

ESIM: an Open Event Camera Simulator

Henri Rebecq, Daniel Gehrig, Davide Scaramuzza

Robotics and Perception Group
Depts. Informatics and Neuroinformatics
University of Zurich and ETH Zurich

Abstract:

Event cameras are revolutionary sensors that work radically differently from standard cameras. Instead of capturing intensity images at a fixed rate, event cameras measure changes of intensity *asynchronously*, in the form of a stream of *events*, which encode per-pixel brightness changes. In the last few years, their outstanding properties (asynchronous sensing, no motion blur, high dynamic range) have led to exciting vision applications, with very low-latency and high robustness. However, these sensors are still scarce and expensive to get, slowing down progress of the research community. To address these issues, there is a huge demand for cheap, high-quality synthetic, labeled event for algorithm prototyping, deep learning and algorithm benchmarking. The development of such a simulator, however, is not trivial since event cameras work fundamentally differently from frame-based cameras. We present the first event camera simulator that can generate a large amount of reliable event data. The key component of our simulator is a theoretically sound, adaptive rendering scheme that only samples frames when necessary, through a tight coupling between the rendering engine and the event simulator. We release an open source implementation of our simulator.

We release ESIM as open source: <http://rpg.ifi.uzh.ch/esim>.

1 Introduction

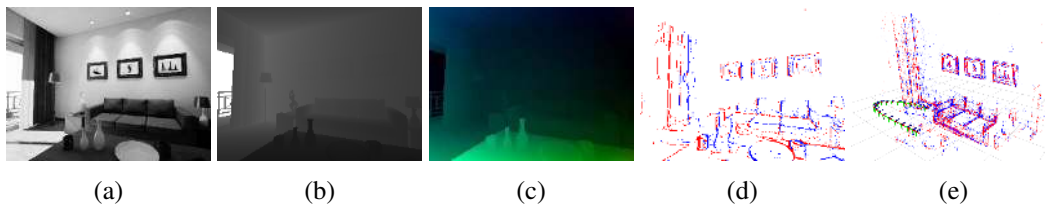


Figure 1: An example of the output of our simulator in a room environment. Our simulator also provides inertial measurements (not shown here). (a) Image. (b) Depth map. (c) Motion field (Color-coded). (d) Events (Positive and Negative). (e) Point cloud (3D events) + Camera trajectory

Vision sensors, such as cameras, have become the sensor of choice for robotic perception. Yet, despite numerous remarkable achievements in the last few years, our robots are still limited by their sensing capabilities. In particular, low-latency perception, HDR (High Dynamic Range) scenes, and motion blur remain a profound issue for perception systems that rely on standard cameras. Event cameras [1] are novel sensors that work fundamentally differently from standard cameras. Instead of capturing intensity images *synchronously*, i.e. at regular time intervals, event cameras measure changes of intensity *asynchronously*, in the form of a stream of *events*, which encode per-pixel brightness changes, with very low latency (1 microsecond). Because of their outstanding properties (low latency, HDR, no motion blur), event cameras promise to unlock robust and high-speed perception in situations that are currently not accessible to standard cameras: for example, balancing a pen [2], tracking features in the blind time between two frames [3], or flying a quadcopter in a dark room [4].

Research on event cameras is still in its infancy and we believe that the progress in this area has been slowed down by several issues. First, event cameras are rare and expensive sensors: to this date, there is only one company selling event cameras ¹ and a recent DAVIS346 sensor [5] costs six thousand dollars, which only a tiny fraction of research groups can afford. Second, commercially available event camera hardware is still at the prototype level and suffers from many practical limitations, such as low resolution (346×260) for the DAVIS346), poor signal-to-noise ratio and complex sensor configuration, which requires expert knowledge. These issues make it difficult to understand the true potential of event cameras for robotics.

In parallel, the rise of deep learning has led to an explosion of the demand for data. To meet this demand, an unprecedented number of simulators (CARLA [6], Microsoft Airsim [7], UnrealCV [8]) has been proposed for standard cameras. Yet, the community is still missing an accurate, efficient, and scalable event camera simulator. Building such a simulator is not trivial because event cameras work fundamentally differently from conventional cameras. While standard camera samples the (time-varying) visual signal at a discrete, fixed sampling rate (in a simulator, this corresponds to *rendering* an image), event cameras perform some form of level-crossing sampling: an event is generated when the signal itself reaches some intensity bounds. As a consequence, the amount of data to be simulated is proportional to the amount of motion in the scene. This calls for a novel simulator architecture to accurately simulate this varying data rate, which is precisely the goal of this work. Specifically, our contributions are:

- a novel event camera simulator architecture that tightly couples the event simulator and the rendering engine to allow for an accurate simulation of events through a novel, adaptive sampling scheme (Section 3),
- a quantitative evaluation of our proposed adaptive sampling scheme against fixed-rate sampling, both in accuracy and efficiency (Section 4),
- validation of the simulated events by training a neural network for optic flow prediction, which generalizes well to real environments (Section 5),
- an efficient, open-source implementation of our proposed event camera simulator.

2 Related Work

In the last few years, a number of event camera datasets and simulators have been introduced ². In this section, we briefly review the most important ones and their specific application scenarios. We then turn to the works on simulation of an event camera.

2.1 Event Camera Datasets

The range of tasks that have been addressed with event cameras can be roughly split into two: low-level vision tasks and high-level vision tasks.

Low-level Vision A number of event datasets has been specifically designed for low-level vision tasks such as visual (inertial) odometry [9, 10, 11], optic flow estimation [12, 13] or depth from stereo [14, 11]. While the scope of the early datasets was severely restricted (for example, [12] contains a dozen sequences of an event camera looking at a calibration pattern, each sequence during less than ten seconds), two recent datasets have recently stood out. [10] was introduced as a benchmark for event-based visual (inertial) odometry. It features 25 sequences recorded with a DAVIS240C sensor [5], and each sequence contains events, frames, and inertial measurements, as well as ground truth camera poses from a motion capture system. However, it does not contain ground truth depth or optic flow, and most of the scenes were captured indoors, targeting AR/VR scenarios. The MVSEC dataset [11] contains 10 sequences from a stereo DAVIS346 sensor, which was mounted on various vehicles (car, quadcopter and motorbike). Images from a standard stereo camera setup, ground truth camera poses, depth maps (derived from a LIDAR) and optic flow maps [15] are also provided. However, the dataset essentially targets visual navigation scenarios for automotive, and the different sensing modalities are only roughly calibrated together (sometimes manually).

¹<https://inivation.com/buy/>

²An extensive list can be found at: https://github.com/uzh-rpg/event-based_vision_resources#datasets-and-simulators-sorted-by-topic

Recognition and Scene Understanding In parallel, using event cameras for high-level tasks has motivated the release of several datasets. Most of these datasets target the task of object recognition [16, 17, 18], or gesture recognition [19]. In particular, [16] presented semi synthetic, event-based versions of the MNIST [20] and Caltech101 datasets. To convert these frame-based datasets to events, they mounted an event camera on a pan-tilt motor, facing a screen displaying standard frames, and recorded the events generated by moving the sensor slightly in different directions using the motor. Finally, the recent DDD17 [21] and N-CARS [18] datasets target automotive applications. While N-CARS [18] specifically focuses on car recognition, DDD17 [21] provides the vehicle speed, GPS position, driver steering, throttle, and brake captured from the car’s on-board diagnostics interface. However, it does not contain semantic labels such as object bounding boxes, nor ground truth data for low-level vision tasks such as depth maps or optical flow.

2.2 Event Camera Simulators

A simple event camera simulator was presented in [22] and implemented as a plugin for the Gazebo simulation software [23]. However, as acknowledged by the authors, this simulator does not implement the principle of operation of an event camera. Instead, it merely thresholds the difference between two successive frames to create edge-like images that resemble the output of an event camera. The asynchronous and low-latency properties of the sensor is not simulated, as all the events from a pair of images are assigned the same timestamp. To the best of the author’s knowledge, [10] and [24] are the only simulators that attempt to simulate an event camera accurately. To simulate an event camera pixel, both works rely on rendering images from a 3D scene at a very high framerate, using either Blender ³ [10], or a custom rendering engine [24]. While this approach allows to mimic the asynchronous output of an event camera, it cannot simulate event data reliably when the brightness signal varies more rapidly than the arbitrarily chosen rendering framerate can handle. A comparison between our approach and this fixed-rate sampling approach [10, 24] is given in Section 4.

3 Proposed Event Simulation Strategy

Unlike standard cameras which capture intensity information from the scene synchronously, in the form of frames, event cameras sample the visual signal asynchronously, i.e., independently for each pixel. Specifically, each pixel of an event sensor will produce an event if the brightness change since the last event fired has reached a given threshold C . This principle is illustrated in Fig. 3. At this point, it is worth clarifying what we mean exactly by “brightness”. Vision sensors measure some function of the radiant flux (or intensity) of light falling per unit area of the sensor, which is referred to as the *irradiance* E . Event cameras operate in the log domain, which allows them to achieve a high dynamic range: instead of measuring changes of irradiance E , they measure changes of log-irradiance $\log E$. Throughout the rest of the paper, we will use the term *brightness* to denote the log-irradiance, and denote it: $\mathcal{L} = \log E$.

3.1 Event Simulation from Adaptive Sampling

To simulate an event camera, one would need access to a continuous representation of the visual signal at every pixel, which is not accessible in practice. To circumvent this problem, past works [10] have proposed to sample the visual signal (i.e. sample frames) synchronously, at a very high framerate, and perform linear interpolation between the samples to reconstruct a piecewise linear approximation of the continuous underlying visual signal, which is used to emulate the principle of operation of an event camera (Fig. 3(a)). We take the same general approach for simulating events, sampling the visual signal (by means of *rendering* images along the camera trajectory), with a key difference: instead of choosing an arbitrary rendering framerate, and sampling frames uniformly across time at the chosen framerate, we propose to sample frames *adaptively*, adapting the sampling rate based on the predicted dynamics of the visual signal.

An adaptative sampling of the visual signal requires a tight integration between the rendering engine, and the event simulator. In the next section, we describe our simulator architecture (Section 3.2) and detail the adaptive sampling scheme we use to ensure that events are generated accurately (Section 3.3).

³<https://www.blender.org/>

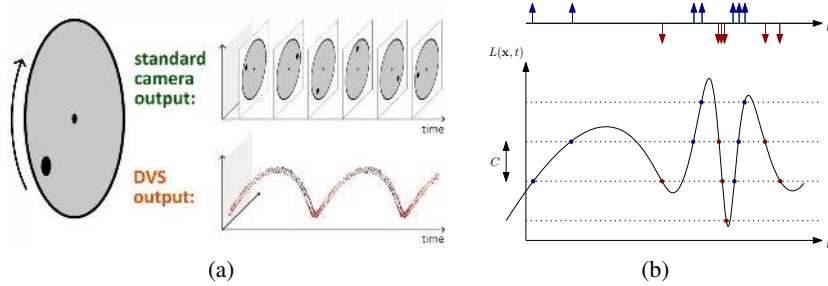


Figure 2: Principle of operation of an event camera. Fig. 2(a): comparison of the output of a standard camera and an event camera when viewing a spinning disk with a black circle. The standard camera outputs frames at a fixed rate, thus sending redundant information when no motion is present in the scene. In contrast, event cameras are data-driven sensors that respond to brightness changes with microsecond latency. Fig. 2(b): Zoom on a pixel x : a positive (resp. negative) event (blue dot, resp. red dot) is generated whenever the (signed) brightness change exceeds the contrast threshold C . Observe how the event rate grows when the signal changes rapidly.

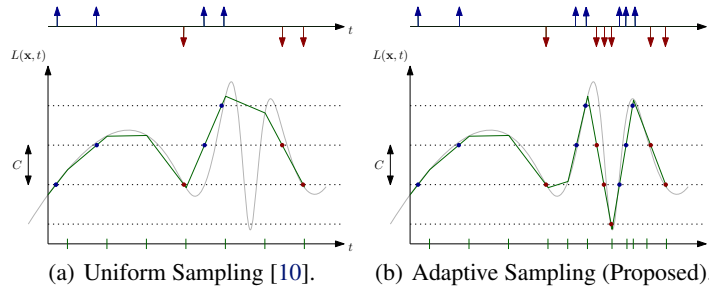


Figure 3: Comparison of uniform sampling (Fig. 3(a)) versus adaptive sampling (Fig. 3(b)). The timestamps at which brightness samples are extracted are shown on the time axis as green markers. While both strategies produce similar events when the signal varies slowly (left part), the uniform sampling strategy fails to faithfully simulate events in the fast-varying part of the signal. By contrast, our proposed adaptive sampling strategy extracts more samples in the fast-varying region, thus successfully simulating events (as can be seen by comparing the simulated events with the ground truth events shown in Fig. 2(b)).

3.2 Simulator Architecture

The architecture of ESIM is illustrated in Fig. 4. It tightly couples the rendering engine and the event simulator, which allows the event simulator to adaptively query visual samples (i.e. frames) based on the dynamics of the visual signal (Section 3.3). Below, we introduce the mathematical formalism we will use to describe, without loss of generality, the various components involved in our simulator. The details of possible implementations of each of these components are given in the appendix (Sections 7.1 and 7.2).

Sensor Trajectory The sensor trajectory is as a smooth function \mathcal{T} that maps every time t to a sensor pose, twist (angular and linear velocity), and acceleration. Following the notation of [25], we denote the pose of the sensor expressed in some inertial frame W by $T_{WB} \in SE(3)$, where $SE(3)$ denotes the group of rigid motion in 3D, the twist by $\xi(t) = ({}_W\mathbf{v}(t), {}_B\omega_{WB}(t))$, and its acceleration (expressed in the inertial frame W) by ${}_B\mathbf{a}(t)$.

Rendering Engine The renderer is a function \mathcal{R} which maps any time t to a rendered image (or irradiance E) of the scene at the current sensor pose. The renderer is parameterized by the environment \mathcal{E} , the sensor trajectory within this environment \mathcal{T} , and the sensor configuration Θ . \mathcal{E} controls the geometry of the scene, as well as its dynamics (it can be for example a still room, a city with moving vehicles, etc.). \mathcal{T} represents the camera trajectory in the environment, which may be generated online based on a dynamical model of the robot and a set of control inputs, or precomputed offline. Θ represents the configuration of the vision sensor simulated, which includes the camera

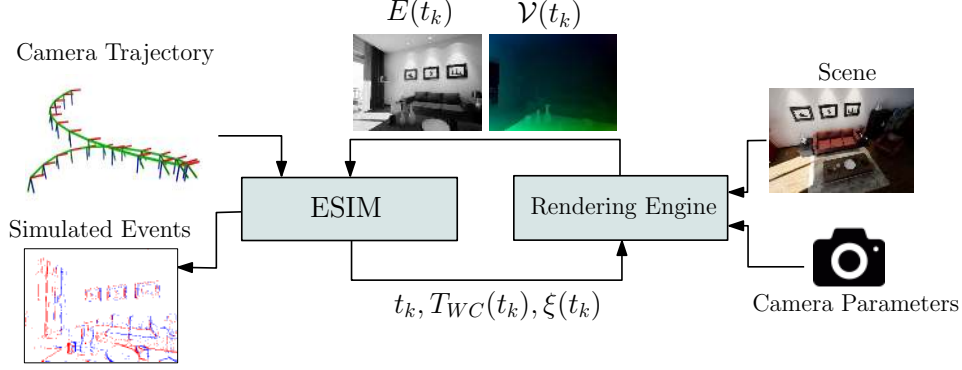


Figure 4: ESIM relies on a tight coupling with the rendering engine to generate events accurately. At time t_k , ESIM samples a new camera pose $T_{WC}(t_k)$ and camera twist $\xi(t_k)$ from the user-defined trajectory and passes them to the rendering engine, which, in turn, renders a new irradiance map $E(t_k)$ and a motion field map $\mathcal{V}(t_k)$. The latter are used to compute the expected brightness change (Eq. (1)), which is used to choose the next rendering time t_{k+1} .

intrinsic (sensor size, focal length, distortion parameters) and the camera extrinsic (pose of the camera with respect to the sensor body, T_{BC}). It also includes sensor-specific parameters (for example, the contrast threshold C for an event camera, the exposure time for a standard camera, etc.). For reasons that will appear clearly in Section 3.3, the renderer additionally provides the motion field $\mathcal{V}(t)$. Figure 1 shows an example of the output of our simulator. Additional examples can be found in the supplementary material.

3.3 Adaptive Sampling Strategies

Based on Brightness Change Under the assumption of lambertian surfaces, a first-order Taylor expansion of the brightness constancy assumption [26] yields:

$$\frac{\partial \mathcal{L}(x; t_k)}{\partial t} \simeq - \langle \nabla \mathcal{L}(x; t_k), \mathcal{V}(x; t_k) \rangle \quad (1)$$

where $\nabla \mathcal{L}$ is the gradient of the brightness image, and \mathcal{V} the motion field. In other words, the expected (signed) brightness change at pixel \mathbf{x} and time t_k during a given interval of time Δt is: $\Delta \mathcal{L} \simeq \frac{\partial \mathcal{L}(x; t_k)}{\partial t} \Delta t$. We want to ensure that for every pixel $\mathbf{x} \in \Omega$, $|\Delta \mathcal{L}| \leq C$ (i.e. the brightness change is bounded by the desired contrast threshold C of the simulated event camera). It can be shown easily from Eq. 1 that choosing the next rendering time t_{k+1} as follows allows for this:

$$t_{k+1} = t_k + \lambda_b C \left| \frac{\partial \mathcal{L}}{\partial t} \right|_m^{-1} \quad (2)$$

, where $\left| \frac{\partial \mathcal{L}}{\partial t} \right|_m = \max_{x \in \Omega} \left| \frac{\partial \mathcal{L}(x; t_k)}{\partial t} \right|$ is the maximum expected rate of brightness change across the image plane Ω , and $\lambda_b \leq 1$ is a parameter that controls the trade-off between rendering speed and accuracy. Essentially, Eq. (2) imposes that the rendering framerate grows proportionally to the maximum expected absolute brightness change in the image. Note that this criterion was derived from Eq. (1), whose correctness depends on the validity of the assumed lambertian surfaces, brightness constancy, and linearity of local image brightness changes. Therefore, it does not provide a strict guarantee that the signal was correctly sampled. To account for these unmodeled non-linear effects, we can enforce a slightly stricter criterion than the one imposed by Eq.(2) by choosing $\lambda_b < 1$ in Eq.(2). In our experiments, we used $\lambda_b = 0.5$.

Based on Pixel Displacement An alternative, simpler strategy to perform adaptive sampling is to ensure that the maximum displacement of a pixel between two successive samples (rendered frames) is bounded. This can be achieved by choosing the next sampling time t_{k+1} as follows:

$$t_{k+1} = t_k + \lambda_v \left| \mathcal{V}(x; t_k) \right|_m^{-1} \quad (3)$$

, where $|\mathcal{V}| = \max_{x \in \Omega} |\mathcal{V}(x; t_k)|$ is the maximum magnitude of the motion field at time t_k , and $\lambda_v \leq 1$. As before, we can choose λ_v to mitigate the effect of unmodeled non-linear effects. In our experiments, we used $\lambda_v = 0.5$.

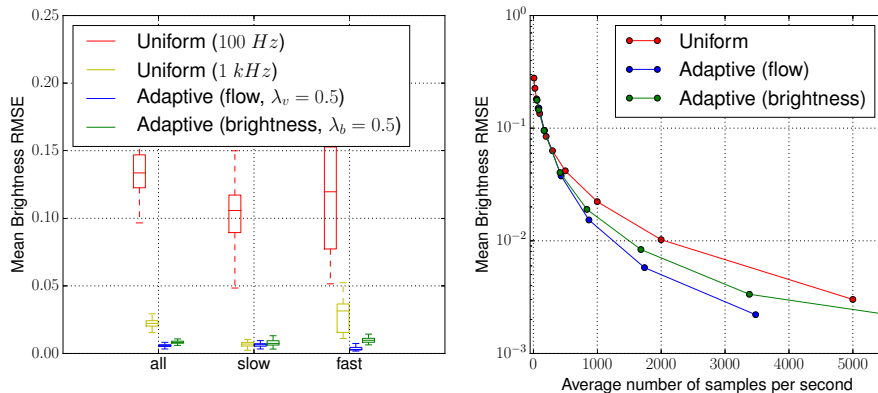


Figure 5: Evaluation of the accuracy and efficiency of the proposed sampling scheme against a fixed-rate sampling scheme. Left: mean RMSE of the reconstructed brightness signal as a function of the number of samples for each sampling strategy, computed on the entire evaluation sequence. Right: mean RMSE of the reconstructed brightness signal as a function of the number of samples for each sampling strategy, computed on the entire evaluation sequence.

3.4 Noise simulation and Non Idealities

We also implement a standard noise model for event cameras, based on an observation originally made in [1] (Fig. 6): the contrast threshold of an event camera is not constant, but rather normally distributed. To simulate this, at every simulation step, we sample the contrast according to $\mathcal{N}(C; \sigma_C)$, where σ_C controls the amount of noise, and can be set by the user. Furthermore, we allow setting different positive (respectively negative) contrast thresholds C_+ (respectively C_-) to simulate a real event camera more accurately, where these values can be adapted independently with the right set of electronic biases. Additional noise effects, such as spatially and temporally varying contrast thresholds due to electronic noise, or limited bandwidth of the event pixels, are still open research questions.

4 Experiments

4.1 Accuracy and Efficiency of Adaptive Sampling

In this section, we show that our adaptive sampling scheme allows achieving consistent event generation, independently of the speed of the camera. Furthermore, we show that the proposed adaptive sampling scheme is more efficient than a fixed-rate sampling strategy [10, 24]. Finally, we compare the two variants of adaptive sampling proposed in Section 3.3, through Eqs. (2), (3).

Experimental Setup To quantify the influence of the proposed adaptive sampling scheme, compared to a uniform sampling strategy, we simulate the same scene (Figure 8), under the same camera motion, with different sampling strategies, and compare the results. To quantify the accuracy of event simulation, one might directly compare the generated events and compare them against "ground truth" events. However, directly comparing event clouds is an ill-posed problem: there exists no clear metric to compute the similarity between two event clouds. Instead, we evaluate the accuracy of a sampling strategy by comparing the estimated brightness signal with respect to a ground truth brightness signal obtained by oversampling the visual signal (at 10 kHz). In order to obtain simulation data that matches the statistics of a real scene, we simulate random camera rotations in a scene simulated using a real panoramic image taken from the Internet, which features various textures, including low-frequency and high-frequency components. We simulated a 5 second sequence of random camera rotation at various speeds, up to 1,000 deg/s, yielding a maximum motion field of about 4,500 px/s on the image plane.

Evaluation Modalities We simulate the exact same scene and camera motion, each time using a different sampling scheme. For each simulated sequence, we consider a set of $N = 100$ pixel locations, randomly sampled on the image plane (the same set is used for all runs). We split the

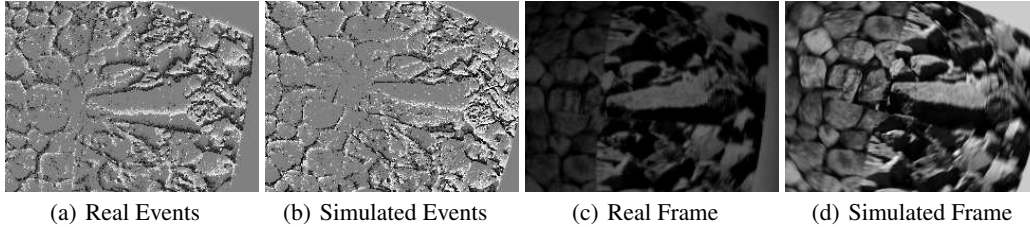


Figure 6: Qualitative comparison of real events and frames (Figs. 6(a), 6(c)) with simulated events and frames (Figs. 6(b), 6(d)) generated during a high-speed motion. While some differences are visible between the real and simulated events, the locations of the simulated events mostly correspond to the real events. A video version of this comparison is available at: <https://youtu.be/VNWFkkTx4Ww>

evaluation sequence in three subsets: a ‘slow’ subset (mean optic flow magnitude: 330 px/s), a ‘fast’ subset (mean optic flow magnitude: 1,075 px/s), and a subset ‘all’, containing the entire sequence (mean optic flow magnitude: 580 px/s). For every pixel in the set, we perform linear interpolation between the brightness samples, and compute the RMSE of the reconstructed brightness signal with respect to the ground truth.

Results Figure 5(a) shows the distribution of the RMSE values for three different sampling schemes: uniform sampling at respectively 100 Hz and 1 kHz, and our proposed adaptive sampling schemes (flow-based and brightness-based) with $\lambda = 0.5$. Sampling at a “low” framerate (100 Hz) yields the highest error, even for the ‘slow’ sequence. Increasing the sampling rate to 1 kHz leads to a significant error reduction on the ‘slow’ sequence, but is not sufficient for faster motions such as those found in the ‘fast’ subsequence, yielding high error overall. Our proposed adaptive schemes do not require any parameter tuning, and perform consistently well on all three subsets, outperforming the uniform sampling strategy by a large margin, especially during fast motions. Figure 5(b) shows the mean RMSE as a function of the number of samples (i.e. the number of frames rendered). It shows that, for a given computation budget, the proposed adaptive sampling strategies have a lower error. For example, the adaptive sampling strategy based on optic flow requires on average 1,260 samples per second to achieve an $\text{RMSE} \leq 0.01$ while the uniform sampling strategy requires on average 2,060 samples per second to achieve the same error bound, which leads to a 60% decrease of the simulation time. Overall, these experiments show that the adaptive sampling strategy based on maximum optic flow achieves a higher accuracy than the one based on brightness for the same computational budget. Following this analysis, we now use the adaptive sampling strategy based on maximum optic flow with $\lambda = 0.5$ in all our next experiments.

4.2 Qualitative Comparison with Real Data

To show qualitatively that our simulated event camera output resembles that of a real event camera, we show a side-by-side comparison of a real dataset from the Event Camera Dataset [10], and a simulated reproduction of the dataset. Details on how we reproduced the scene in simulation are provided in the appendix. Fig. 6 shows a side-by-side comparison of simulated data (events and frames) with real data. The events are visualized using an exponential time surface [27] with exponential decay $\tau_c = 3.0 \text{ ms}$.

Discussion As Fig. 6 shows, the main moving structures are correctly captured in the simulated events. However, small differences between the real and simulated event streams show that the simulation is not perfect. There are two main sources of error. First, the simulated and real setup do not match exactly: (i) the contrast thresholds and noise level of the real event camera are not known and were only guessed from data, and (ii) the poster position and texture do not exactly match the real poster used in the dataset. Second, our simulator features a simple noise model (3.4), which does not account for more complex noise sources in a real event camera coming from non-idealities in the electronics. For example, at high speeds, it has been observed that the refractory period of the event pixels may cause the loss of events [28], which is not modelled by our implementation.

5 Example Application: Learning Optic Flow

Affine Optical Flow Dataset In this section we explore an application of the event camera simulator which is supervised learning of globally affine optic flow. We generate a large dataset (10,000 sequences with 0.04 s duration each) of event data simulated by applying random affine transformations to a set of real images (from the Internet). As in [29], we select the random affine parameters such that the resulting flow distribution matches that of the Sintel dataset. We train a neural network that regresses the dense motion field from the events, using the ground truth optic flow maps as labels. Our network architecture is the same as FlowNet Simple [29] and we train the network by minimizing the end-point-error (EPE) between predicted and ground truth optic flow. Similarly, we perform online geometric augmentation (ie, random rotation and scaling) during training. As event-based input to the network, we convert each event sequence to a tensor with 4 channels as proposed in [15] and report the validation and test EPE only on pixels where events were fired. Comparisons of predicted and ground truth optic flow from the validation set are shown in Fig. 7 and Fig. 10 in the appendix. The network achieves a final EPE of about 0.3 pixels on the validation set.

Sim2Real Transfer We evaluated the trained network directly on *real event data* without any fine-tuning. Specifically, we use the *boxes.rotation* sequence from the Event Camera Dataset [10], for which the optic flow can be approximated as an affine flow field. We use the IMU data to measure angular velocity which allows us to compute the ground truth motion field for testing. The network performs well, both qualitatively and quantitatively: the EPE being close to 4.5 pixels on the real event dataset. A comparison of predicted and ground truth optic flow can be found in Fig. 7 (bottom row) and 10 in the appendix. It is important to stress that the network was not trained on real event data, nor trained on scenes that look similar to the scene used for testing. We therefore showed that our proposed simulator is useful to train networks that will perform relatively well in the real world. In practice, such a network would be trained on a large amount of synthetic data and fine-tuned on a smaller amount of real event data.

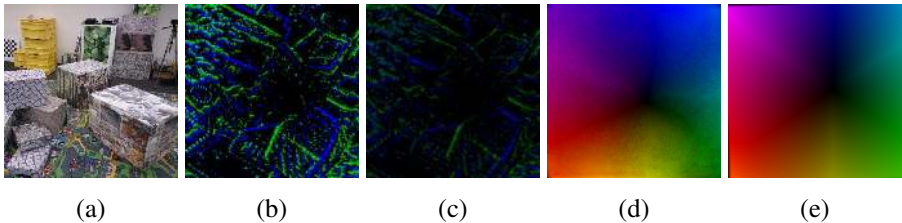


Figure 7: Results from learning globally affine flow from simulated event data and ground truth optic flow. The top row shows the network output on the validation set (simulated) and the bottom row on the test set (real). The columns show a preview of the scenes (a), inputs to the network, comprising event time surfaces and event count frames (b-c) [15], and the predicted (d) and ground truth (e) optic flow.

6 Conclusions

We presented the first event camera simulator that can simulate events reliably and efficiently, and that can simulate arbitrary camera trajectories in arbitrary 3D scenes. The key component of our simulator is a theoretically sound, adaptive rendering scheme that only samples frames when necessary, through a tight coupling between the rendering engine and the event simulator. We release an open source implementation of our simulator. We believe our simulator will be of considerable interest for the vision community; allowing, for the first time, to collect large amounts of event data with ground truth, opening the door to accurate benchmarking of existing computer vision methods that operate on event data, as well as to supervised learning approaches with event cameras.

Acknowledgments

We thank Raffael Theiler and Dario Brescianini for helping with the open source implementation of ESIM, and Antonio Loquercio and Kosta Derpanis for fruitful discussions, advice and review. This work was supported by the the Swiss National Center of Competence Research Robotics (NCCR), Qualcomm (through the Qualcomm Innovation Fellowship Award 2018), the SNSF-ERC Starting Grant and DARPA FLA.

References

- [1] P. Lichtsteiner, C. Posch, and T. Delbruck. A 128×128 120 dB 15 μ s latency asynchronous temporal contrast vision sensor. *IEEE J. Solid-State Circuits*, 43(2):566–576, 2008. doi:10.1109/JSSC.2007.914337.
- [2] J. Conradt, M. Cook, R. Berner, P. Lichtsteiner, R. J. Douglas, and T. Delbruck. A pencil balancing robot using a pair of AER dynamic vision sensors. In *IEEE Int. Symp. Circuits Syst. (ISCAS)*, pages 781–784, 2009. doi:10.1109/ISCAS.2009.5117867.
- [3] D. Gehrig, H. Rebecq, G. Gallego, and D. Scaramuzza. Asynchronous, photometric feature tracking using events and frames. In *Eur. Conf. Comput. Vis. (ECCV)*, 2018.
- [4] A. Rosinol Vidal, H. Rebecq, T. Horstschaefer, and D. Scaramuzza. Ultimate SLAM? combining events, images, and IMU for robust visual SLAM in HDR and high speed scenarios. *IEEE Robot. Autom. Lett.*, 3(2):994–1001, Apr. 2018. doi:10.1109/LRA.2018.2793357.
- [5] C. Brandli, R. Berner, M. Yang, S.-C. Liu, and T. Delbruck. A 240×180 130dB 3 μ s latency global shutter spatiotemporal vision sensor. *IEEE J. Solid-State Circuits*, 49(10):2333–2341, 2014. ISSN 0018-9200. doi:10.1109/JSSC.2014.2342715.
- [6] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun. CARLA: An open urban driving simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*, pages 1–16, 2017.
- [7] S. Shah, D. Dey, C. Lovett, and A. Kapoor. Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In *Field and Service Robotics*, 2017. URL <https://arxiv.org/abs/1705.05065>.
- [8] Q. Weichao, Z. Fangwei, Z. Yi, Q. Siyuan, X. Zihao, S. K. Tae, W. Yizhou, and Y. Alan. Unrealcv: Virtual worlds for computer vision. *ACM Multimedia Open Source Software Competition*, 2017.
- [9] D. Weikersdorfer, D. B. Adrian, D. Cremers, and J. Conradt. Event-based 3D SLAM with a depth-augmented dynamic vision sensor. In *IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 359–364, June 2014. doi:10.1109/ICRA.2014.6906882.
- [10] 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. *Int. J. Robot. Research*, 36:142–149, 2017. doi:10.1177/0278364917691115.
- [11] A. Z. Zhu, D. Thakur, T. Ozaşlan, B. Pfrommer, V. Kumar, and K. Daniilidis. The multivehicle stereo event camera dataset: An event camera dataset for 3D perception. *IEEE Robot. Autom. Lett.*, 3(3):2032–2039, July 2018. doi:10.1109/Lra.2018.2800793.
- [12] F. Barranco, C. Fermüller, Y. Aloimonos, and T. Delbruck. A dataset for visual navigation with neuromorphic methods. *Front. Neurosci.*, 10:49, 2016. doi:10.3389/fnins.2016.00049.
- [13] B. Rueckauer and T. Delbruck. Evaluation of event-based algorithms for optical flow with ground-truth from inertial measurement sensor. *Front. Neurosci.*, 10(176), 2016. doi:10.3389/fnins.2016.00176.
- [14] Z. Xie, S. Chen, and G. Orchard. Event-based stereo depth estimation using belief propagation. *Front. Neurosci.*, 11, Oct. 2017. doi:10.3389/fnins.2017.00535.
- [15] A. Z. Zhu, L. Yuan, K. Chaney, and K. Daniilidis. Ev-flownet: Self-supervised optical flow estimation for event-based cameras. In *Robotics: Science and Systems (RSS)*, July 2018.

- [16] G. Orchard, A. Jayawant, G. K. Cohen, and N. Thakor. Converting static image datasets to spiking neuromorphic datasets using saccades. *Front. Neurosci.*, 9:437, 2015. doi:10.3389/fnins.2015.00437.
- [17] T. Serrano-Gotarredona and B. Linares-Barranco. Poker-DVS and MNIST-DVS. their history, how they were made, and other details. *Front. Neurosci.*, 9:481, 2015. doi:10.3389/fnins.2015.00481.
- [18] A. Sironi, M. Brambilla, N. Bourdis, X. Lagorce, and R. Benosman. HATS: histograms of averaged time surfaces for robust event-based object classification. In *IEEE Int. Conf. Comput. Vis. Pattern Recog. (CVPR)*, June 2018.
- [19] A. Amir, B. Taba, D. Berg, T. Melano, J. McKinstry, C. D. Nolfo, T. Nayak, A. Andreopoulos, G. Garreau, M. Mendoza, J. Kusnitz, M. Debole, S. Esser, T. Delbruck, M. Flickner, and D. Modha. A low power, fully event-based gesture recognition system. In *IEEE Int. Conf. Comput. Vis. Pattern Recog. (CVPR)*, pages 7388–7397, July 2017. doi:10.1109/CVPR.2017.781.
- [20] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, Nov. 1998.
- [21] J. Binas, D. Neil, S.-C. Liu, and T. Delbruck. DDD17: End-to-end DAVIS driving dataset. In *ICML Workshop on Machine Learning for Autonomous Vehicles*, 2017.
- [22] J. Kaiser, T. J. C. V., C. Hubschneider, P. Wolf, M. Weber, M. Hoff, A. Friedrich, K. Wojtasik, A. Roennau, R. Kohlhaas, R. Dillmann, and J. Zöllner. Towards a framework for end-to-end control of a simulated vehicle with spiking neural networks. In *2016 IEEE International Conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAN)*, pages 127–134, Dec. 2016.
- [23] N. Koenig and A. Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*, volume 3, pages 2149–2154, Sept. 2004. doi:10.1109/IROS.2004.1389727.
- [24] W. Li, S. Saeedi, J. McCormac, R. Clark, D. Tzoumanikas, Q. Ye, Y. Huang, R. Tang, and S. Leutenegger. Interiornet: Mega-scale multi-sensor photo-realistic indoor scenes dataset. In *British Machine Vis. Conf. (BMVC)*, page 77, Sept. 2018.
- [25] C. Forster, L. Carlone, F. Dellaert, and D. Scaramuzza. On-manifold preintegration for real-time visual-inertial odometry. *IEEE Trans. Robot.*, 33(1):1–21, 2017. doi:10.1109/TRO.2016.2597321.
- [26] R. Szeliski. *Computer Vision: Algorithms and Applications*. Texts in Computer Science. Springer, 2010. ISBN 9781848829343.
- [27] X. Lagorce, G. Orchard, F. Gallupi, B. E. Shi, and R. Benosman. HOTS: A hierarchy of event-based time-surfaces for pattern recognition. *IEEE Trans. Pattern Anal. Machine Intell.*, 39(7): 1346–1359, July 2017. doi:10.1109/TPAMI.2016.2574707.
- [28] C. Brandli, L. Muller, and T. Delbruck. Real-time, high-speed video decompression using a frame- and event-based DAVIS sensor. In *IEEE Int. Symp. Circuits Syst. (ISCAS)*, pages 686–689, June 2014. doi:10.1109/ISCAS.2014.6865228.
- [29] A. Dosovitskiy, P. Fischer, E. Ilg, P. Häusser, C. Hazırbaş, V. Golkov, P. van der Smagt, D. Cremers, and T. Brox. FlowNet: Learning optical flow with convolutional networks. In *Int. Conf. Comput. Vis. (ICCV)*, pages 2758–2766, 2015. doi:10.1109/ICCV.2015.316.
- [30] P. Furgale, J. Rehder, and R. Siegwart. Unified temporal and spatial calibration for multi-sensor systems. In *IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*, 2013.
- [31] G. Gallego, C. Forster, E. Mueggler, and D. Scaramuzza. Event-based camera pose tracking using a generative event model. arXiv:1510.01972, 2015.
- [32] R. A. Newcombe. *Dense Visual SLAM*. PhD thesis, Imperial College London, London, UK, Dec. 2012.

ESIM: an Open Event Camera Simulator

Supplementary Material

7 Implementation Details

This section describes the implementation choices we made in the open source implementation of ESIM, available at: <http://rpg.ifi.uzh.ch/esim>.

7.1 Sensor Trajectory

To implement the trajectory function \mathcal{T} , we use a continuous time representation for the camera trajectory, based on splines in $SE(3)$ ([30]). The user can either decide to generate a random trajectory or load a trajectory from a file containing a discrete set of camera poses. In the first case, we sample randomly a fixed number of camera poses in $SE(3)$ and fit a spline to the poses. In the second case, a spline is fitted to the waypoint poses provided by the user. The spline representation allows us to sample poses continuously in time. Furthermore, splines provide analytical formulas for the sensor twist ξ and acceleration ${}_B\mathbf{a}$ along the trajectory, from which we can compute the motion field (7.3) and simulate inertial measurements (7.4).

7.2 Rendering Engines

Our open-source implementation comes with multiple rendering engines that have different characteristics. A full list is available on the simulator website⁴. Here, we briefly describe the two most important ones. The first rendering backend is based on pure OpenGL code, and is designed to simulate a large amount of data fast, at the price of simulating only simple 3D scenes with textures. The second is based on Unreal Engine, and is designed to simulate photorealistic data; however, it is relatively slow compared to the former.

OpenGL Rendering Engine The first available rendering engine is based on pure OpenGL, i.e. it is based on rasterization, implemented on the GPU. A vertex shader projects a user-provided 3D point cloud in the desired camera frame, and projects it on the image plane. A fragment shader evaluates the intensity to assign to each pixel in the image plane, by sampling textures (provided UV mappings) provided with the 3D model. The current implementation supports loading textured 3D models from OBJ files or directly from Blender, and also supports loading camera trajectories directly from Blender. A rendering pass is extremely fast (1ms on a laptop), which allows the event simulation to run in real-time in most scenarios. An example of the output of this renderer can be found in the third row of Fig. 9.

Photorealistic Rendering Engine We additionally provide a rendering engine based on Unreal Engine⁵, through an UnrealCV [8] project. We implemented a C++ client that interacts with an UnrealCV server and queries images (which we assimilate to irradiance maps) and depth maps along camera poses sampled along a user-defined trajectory. Furthermore, we supply a user interface which allows users to navigate through a scene interactively, set up waypoints by clicking on the screen, and then simulate a camera following an interpolated trajectory passing through these waypoints offline. While the generated event data is of very high quality (photorealistic), the event generation is quite slow (about 2 min computing time for one second of simulated data on a laptop computer). Example outputs of this renderer can be found in the first and second rows of Fig. 9.

7.3 Computation of Motion Field

As anticipated in Section 3, our adaptive sampling scheme requires the computation of the motion field on the image plane. Here we briefly describe how to this from the camera twist ξ and depth map of the scene Z . The motion field \mathcal{V} is the sum of two contributions: one term comes from the camera egomotion itself (apparent motion of the scene with respect to the camera) \mathcal{V}_{ego} , and the second arises from moving objects in the scene \mathcal{V}_{dyn} . It follows that $\mathcal{V} = \mathcal{V}_{ego} + \mathcal{V}_{dyn}$. Below

⁴<http://rpg.ifi.uzh.ch/esim>

⁵<https://www.unrealengine.com/en-US/what-is-unreal-engine-4>

we give a closed-form expression for the computation of \mathcal{V}_{ego} . A similar expression can be derived for \mathcal{V}_{dyn} . \mathcal{V}_{ego} can be computed from a depth map of the scene Z and the camera twist vector $\xi(t) = ({}_W\mathbf{v}(t), {}_B\omega_{WB}(t))$ as follows [31]:

$$\mathcal{V}_{ego}(\mathbf{x}, t) = B(\mathbf{x}, t)\xi(t) \quad (4)$$

where

$$B = \begin{bmatrix} -Z^{-1} & 0 & uZ^{-1} & uv & -(1+u^2) & v \\ 0 & -Z^{-1} & vZ^{-1} & 1+v^2 & -uv & -u \end{bmatrix}$$

is the interaction matrix, with $Z := Z(\mathbf{x}, t)$ denoting the depth at pixel \mathbf{x} , time t , and $\mathbf{x} = (u, v)$ denoting a pixel location in the image plane, expressed in calibrated coordinates.

7.4 Additional Sensors

Our open-source simulator can also simulate additional sensors besides an event camera, namely a standard camera, and an inertial measurement unit. We now briefly describe how we simulate both of these sensors.

Standard Camera Our camera sensor takes as an input a sequence of irradiance maps rendered from the rendering engine \mathcal{R} , and produces a simulated camera image, by integrating the sequence of irradiance samples (with times t_k) collected over a user-defined exposure time ΔT . Specifically, a camera image I is generated as follows:

$$I = f \left(\sum_{\substack{t_k \leq t + \Delta t/2 \\ t_k \geq t - \Delta t/2}} E(t_k)(t_{k+1} - t_k) \right)$$

where f is the camera response function that maps integrated irradiance values to pixel intensities [32]. In our implementation, we used $f = Id$ which matches the linear response of the DAVIS active pixel sensor.

Inertial Measurement Unit We simulate an inertial measurement unit using additive Gaussian noise on the ground truth angular velocity ${}_B\omega_{WB}$ (respectively acceleration ${}_B\mathbf{a}$) values sampled at 1 kHz from the trajectory \mathcal{T} . Additionally, we offset the angular velocity (and acceleration) measurements with a gyroscope and accelerometer bias. In real sensors the dynamics of these biases are well characterized by random walks. To simulate them we generate random walks at 200 Hz by adding Gaussian noise and fit a spline through these points to provide continuous-time values.

8 Additional Details on the Experiments

8.1 Quantitative Comparison

Figure 8 shows a preview of the simulation scenario used for the quantitative evaluation of the adaptive sampling strategy proposed in Section 4.1 the paper.

8.2 Qualitative Comparison

For the qualitative comparison with real event data presented in Section 4.2, we replicated the *poster_6dof* sequence from the Event Camera Dataset [10], which consists of a planar poster attached to a wall. We recreated the scene in 3D by downloading high-resolution pictures of the original poster, and converting them from sRGB to a linear color space. Furthermore, we used the provided camera calibration and trajectory (the trajectory was recorded with a motion capture system) to simulate the camera motion in the scene. Since neither the contrast thresholds of the event camera, nor the exposure time of the standard camera used in the dataset were disclosed, we guessed these values based on observing the event output, and chose respectively $C_+ = 0.55$, $C_- = 0.41$, with Gaussian noise on the thresholds $\sigma_C = 0.021$. For the exposure time of the standard camera simulated, we used $\Delta t = 7.5 \text{ ms}$.

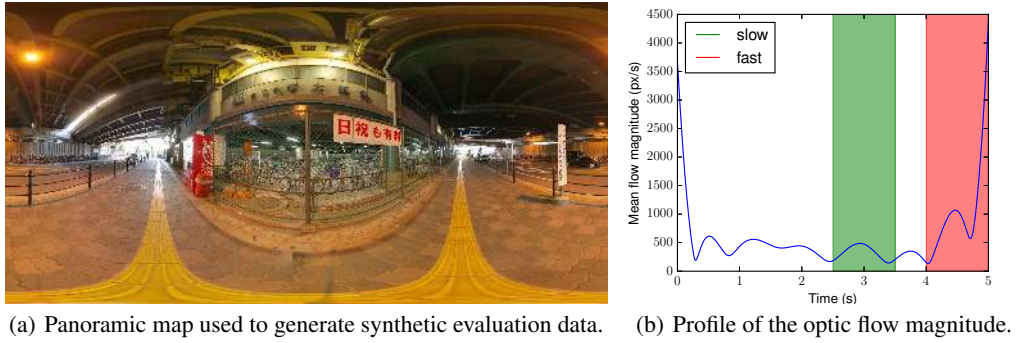


Figure 8: Simulation scenario used for quantitative evaluation of the adaptive sampling strategy proposed. Fig. 8(a): Real panoramic image that was used to simulate an event camera performing random rotations at various speeds. This scene was chosen because of it features both mostly uniform regions, but also regions with much finer, high frequency details such as the grillage close to the bikes. Fig. 8(b): average magnitude of the optic flow as a function of time. The red and green shaded areas mark the boundaries of the "slow" and "fast" subsets used mentioned in Section 4.1.

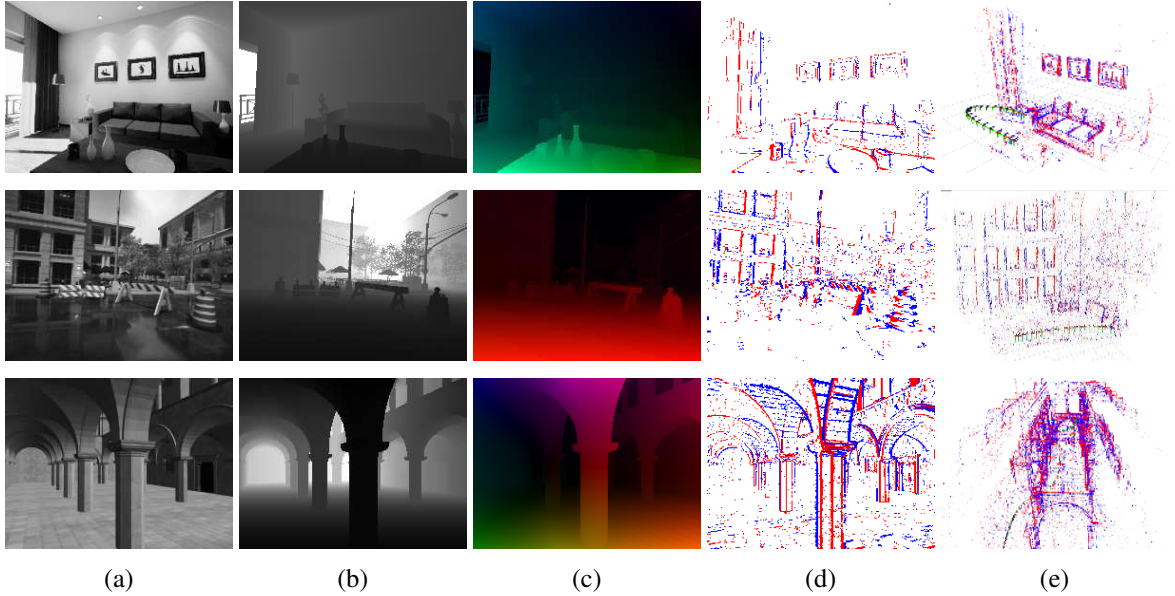


Figure 9: (a) Image. (b) Depth map. (c) Optical Flow (Color-coded). (d) Events (Positive and Negative). (e) Point cloud (3D events) + Camera trajectory

8.3 Additional qualitative results for predicting affine flow

Figure 10 shows additional qualitative examples of the machine learning application presented in Section 5. We show results for both synthetic sequences from the evaluation set, and real event data.

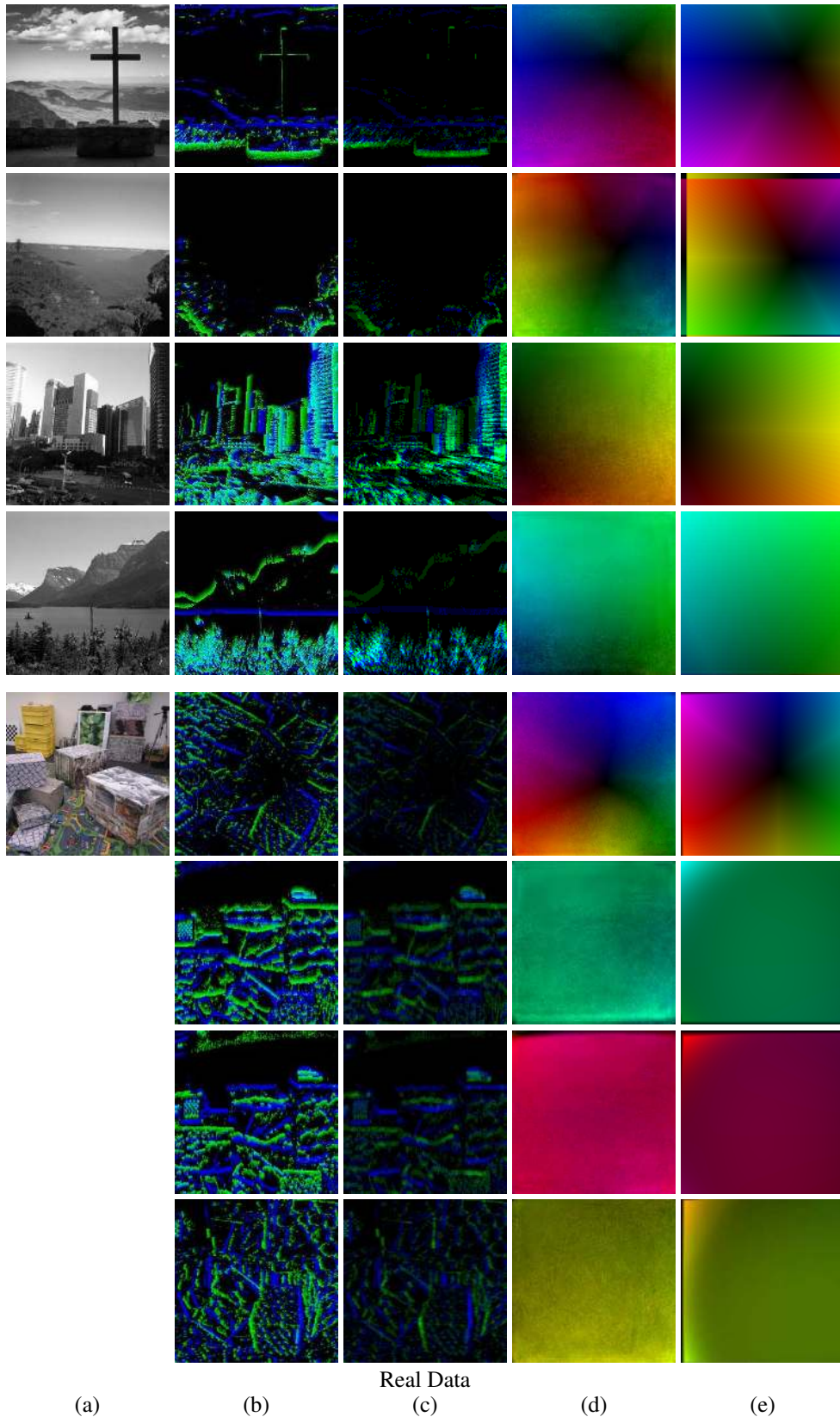


Figure 10: Results from learning globally affine flow from simulated event data and ground truth optic flow. The top 4 rows show network outputs on the validation set (simulated) and the bottom rows on the test set (real). The columns show a preview of the scenes (a), inputs to the network, comprising event time surfaces and event count frames (b-c) [15], and the predicted (d) and ground truth (e) optic flow.