https://doi.org/10.2352/EI.2026.38.15.COIMG-163

NUTIK: A testbed for functional post-capture manipulation of time and gain

Paul S. Eberhart, Henry Deitz; University of Kentucky; Lexington, Kentucky

Abstract

Conventional photographic imaging technologies fix the time, interval, and gain of an exposure at the instant of capture, spoiling images if these parameters are not correctly anticipated. Techniques, such as TDCI (Time-Domain Continuous Imaging) allow the integration of an image to be decoupled from the capture of scene data, creating an opportunity to not only adjust the timing and gain of an image after the fact, but manipulate those parameters in previously impossible ways. To facilitate exploration of these new dimensional freedoms, we have created NUTIK, a tool which allows scene data to be captured and computationally post-processed to expose 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. This paper documents the design and operation of NUTIK, and makes an initial exploration of useful and interesting new photographic techniques enabled by such a tool.

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 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. The publication record for TDCI technology begins in 2014 [1].

Instead of attempting to obtain a single, temporally 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).

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 [2] 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" [3]. TDCI hinges on the observation that factors such as scene constancy and photon shot noise limit the information content of those signals. An IMEV only needs to store changes in the expected value of each sensel, allowing a significantly reduced amount of *data* to still capture all the *information* provided about the evolution of the scene appearance over time as sampled by the arriving photons.

One of the most exciting features of TDCI capture is the possibility of rendering frames in ways impossible in a conventional camera, which convolves the process of capture and integration, limiting flexibility on both.

TIK

There have been a number of publications related to *implementations* of TDCI systems. The most consequential has been a simulation testbed, TIK, slightly unfortunately, 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" [4].

The TIK software and formats create a testbed for performing time domain continuous imaging using conventional still images and/or video captures and was 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.

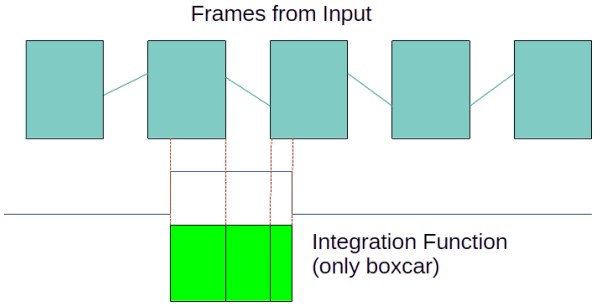


Figure 1: TIK Data Model

These three functions are performed offline, using recorded images, and taking nontrivial time and computational resources to perform the renderings. Figure 1 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.

An Early Prototype

In order to explore the possibilities of non-uniform rendering, a proof-of-concept implementation was constructed as an Octave script 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 particular, this initial prototype did not attempt to interpolate between samples, as prototypes like TIK do, resulting in somewhat coarse output images, and the simple internal data model required minutes of rendering time per output frame, impeding experimentation.

The results of this early experiment were presented in a paper at EI2020 [5]. This first experiment demonstrated the feasibility of using the TDCI paradigm to generate virtual exposures with physically unrealizable exposure parameters. This mechanism shows promise both for a variety of scientific and creative applications. 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 made the most common cases awkward. The time properties of conventional exposures look like slightly slew-limited boxcar functions: roughly 0 except during the exposure interval, which starts and stops abruptly. As indicated by an earlier study of shutter artifacts [6], these functions are not perfectly square, as the 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-like functions, while allowing more complex functions to be represented with additional human effort, or the aid of software to generate the descriptions.

The earlier prototype 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 is generally not supported. This is typically treated as a normalization problem; if the output of integration produced negative, saturated, or otherwise unpresentable values, there is a choice of how to handle the case. NUTIK, the subject of this paper, 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.

NUTIK

Based on the positive indications from this initial prototype, a second-generation non-uniform integration tool has been constructed, integrating the lessons from the Octave-based non-uniform integration prototype into the TIK tooling.

This second-generation prototype, called **N**on-**U**niform **T**IK (NUTIK), was constructed by first significantly updating, and then extending, TIK. The updates, mainly in 2022, centered on converting TIK to use the OpenCV library [7] 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 and 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 `-m`*`MaskFile`*`.pgm` and Function files are specified to nutik with the command line option `-k`*`FnFile`*`.fn`.

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 specific exposure processing function by the literal 0-255 encoding of the grey level.

This mask must be of of the same spatial resolution as the IMEV, and thus also as 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. 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 — often with biz-

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

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 integrated into the NUTIK codebase for generating exposures, and another in a support tool for plotting functions, `FnPlotter.py`, 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 [8] 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 create a `.tik` file from an input video stream, then performs function-controlled rendering of output frames from that `.tik` file. A higher frame rate for the input video stream generally results in a more precise 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` *ClipName.mp4 ClipName*`.tik`, specifying a frame-rate of $120 fps$ (`-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 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 2.

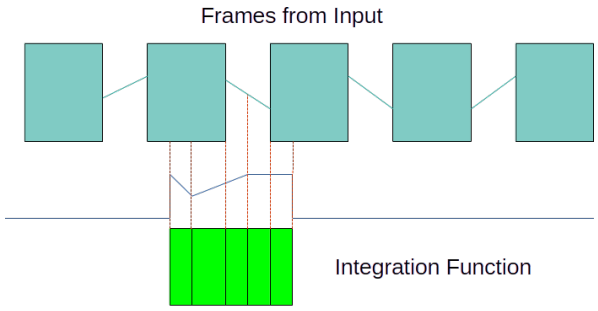


Figure 2: NUTIK Data Model

People do not generally have an intuitive understanding of the relationship between integration functions and the images they produce. 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 product 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.

Samples

The utility of the ability to perform post-capture integration with arbitrary functional control of time and gain is best shown via examples. Most of the remainder of this paper describes a small number of illustrative cases which produce desirable photographic features not practically realizable with conventional imaging technologies.

A Negative Gain

As an example of negative gain's utility for feature extraction, consider an exposure rendered from a brightly-colored plush dinosaur being shaken violently in front of a machine room window, as a stand-in for any sort of rapidly and organically moving feature. A conventional frame from this recording is shown in Figure 3.

Figure 3: A conventional exposure of a shaken orange dinosaur

To isolate motion, we can instead create a rendering which is non-uniform in time (but not space) exposed with only a single function which combines a positive $1/30s$ exposure with gain 1 surrounded by $1/60s$ exposures with gain -0.5 to either side, all occurring 1.5 seconds into the sampled interval. This exposure is described by Figure 4, which is visualized as a graph in Figure 5. 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.

```
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 4: FeatureExtraction.fn

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 6. While the total subtracted interval is less than the added interval, the gain is slightly lower, leaving a desaturated image of the static features, but strongly exaggerating the changed regions.

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

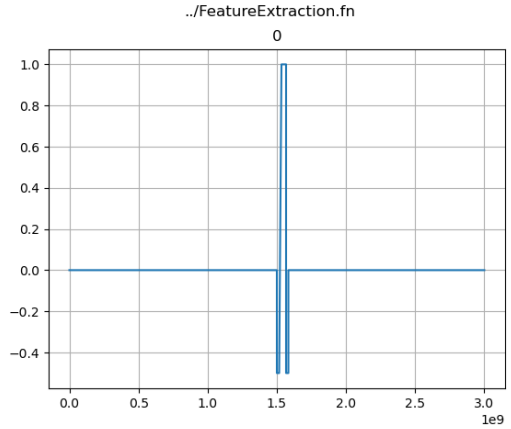


Figure 5: Visualization of FeatureExtraction.fn

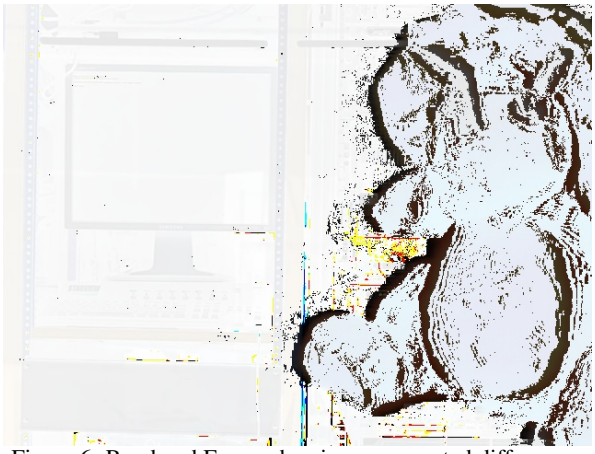


Figure 6: Rendered Frame showing exaggerated differences

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 7 and plotted in Figure 8. The result produces is the image Figure 9. This processing is much easier to implement and more flexible than attempting to synchronize a shutter or 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.

```
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 7: BallMultiple.fn

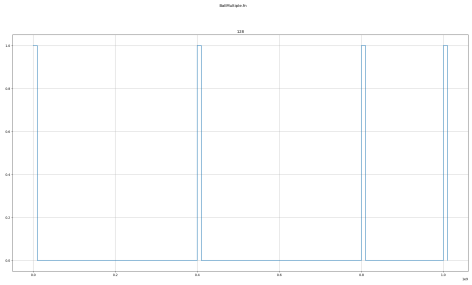


Figure 8: Plot of the function generating a virtual multiple exposure image

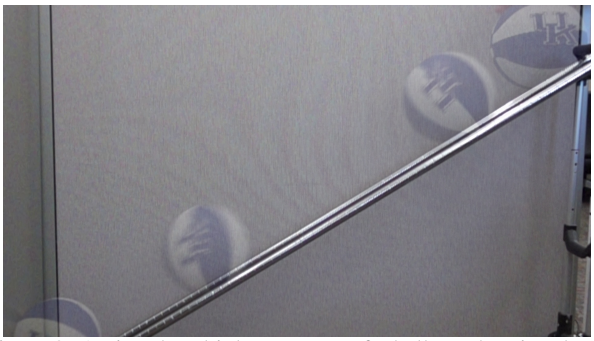


Figure 9: A virtual multiple exposure of a ball accelerating down a ramp.

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 motion blur for the entire second from the IMEV. This final image is shown in Figure 10.

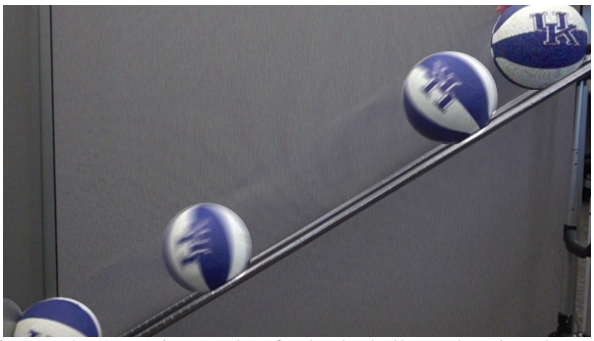


Figure 10: A motion study of a basketball accelerating down a ramp

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 11.

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

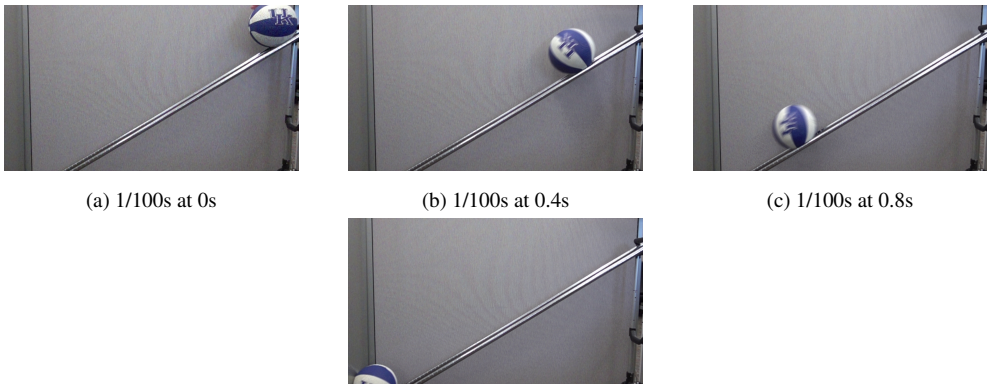
Once the four individual frames were exposed, they were loaded into an image editor, and the area occupied by the ball in each frame was selected with standard selection tools. Those selected areas were then transferred to a mask with a background of value 0, and bucket filled with tag values of 64, 128, 196, and 255 respectively to create BallMask.pgm shown in Figure 12.

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 13. Note that there is a small patch of corruption due to an unrepresentable 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 14, a plot of which generated with `FnPlotter.py BallComposite.fn` is shown in Figure 15.

Finally, NUTIK is run one more time, as `./tik -mBallMask.pgm -kBallComposite.fn BallRoll.tik`. This creates the final exposure shown in Figure 10. The final image rendered from the scene model selectively composites the fading motion trail of the exaggerated long exposure with shorter exposures from several short sub-intervals, all of which are of time and duration chosen interactively after the time of capture, to create the final motion study.

The authors are 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.



(a) 1/100s at 0s (b) 1/100s at 0.4s (c) 1/100s at 0.8s

(d) 1/100s at 1s

Figure 11: Virtual exposures rendered in the process of designing this motion study

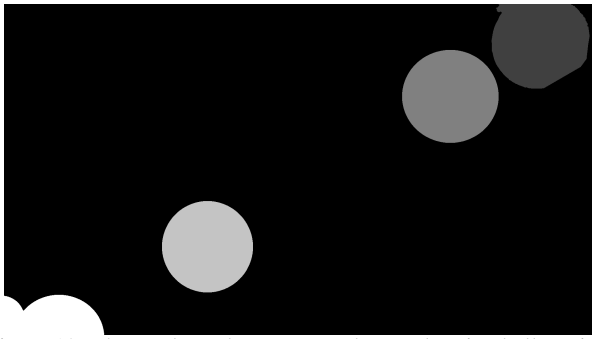


Figure 12: The mask used to generate the accelerating ball motion study

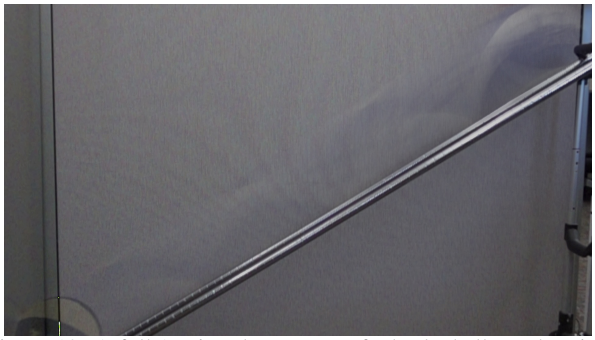


Figure 13: A full 1s virtual exposure of a basketball accelerating down a ramp

```
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 14: BallComposite.fn

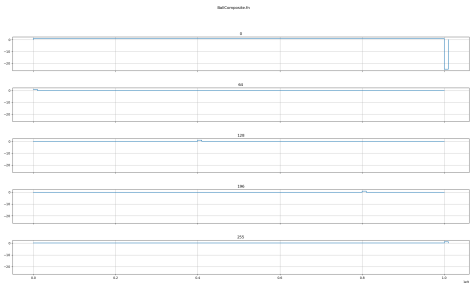


Figure 15: Plot of the function specification for the motion study

```
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 16: BallCompositeOneBlur.fn

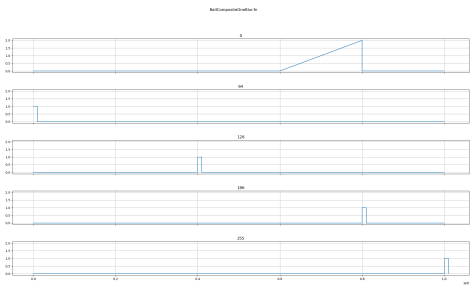


Figure 17: Plot of functions to produce blur only leading to one ball position

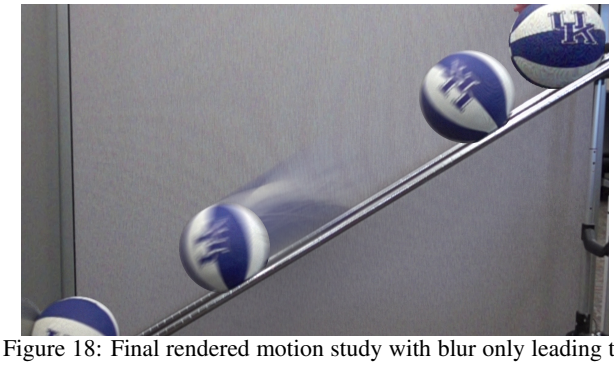


Figure 18: Final rendered motion study with blur only leading to one position

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 16, plotted in Figure 17 resulting in the image shown in Figure 18.

Conclusion

The current work has demonstrated that an imaging system which decouples capture from integration, and allows arbitrary functional manipulation of the time and gain dimensions of the integration process, is both possible and compelling. Extended coverage of the surrounding work can be found in [9]. The latest source and updates about this and related imaging work are published online at `https://aggregate.org/DIT/`.

A number of other related projects are suggested or ongoing. One obvious future direction is the addition of a GUI layer for the integration process. While the absolute flexibility and control of masks constructed in ones' image editor of choice and arbitrary functions specified control point by control point is exciting, a layer of simplifying automation would make it much easier to obtain desired results in all but the most obscure cases. A front-end supporting features like specifying functions by graphically selecting and dragging control points, automatically rendering a preview image in response to adjustments to masks and functions, and the ability to scrub a boxcar of specified width across the IMEV interval to easily locate events of interest would all make the process much more accessible. In general, automating the process of adjusting parameters, rendering, evaluating the result, and repeating to obtain the desired image will be necessary to transform a powerful demonstrator into a useful tool.

In parallel, there are several efforts to construct IMEV-native cameras, including experiments with LED-sensels, underway. These offer opportunities to produce better IMEV models, and do so with much more efficient use of bandwidth and storage.

The rendering infrastructure is also subject to improvement. While NUTIK's performance is not a serious bottleneck for experiments, NUTIK does not currently use any parallel execution. Most of the algorithms and data structures are amenable to massively-parallel execution, so as the interest shifts from designing and altering the exposure mechanisms to user focused application, it is likely performance can be readily improved further via relatively straightforward parallelization.

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. 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.

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

[4] 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.

[5] 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.

[6] 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.

[7] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

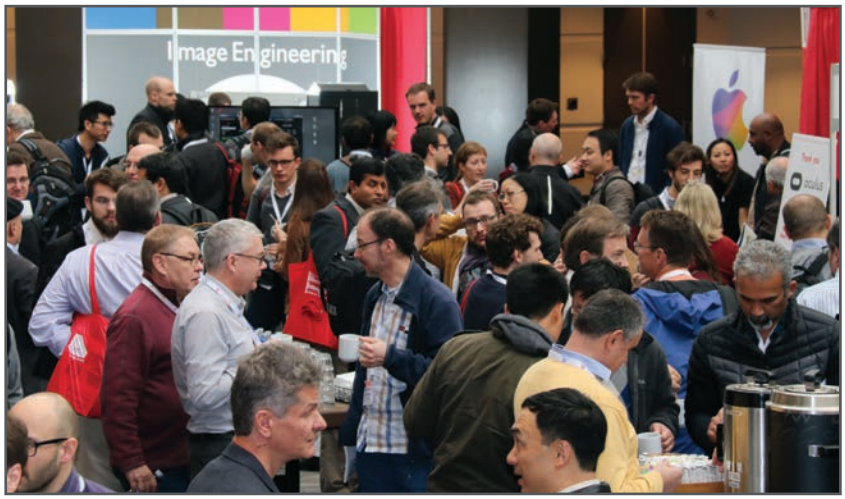
[8] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007. DOI: 10.1109/MCSE.2007.55.

[9] P. Eberhart, "Post-capture synthesis of images using manipulable integration functions," Ph.D. dissertation, University of Kentucky, Sep. 2024. [Online]. Available: https://uknowledge.uky.edu/cs_etds/146/.

JOIN US AT THE NEXT EI!

electronic IMAGING

Imaging across applications . . . Where industry and academia meet!



• SHORT COURSES • EXHIBITS • DEMONSTRATION SESSION • PLENARY TALKS •
• INTERACTIVE PAPER SESSION • SPECIAL EVENTS • TECHNICAL SESSIONS •

www.electronicimaging.org

ADVANCING IMAGING SCIENCE 75 1947 IS&T 2022 imaging.org