Graph-based Asynchronous Event Processing for Rapid Object Recognition

Yijin Li, Han Zhou, Bangbang Yang, Ye Zhang, Zhaopeng Cui, Hujun Bao, Guofeng Zhang*
State Key Lab of CAD&CG, Zhejiang University†

Abstract

Different from traditional video cameras, event cameras capture asynchronous events stream in which each event encodes pixel location, trigger time, and the polarity of the brightness changes. In this paper, we introduce a novel graph-based framework for event cameras, namely SlideGCN. Unlike some recent graph-based methods that use groups of events as input, our approach can efficiently process data event-by-event, unlock the low latency nature of events data while still maintaining the graph's structure internally. For fast graph construction, we develop a radius search algorithm, which better exploits the partial regular structure of event cloud against k-d tree based generic methods. Experiments show that our method reduces the computational complexity up to 100 times with respect to current graph-based methods while keeping state-of-the-art performance on object recognition. Moreover, we verify the superiority of event-wise processing with our method. When the state becomes stable, we can give a prediction with high confidence, thus making an early recognition.

1. Introduction

Rapid object recognition is essential for a variety of applications, such as autonomous driving and flying drones. For instance, when an autonomous vehicle is driving at high speed, the low latency is desirable to identify obstacles or moving objects once they appear. Due to its low frame rate, the standard video camera is not ideal for this task. Fast-speed video cameras can have more than 1000 frames per second, while they are normally very expensive and the information is also highly redundant. As a result, event cameras [3, 28, 16] attracts more attention recently due to their high temporal resolution and low latency (both in the order of microseconds) as well as high dynamic range without motion blur. Compared with video cameras that output images with a specific frame rate, event cameras are event-driven. When a certain brightness change occurs on a pixel, the event camera will trigger an individual event. In this way, they naturally discard redundant information by only measuring brightness.

However, since the output of an event camera is a sparse asynchronous events stream, existing efficient methods [15, 37] which typically work on frames can not be directly applied for event cameras. As a result, most works [11, 8, 33] transform such events stream to regular 2D event frames or 3D voxel grids before processing. However, these data representation transformations discard the sparsity of events data and quantify event timestamps, which are likely to obscure the natural invariance of the data Another type of approach is directly tailored to the sparse and asynchronous nature of event-based data. Time-surface-based methods [17, 36] and Spiking Neural Networks (SNNs) [27, 19, 1] are two dominant classes of methods for event-by-event processing. Despite keeping low latency, both methods have limited accuracy in high-level tasks, mainly due to their sensitivity to tuning and difficulty in the training procedure, respectively. To fully utilize the spatial-temporal sparsity of event data, some recent methods [41, 34, 4, 22] introduce a compact graph representation that interprets an event sequence as a graph on event cloud and employs graph convolutional networks. Although these graph-based methods, *e.g.*, [4, 22], reach state-of-the-art performance, they rely on integrating events over a certain number of events or events within a period. They gather the information contained in groups of events at the cost of discarding the low latency nature of events data.

Based on all these observations, in this paper, we propose a novel graph-based recursive algorithm with a sliding window strategy that can process the stream event-by-event efficiently while maintaining high accuracy. However, it is non-trivial to apply the sliding window strategy for graph-based and event-wise processing. The naive sliding window strategy is inefficient because it needs to process all the nodes in the graph even with a minor change, although many nodes' features don't change. Moreover, graph construction is prerequisite for graph neural networks, and the radius search is normally adopted [4, 22] to determine

*Corresponding author: Guofeng Zhang.
†Emails: {eugenelyj, hanzhou, ybbbbt, yezhang509, zhpcui, baohujun, zhangguofeng}@zju.edu.cn. The authors except Zhaopeng Cui are also affiliated with ZJU-SenseTime Joint Lab of 3D Vision. This work was partially supported by NSF of China (No. 61822310 and 61932003).

nodes' connection, which is very slow. Take the k-d tree based search as an example, frequent insertion and deletion will make it unbalanced and cause query performance to drop while rebuilding the index will bring the extra cost to insertion.

To solve these problems, we first propose a novel incremental graph convolution, namely slide convolution, that exploits the local spatial connectivity of convolution and reuses previous calculations in order to avoid processing all nodes. For a single layer, it is rather simple to just compute the features around newly added nodes. For multiple-layer GCN, we need to solve the propagation of modified features between layers with different graph topologies. Thus we derive a series of propagation rules. In this way, we reduce the computational complexity up to 100 times in comparison with the naive sliding window strategy. Moreover, considering that events locate in the image grid (which consists of two limited and discrete dimensions) rather than generic 3D continuous metric spaces, we introduce a novel radius search algorithm for the structure of event cloud, cutting the search cost by half and reducing the cost of insertion and deletion operations to O (1).

A straightforward application of event-wise processing is early object recognition, as when enough information is received, the prediction result becomes stable and it is not necessary to process more events. Previous works either focus on how to process event-by-event efficiently or reach a certain level of accuracy with less information, but lack the ability of early recognition. In this paper, we further apply our graph-based recursive method to early object recognition by designing a state-aware module to predict whether it reaches the stable state. In this way, we can enable accurate recognition with confidence as early as possible. To the best of our knowledge, we are the first ones to verify the superiority of event-wise processing in early object recognition.

To summarize, the contributions of this paper are as follows:

- We propose a novel graph-based recursive algorithm that enables efficient event-wise processing for event cameras.
- We introduce a novel incremental graph convolution for event-wise processing. It reduces the computational complexity up to 100 times compared to the naive sliding-window-based graph convolution.
- We propose an event-specific radius search algorithm that reduces query and insertion/deletion costs to make graph construction faster.
- Experiments demonstrate that our efficient event-wise algorithm achieves similar performance with batch-wise methods on standard recognition task while enabling early object recognition with confidence.

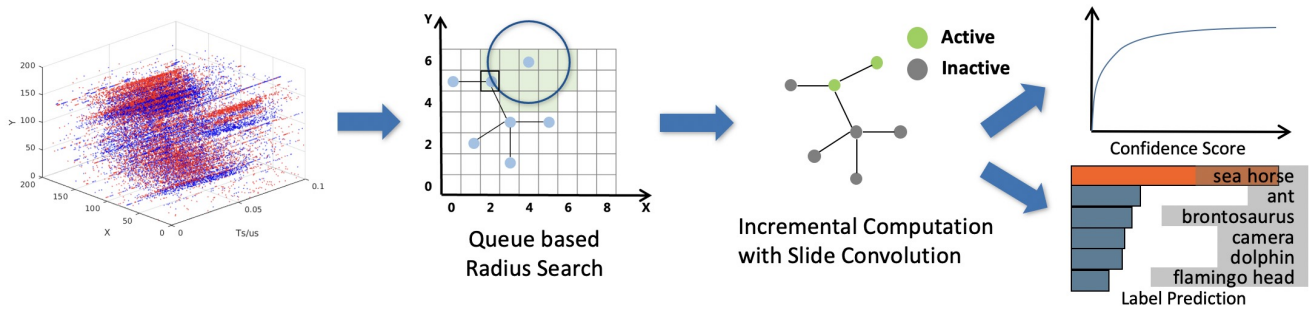
2. Related Work

Here we review the existing representation for event-based data in three parts: (1) event-specific representation; (2) event images and voxel grid; (3) point set and graph.

Event-Specific Design. Traditional methods have designed an event-specific representation, namely time surface [17], which is manifested as a 2D map formed with the timestamps of the most recent events. Typically followed by a lightweight model, this representation can be easily updated with each newly arrived event, unlocking the low latency advantage of the event camera. Time surface has been applied in different tasks, *e.g*., stereo event-based SLAM [44] and image reconstruction [24]. While many variants [36, 20] have been developed, their performance degrades on highly textured scenes [23] due to the "motion overwriting" problem. Another type of method tailored for event cameras adopts the Spike Neural Network (SNN) [19, 25, 27, 1], which is also bio-inspired designed like event cameras. SNN exploits the sparse and asynchronous nature of events data, but training such networks is difficult due to their non-differentiable character.

Event Images and Voxel Grid. These methods [8, 11, 33] try to make event data compatible with frame-based technologies. Earlier approaches use simple ways (*e.g*., counting events or accumulating pixel-wise polarity) to convert the event stream into 2D event frames [9]. Such event frames, which reveal spatial information of scene edges, have been applied to several tasks, *e.g*., visual odometry [32], feature tracking [12]. However, they quantify the timestamp and discard the sparsity property of events data. To improve the temporal resolution, Zhu *et al*. [45, 40] suggest discretizing the time dimension into consecutive temporal bins. They accumulate events into a voxel grid through a linearly weighted accumulation similar to bilinear interpolation. Messikommer *et al*. [21] further exploit spatial and temporal sparsity by adopting sparse convolution [13] and developing a recursive convolution formula. However, their operations are still on sparse volumes. It's challenging for them to process vast event clouds due to the expensive computation cost of 3D convolution.

Point Set and Graph. Ryad *et al*. [2] solve optical flow estimation by plane fitting to the event point cloud, an early work that interprets an event sequence as 3D point clouds. Recent works, for example, Wang *et al*. [41] further use a PointNet [29, 30]-like framework, which utilizes multi-layer-perceptron to learn features of each point separately, and then outputs object-level responses (*e.g*., classification labels) through global max operations. For event-wise processing, Sekikawa *et al*. [34] first develop a recursive architecture, namely EventNet. Specifically, it formulates de-



(a) Incoming Event Stream (b) Graph Construction (c) Graph Convolution (d) Network Prediction

Figure 1. **Our graph-based Asynchronous Event Processing Framework.** It can efficiently process in an event-wise manner and enable early object recognition, which is mainly thanks to (b) an event-specific radius search algorithm for graph construction, (c) incremental graph convolution for efficient event-wise processing, and (d) bottom branch for object recognition prediction and top branch, *i.e.*, a state-aware module predicting whether it reaches the stable state.

pendence on causal events to the output recursively using a novel temporal coding and aggregation scheme and precomputes the node features corresponding to specific spatial coordinates and polarities. However, due to its approximate calculation and the lack of hierarchical architecture, extending EventNet to other high-level tasks is challenging. To better exploit the topological structure, [43, 4] interpret the event cloud in the form of space-time graphs. In particular, Bi *et al*. [4] show that such compact graph representation requires less computation and memory than conventional CNNs while achieving superior results to the state-of-the-art in various datasets. Mitrokhin *et al*. [22] show that capturing the changes over large time intervals can resolve motion ambiguities. However, such a large time interval will result in a very low response frequency. A recursive formula for graph-based processing needs to be studied, which motivates this paper.

3. Preliminaries

We first introduce how to build a graph from the event stream. Then we introduce spatial graph convolution, which is the basis of our slide convolution.

3.1. Event Graph

Event cameras respond to changes in the logarithmic brightness signal $L(u_i, t_i) \doteq logI(u_i, t_i)$ asynchronously and independently for event pixel [10]. An event is triggered at pixel $u_i = (x_i, y_i)$ and at time t_i as soon as the brightness increment since the last event at the pixel reaches a threshold $\pm C$ (with $C > 0$):

$$L(u_k, t_k) - L(u_k, t_k - \Delta t) \geq p_k C, \tag{1}$$

where $p_i \in \{-1, 1\}$ is the polarity of the brightness change and Δt is the time since the last event at u_i. An asynchronous event stream can be expressed as a sequence of events:

$$\{\text{event}_i\}_B = \{x_i, y_i, t_i, p_i\}_B, \tag{2}$$

where B is the length of events sequence.

From an event stream, we can construct a graph which is denoted as $G = \{V, E\}$ where V and E represent nodes and edges, respectively. Each event is a node in the event graph, which contains a 3D coordinate (x_i, y_i, t_i) and nodes attribute (p_i). It is also possible to remove or include additional attributes like event-surface normals.

The connectivity of nodes in the graph is usually established by the radius-neighborhood graph strategy. Namely, neighboring nodes v_i and v_j are connected with an edge only if their weighted Euclidean distance $d_{i,j}$ less than radius distance R. Before radius search, the temporal axis of the event cloud is upscaled by a factor to keep the density of events more uniform across the x, y, t axes. Each edge has its own attribute e_{ij}, which is often computed by relative Cartesian coordinates of linked nodes. To limit the size of the graph, the connectivity degree for each node is usually constrained to a parameter D_{max}.

3.2. Spatial Graph Convolution

Spatial Graph Convolution [14, 39] works by constructing a local neighborhood graph and applying convolution-like operations on the edges connecting neighboring pairs of points. Formally, it aggregates a new feature vector for each vertex using its neighborhood information weighted by a trainable kernel function. By using summation as the aggregation operation, it can be defined as:

$$\begin{aligned} (f \otimes g)(i) &= \sum_{j \in E(i)} f(j) h_\theta, \\ h_\theta &= h_\theta(f(i), f(j), e_{ij}), \end{aligned} \tag{3}$$

where $\otimes$ is the graph convolution operator, g is kernel function, f is node feature. $E(i)$ is the set of node i's neigh-

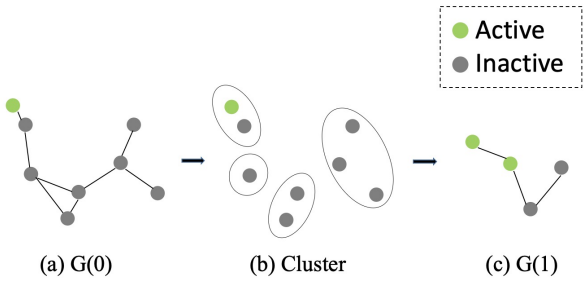


Figure 2. **An example of the propagation of modified features between different layers.** (a) A newly active node in layer 0 (denoted as G(0)). The active state means that it needs to be updated. (b) Graph pooling which causes a change in the topology. (c) Active nodes in G(1).

bor and h_θ is a function determining how the features are aggregated by making use of two node features and edge attributes.

4. Method

Inspired by [34, 21], we develop a recursive formula for spatial graph convolution, namely slide convolution. Specifically, slide convolution takes events one by one as input and responds in an event-wise manner while maintaining the structure of the past graph internally (Section 4.1). To make graph construction faster, we develop a radius search algorithm which better exploits the structure of events cloud against generic 3D continuous metric space (Section 4.2). In Section 4.3, we introduce how to apply our graph-based method to early object recognition by combining it with a state-aware module. Fig. 1 overviews the proposed pipeline.

4.1. Slide Convolution

To enable spatial graph convolution (denoted as convolution for short in the following) work in an event-wise manner, one straightforward idea is to use a sliding window strategy, i.e., consecutively updating the graph by sliding new events in and sliding events out, then apply convolution on the full graph. This way, however, is infeasible because it requires processing the entire window of events again and again at a high event rate. A simple way to improve it will be just computing the features around the newly active or inactive nodes (corresponding to the events of sliding in and sliding out). But it only works for the case of single layer, while current modern architecture usually contains multiple layers, which even cause a change in graph topology (Fig. 2). Slide convolution solves these problems by deriving a series of propagation rules which helps to propagate the changes from the input layer to deeper layers. The following will focus on how to derive these propagation rules.

Firstly we rewrite the convolution in a multi-layer architecture as:

$$f_{n+1}(i) = \sum_{j \in N(i)} f_n(j) h_\theta, for\ i \in A_{n+1}, \tag{4}$$

where f_n and f_{n+1} are node features at layer n and $n+1$ respectively (layer 0 is input layer). A_{n+1}, namely existing set, represents all existing nodes in the graph at layer $n+1$ (which will change with different sliding windows). $N(i)$ is a map that stores which nodes at layer n contribute to node i at layer $n+1$. Here for convolution, $N(i)$ is one-hop neighbour of node i.

Eq. (4) leads of course to redundant computation. We seek to leverage the temporal sparsity of the event stream, *i.e.*, some nodes stay same values at two consecutive timestamp, for efficient computing, which has the following form:

$$\begin{aligned} f_{n+1}^{t+1}(i) &= f_{n+1}^{t}(i) + \Delta_{n+1}(i), \\ \Delta_{n+1}(i) &= \sum_{(j,i) \in E_{n+1}} (f_n^{t+1}(j) - f_n^t(j)) h_\theta, \end{aligned} \tag{5}$$

The most critical part is E_{n+1}, a set of directed edges containing all the edges that point to modified nodes. If we know E_{n+1}, we can calculate the change of features at time $t+1$ compared to that at time t, *i.e.*, Δ_{n+1} and update the node features.

Notice that for newly active nodes, their states at time t are undefined (similar for newly inactive nodes). In order to distinguish these nodes, we divide nodes that need to be updated into three categories: the ones deleted from the graph, the ones newly added to the graph, and the nodes that locate in the receptive fields of these two types of nodes. We use V^{del}, V^{add}, V^{up} to represent these three kinds of nodes and further split the E into E^{del}, E^{add}, E^{up} according to which node it points to.

At time $t+1$, for layer 0, V_0^{add}, V_0^{del}, V_0^{up} is initialized as events sliding into the window, events sliding out of the window and an empty set, respectively. E_0 is initialized as an empty set. Then we can deduce layer $n+1$ through simple set operations when layer n is given:

$$\begin{aligned} &V_{n+1}^{add} = V_n^{add}, V_{n+1}^{del} = V_n^{del}, \\ &V_{n+1}^{up} = \{i \mid for\ i \in A_{n+1}^t \setminus V_{n+1}^{del},\ if\ \exists j \in N(i) \wedge j \in V_n\}, \\ &V_{n+1} = V_{n+1}^{add} \cup V_{n+1}^{del} \cup V_{n+1}^{up}, \\ &A_{n+1}^{t+1} = A_{n+1}^t \cup V_{n+1}^{add} \setminus V_{n+1}^{del}, \end{aligned} \tag{6}$$

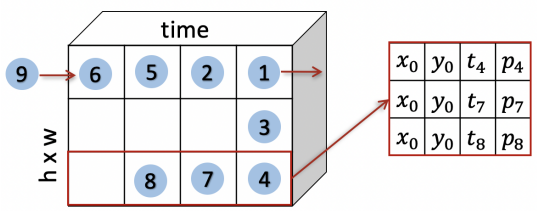


Figure 3. The **pixel queue** stores events, where each event is a four-tuple. The numbers on the ball represent the order of events (figure adapted from [38]).

$$
\begin{aligned}
E_{n+1}^{del} &= \{(j,i) \mid for\ i \in V_{n+1}^{del}\ then\ \forall j \in N(i) \wedge j \in A_n^{t+1}\},\\
E_{n+1}^{add} &= \{(j,i) \mid for\ i \in V_{n+1}^{add}\ then\ \forall j \in N(i) \wedge j \in A_n^{t+1}\},\\
E_{n+1}^{up} &= \{(j,i) \mid for\ i \in V_{n+1}^{up}\ then\ \forall j \in N(i) \wedge j \in V_n\},\\
E_{n+1} &= E_{n+1}^{add} \cup E_{n+1}^{del} \cup E_{n+1}^{up},
\end{aligned}
\tag{7}
$$

In Eq. (6), V_{n+1}^{add} and V_{n+1}^{del} are directly inherited from last layer since the convolution does not change graph topology. Notice that each time V_{n+1} and E_{n+1} is built from state at previous layer, so we do not need to keep their state at previous moment. A_{n+1}, on the contrary, evolved from the state at previous moment, thus we need to distinguish between A_{n+1}^t and A_{n+1}^{t+1}. After reducing V and E, we also need to augment those undefined nodes for calculating $\Delta_{n+1}(i)$. Specifically, we expand feature map f_{n+1}^t and assign to zeros for V_{n+1}^{add} while for V_{n+1}^{del}, we assign f_{n+1}^{t+1} to zeros. Now we know how to derive E_{n+1}, but we can only deal with convolution. To extend to pooling operations, we need to know the corresponding set of neighborhoods $N(i)$. Take voxel grid pooling as an example. For nodes located in the same voxel (denoted as a set $S(voxel)$), it will be clustered to a center node. Thus for this center node, its corresponding $N(i)$ is $S(voxel)$.

In the supplementary, we prove that using Eq. (5) to process events one by one is equivalent to using Eq. (4) to process all events at once. In this way, we can replace spatial convolution with our sliding convolution, leveraging existing graph-based architecture to process events one by one efficiently without sacrificing accuracy. Note that though we use summation for aggregation in the above formulation, it is easy to replace summation by another way, such as max/min aggregation, as long as we know how E_{n+1} evolves.

4.2. Pixel Queue based Graph Construction

Recent graph-based methods on event cameras define node connectivity in the graph based on the radius-neighborhood graph strategy, namely radius search. It is usually done by k-d tree, a space-partitioning data structure for organizing points in a k-dimensional space. However, we argue that k-d tree does not leverage the structure of the event cloud. What's worse, k-d tree is inefficient when frequent insertions and deletions occur. This is because adding points will make the tree unbalanced, resulting in performance degradation. The same is true for deletion.

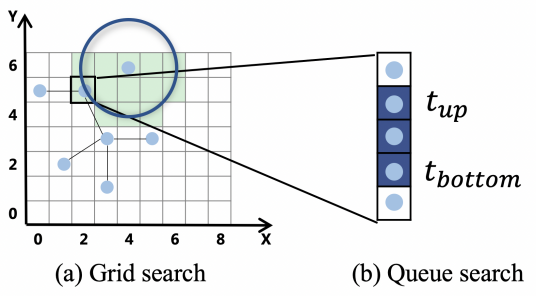


Figure 4. **A two-stage radius search based on pixel queue**. To search events given the radius, we first determine candidate pixels in the image grid, as shown in (a). Candidate pixels are represented in lime. Secondly, for each pixel queue, we collect events between lower bounds t_{bottom} and upper bounds t_{up}, as shown in (b).

Instead of k-d tree, we employ pixel queue (See Fig. 3) to store events, which is an event-specific data structure that has been used in a number of works [38, 44, 43]. Pixel queue stores the most recent events at each location sorted by the time of their arrival. Based on pixel queue, we propose a two-stage radius search algorithm (see Fig. 4). The first step is to search in the image grid and filter out the candidate pixel queues that contain the events we want. It can be done with the help of distance field, which describes how far the other pixels away from an anchor pixel. The distance field is similar to the partitioning technology used in k-d tree but only needs to be calculated once. For a query event (x_0, y_0, t_0) and the radius R, we can determine candidate pixels and corresponding queues whose spatial distance is smaller than the radius by looking up the pre-computed distance field.

In the second step, we traverse these candidate pixel queues. For a candidate pixel queue with spatial offset $(\delta x, \delta y)$ from the query event, the target events contained in it must have lower bound $t_0 - \sqrt{R^2 - (\delta x^2 + \delta y^2)}$ (denoted as t_{bottom}) and upper bound $t_0 + \sqrt{R^2 - (\delta x^2 + \delta y^2)}$) (denoted as t_{up}). We get these events by finding the index of t_{up} and t_{bottom} using binary search. Collecting events in all candidate pixel queues gives a final query result. We evaluate the performance of our method compared to the k-d tree based method in Section 5.1 and analyze the computation complexity in the supplementary.

4.3. State-aware Module

With the increase of input events and information, the prediction result will be stable from a certain moment. By this time, it is meaningless to process more events, and the system should give an early recognition result. We fulfill this goal with the help of our state-aware module, *i.e*., top branch in Fig. 1-(d). Specifically, we use a multi-layer perceptron, *i.e*. MLP, to represent a state-aware function which maps the graph feature map to a binary prediction. The prediction result means whether it achieves the stable status. Then during inference, we interpret the value after activation as the confidence score.

Given the object recognition branch pre-trained, we can generate ground truth labels for training state-aware module. One possible way is to analyze prediction as a function of event index. When the prediction does not change with event index increasing, we consider it stable and define the ground truth as one, otherwise, we define it as zero. For simplification, we adopt an approximate approach, *i.e*., considering a prediction stable if it is equal to that at the last event index. The ground truth will be compared with network prediction by Binary Cross Entropy with Logits Loss. As for the training data, we randomly crop sequences to variable lengths (from 5ms to 50ms) and hope it learn to predict corresponding confidence in different states. It is noteworthy that we do not crop sequences when we train the object recognition branch.

5. Experiments

5.1. Object Recognition

(1) Dataset and Evaluation Metrics. Event-based object recognition is to predict the category of the object through the input event stream. We verify our method's superiority on four datasets, including N-Caltech101 [26], N-Cars [36], CIFAR10-DVS [18], and MNIST-DVS [35].

N-Caltech101, CIFAR10-DVS, and MNIST-DVS are converted from standard frame-based datasets. This is done by displaying a moving image on a monitor and recording with a fixed event camera or fixing the monitor instead of the camera. The same as the original Caltech101, N-Caltech101 contains 8246 samples and 101 categories. CIFAR10-DVS, on the contrary, randomly select one-sixth of the original frame-based image dataset, thus containing 6,000 samples per class and 60,000 samples in total. MNIST-DVS consists of 10,000 symbols sampled from the standard MNIST 70,000-picture database, with each of the 10,000 symbols displayed at three different scales, thus containing 30,000 samples in total. Different from the former, N-Cars are created by directly recording objects in real-world environments with an event camera. N-Cars comprises two class labels, namely 12,336 car samples and 11,693 non-car samples (background). We sample some sequences from these datasets for visualization in the supplementary material.

To evaluate task performance and the potential ability of event-by-event processing, we consider two metrics: prediction accuracy and floating point operations per second (FLOPs). While the first indicates the quality of the prediction, the second shows the computational complexity required for each event update.

(2) Implementation Details. We implement two graph convolution networks to evaluate our SlideGCN. Adapting from [4], our first architecture, namely NVS, consists of two parts: a backbone and a prediction head. The backbone is comprised of 4 "GraphConv-ELU-Bn" layers, where "GraphConv" will be replaced by our slide convolution during inference. The prediction head comprises one fully connected layer to map the features to classes. In order to reduce overfitting, a dropout layer with a probability of 0.3 is added after the first fully connected layer. Compared with the original architecture, we replace the cluster-level pooling layer with a readout function, which summarizes the graph-level representation by taking both the max/mean of hidden representations of sub-graphs [42]. The second architecture, which is inspired by [22], is named EvS . Following the idea from [22], we used two constraints when computing edges for EvS. One of them is to keep only the points lying in the upper (along the temporal axis) hemisphere of a point, and the other is to filter the edges so that they are parallel to the event surface. This preserves most of an event's temporal motion information while obtaining sparser edges. We also add the normal of the event surface to the input. We use these two networks as our baseline and then replace their convolution layers with our slide convolution during evaluation. Please refer to the supplementary material for more details about the parameters like time interval and network depths.

(3) Comparison to the State-of-the-Art. Table 1 compares our results with other state-of-the-art methods. All these methods are able to process event stream event-by-event. Thanks to the effective expression of the graph structure, the EvS(baseline) we implemented has reached state-of-the-art on the challenging datasets N-Caltech101 and CIFAR10-DVS. The graph convolution layer is further replaced with our slide convolution, namely EvS(SlideGCN), which reduces the computational complexity up to 100 times without sacrificing the original performance. Another method, *i.e*., NVS does not perform on par with state-of-the-art, but it is lightweight and requires less computation.

Our method strikes a balance between event-specific low-latency and high-performance high-latency methods. On the one hand, it has less calculation (11.5 vs. 202 on N-Caltech101 and 33.2 vs. 103 on CIFAR10-DVS) than other second-best methods. On the other hand, it achieves 15.6% higher accuracy on N-Caltech101 and 29.7% higher

Methods	Representation	N-Caltech101		CIFAR10-DVS		MNIST-DVS		N-Cars	
		Acc ↑	Mps/ev ↓	Acc ↑	Mps/ev ↓	Acc ↑	Mps/ev ↓	Acc ↑	Mps/ev ↓
H-First [27]	Spike	0.054	-	0.077	-	0.595	-	0.561	
Gabor-SNN [6, 36]	Spike	0.196	-	0.245	-	0.824	-	0.789	-
HOTS [17]	TimeSurface	0.210	54.0	0.271	26	0.803	26	0.624	14.0
HATS [36]	TimeSurface	0.642	4.3	0.524	0.18	0.984	0.18	0.902	0.03
DART [31]	TimeSurface	0.664	-	0.658	-	0.985	-	-	-
YOLE [7]	VoxelGrid	0.702	3659	-	-	0.961	-	0.927	328.16
Asynet [21]	VoxelGrid	0.745	202	0.663	103	0.994	112	0.944	21.5
NVS-B (Ours)	Graph	0.670	221	0.602	601	0.986	154	0.915	57.9
NVS-S (Ours)	Graph	0.670	7.8	0.602	22.8	0.986	10.1	0.915	5.2
EvS-B (Ours)	Graph	0.761	1152	0.680	3020	0.991	548	0.931	251
EvS-S (Ours)	Graph	0.761	11.5	0.680	33.2	0.991	15.2	0.931	6.1

Table 1. **Comparison with different representations for object recognition.** We color code each row as best, second best and third best. *-B means baseline and *-S means SlideGCN. Our graph-based baseline (EvS-B) achieves the state-of-the-art-performance (i.e. 0.761 on N-Caltech101 and 0.680 on CIFAR10-DVS). Replacing by our slide convolution, the computational complexity reduces up to two orders of magnitude (1152 vs. 11.5 and 3020 vs. 33.2). As a result, our method (EvS-S) strikes a balance between event-specific low-latency and high-performance high-latency methods.

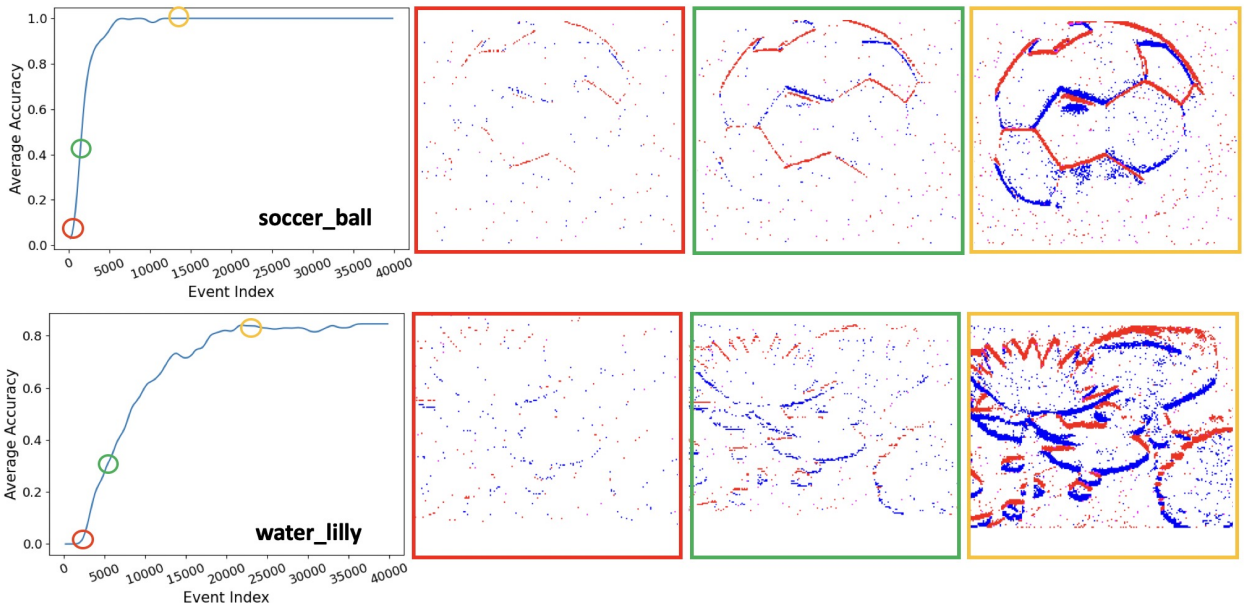


Figure 5. **Qualitative results of stream-based object recognition.** The accuracy improves and stabilizes as events accumulate. We selected points at three moments in this process, which are circled by red, green, and orange.

accuracy on CIFAR10-DVS, than hand-crafted and event-specific methods, *i.e.*, HATS [36], which have a low-level computation complexity.

Besides the computation complexity, we timed our experiments conducted on N-Caltech101 by measuring the processing time for each event update on an i7-9700K CPU (using a single core). Our method requires 16.9 ms, while the baseline needs 130.4 ms. Therefore, our method is roughly 8 times faster by reusing previous calculations. We expect that our method will significantly reduce the running time on the GPU or specific hardware as its lower number of FLOPs. Please refer to our supplementary material for more comparisons with the methods that cannot efficiently process event data.

(4) Efficiency of Pixel Queue based Radius Search. We evaluate the efficiency of our pixel queue based radius search with nanoflann [5] , which is a popular k-d tree implementation supporting dynamic update. Specifically, we use a window of 100,000 events, and each time slide by 100 events, followed by a radius search on the newly sliding-in events. We repeat sliding 1,000 times and show the cumulative cost of insertion, deletion, and searching in Fig. 6.

The insertion and deletion cost of nanoflann is an order of magnitude higher than our method on average, and

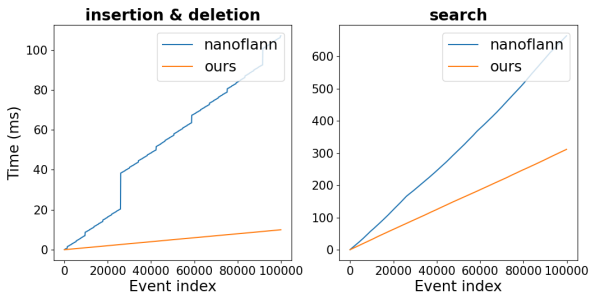


Figure 6. **Comparison of our radius search method and k-d tree based method** (with nanoflann implementation).

it rises rapidly sometimes. That is because nanoflann uses lazy deletion, which does not rebuild the index immediately after removing elements. The cost will have a significant rise when it rebuilds the index. As for searching, our method reduces the time cost by half by leveraging the spatial locality of the event cloud.

5.2. Stream based Object Recognition

Method	Batch size (ms)	Accuracy	Index
Batch-wise	10	0.362	3829
	20	0.490	7611
	30	0.615	11041
	40	0.718	14771
	50	0.761	19154
SlideGCN	12^*	0.669	3118

Table 2. **Comparison of SlideGCN and batch-wise method on stream based object recognition.** 12^* means that time consuming is 12 ms, not batch size.

Methods	Size	Cum MFLOPS	Avg MFLOPS	Index
SlideGCN	1	17711	5.68	3118
	10	8936	28.58	3127
	100	3041	95.64	3170
Batch-wise	-	1152	-	19154

Table 3. **Cumulative MFLOPS with different mini-batch sizes.**

We designed the stream based object recognition task to verify the effectiveness of our event-wise processing. In this task, we evaluate the prediction accuracy when the algorithm claims it gives a reliable response. Specifically, we consider it a reliable result for our event-wise method when our state-aware module provides a high confidence score. While for the batch-wise method, we consider that the confidence of each processing is equal to one. In this way, we measure the accuracy and the latency at the same time.

(1) Comparison to the batch-wise way. We use a window configuration of 50 milliseconds to train the network. Using the same network, we test in the batch-wise way and event-wise way (by replacing with our slide convolution) separately. As shown in Table 2, for the batch-wise way, decreasing batch size reduces latency but also causes drops in accuracy. On the contrary, our SlideGCN performs close accuracy (0.669 vs. 0.761) with the best configured batch-wise manner but response much earlier (3118 vs. 19154 in terms of event index and 12ms vs. 50ms in terms of time). Here we analyze how our method works. In Fig. 5 we show how the accuracy increases with cumulative events for two kinds of objects. The curves vary because the texture richness of the objects is different. Different kinds of motion also cause discrepancies. As a result, it is not trivial to tune a perfect batch size for batch-wise methods. While choosing a big batch size ensures high accuracy (a big batch size means that it receives enough information for most of the objects), it requires too many events as input. Choosing a small batch size, on the contrary, can not guarantee to receive enough information for many objects, thus resulting in low accuracy. Unlike the batch-wise way, our method is not limited to fixed batch size and works in an event-wise manner. Combined with our state-aware module, it can process event-by-event, predict a confidence score simultaneously. As soon as it detects a stable state with high confidence, we can stop processing the following events and make an early recognition.

(2) Trade-off between latency and computational effort. In practice use, there is a trade-off between the latency and the computation load. Event-wise processing minimizes the latency of the data, but it also makes the computation load very large. We declare that our method is not limited to event-wise processing but can be extended to mini-batch and batch-wise. In Table 3 we compare cumulative MFLOPS (denoted as Cum MFLOPS) with different mini-batch sizes. The table shows that although the event-wise method owns the lowest latency, it causes higher cumulative FLOPs. Increasing the mini-batch size will reduce cumulative FLOPs but in the cost of bringing more latency (an extreme case is that the entire window is used as a batch). In practical use, we can set a mini-batch size of 100 to achieve a balance between computation load and latency.

6. Conclusion

In this paper, we introduce a novel graph-based recursive algorithm for event cameras, which is able to keep a high performance of graph convolution networks as well as the ability of event-by-event processing. To achieve this, we propose a novel incremental convolution method that significantly reduces computational complexity compared to the naive sliding window strategy. To make graph construction faster, we also exploit the structure of the events cloud and develop an event-specific radius search algorithm based on pixel queue. The experiments demonstrate that our efficient event-wise algorithm achieves similar performance with batch-wise methods on standard recognition task while enabling early object recognition with confidence.

References

[1] Arnon Amir, Brian Taba, David J. Berg, Timothy Melano, Jeffrey L. McKinstry, Carmelo di Nolfo, Tapan K. Nayak, Alexander Andreopoulos, Guillaume Garreau, Marcela Mendoza, Jeff Kusnitz, Michael DeBole, Steven K. Esser, Tobi Delbrück, Myron Flickner, and Dharmendra S. Modha. A low power, fully event-based gesture recognition system. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 7388–7397. IEEE Computer Society, 2017. 1, 2

[2] Ryad Benosman, Charles Clercq, Xavier Lagorce, Sio-Hoi Ieng, and Chiara Bartolozzi. Event-based visual flow. *IEEE Trans. Neural Networks Learn. Syst.*, 25(2):407–417, 2014. 2

[3] Raphael Berner, Christian Brandli, Minhao Yang, Shih-Chii Liu, and Tobi Delbruck. A 240× 180 10mw 12us latency sparse-output vision sensor for mobile applications. In *2013 Symposium on VLSI Circuits*, pages C186–C187. IEEE, 2013. 1

[4] Yin Bi, Aaron Chadha, Alhabib Abbas, Eirina Bourtsoulatze, and Yiannis Andreopoulos. Graph-based object classification for neuromorphic vision sensing. In *Proceedings of IEEE/CVF International Conference on Computer Vision*, pages 491–501. IEEE, 2019. 1, 3, 6

[5] Jose Luis Blanco and Pranjal Kumar Rai. nanoflann: a C++ header-only fork of FLANN, a library for nearest neighbor (NN) with kd-trees. https://github.com/jlblancoc/nanoflann, 2014. 7

[6] Alan C. Bovik, Marianna Clark, and Wilson S. Geisler. Multichannel texture analysis using localized spatial filters. *IEEE Trans. Pattern Anal. Mach. Intell.*, 12(1):55–73, 1990. 7

[7] Marco Cannici, Marco Ciccone, Andrea Romanoni, and Matteo Matteucci. Asynchronous convolutional networks for object detection in neuromorphic cameras. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 1656–1665. Computer Vision Foundation / IEEE, 2019. 7

[8] Marco Cannici, Marco Ciccone, Andrea Romanoni, and Matteo Matteucci. A differentiable recurrent surface for asynchronous event-based data. In *Proceedings of European Conference on Computer Vision*, volume 12365 of *Lecture Notes in Computer Science*, pages 136–152. Springer, 2020. 1, 2

[9] Matthew Cook, Luca Gugelmann, Florian Jug, Christoph Krautz, and Angelika Steger. Interacting maps for fast visual interpretation. In *Proceedings of International Joint Conference on Neural Networks*, pages 770–776. IEEE, 2011. 2

[10] Guillermo Gallego, Tobi Delbrück, Garrick Orchard, Chiara Bartolozzi, Brian Taba, Andrea Censi, Stefan Leutenegger, Andrew J. Davison, Jörg Conradt, Kostas Daniilidis, and Davide Scaramuzza. Event-based vision: A survey. *CoRR*, abs/1904.08405, 2019. 3

[11] Daniel Gehrig, Antonio Loquercio, Konstantinos G. Derpanis, and Davide Scaramuzza. End-to-end learning of representations for asynchronous event-based data. In *Proceedings of IEEE/CVF International Conference on Computer Vision*, pages 5632–5642. IEEE, 2019. 1, 2

[12] Daniel Gehrig, Henri Rebecq, Guillermo Gallego, and Davide Scaramuzza. EKLT: asynchronous photometric feature tracking using events and frames. *Int. J. Comput. Vis.*, 128(3):601–618, 2020. 2

[13] Benjamin Graham, Martin Engelcke, and Laurens van der Maaten. 3D semantic segmentation with submanifold sparse convolutional networks. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 9224–9232. IEEE Computer Society, 2018. 2

[14] William L. Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Proceedings of Neural Information Processing Systems*, pages 1024–1034, 2017. 3

[15] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017. 1

[16] Giacomo Indiveri. Neuromorphic engineering. In *Springer Handbook of Computational Intelligence*, Springer Handbooks, pages 715–725. Springer, 2015. 1

[17] Xavier Lagorce, Garrick Orchard, Francesco Galluppi, Bertram E. Shi, and Ryad Benosman. HOTS: A hierarchy of event-based time-surfaces for pattern recognition. *IEEE Trans. Pattern Anal. Mach. Intell.*, 39(7):1346–1359, 2017. 1, 2, 7

[18] Hongmin Li, Hanchao Liu, Xiangyang Ji, Guoqi Li, and Luping Shi. Cifar10-dvs: an event-stream dataset for object classification. *Frontiers in neuroscience*, 11:309, 2017. 6

[19] Qianhui Liu, Haibo Ruan, Dong Xing, Huajin Tang, and Gang Pan. Effective AER object classification using segmented probability-maximization learning in spiking neural networks. In *Proceedings of AAAI Conference on Artificial Intelligence*, pages 1308–1315. AAAI Press, 2020. 1, 2

[20] Jacques Manderscheid, Amos Sironi, Nicolas Bourdis, Davide Migliore, and Vincent Lepetit. Speed invariant time surface for learning to detect corner points with event-based cameras. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 10245–10254. Computer Vision Foundation / IEEE, 2019. 2

[21] Nico Messikommer, Daniel Gehrig, Antonio Loquercio, and Davide Scaramuzza. Event-based asynchronous sparse convolutional networks. In *Proceedings of European Conference on Computer Vision*, volume 12353 of *Lecture Notes in Computer Science*, pages 415–431. Springer, 2020. 2, 4, 7

[22] Anton Mitrokhin, Zhiyuan Hua, Cornelia Fermüller, and Yiannis Aloimonos. Learning visual motion segmentation using event surfaces. In *Proceedings of IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14402–14411. IEEE, 2020. 1, 3, 6

[23] Elias Mueggler, Chiara Bartolozzi, and Davide Scaramuzza. Fast event-based corner detection. In *Proceedings of British Machine Vision Conference*. BMVA Press, 2017. 2

[24] Gottfried Munda, Christian Reinbacher, and Thomas Pock. Real-time intensity-image reconstruction for event cameras using manifold regularisation. *Int. J. Comput. Vis.*, 126(12):1381–1393, 2018. 2

[25] Garrick Orchard, Ryad Benosman, Ralph Etienne-Cummings, and Nitish V. Thakor. A spiking neural network architecture for visual motion estimation. In *Proceedings of IEEE Biomedical Circuits and Systems Conference (BioCAS)*, pages 298–301. IEEE, 2013. 2

[26] Garrick Orchard, Ajinkya Jayawant, Gregory Cohen, and Nitish V. Thakor. Converting static image datasets to spiking neuromorphic datasets using saccades. *CoRR*, abs/1507.07629, 2015. 6

[27] Garrick Orchard, Cedric Meyer, Ralph Etienne-Cummings, Christoph Posch, Nitish V. Thakor, and Ryad Benosman. Hfirst: A temporal approach to object recognition. *IEEE Trans. Pattern Anal. Mach. Intell.*, 37(10):2028–2040, 2015. 1, 2, 7

[28] Christoph Posch, Daniel Matolin, and Rainer Wohlgenannt. A QVGA 143 db dynamic range frame-free PWM image sensor with lossless pixel-level video compression and time-domain CDS. *IEEE J. Solid State Circuits*, 46(1):259–275, 2011. 1

[29] Charles Ruizhongtai Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 77–85. IEEE Computer Society, 2017. 2

[30] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *Proceedings of Neural Information Processing Systems*, pages 5099–5108, 2017. 2

[31] Bharath Ramesh, Hong Yang, Garrick Orchard, Ngoc Anh Le Thi, Shihao Zhang, and Cheng Xiang. DART: distribution aware retinal transform for event-based cameras. *IEEE Trans. Pattern Anal. Mach. Intell.*, 42(11):2767–2780, 2020. 7

[32] Henri Rebecq, Timo Horstschaefer, Guillermo Gallego, and Davide Scaramuzza. EVO: A geometric approach to event-based 6-dof parallel tracking and mapping in real time. *IEEE Robotics Autom. Lett.*, 2(2):593–600, 2017. 2

[33] Henri Rebecq, René Ranftl, Vladlen Koltun, and Davide Scaramuzza. Events-to-video: Bringing modern computer vision to event cameras. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 3857–3866. Computer Vision Foundation / IEEE, 2019. 1, 2

[34] Yusuke Sekikawa, Kosuke Hara, and Hideo Saito. Eventnet: Asynchronous recursive event processing. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 3887–3896. Computer Vision Foundation / IEEE, 2019. 1, 2, 4

[35] Teresa Serrano-Gotarredona and Bernabé Linares-Barranco. Poker-dvs and mnist-dvs. their history, how they were made, and other details. *Frontiers in neuroscience*, 9:481, 2015. 6

[36] Amos Sironi, Manuele Brambilla, Nicolas Bourdis, Xavier Lagorce, and Ryad Benosman. HATS: histograms of averaged time surfaces for robust event-based object classification. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 1731–1740. IEEE Computer Society, 2018. 1, 2, 6, 7

[37] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *Proceedings of International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 6105–6114. PMLR, 2019. 1

[38] Stepan Tulyakov, François Fleuret, Martin Kiefel, Peter V. Gehler, and Michael Hirsch. Learning an event sequence embedding for dense event-based deep stereo. In *Proceedings of IEEE/CVF International Conference on Computer Vision*, pages 1527–1537. IEEE, 2019. 5

[39] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. *CoRR*, abs/1710.10903, 2017. 3

[40] Lin Wang, S. Mohammad Mostafavi I., Yo-Sung Ho, and Kuk-Jin Yoon. Event-based high dynamic range image and very high frame rate video generation using conditional generative adversarial networks. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 10081–10090. Computer Vision Foundation / IEEE, 2019. 2

[41] Qinyi Wang, Yexin Zhang, Junsong Yuan, and Yilong Lu. Space-time event clouds for gesture recognition: From RGB cameras to event cameras. In *Proceedings of IEEE Winter Conference on Applications of Computer Vision*, pages 1826–1835. IEEE, 2019. 1, 2

[42] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks. *IEEE Trans. Neural Networks Learn. Syst.*, 32(1):4–24, 2021. 6

[43] Yi Zhou, Guillermo Gallego, Xiuyuan Lu, Siqi Liu, and Shaojie Shen. Event-based motion segmentation with spatio-temporal graph cuts. *CoRR*, abs/2012.08730, 2020. 3, 5

[44] Yi Zhou, Guillermo Gallego, and Shaojie Shen. Event-based stereo visual odometry. *IEEE Transactions on Robotics*, 2021. 2, 5

[45] Alex Zihao Zhu, Liangzhe Yuan, Kenneth Chaney, and Kostas Daniilidis. Unsupervised event-based learning of optical flow, depth, and egomotion. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 989–997. Computer Vision Foundation / IEEE, 2019. 2