Supplementary material
HUGNet: Hemi-Spherical Update Graph Neural Network applied to low-latency event-based optical flow

Thomas Dalgaty[1] * Thomas Mesquida[1] Damien Joubert[2] † Amos Sironi[2] Pascal Vivet[1] Christoph Posch[2]
[1] Université Grenoble-Alpes, CEA-List, Grenoble, France, [2] Prophesee, Paris, France
thomas.dalgaty@cea.fr thomas.mesquida@cea.fr cposch@prophesee.ai

1. The Rock Scenes dataset

The Rock Scene (RS) dataset was generated to benchmark the different methods with a lightweight yet complex dataset. Classic rock album covers move in front of a background scene, either natural (leaves, trees etc.) or artificial (buildings, stained glass, etc.). Both the cover and background move independently and undergo sharp, uncorrelated direction changes. An example of the frames, and the generated events, is shown in Fig.1(a). This 2D dataset is simple to generate and does not require complex rendering engines. The motion of both the cover and the background are uniformly sampled between 100 and 1000 px/s, and the acceleration is not continuous between two motions. The resulting distribution of x and y speeds are plotted in Fig.1(c) and three examples of object trajectories for three different sequences are traced in Fig.1(d). The RGB images are converted into the Lab color space and the event simulator, similar to [4] or [5], uses the luminance channel. The sensitivities of the 100×100 pixels are set to 0.5 on average with a standard deviation equal to 0.03. The mean latency is set to 300 μs and the jitter to 100 μs. Both pixel non-idealities such as refractory period or cut-off frequency are enabled in the tool. The events are then filtered using an algorithm keeping the second event with the same polarity after a polarity change [2]. This is illustrated in Fig.1(b) where red events are filtered out since they are generated within 10ms of the previous blue event of the same polarity. The threshold of the filter, which is the maximum delay between the first and second event after a polarity change, is set to 10ms. For Rock scenes, this approach reduces the number of events on average by x10, while preserving the edges and motion information. Ultra-fast edges with the same relative contrast can be removed by this filter, but the overall performance using HUGNet is not significantly impacted. In the MVSEC dataset, this filtering strategy is too strong. We use instead another filter which keeps the first event of a burst of events. As these filters are available in state of the art sensors [1], it will also considerably reduce the size of event-based datasets used for further motion analysis tasks too.

*Corresponding author
†Contact djoubert@prophesee.ai for access to Rock Scenes
This work is partly funded thanks to the French national program "Programme d'Investissements d'Avenir, IRT Nanoelec" ANR-10-AIRT-05.

2. Detail on Seq-flownet

The different components of the network topology of Seq-flownet is shown in Fig. 2. The architecture uses the same UNet structure as Ev-flownet with a few notable differences. First, each downsampling block (Fig.2(a)) is composed of 2 convolution layers and a pooling layer. Also, each upsampling block is composed of a transposed convolutional layer. In the upsampling layers, the lower resolution feature map is concatenated with the feature map coming from the same level in the encoder stage and then passed through a convolution layer - similar to the decoder stage of [6] and depicted in Fig.2(b). In contrast EV-flownet uses bi-linear interpolation for upsampling. It should be noted that, in the main text, the computational cost of this upsampling is not considered for EV-flownet whereas it does contribute to the MAC/s reported for Seq-flownet. The number of MAC/s for EV-flownet is therefore likely under-reported. In addition, EV-flownet uses two residual blocks at the bottleneck point of the topology whereas Seq-flownet has only a single block. The kernel size is 3×3 for all filters besides the output layer. The output feature maps, one feature map for each of the optical flow x and y components, are calculated by performing 1×1 convolutions with the output of the final upsampling block.

Furthermore, EV-flownet adds a single convolutional layer to predict the optical flow prediction at the output of each up-sampling block. The predictions at each level are used to train the network in a self-supervised manner: minimising the difference between the next image and the previous image warped by the predicted flow at each level. Seq-flownet is trained in a fully-supervised manner: minimising

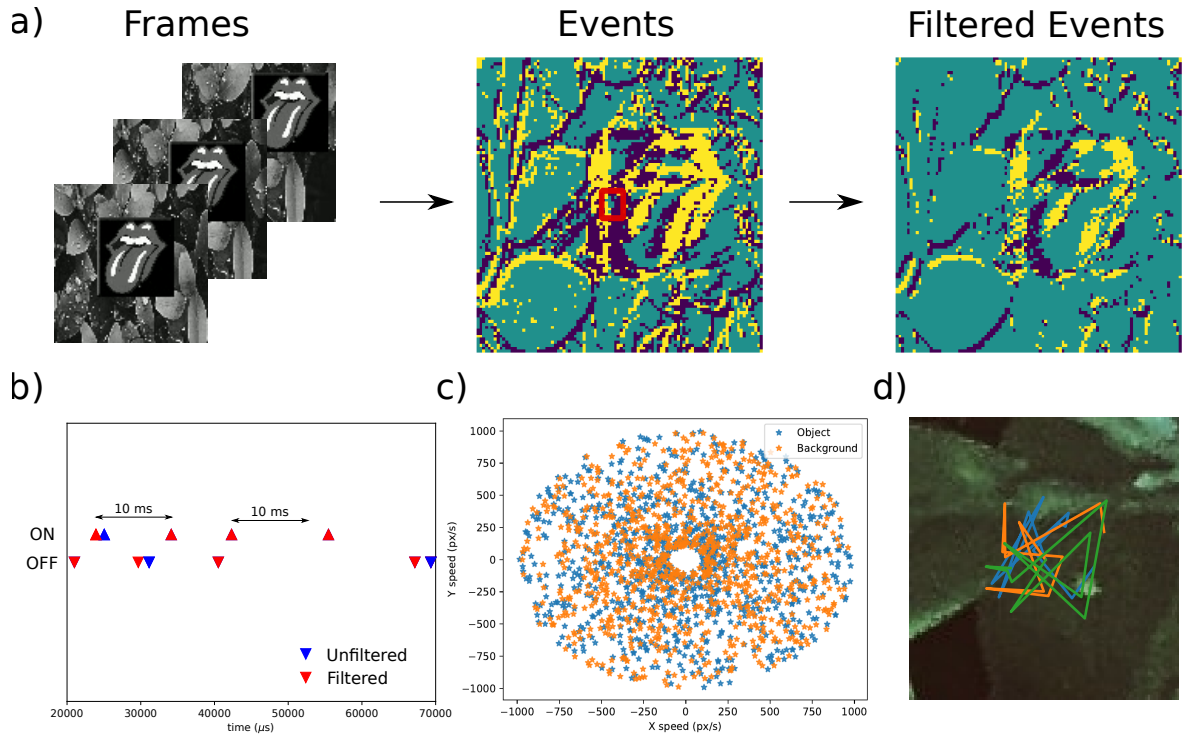

Figure 1. **Rock Scene dataset**. **a)** A sequence of images is generated at 5 kHz to simulate the events produced by an event camera. The resulting events are then filtered using a basic algorithm to discard repeated events outside a polarity change. **b)** Example of the filtering for the pixel at the centre of the red square in part (a): the events in red are removed because they arrive less than 10 ms after an event with the polarity. The double arrow represents the 10 ms filtering threshold. **c)** Distribution of the speeds of the objects and the backgrounds. **d)** Example of three trajectories of the album cover in one scene: each colour represents a trajectory.

the predicted optical flow to the ground truth available in the dataset in same fashion as described for HUGNet in the main text. A mask is used such that the loss is only calculated over active pixels - i.e., pixels having a non-zero event count. Seq-flownet is therefore useful for a direct comparison with HUGNet since the two approaches are trained using the same data and loss function.

A final difference is that Seq-flownet uses as its input the concatenation of five successive dense frames, each frame representing the positive and negative event count in two polarity channels during 100ms. This results in a total of ten input channels. These frames are created at a rate of 100Hz which was found to be required in order for the model to model scene movements (i.e., using previous scene states) and provide more accurate optical flow results. On the other hand, EV-flownet uses a single frame with four channels as input: two channels representing the event count for the period of the frame and another two channels storing the absolute timestamp of the most recent activity per pixel. As EV-flownet uses RGB frames for the self-supervised training, dense event frames use the same framerate as the standard frames used for warping.

3. Details about Algorithm 1 and [7]

Algorithm 1 is not an exact replication of the method proposed in [7]. Indeed, their approach is to asynchronously update not only the graph features but also the task head. Algorithm 1 modifies this point waiting for the graph features to be frozen to compute the task head. While both methods will produce the same output, waiting for the graph to be frozen to compute task head decreases the number of operation used per event and permitted the fairest comparison to this approach in terms of MAC/s.

4. Event-Graph Charbonnier Smoothness loss - Ablation study

Fig.3 shows the ablation study performed for the Charbonnier Smoothness loss adapted to event-graph computation. The loss is weighted by different values between zero to one to understand the optimum value of its contribution on the MVSEC indoor split. While Charbonnier smoothness loss aims to smooth Optical Flow prediction between neighbouring pixels, our adaptation aims to smooth prediction between neighbouring node in the event-graph. The average prediction of neighbours is acquired through an identity weighted graph convolutional layer as described in the

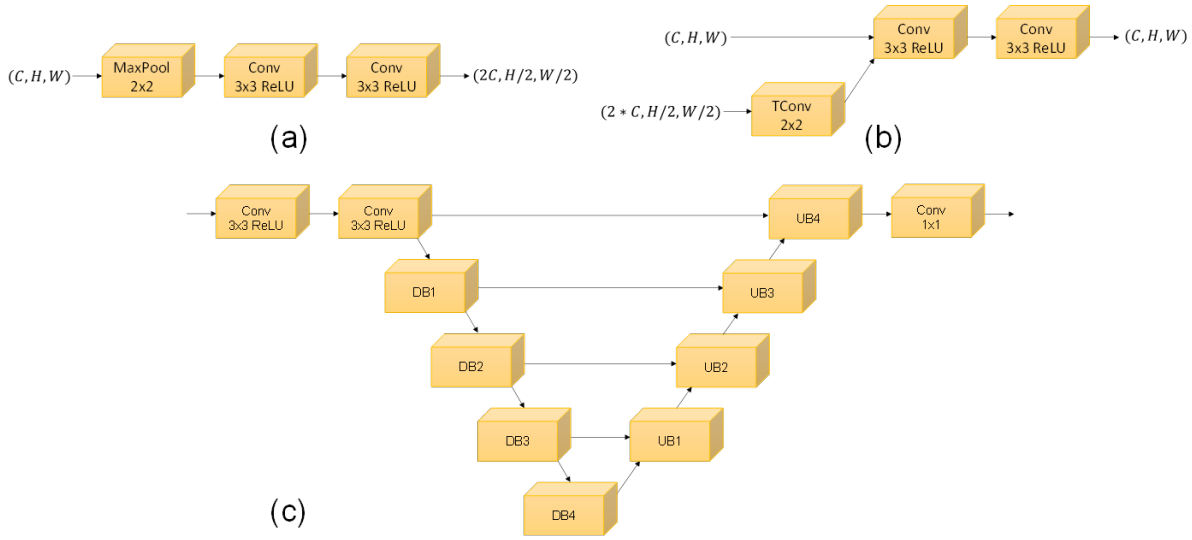

Figure 2. **Seq-flownet network topology**. (a) The DownBlock (DB) used for downsampling. (C, H, W) represent the feature size at each input and output. (b) the UpBlock (UB) used for upsampling. A transposed convolution is used to upscale the input coming from lower resolution. (c) Seq-flownet topology. DBi and UBi represents successive downblocks and upblocks respectively. 16 channels are used for the first 2 conv and each downsampling multiplies the number of channels by 2 and divides image size by 2. A final pointwise convolution layer is used to output optical flow prediction.

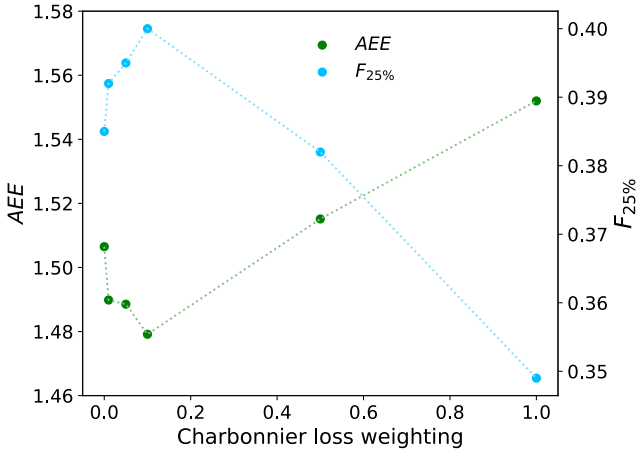

Figure 3. The average endpoint error (green) and the flow prediction accuracy at 25% (blue) plotted as a function of the graph Charbonnier smoothness loss. The MVSEC indoor split was used for this study.

main text.

5. Qualitative HUGNet optical flow examples

Qualitative examples of optical flow predictions made by HUGNet on Rock scenes and MVSEC test sequences are shown in Fig.4 and Fig.5 respectively. The central column contains the predicted vectors, coloured based on their flow accuracy at 25%. On the right-hand side, the ground truth vectors are shown and on the left the RGB corresponding image.

It can be instantly seen that, globally, the predicted flow by HUGNet corresponds well to the movement in the scene. This is particularly impressive in Rock scenes where the background and the foreground object have different directions and velocities. The coloration of the flow vector arrows (green: AEE magnitude $<$ %25 of ground truth magnitude, red: the inverse) also highlights the challenging nature of the flow accuracy metric. Flow vector arrows that are visibly very close to the ground truth are in fact often outwith this limit.

6. Graph neural network timing codes

The graph update latency and the number of MAC/s reported in the main text were measured using Python codes that reproduce exactly the sequence of steps required to incorporate an event into an existing event-graph. The time Python library was used to measure the elapsed time for the critical code block. Each code was run 1000 times on the same CPU to calculate an average graph update latency and MAC/s. The three codes are *Sub-graph Full*, *Sub-graph Sparse* and *Sub-graph HUGNet*.

Sub-graph Full: An entire graph G from an input recording is initially loaded. The graph is already built (since they are the same used for training and testing) so an event is randomly sampled within the graph and the past sub-graph that relevant to this event is isolated from the full structure. This corresponds to all events and edges within L time and pixel radii - i.e., all possible events that interact with the sampled event. We then start the timer. For the newly generated event and all events within a single time

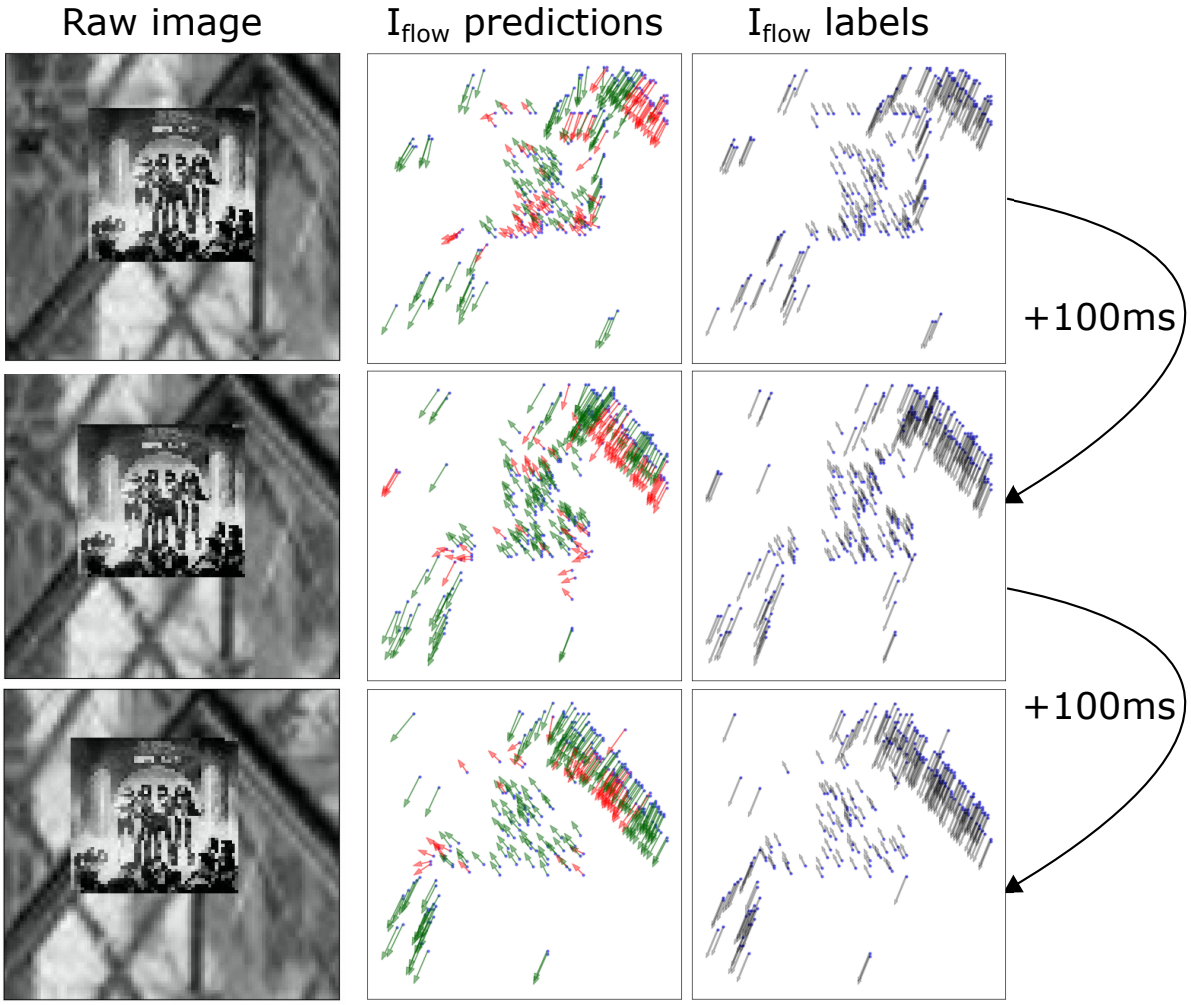

Figure 4. **Qualitative example (Kiss album cover moving over a Louvres pyramid background) on Rock scenes**. The figure is composed to a three by three grid of imagettes. The left-most column features the raw images recorded at intervals of 100ms at a particular point of one of the test sequences. The central column show optical flow vectors predicted by HUGNet. These predicted flow are shown using arrows that correspond to the sum of the predicted x and y components. These arrows are coloured either green or red corresponding to whether they are inferior to, or exceed, the flow accuracy at 25%. Events are plotted as blue dots at the base of each flow vector arrow. The right-most column show the ground truth flow vectors - plotted in black.

and pixel radius we perform a KD-tree K-nearest neighbour search using the $search\_radius\_vector\_3d()$ function from the library Open3D [8]. This returns all events within the search volume sorted by distance. These events are then filtered such that only the nearest (L2-distance) K events are kept. For the new event the formed edges are added to the graph and for previously generated events the edges are updated if the new event becomes one their K-nearest neighbours. After this point, all of the node embeddings in each of the L layers are updated by applying the graph convolutional layers as detailed in the main paper - the multi-layer perceptron used to predict optical flow is not executed. The graph convolutions are implemented using the PyTorch geometric framework [3]. The number MAC required to update all of the event node embeddings is counted. At this point, the timer is stopped and the execution time and the total number of MAC operations are recorded. In order to compute the MAC operations per second (MAC/s), the average number of MAC required per graph update is multiplied by the average event-rate over the test data.

Sub-graph Sparse: The sparse method is equivalent to the full method described above besides the fact that only a subset of node embeddings are updated. This is based on the method described in [7] and makes us of the function $k\_hop\_subgraph()$ from PyTorch geometric [3]. Given a graph node, this function returns the nodes to which it is connected in the graph over K-hops over intermediate nodes. Over L iterations (i.e., for L layers) all of the nodes found within an increasing number of hops (1,2,3,4 for five graph convolutional layers) from the newly arrived event as

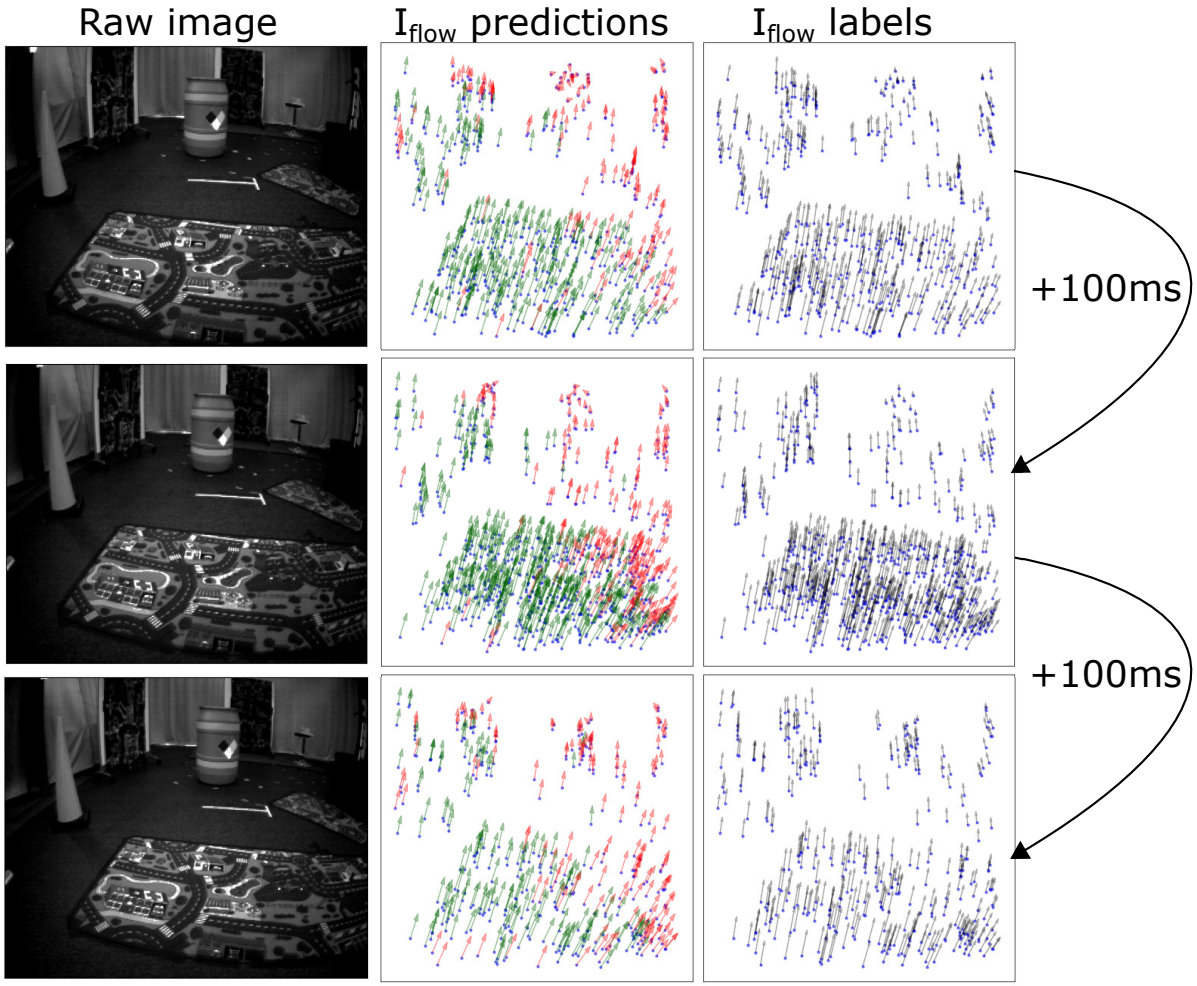

Figure 5. **Qualitative example on MVSEC**. The figure is composed to a three by three grid of imagettes. The left-most column features the raw images recorded at intervals of 100ms at a particular point of one of the MVSEC indoor test sequences. The central column show optical flow vectors predicted by HUGNet. These predicted flow are shown using arrows that correspond to the sum of the predicted x and y components. These arrows are coloured either green or red corresponding to whether they are inferior to, or exceed, the flow accuracy at 25%. Events are plotted as blue dots at the base of each flow vector arrow. The right-most column show the ground truth flow vectors - plotted in black.

well as any event with updated edges within a single time and pixel radius are found. The node embedding, in a layer l, of each concerned node is then updated. This means not all graph convolutional layers are required to be applied to all nodes - reducing considerably the number of MAC operations per second.

Sub-graph HUGNet: Unlike the previous two methods, the HUGNet approach does not require an existing graph as a starting point. Rather, from a sampled event within the raw 3D-pointcloud, the subset of past events within one time and pixel radius are isolated from the full pointcloud. The timer begins at this point. A KD-tree search is then used to find the K-nearest neighbours as described above. The K node embeddings of the these events are then directly used to calculate only the L node embeddings of the newly arrived event only using PyTorch. The timer is then stopped. As for the other codes, this code is repeated 1000 times and an average graph update latency is taken.

References

[1] Event signal processing. https://docs.prophesee.ai/stable/hw/manuals/esp.html. Accessed: 2021-10-22. 1

[2] Spatiotemporal contrast algorithm. https://docs.prophesee.ai/stable/metavision_sdk/modules/cv/python_api/bindings.html#metavision_sdk_cv.SpatioTemporalContrastAlgorithm. Accessed: 2021-10-24. 1

[3] Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric. *Proceedings of the International Conference on Learning Representation*, 2019. 4

[4] Yuhuang Hu, Shih-Chii Liu, and Tobi Delbruck. v2e: From video frames to realistic dvs events. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1312–1321, 2021. 1

[5] Damien Joubert, Alexandre Marcireau, Nic Ralph, Andrew Jolley, André van Schaik, and Gregory Cohen. Event camera simulator improvements via characterized parameters. *Frontiers in Neuroscience*, page 910, 2021. 1

[6] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015. 1

[7] Simon Schaefer, Daniel Gehrig, and Davide Scaramuzza. Aegnn: Asynchronous event-based graph neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12371–12381, 2022. 2, 4

[8] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. Open3d: A modern library for 3d data processing. *arXiv preprint arXiv:1801.09847*, 2018. 4