Graph-based Asynchronous Event Processing for Rapid Object Recognition - Supplementary Material -

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

In this supplementary document, we provide additional ablation studies and justify the choice of parameters used for our experiments in Sec. 1. Then we show more comparisons with state-of-the-art methods which work in a batch-wise way in Sec. 2. In Sec. 3, we present more training details and in Sec. 4, we introduce FLOPs computation. Then we analyze the computation complexity of our pixel queue based radius search in Sec. 5 and prove that using our slide convolution to process events one by one is equivalent to using original spatial convolution to process all events at once in Sec. 6. Finally, we visualize some examples of datasets used in our experiments in Sec. 7. In this document, references that point to the main manuscript will be referenced as "P-".

1. Ablation study

In this section, we explore how performance and complexity are affected when changing the key parameters of our approach. First of all, we explore the performance when changing the depth of the graph convolution layer. Concerning the graph construction, we also study the time interval under which we extract events and the radius distance used to define the connectivity of the nodes. All experiments reported are conducted on the N-Caltech101 dataset, since it has the highest number of classes among all datasets.

Depth. In the implementation details of Sec.P-5.1, we describe the network architecture, whose backbone is comprised of D blocks (a block consists of "GraphConv-ELU-Bn" layers), each block followed by a voxel grid pooling layer. In the following experiment, we vary D from 2 to 5, to find the best setting of depth with respect to accuracy and complexity. The number of output channels in each convolution layer and the voxel size in each pooling layers are shown in Tab. 1. The results are shown in Tab 4. While the highest accuracy is obtained when the depth is 5, the complexity of the network substantially increases in comparison to $D = 4$. Therefore, in our paper, we set the depth of the graph convolution layer to $D = 4$. The network saturates when the depth is larger than five, which is in line with the well-known limitation of "over-smoothing" [3]. This problem is beyond our topic because it is a general problem of GCN. Except for the accuracy, we also evaluate the ratio of baseline's FLOPs divided by SlideGCN's FLOPs (notice that the difference between SlideGCN and baseline is that SlideGCN uses slide convolution), which is denoted as B/S ratio. The ratio decrease with deeper layers because generally V_n expands along the receptive field while the number of nodes decreases because of pooling layers (Please refer to the ratio shown in the fourth and seventh rows in Tab. 2). Currently, many graph-based architectures are still shallow due to "over smoothing" problem. In this depth setting, our method brings excellent gain. In future work, we will explore the technology to keep the gain in a deeper layer.

D	Output Channels	Voxel Size
2	$(64, 128)$	$(4, 8)$
3	$(64, 128, 256)$	$(4, 8, 16)$
4	$(64, 128, 256, 512)$	$(4, 8, 16, 32)$
5	$(64, 128, 256, 512, 512)$	$(4, 8, 16, 32, 64)$
6	$(64, 128, 256, 512, 512, 512)$	$(4, 8, 12, 24, 32, 64)$

Table 1. **Parameters setting for different depths**.

i^{th} layer	1	2	3	4
N_a	18392	1285	429	124
N_v	1	5	8	10
N_a/N_v	18392.0	257.0	53.6	12.4
N_r	146591	3877	1170	345
N_e	7	4	12	17
N_r/N_e	20941.6	969.3	97.5	20.3

Table 2. **An example of number of nodes/edges at different layers**. Here N_a and N_r are the number of nodes at that layer and the number of edges at that layer respectively. N_v and N_e are the number of V_n and E_n in Eq.P-6 and Eq.P-7.

Time Interval. For each event stream, we extract events within a fixed time interval as input. In this study, we test under various time intervals, i.e., 10, 30, 50 and 70 milliseconds, to see their effect on the accuracy and computation. The results are shown in Tab. 5. With a similar reason on choosing depth, we opted for extracting 50 ms-events, to strike a balance between accuracy and complexity.

*Corresponding author: Guofeng Zhang.

Methods	Representation	N-Caltech101		N-Cars	
		Acc ↑	Mps/ev ↓	Acc ↑	Mps/ev ↓
SSC [4]	VoxelGrid	0.761	1621	0.945	321
EST [2]	VoxelGrid	0.817	4150	0.925	1050
Matrix-LSTM [1]	VoxelGrid	0.857	2086	0.9565	616.319
Ev2Vid [5]	Image	0.866	21585	0.91	6126
NVS-B (Ours)	Graph	0.670	221	0.915	57.9
NVS-S (Ours)	Graph	0.670	7.8	0.915	5.2
EvS-B (Ours)	Graph	0.761	1152	0.931	251
EvS-S (Ours)	Graph	0.761	11.5	0.931	6.1

Table 3. **Comparison with batch-wise methods.**

Radius Distance. In this ablation study, we vary the radius distance as $R = \{2, 3.5, 5, 6.5\}$, to find the best distance with respect to accuracy and complexity. The results are shown in Tab 6, where we demonstrate that radius distance above 5 improves the model performance little while incurring increased complexity. Therefore, in our paper, we set the radius distance to 5. Note that when radius distance changes from 5 to 6.5, the required computation increases only slightly because the maximum connectivity degree D_{max} is set to 30.

It is worth noting that B/S ratio increases with longer time intervals and bigger radius, which demonstrates that the SlideGCN's complexity grows slower than the baseline method when the graph size of nodes and edges increases.

Depth	2	3	4	5	6
Accuracy	0.630	0.702	0.761	0.774	0.771
MFLOPs-B	564	887	1152	1774	2420
MFLOPs-S	4.1	7.1	11.5	22.7	36.2
B/S Ratio	137.6	125.0	100.2	78.1	66.9

Table 4. **The performances with different depths of convolution layers**. *-B denotes baseline method, *-S denotes SlideGCN.

Interval	10	30	50	70
Accuracy	0.635	0.710	0.761	0.766
MFLOPs-B	415	720	1152	1402
MFLOPs-S	5.14	8.1	11.5	12.9
B/S Ratio	80.7	88.9	100.2	108.7

Table 5. **The performances with different time intervals**. *-B is denoted as baseline method, *-S is denoted as our method with slide convolution.

Radius	2	3.5	5	6.5
Accuracy	0.675	0.724	0.761	0.763
MFLOPs-B	522	820	1152	1302
MFLOPs-S	7.73	9.7	11.5	12.9
B/S Ratio	67.5	84.5	100.2	100.9

Table 6. **The performances with different radius distances**. *-B is denoted as baseline method, *-S is denoted as our method with slide convolution.

2. More Comparisons

In this section, we show comparisons with state-of-the-art methods that work in a batch-wise way, as shown in Table 3. Generally, SlideGCN is much more efficient on event data than batch-wise methods (ours 11.5 MFLOPs v.s. 21585 MFLOPs of Ev2Vid [5]), although sacrificing some accuracy (ours 0.76 v.s. 0.86). It is worth noting that a ConvLSTM [6]-like method is included in our comparison, *i.e.*, Matrix-LSTM [1]. Some readers may think that ConvLSTM-like methods is similar to ours as they also rely on a buffer of events but in the way of memorizing hidden state. We respectfully argue that ConvLSTM is designed to build long-term dependencies but cannot process sparse and asynchronous data, which is the nature of events. Specifically, although Matrix-LSTM have been improved upon original ConvLSTM, its efficiency is far below our SlideGCN in an event-wise processing manner (2086 MFLOPSs v.s. ours 11.5 MFLOPs).

3. Training Details

In this section, we present more training details used in our experiments in Sec.P-5. Our training consists of two steps. First of all, We train the main branch with 60 epochs and a batch size of 16. The first five epochs is trained with a warm-up strategy, with the learning rate increased linearly from zero to the initial learning rate. Our initial learning rate is set to 5e-3 and decayed with a polynomial learning rate schedule. Secondly, we freeze the main branch and start training the state-aware module. We set the initial learning rate to 1e-5. Other parameters like batch size and training epochs stay unchanged.

4. FLOPs Computation

In Tab. 7, we show the number of FLOPs for different operations of baseline and our slide convolution. The FLOPs needed for the baseline can be split into two parts: message passing (denoted as MP) and aggregation (denoted as Aggr). The FLOPs needed for message passing is $N_r C_{out}(2C_{in} - 1)$, which is the result of performing C_{in} multiplications and $C_{in} - 1$ additions for each edge and

each output channel. For each node's aggregation, we consider adding transformed root node features to the output, which needs to perform $C_{in}C_{out}$ multiplications. We need to perform C_{out} additions to sum up root node features and edge features and perform C_{out} divisions to normalize final node features with node degree. The total FLOPs needed for aggregation is $N_aC_{out}(2 + C_{in})$ (excluding bias). We can see for FLOPs of slide convolution, it is simply replacing N_r with N_e (similar for N_a). This is because the slide convolution works only on a part of nodes/edges instead of computing full nodes/edges.

	Baseline	Slide Convolution
MP	$N_rC_{out}(2C_{in} - 1)$	$N_eC_{out}(2C_{in} - 1)$
Aggr	$N_aC_{out}(2 + C_{in})$	$N_vC_{out}(2 + C_{in})$
Pooling	N_aC_{out}	N_vC_{out}
FC	$2C_{in}C_{out}$	$2C_{in}C_{out}$

Table 7. **FLOPs computation at each layer**. Here N_a and N_r are the number of nodes at that layer and the number of edges at that layer respectively. N_v and N_e are the number of V_n and E_n in Eq.P-6 and Eq.P-7.

5. Pixel Queue based Radius Search

In this section, we further analyze the computation complexity of our pixel queue based radius search. We maintain a queue for each pixel (namely pixel queues), accessed by the (x, y) index. When inserting an event, we first find the queue at the cost of $O(1)$, and then append the event to the end of the queue (notice that in the main manuscript, we show that event data may be stochastic, but its timestamp must be in increasing order). To support delete operations, we maintain a global queue. When an event slides out of the window, we pop it from the global queue at the cost of $O(1)$ because it is the "oldest" event. We obtain this event, subsequently, find its corresponding pixel queue, and pop it from this local queue. All these operations can be done at the cost of $O(1)$, so both the cost of insertion and deletion is $O(1)$.

As for searching, we use a two-stage algorithm. In grid search, for a given radius R and a query event (x_0, y_0, t_0), we find candidate pixels whose distance is less than R' on the image grid. We can precompute candidate pixels for a specific integer radius. So this step can be done at the cost of $O(1)$. In the second step, for each queue corresponding to the candidate pixels, we need to find events whose timestamp are in range $(t_0 - \sqrt{R^2 - (\delta x^2 + \delta y^2)}, t_0 + \sqrt{R^2 - (\delta x^2 + \delta y^2)})$. $(\delta x, \delta y)$ is 2D offset relative to the query event. This can be done by searching the bound through binary search. Therefore, the total cost of the search is $\pi * R'^2 * \log M$, where M is the average queue length. The cost can be optimized by searching candidate queues from inner to outer with an early stop if connectivity degree is limited to D_{max}. The experiment shown in Fig.P-5 is evaluated on an i7-9700K CPU (using a single core).

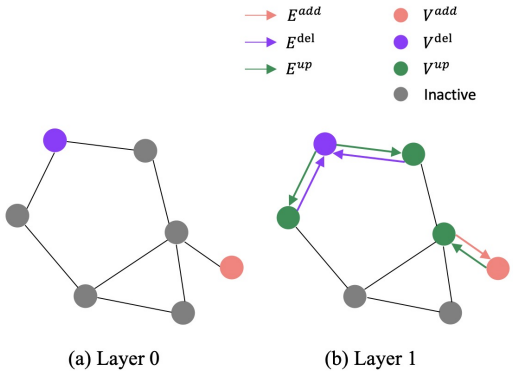

Figure 1. **An example for recursive update of V and E.** The left figure shows a newly added and newly deleted node at Layer 0. The right figure shows the update of V and E according to Eq.P-6 and Eq.P-7.

6. Slide Convolution

In the main manuscript, we use Eq.P-5 to represent the recursive processing, whose key idea is evolving V_n and E_n (See the example in Fig. 1). The equation is suitable for V^{up}. While for V^{add} and V^{del}, we directly compute X^{t+1} for V_n^{add} and assign X^{t+1} to zero for V_n^{del}. The complete algorithm is described in Algo. 1.

In the following, we will prove that processing events one by one with this algorithm is equivalent to processing all events at once based on Eq.P-4.

For convenience, we use (set_b, set_a) to represent $\{(j, i) \mid for\ i \in set_a\ then\ \forall j \in N(i) \wedge j \in set_b\}$. Firstly we rewrite equation Eq.P-4 as following:

$$x_{n+1}^{t+1}(i) = \sum_{(j,i) \in (A_n^{t+1}, A_{n+1}^{t+1})} x_n^{t+1}(j) h_\theta$$

According to Eq.P-6, we can split $(A_n^{t+1}, A_{n+1}^{t+1})$ into $(A_n^{t+1}, V_{n+1}^{add}) \cup (A_n^{t+1}, A_{n+1}^{t}) \setminus (A_n^{t+1}, V_{n+1}^{del})$. (A_n^{t+1}, A_{n+1}^{t}) can be further written as $(A_n^{t+1}, A_{n+1}^{t} \setminus V_{n+1}^{del}) \cup (A_n^{t+1}, V_{n+1}^{del})$. Substitute these into $x_{n+1}^{t+1}(i)$, we get:

$$
\begin{aligned}
x_{n+1}^{t+1}(i) = & \sum_{(j,i)\in(A_n^{t+1},V_{n+1}^{add})} x_n^{t+1}(j)h_\theta \\
& + \underbrace{\sum_{(j,i)\in(A_n^{t+1},A_{n+1}^{t}\setminus V_{n+1}^{del})} x_n^{t+1}(j)h_\theta}_{marked\ as\ {}_{wodel}x_{n+1}^{t+1}(i)} \\
& + (\sum_{(j,i)\in(A_n^{t+1},V_{n+1}^{del})} x_n^{t+1}(j)h_\theta \\
& - \sum_{(j,i)\in(A_n^{t+1},V_{n+1}^{del})} x_n^{t+1}(j)h_\theta)
\end{aligned}
$$

The first and last two lines correspond to lines 12-13 and 14 of the algorithm 1, respectively. We know both ${}_{del}x_{n+1}^{t+1}$ and ${}_{add}x_n^t$ are zeros, so we can rewrite ${}_{wodel}x_{n+1}^{t+1}(i)$ as Eq. (1).

Apparently, $\Delta n(j)$ is only non-zero for $j \in V_n$, and we know $V_n \subset \{A_n^{t+1} \cup V_n^{del}\} = \{A_n^t \cup V_n^{add}\}$, so we can divide $A_n^{t+1} \cup V_n^{del}$ into two parts, one of which is V_n. Therefore, We have:

$$
\begin{aligned}
& {}_{wodel}x_{n+1}^{t+1}(i) \\
= & {}_{wodel}x_{n+1}^{t}(i) + \sum_{(j,i)\in(A_n^{t+1}\cup V_n^{del},A_{n+1}^{t}\setminus V_{n+1}^{del})} \Delta n(j)h_\theta \\
= & {}_{wodel}x_{n+1}^{t}(i) + \underbrace{\sum_{(j,i)\in(V_n,A_{n+1}^{t}\setminus V_{n+1}^{del})} \Delta n(j)h_\theta}_{\Delta_{n+1}(i)} + 0 \\
= & \underbrace{{}_{wodel}x_{n+1}^{t}(i) + \Delta_{n+1}(i)}_{line\ 9-11\ in\ Algo.1}
\end{aligned}
$$

So far, we have proved that incrementally updating from t to $t+1$ with Algo. 1 is equivalent to processing all the events at $t+1$ via Eq.P-4. In this way, we can readily obtain the immediate result at any time point of event stream through this incremental mechanism, instead of calculating a period of events from scratch. This is also verified through our experiment (Sec.P-5) in the main manuscript.

7. Visualization of Datasets

We sample some sequences from used datasets for visualization, as shown in Fig. 2. Among them, N-Cars have only two categories. In particular, the car category has a similar texture, which makes it easy to recognize. On the contrary, N-Caltehc101 and CIFAR10-DVS have a variety of different appearances, which is even noisy and fuzzy, thus leading to low accuracy. It is worth noting that our proposed method can also recognize an object in foreground from events triggered by both foreground and background patterns. We show eight samples in Fig. 3, which are all successfully identified.

Algorithm 1 slide convolution at layer n

Input: V_{n-1}, X_{n-1}^{t+1}
Output: V_n, X_n^{t+1}

if n equals to 0 **then**
  $V_n = \{added : new\ events, deleted : events\ out\ of\ window\}$
  $X_n^{t+1} = \{added : input\ feature, deleted : 0\}$
else
  compute V_n using Eq.P-6 with V_{n-1}, A_n^t
  update A_n^{t+1} using Eq.P-6 according to V_n
  compute E_n using Eq.P-7 with V_{n-1}, V_n and A_n^{t+1}
  expand X_{n-1}^t according to V_{n-1}^{add} and assign new part to zeros
  for $i \in V_n^{up}$ **do**
    compute $\Delta n(i)$ using Eq.P-5 with E_n^{up}, X_{n-1}^t, X_{n-1}^{t+1}
    update $X_n^{t+1}(i)$ using Eq.P-5 with X_n^t, $\Delta n(i)$
  for $i \in V_n^{add}$ **do**
    compute $X_n^{t+1}(i)$ using Eq.P-4 with E_n^{add}
  set deleted vertex(V_n^{del}) in X_n^{t+1} to zeros
Return V_n, X_n^{t+1}

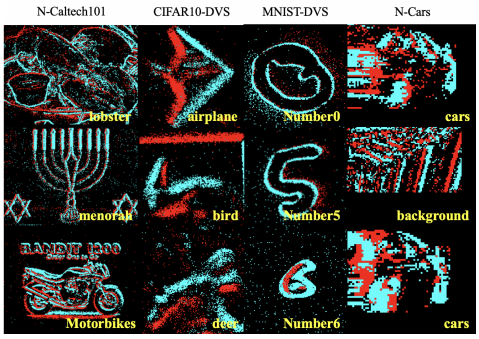

Figure 2. **Examples of datasets used in our experiments**. (Red/Cyan: ON/OFF events).

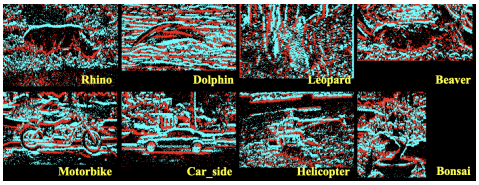

Figure 3. **Some examples with rich background.** (Red/Cyan: ON/OFF events).

$$
\begin{aligned}
{}_{wodel}x_{n+1}^{t+1}(i) &= \sum_{(j,i)\in(A_n^{t+1},A_{n+1}^t\setminus V_{n+1}^{del})} x_n^{t+1}(j)h_\theta + \underbrace{\sum_{(j,i)\in(V_n^{del},A_{n+1}^t\setminus V_{n+1}^{del})} x_n^{t+1}(j)h_\theta}_{0} \\
&= \sum_{(j,i)\in(A_n^{t+1}\cup V_n^{del},A_{n+1}^t\setminus V_{n+1}^{del})} (x_n^t(j)+\Delta n(j))h_\theta \\
&= \sum_{(j,i)\in(A_n^{t+1}\cup V_n^{del},A_{n+1}^t\setminus V_{n+1}^{del})} x_n^t(j)h_\theta - \underbrace{\sum_{(j,i)\in(V_n^{add},A_{n+1}^t\setminus V_{n+1}^{del})} x_n^t(j)h_\theta}_{0} \\
&+ \sum_{(j,i)\in(A_n^{t+1}\cup V_n^{del},A_{n+1}^t\setminus V_{n+1}^{del})} \Delta n(j)h_\theta \\
&= \sum_{(j,i)\in(\underbrace{A_n^{t+1}\cup V_n^{del}\setminus V_n^{add}}_{A_n^t},A_{n+1}^t\setminus V_{n+1}^{del})} x_n^t(j)h_\theta + \sum_{(j,i)\in(A_n^{t+1}\cup V_n^{del},A_{n+1}^t\setminus V_{n+1}^{del})} \Delta n(j)h_\theta \\
&= \underbrace{\sum_{(j,i)\in(A_n^t,A_{n+1}^t\setminus V_{n+1}^{del})} x_n^t(j)h_\theta}_{{}_{wodel}x_{n+1}^t(i)} + \sum_{(j,i)\in(A_n^{t+1}\cup V_n^{del},A_{n+1}^t\setminus V_{n+1}^{del})} \Delta n(j)h_\theta
\end{aligned}
\tag{1}
$$

References

[1] 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. 2

[2] 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. 2

[3] Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In *Proceedings of AAAI Conference on Artificial Intelligence*, pages 3538–3545. AAAI Press, 2018. 1

[4] 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

[5] 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. 2

[6] Xingjian Shi, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-Kin Wong, and Wang-chun Woo. Convolutional LSTM network: A machine learning approach for precipitation nowcasting. In *Proceedings of Neural Information Processing Systems*, pages 802–810, 2015. 2