the period each virtual exposure should span. This can be specified most straightforwardly as a time offset for when the exposures should begin, a frame rate, a number of frames, and a shutter speed (or shutter angle).

From that simple specification, the noise model, a model of value confidence (if there is more than one TDCI stream input), one of several possible interpolation schemes, etc. are applied to compute each image. Note that there is no need for the virtual exposures to align temporally with change records in a TDCI stream. For example, consider the images of a fast-moving pink dragon in Figures 3, 4, and 5. Despite the pink dragon moving faster than any human is likely to, these virtual exposures appear fairly natural, with credibly smooth motion blur when the rendered framerate dropped below the speed of the puppet's motion. All these images are extracted by `tik` from the same `.tik RGB` file created by `tik` from a 240FPS video capture, yet different framerates and virtual exposure intervals produced good quality even when the timing did not line-up with the 1/240s frame intervals. In fact, the 240FPS captures used a shutter speed of 1/251s, so interpolation is required even for 1/240s frames.

Figure 3. *Five sequential frames at 240FPS as captured with shutter 1/251s*

Figure 4. *Five sequential virtual exposures rendered at 24FPS, 25FPS, and 30FPS (approximately 360 degree shutter)*

Figure 5. *Five sequential virtual exposures rendered at 25FPS with shutter 1/50s, 1/100s, and 1/500s*

In general, if a TDCI stream is to be created from a single conventional video, there will be temporal gaps between the frames. These gaps might even be large; for example, a 24FPS video shot in bright daylight might use a shutter speed of 1/500s (i.e., a shutter angle of 17 degrees), meaning that less than 5% of the video's elapsed time is sensed at all. These gaps, if present in the TDCI stream, are currently interpolated by assuming that the value of a pixel linearly changes between adjacent temporal samples. Although better interpolation schemes are being investigated, linear interpolation in the time domain produces surprisingly good results when the sampling frequency is relatively high (i.e., when the original video capture was made at a framerate that is fast compared to the rate of scene change). For temporal sampling far below a Nyquist sampling of the scene change, such as the 24FPS 1/500s example mentioned earlier is likely to be, linear temporal interpolation is not sufficient to produce good quality; more sophisticated spatio-temporal interpolation would be necessary, and even that might prove insufficient.

The computation of the virtual exposure pixel values is done in an approximately linear gamma space using double-precision floating-point values. Because many pixel values are often averaged over many samples or otherwise smoothly interpolated (as constrained by a noise model), signal to noise ratio and effective dynamic range can both be much better than the original captures would suggest. Thus, the resulting virtual exposure is really a high dynamic range (HDR) image. Although we may later output virtual exposures as HDR images, currently, they are straightforwardly mapped into the dynamic range of the output file format.

Despite the various approximations made by `tik`, perhaps the most surprising aspect of TDCI virtual exposure rendering is that a virtual exposure will generally have better signal-to-noise ratio than the actual exposure representing the same time interval. This is the effect discussed earlier in the paper and shown in Figure 1.

Conclusion

This paper is intended to serve both to report preliminary experimental test results for how well TDCI concepts can be applied to conventionally-captured images and to invite to others to use, experiment with, and contribute improvements to the `tik` tool and `.tik` file format. The full source code for the system will be openly available from `aggregate.org`.

There are some examples and performance analysis in this paper, but we also intended to give live demonstrations of `tik` at Electronic Imaging 2017.

Acknowledgments

This work is supported in part under NSF Award #1422811, *CSR: Small: Computational Support for Time Domain Continuous Imaging.*

References

[1] Henry Gordon Dietz, Frameless, time domain continuous image capture, *Proc. SPIE 9022, Image Sensors and Imaging Systems 2014, 902207 (March 4, 2014); doi:10.1117/12.2040016.* (2014).

[2] Richard L. White1, David J. Helfand2, Robert H. Becker, Eilat Glikman, and Wim de Vries, Signals from the Noise: Image Stacking for Quasars in the FIRST Survey, *The Astrophysical Journal, Volume 654, Number 1; http://stacks.iop.org/0004-637X/654/i=1/a=99* (2007).

[3] Henry Gordon Dietz, Frameless representation and manipulation of image data, *Proc. SPIE 9410, Visual Information Processing and Communication VI, 94100R (March 4, 2015); doi:10.1117/12.2083468.* (2015).

[4] Deep Sky Stacker, *http://deepskystacker.free.fr/* (accessed November 26, 2016).

[5] Doug Kerr, APEX - The Additive System of Photographic Exposure, Issue 7 (August 4, 2007); *http://dougkerr.net/Pumpkin/index.htm#APEX* (accessed November 26, 2016).

[6] Adobe Systems Incorporated, Digital Negative (DNG) Specification, Version 1.4.0.0; *http://www.adobe.com/* (June 2012).

[7] Jef Poskanzer, NETPBM: Extended portable bitmap toolkit, (1993).

[8] Dave Coffin, Decoding raw digital photos in Linux, *http://www.cybercom.net/d̃coffin/dcraw/* (2016).

[9] Canon Hack Development Kit (CHDK), *http://chdk.wikia.com/wiki/CHDK* (accessed November 26, 2016).

[10] Henry Gordon Dietz, Zachary Snyder, John Fike, and Pablo Quevedo, Scene appearance change as framerate approaches infinity, *Electronic Imaging, Digital Photography and Mobile Imaging XII, pp. 1-7 (February 14, 2016);* (2016).

Author Biography

Henry (Hank) Dietz is a Professor in the Electrical and Computer Engineering Department of the University of Kentucky. He and the student co-authors of this paper, Paul Eberhart, John Fike, Katie Long, Clark Demaree, and Jong Wu, have been working to make Time Domain Continuous Image capture and processing practical. See `Aggregate.Org` for more information about their research on TDCI and a wide range of computer engineering topics.