EE66110 Segment B CA: Simulating Event Videos from High-FPS Videos

Lai, Yan Kai
Department of Electrical and Computer Engineering, National University of Singapore
Singapore
Email: lai.yankai@u.nus.edu

***Abstract*—This work attempts to create event videos depicting asynchronous events generated from normal, high FPS videos. Concepts are adapted from v2e and improved upon in the implementation. By comparing the current work and an existing v2e implementation, we show that the generation of asynchronous events and their interpolation is better (probably) than v2e. Most significantly, a 960 FPS video can be converted to 5000 FPS event video with smoother interpolation of events as compared to the v2e implementation.**

I. Introduction

The work explores ways to generate asynchronous events from high FPS videos, and then produce a higher FPS video depicting these asynchronous events. The work adapts concepts from v2e, and provides improvements to give slightly better generation of asynchronous events. Section II explains the concepts related to simulating the DVS sensors in event cameras. Section III describes the codes and functions designed to implement these concepts. Section VI describes observations and attempts to explain them.

II. Problem Formulation

The derivation of the physical model is largely identical to v2e. A video is first converted to a grayscale value to get intensity values, to be conveniently called luma. The grayscaling is done by MATLAB's implementation of ITU-R Recommendation BT.601-7. This gray scaling standard is used by other sources [1]–[3].

The event camera consists of DVS sensors on every pixel that tracks fold-changes in luma. When the fold-changes exceeds a threshold, each pixel is triggered with a reset event. An ON reset event triggers in a pixel when the fold-changes are positive, and OFF when negative. There is some latency due to the capacitors in the sensors.

Each pixel sensor can be approximately modelled as containing comparators after a low pass filter. The comparators measure the low pass filter's output voltage L_{lp} against the voltage L_m across a capacitor. If the difference $|L_l p - L_m|$ between the output voltage and voltage across the capacitor exceeds a threshold θ, a reset event triggers. When this occurs, L_m stores L_{lp}, which is approximately equivalent to adding L_m with θ if ON and subtracting with θ if OFF.

A. Low-pass Filter Modelling

L_{lp} can be approximately obtained from the low-pass, infinite-impulse response (IIR) model of the sensor. The sensor is similar to a second-order system with a dominant pole. For low luma values, it behaves like an IIR filter. For higher luma values, resonant peaking occurs, but flattens out for even higher luma values. Hence, it can be simply approximated as an IIR filter. The -3dB bandwidth f_{BW} of this filter increases with luma. The behavior is plotted in [4]Fig. 9b. The low pass filtering explains most of the latency seen in DVS sensors [1], [4].

Like v2e [1], $f_{\text{BW}} = 200Hz$ is chosen for luma values of 255, which decreases to $c_{\text{BW}} f_{\text{BW}} = 20$ for luma values of 0, where $c_{\text{BW}} = 0.1$. The latter is necessary to avoid zero bandwidth, which means all input luma are fully attenuated.

The discrete modelling of the low pass filter may be adapted from a simple IIR filter – the RC low-pass filter. This leads to the following equations and discretisation:

$$L_{\text{lp}}(f) = L_{\text{in}}(f) - Ri(f) \tag{1}$$

$$= L_{\text{in}}(f) - R\frac{dCL_{\text{lp}}(t)}{dt} \tag{2}$$

$$\approx L_{\text{in}}(f) - RC\frac{L_{\text{lp}}(f) - L_{\text{lp}}(f-1)}{t_f} \tag{3}$$

where f represents the frame number, and the time $t = ft_f$ where $t_f = 1/f_f$ is the reciprocal of the true FPS f_f. This can the be rearranged as the following weighted average recurrence:

$$L_{\text{lp}}(f)[t_f + RC] = L_{\text{in}}(f)t_f + RCL_{\text{lp}}(f-1) \tag{4}$$

$$L_{\text{lp}}(f) = L_{\text{in}}(f)\frac{t_f}{t_f + RC} + L_{\text{lp}}(f-1)\frac{RC}{t_f + RC} \tag{5}$$

$$= pL_{\text{in}}(f) + (1-p)L_{\text{lp}}(f-1) \tag{6}$$

where p is the forgetting factor. Since $f_{\text{BW}} = (2\pi RC)^{-1}$ for this simple filter, p can be derived as:

$$p = \frac{t_f}{RC + t_f} = \frac{t_f}{(2\pi f_{\text{BW}})^{-1} + t_f} = \frac{2\pi f_{\text{BW}}}{2\pi f_{\text{BW}} + f_f} \tag{7}$$

B. Linear-log Mapping of Luma

The photodiode converts the luma to a voltage output L_{in} that is logarithmic in the luma [4]. However, since we do not have the true luma in a picture due to a limited dynamic range and other camera settings, a pixel intensity of 0 may not be interpreted as infinitely dark, and an intensity of 255 may not be interpreted as infinitely bright. This conveniently does not

disallow an arbitrary conversion of $L_{\text{in}} = \log(v)$, where v is the pixel value [4].

Since a log near zero will result in large differences between the logs of adjacent pixel values, a linear model of up to a pixel value of 20 may be chosen for the mapping [1], [5]. This is likewise conveniently possible from the previous assumption.

The mapping for the first 40 pixel values are plotted in Fig. 1

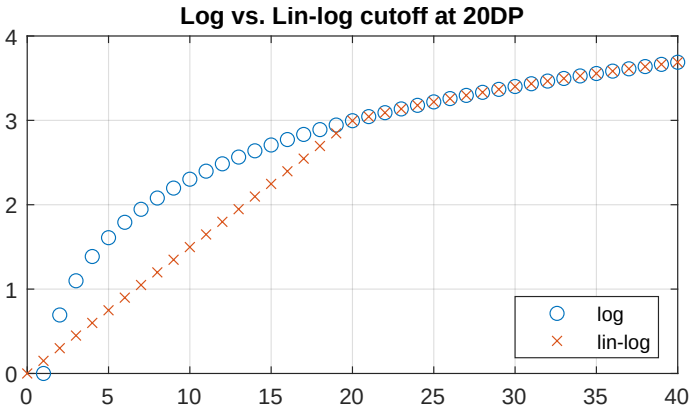

Fig. 1: Linear-log mapping with a cutoff at 20

C. Thresholding and Hot Pixels

The electrical characteristics for each sensor will inevitably differ due to manufacturing mismatches. The reasons behind these mismatches can be found in [4]. These results in a variation of about 2 to 3% in threshold values across all pixels. Like v2e, the variation can be modelled as a Gaussian with a mean $\theta^* = 0.3$ and a standard deviation $\sigma_\theta = 0.03$ [1], [4]. The Gaussian model is used to find every pixel's θ at the beginning of the simulation and it remains constant throughout.

Hot pixels arise from very small θ due to the mismatches. They are termed as such because they trigger frequently to small changes in log intensity, so are highly active.

D. Shot Noise

Shot noise arises due to the quantum nature of light in the photoelectric effect. In a dimly lit setting, the changes in log luma becomes more discrete to reflect the change in intensity due to a single photon. This results in abrupt ON or OFF events in dimly lit settings, but much less so in bright settings where changes in log luma from a photon are exponentially smaller. These abrupt ON or OFF events are known as shot noise.

This was claimed to occur at 1Hz in dark settings in [1], decreasing to around 0.25Hz in bright settings. In [1], the former occurs at luma of 0, and the latter at 255, with other values being linearly distributed between. This frequency is then multiplied by t_f to model an approximate probability p of an OFF event occuring between the frames. A uniform random sampler then samples a value r between 0 and 1. If $r < p$, the simulator generates a spontaneous OFF event, setting $L_{\text{lp}}(f)$ and $L_m(f)$ at $L_m(f-1) - \theta$. If $r > 1 - p$, the simulator generates an ON event, setting $L_{\text{lp}}(f)$ and $L_m(f)$ at $L_m(f-1) + \theta$.

Unlike v2e, this work additionally sets this event **at a random time between the current and previous frame to mimic the asynchronous nature**. The time is obtained from another uniform random sampler between the times $(f-1)t_f$ and ft_f.

It is not clear how v2e implements the shot noise. In this work, shot noise is simulated only when no reset occurs, i.e. $|L_{\text{lp}} - L_m| < \theta$. This is because it is a result of the quantised nature of light, which is highly unlikely to occur when the log difference is high and results in resets. Implementing this also requires checks for fold changes after the event. Moreover, the FPS is very high, so the resulting probability is very small. Hence, there is no practical impetus to implement it when reset alrady occurs.

E. Leak Events

A leak current occurs on the circuit memorising L_m. The leak rate f_{leak} is approximately 0.1 Hz, causing L_m to decrease over time [1], [6]. From [6]Eq. 11, the decrease is modelled as an increase of L_{lp} instead:

$$L'_{\text{lp}}(f) \approx L_{\text{lp}}(f) + \frac{f_{\text{leak}}}{\theta_{\text{ON}}} t_f \tag{8}$$

which is unlike v2e, where the leak is implemented over L_m. Eq. (8) is easier to implement because $L_m em$ does not change other than during resets, resulting in a simpler derivation for the interpolation of events later.

F. Interpolation of Events

The first frame is $f = 0$ and all frames have timestamps given as $t = ft_f$. The first frame is used to initialise $L_{\text{lp}}(0)$ and $L_m(0)$ of all pixels, so for each pixel, both have the same value at $t = 0$. This is to prevent an incorrect event avalanche in the first few frames.

To simulate asynchronous events, we first find the number of events as:

$$N_e = \left\lfloor \frac{L_{\text{lp}}(f) - L_m(f-1)}{\theta} \right\rfloor \tag{9}$$

where $N_e > 0$ for ON events and $N_e < 0$ for OFF events.

Consider a similar interpolation to [3] in Fig. 2. Between frame $f-2$ to $f-1$, notice that $L'_{\text{lp}}(f-2)$ (diamond) does not intersect $L_m(f-2)$ (cross) when changing. Between frame $f-1$ to f, $L'_{\text{lp}}(f-1)$ intersects $L_m(f-1)$. Eq. (9) stands for both situations.

The events are then linearly distributed between $L'_{\text{lp}}(f-1)$ and $L'_{\text{lp}}(f)$. Since a large change occurs, we can simply find a non-zero gradient of the linear distribution as:

$$g(f) = \frac{L'_{\text{lp}}(f) - L'_{\text{lp}}(f-1)}{t_f} \neq 0 \tag{10}$$

Rounding errors can produce situations downstream that result in zero gradients, so additional thresholded floating point

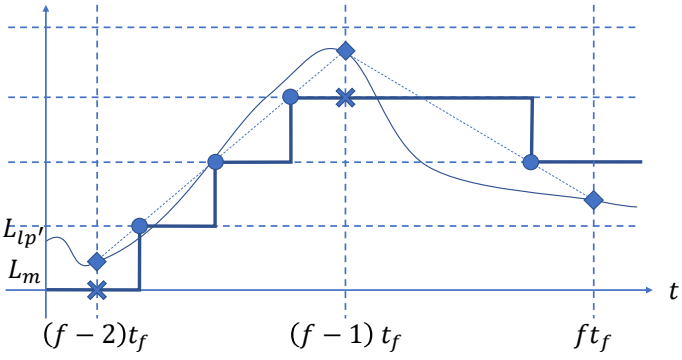

Fig. 2: Time interpolation of events

checks need to be carefully implemented. Using this gradient, we find the time of reset as:

$$t = (f-1)t_f + \frac{L_m(f-1) + e\theta}{g(f)} \quad \forall e \in \text{sign}(N_e) \times \{1, 2, \cdots, |N_e|\} \tag{11}$$

which explains why it is important for $g(f) \neq 0$.

Strangely, v2e does not appear to implement the interpolation correctly, contrary to claims in [1]. The events occur at regular intervals between frames, and may not be evenly distributed, for example, occuring at the $0.25t_f$ and $0.75t_f$ mark from $(f-1)t_f$. The implementation was designed by the authors in [7].

When implemented correctly, one should see a constant stream of events due to noise due to the asynchronous nature. However, v2e events can be shown to occur at fixed intervals. Evidence can be found in the supplementary material `data/lego_comb_clean_interp.mp4`. The left video is v2e. The right is from the current work.

G. No Slow-motion Interpolation

Unlike v2e, no slow-motion interpolation is implemented in this work. This may explain the fixed interval problem in the interpolation – the frame rate may be high enough to not bother about such linear distributions.

III. Methodology

A. v2e Implementation

The work is compared to a Python implementation of v2e on Google Colab at [7]. The relevant output are stored in `.h5` files in the folder `data`.

The configurable parameters are equivalent to `ThetaMean`, `LeakNoiseRate`, `ShotNoiseRate` and `LPFMaxCutoff` from Table I. The other parameters are not configurable but are assumed to be those values in the table, as these are taken directly from its paper [1]. These values for the parameters can be found in Table I.

A *clean* setting is possible, where there is no low pass filter, no noises and no variation in θ. The last three parameters can be set at zero to trigger their *clean* settings.

B. Current work (CA)

This section contains details about the code and implementation.

1) Initialisations: The script `init` is called at every important segment. Add paths, clear all variables, close all windows, clear the command window, and load the colormap.

2) Latex: The method `latexfig` is used for opening LaTeXfriendly figures in Matlab. `latexexp` exports these figures to PDF.

3) Preprocess Super Slowmo Video: The Samsung Galaxy S21+ "Super Slowmo" Video records videos for 2.5s, with the centre 0.5s being recorded in a burst of 960FPS and the rest in 30FPS. The result is played back in 30 FPS, so the video becomes 18s long, with the middle 16s the original 480 frames from the burst. The function `parse_ssm_video` loads and trims the first and last 30 frames, crops it, resizes the cropped result, and applies adaptive contrast if opted.

4) Get V2E Events: Used for loading H5 files exported from the authors' Python implementation on Google Colab on [7], and extract the asynchronous $4 \times N_e$ matrix, where N_e is the number of events. It outputs an `EventInfo` data structure containing options passed to the function in `Options` and the matrix in the `Async` properties.

5) Get CA Events: CA stands for "continuous assessment", and refers to the simulation created for this project. This simulation is implemented in the method `generate_events`. It accepts simulation options in a data structure `EventOptions`. The parameters are shown in Table I.

The function then outputs an `EventInfo` data structure containing the options passed to the function in `Options` and the asynchronous event matrix in the `Async` properties. Floating point problems during logical comparisonns were fixed in the code to prevent downstream calculation issues.

The first frame in the original frame is used to initialise $\mathbf{L}_{\text{m}}$ and $\mathbf{L}_{\text{lp}}$ at $t = 0$. This is to prevent an avalanche of events at the start of the simulation.

The simulation may be selected for a *clean* setting, where low pass filter is removed so no attenuation and latency occurs. There are also no variation in θ and no noises, like the v2e clean setting. Hence, parameters after but not including `ThetaMean` in Table I are not used when `Noisy` is set to `false`.

6) Generate Grayscale Event Video: The function `generate_event_video` takes the `EventInfo` data structure output by `get_v2e_events` or `generate_events` as input. It gathers events that fall within the frame duration `outputFrameDuration` specified by the user in milliseconds. If a user chooses a value of t_o ms, all events that lie between $(f-1)t_o$ms and ft_oms will be displayed on the f frame, where $f \in 0, 1, \cdots$. If 0 was chosen, it derives the duration from the FPS from `EventInfo.Options.FPS` property, which are filled by the two aforementioned functions. The last frame may be empty due to rounding errors.

TABLE I: *EventOptions* parameters, for use in methods *generate_events* and *generate_event_video*

Parameter	Symbol	Default	Description
`InputPath`	–	""	The path to the source video.
`FPS`	f_f	960	The original recorded frames per second.
`LinLogCutoff`	–	20	The luma value where the linear model changes to the log model.
`ThetaMean`	θ^*	0.3	The nominal, ON and OFF threshold magnitude across all pixels.
`ThetaStd`	σ_θ	0.03	The standard deviation which the nominal threshold value varies across all pixels.
`ThetaMin`	θ_{min}	0.01	The minimum threshold value the Gaussian model can vary the threshold value to.
`ShotNoiseRate`	$f_{\text{shot},0}$	1	The expected frequency that a reset occurs over a pixel when its luma value is 0.
`ShotNoiseReduction`	c_{shot}	0.25	Multiply with `ShotNoiseRate` to get the expected reset frequency for a luma of 255.
`LeakNoiseRate`	f_{leak}	0.1	The rate at which leak current occurs over the reset junction.
`LPFMaxCutoff`	f_{BW}	200	The -3dB bandwidth for the infinite-response filter for each pixel when its luma value is 255.
`LPFMinCutoffFraction`	c_{BW}	0.1	Multiply this with `LPFMaxCutoff` to get the -3dB bandwidth for a luma of 0.
`Noisy`	–	`true`	True for noisy simulation, false for clean. Clean ignores all parameters after `ThetaMean`.

The grayscale intensity is affected by the number of events that are gathered on every frame. Each OFF event is counted as -1, while ON event is counted as 1. If 2 ON and 3 OFF events are gathered on the same pixel and frame, the sum of the counts are $\Sigma_{(x,y,f)} = 2 - 3 = -1$. This carries on for all other pixels in every frame. The output video is a 3D matrix V of size $N_x \times N_Y \times N_f$ where N_x is the height or number of pixels along x-axis, N_y is the width and N_f is the number of frames. The largest magnitude of the sum across all frames and pixels, Σ_{max} is used to scale the entire video to the interval $[0, 1]$. This is done by dividing V with $2\Sigma_{\text{max}}$ and then adding 0.5.

Because Σ_{max} may not occur frequently, it is helpful to increase the contrast by multiplying it with a `outputContrast`, c, before adding 0.5. All pixel values are then trimmed to the interval $[0, 1]$.

Hence, the output video is

$$V_{\text{gray}} = \max\left(\min\left(\frac{Vc}{2\Sigma_{\text{max}}} + 0.5, 1\right), 0\right) \tag{12}$$

Thus, pixels with no events will have values of 0.5. Pixels with ON events will be larger than 0.5, while OFF events will be smaller. The larger the value deviates from 0.5, the higher the number of corresponding events. The output matrix are `double` values between 0 and 1.

7) Color Grayscale Event Video with Colormap: The method `color_event_video` takes V_{gray} as `Vin` and converts it into a 4D matrix V_{col} of size $N_x \times N_y \times 3 \times N_f$ depicting the colored video. The video is colored by a $N_{\text{col}} \times 3$ colormap matrix `customColormap` identical to MATLAB's definition. A custom colormap `eventVideoColormap` was designed to color ON events red and OFF events blue. The color becomes redder or bluer if there are more of the corresponding events. The output matrix are now integers between 0 and 255.

8) Video Export: The method `export_video` exports either V_{gray} or V_{col} into a 30FPS MPEG-4 video.

IV. Input Videos

The "Super Slowmo" setting captures videos at 960FPS for 0.5s. For fairer comparison with v2e, *dot* was chosen as it was used by $v2e$. Its low variety of intensities also provides a good comparison against the noisy backgrounds of the other samples. *faucet_dark* was darkened from *faucet* to test situations with dark settings.

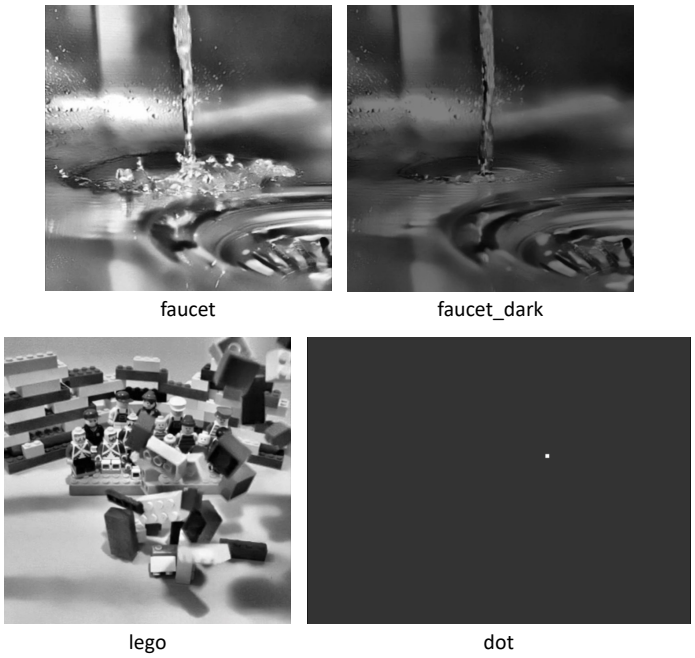

Fig. 3: All videos except *dot* were recorded from a Samsung Galaxy S21+ with the "Super Slowmo" setting. *faucet_dark* is obtained from *faucet* by darkening it (half the pixel values). *dot* is box-moving-2 from [8]

V. Matlab Script

For each video, a script is created to simulate in Matlab and export the v2e videos. They are `faucet.m`, `faucet_dark.m`, `lego.m` and `dot.m`.

Each script produces the v2e videos from its h5 files for both *clean* and *noisy* settings. The `.mp4` videos are suffixed `_v2e_clean` and `_v2e` respectively. The prefixes are the video names. The number of events are also counted.

The clean and noisy videos are also generated using the current work. These values for the parameters can be found in Table I, and are identical to those used for the v2e implementation. The videos are suffixed `_ca_clean` and `_ca` respectively. The number of events are counted.

After that, the source video, v2e and simulated videos are combined for side-by-side comparison. The combined clean videos are suffixed `_comb_clean` while the noisy are suffixed `_comb`.

VI. Results and Discussion

The number of events for each video and setting is shown in Table II

TABLE II: The number of events for each method and setting. *CA* refers to the current work. Noisy settings result in a difference of about few thousand events for both methods in different runs. Clean settings result in more or less consistent number of events for v2e, and exactly consistent for CA

Method	Video	Parameters	Number of Events
v2e	dot	Noisy	302 000
CA	dot	Noisy	214 000
v2e	dot	Clean	50 804
CA	dot	Clean	105 992
v2e	faucet	Noisy	1 465 000
CA	faucet	Noisy	1 063 000
v2e	faucet	Clean	1 976 460
CA	faucet	Clean	2 439 871
v2e	faucet_dark	Noisy	326 000
CA	faucet_dark	Noisy	922 000
v2e	faucet_dark	Clean	2 199 386
CA	faucet_dark	Clean	2 721 327
v2e	lego	Noisy	2 649 000
CA	lego	Noisy	2 394 000
v2e	lego	Clean	3 436 129
CA	lego	Clean	4 318 542

The events vary in a similar manner for both methods, so the current method appears to be correct. Fpr example, less events are generated for *faucet_dark* than *faucet*, which is expected for a smaller f_{BW} for darker settings when noisy parameters are used, leading to attenuation of log luma changes.

In contrast, the number of clean events are consistent for both videos, which show the effectiveness of the noisy parameters. However, v2e generates much less events for noisy *faucet_dark*, and may be a result of a wrong interpretation of v2e on my part.

v2e also consistently generates less events for Clean, which may be due to implementation issues arising from the aforementioned regular spread of events (in that some events may be discarded).

Fig. 4: Zoomed-in screen grabs for *dot* for v2e (left) and CA(right) on clean settings.

Fig. 4 appears to show a more sensitive threshold change, or some latency effect that was simulated. By verifying against a run made by setting `ThetaMean = 1`, the tail disappears in v2e. The video is shown in `dot_v2e_clean_thresh1.mp4`. Therefore, it may be concluded that the v2e implementation is correct, but appears to implement a smaller version of θ than the current work. However, this fails to explain why there are consistently less results for v2e. Upon further investigation, the pixels for the *dot* changes by more than 1, so the reset events should clearly define the square even in `dot_v2e_clean_thresh1.mp4`. Hence, it may be an unknown bug in this v2e implementation that is affecting its performance.

A. Asynchronous Interpolation of Events

As mentioned, the time interpolation in v2e is regular and not asynchronous, resulting in events that occur asynchronously. This can be easily verified by setting the `outputFrameDuration` to 0.2 instead of 0 for both the v2e video generation and simulation generation.

For example, in `lego.m`, first set line 151 to `data/lego_ca_clean_interp.mp4`, lines 109 and 152 to 0.2 (milliseconds, so 5000 FPS), and line 108 to `data/lego_v2e_clean_interp.mp4`. Then, run lines 103 to 175. After that, uncomment lines 259 to 293 (end), and execute them.

The left video shows the v2e events and the right video shows the events from the current work at 5000FPS. **The asynchronous nature of the events from the current work enables a better interpolation of events at high FPS**. The video is saved as `lego_ca_clean_interp.mp4`. The noises also do not occur regularly but randomly. These thus show the correctness of the asynchronous design and interpolation.

Artifacts still exist, which can be managed with optical flow calculations but is not included since this is a CA. For example, the events may stretch across the last matching pixel instead of just the existing pixel to avoid 'jumping' of objects.

B. Consistency of Clean Events

In a clean setting, an ideal simulation is run with no latency and noises. Hence, the number of events should be consistent. However, this is not consistent for v2e videos, which differ by a few hundred out of about a million events. Even though a specific Nvidia device is requested in Colab, different backend hardware may still result in the small discrepancy. For the current work, the number of events counted are consistent across different runs as the same computer is used.

VII. Conclusion

This work created an event video simulator that converts normal, high FPS videos to simulated asynchronous events and videos depicting these events. The concepts are identical to v2e, but with several improvements to the implementation. The implementation steps are listed in detail in this work.

Results show that it has better interpolation of asynchronous events than the implementation of v2e. This results in better videos depicting these events at high FPS of around 5000.

REFERENCES

[1] Y. Hu, S.-C. Liu, and T. Delbruck, "V2e: From video frames to realistic dvs events," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 1312–1321.

[2] H. Rebecq, D. Gehrig, and D. Scaramuzza, "Esim: an open event camera simulator," in *Conference on Robot Learning*. PMLR, 2018, pp. 969–982.

[3] E. Mueggler, H. Rebecq, G. Gallego, T. Delbruck, and D. Scaramuzza, "The event-camera dataset and simulator: Event-based data for pose estimation, visual odometry, and slam," *The International Journal of Robotics Research*, vol. 36, no. 2, pp. 142–149, 2017.

[4] P. Lichtsteiner, C. Posch, and T. Delbruck, "A 128x 128 120 db 15 us latency asynchronous temporal contrast vision sensor," *IEEE Journal of Solid-State Circuits*, vol. 43, no. 2, pp. 566–576, 2008.

[5] M. L. Katz, K. Nikolic, and T. Delbruck, "Live demonstration: Behavioural emulation of event-based vision sensors," in *2012 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2012, pp. 736–740.

[6] Y. Nozaki and T. Delbruck, "Temperature and parasitic photocurrent effects in dynamic vision sensors," *IEEE Transactions on Electron Devices*, vol. 64, no. 8, pp. 3239–3245, 2017.

[7] "v2e tutorial," (Date last accessed 1-Oct-2021). [Online]. Available: https://colab.research.google.com/drive/1czx-GJnx-UkhFVBbfoACLVZs8cYlcr_M?usp=sharing

[8] "v2e," (Date last accessed 1-Oct-2021). [Online]. Available: https://sites.google.com/view/video2events/home