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

Thomas Dalgaty\textsuperscript{1} \textsuperscript{*} Thomas Mesquida\textsuperscript{1} Damien Joubert\textsuperscript{2} † Amos Sironi\textsuperscript{2} Pascal Vivet\textsuperscript{1} Christoph Posch\textsuperscript{2}  
\textsuperscript{1} Université Grenoble-Alpes, CEA-List, Grenoble, France, \textsuperscript{2} Prophesee, Paris, France  

thomas.dalgaty@cea.fr thomas.mesquida@cea.fr cposch@prophesee.ai

Abstract  

Event camera pixels asynchronously output binary events corresponding to local light intensity changes in time. While encoding visual information in this fashion increases sparsity and the temporal detail of motion with respect to frame-based cameras, there is not yet an established machine learning method capable of exploiting these features to increase efficiency, reduce latency and, ultimately, perform optimally in event-based tasks. Graph neural networks are a promising avenue for such a method, but current solutions are too slow to be compatible with the continuous streaming nature of event-data. In this study, we propose a hemi-spherical update event-graph neural network that significantly reduces the complexity and latency of graph updating and event-level prediction. We compare our approach to existing graph neural network methods, as well as to dense-frame convolutional neural networks, on optical flow estimation tasks. Relative to the previous state of the art in event-graphs, we reduce event-graph update latency by more than four orders of magnitude and reduce the number of neural network calculations per second by 70\times while predicting optical flow more accurately.

1. Introduction  

From initial work where the analogue properties of transistors were used to mimic the early-stages of the mammalian visual system [6,22,26,34], we now have at our disposal industrial event-cameras [11,29,33,39]. In contrast to conventional frame-based cameras, where pixels periodically integrate photon-generated charges to record absolute light intensity, event-camera pixels asynchronously generate binary flags, referred to as events, upon the detection of relative light intensity changes (i.e., temporal contrast).

\textsuperscript{*}Corresponding author  
\textsuperscript{†}Contact d.joubert@prophesee.ai for access to Rock Scenes  
This work is partly funded thanks to the French national program “Programme d’Investissements d’Avenir, IRT Nanoelc” ANR-10-AIRT-05.
Previous event-graph works however have not reported on the additional computational burden inherent to the incorporation of new events into a continuously evolving event-graph, where events may connect to previously generated and future events. The resulting high per-event latency required to update the event-graph and to output a prediction makes existing approaches poorly adapted for many applications where event cameras are well suited (i.e., embedded and edge AI). In this paper, we solve these problems by introducing HUGNet - an event-graph neural network method which uses only the information instantaneously available in a semi-spherical volume of past events to create a tiny sub-graph for each newly generated event. We compare HUGNet to fully-spherical event-graph approaches and to dense-frame convolutional neural networks on optical flow estimation tasks. Compared to the previous state-of-the-art event-graphs, we observe a remarkable four order of magnitude reduction in event-graph update latency and eliminate event-level prediction latency altogether, while predicting optical flow more accurately. We also reduce the number of multiply-and-accumulate operations per second by 70×. Furthermore, we obtain favourable results relative to dense-frame convolutional approaches with significantly fewer MAC operations. The specific contributions of this work are as follows:

- We restrict event-graphs nodes to form directed edges (past to future) to past events only which drastically reduces event-graph update latency and the number of MAC operations per second.
- We demonstrate for the first time the effectiveness of graph neural networks for event-based optical flow prediction.
- We introduce a lightweight synthetic dataset for optical flow and motion segmentation called ROCK scenes. The dataset features a very high rate of ground truth labelling and challenging fast direction changes which are not readily available in existing datasets. Rock Scenes will be shared upon request.

2. Related Work

Hand-crafted event-camera algorithms: Under certain assumptions, mathematical models can be developed that solve some computer vision tasks using event-camera data. For instance, previous works introduced optical-flow estimation based on local partial derivatives of event-surfaces [4], tracking via probabilistic filtering [17] and depth estimation using relative timing differences from a pair of event cameras [35]. Such approaches often do not generalise well to real-world settings [46].

Dense-frame convolutional networks: Convolutional neural networks are not adapted to deal with event-based data. In order to render them compatible with what are effectively 3D-pointclouds, events within temporal slices are integrated into a sequence of 2D dense frames [13, 20, 23, 25, 46]. Dense-frame CNNs have been applied to dense optical flow prediction in this fashion [14, 46]. While these approaches allow the use of existing CNN architectures and optimised hardware implementations, the fine spatiotemporal detail captured by event cameras is effectively discarded. While in many tasks (i.e., object detection and classification) the spatial information alone may be sufficient to solve a task well, others such as optical flow prediction may be negatively impacted. Dense-frame CNNs also do not readily leverage the inherent sparsity of event-data to reduce computational requirements. Although techniques such as sub-manifold sparse convolutions [15] have been applied to event-data [27], their additional computational and memory requirements are not yet clear.

Spiking neural networks: Spiking models [21, 32, 43] are similar in conception to recurrent neural networks, but instead neurons are modelled using a step function that rectifies the state of a leaky-integrator model - generating an output spike. After a spike, neurons typically undergo a refractory period in which its input is reset to zero and it does not integrate input spikes from other neurons for a limited time. Training SNNs, typically using surrogate-gradient approximations and backpropagation through-time [42], can be time-consuming since this dynamical system must be modelled laboriously over a series of fine-grain timesteps. SNNs have been applied to optical flow estimation by incorporating spiking neuron models into the UNet architectures [16, 20].

Event-graph neural networks: Graph neural networks (GNNs) have recently been applied to event-camera data, as well as other event datasets [3], by building graphs from 3D-pointclouds. Layers of graph convolutions [18] are then applied in order to find useful embeddings for events for use in a downstream task. Furthermore, the fine spatiotemporal structure of the event-data can be stored, and leveraged, in the graph edges. Event-graphs are typically constructed by performing a K-nearest neighbour search around each point and connecting the nearest events through edges. Edges can be binary flags [28] or they can contain vectors describing, for example, local spatiotemporal differences [5, 37].

Despite promising early results [5, 28, 37], their suitability for processing a continuous stream of events has not yet been fully considered. Current methods build graphs using edges to past and to future events - i.e., a fully-spherical search radius (Fig.1(a)). This has two major drawbacks. First, event-level predictions cannot be made instantaneously. A newly arrived event may be updated by an event arriving within a time equal to the search radius multiplied by the number of graph layers. For example, for five layers and a temporal search radius of 20ms, an event
may be impacted by future events arriving within 100ms. Secondly, the node embeddings of all previously arrived events within this same temporal window are subject to change [37] and, after incorporating a new event, graph convolutions must be re-applied. This must be done each time an event arrives and is particularly problematic for high-resolution event cameras where event-rates can be on the order of millions of events per second [11].

**Event-based optical flow**: We evaluate HUGNet using the case study of event-based optical flow. Intuitively, it is a task which should be reliant on the fine temporal information captured by event cameras and for which the event-by-event prediction capabilities of graph neural networks may be highly desirable. While many event-based optical flow processing approaches based on the supervised and self-supervised training of convolutional and spiking neural networks have already been proposed [7, 12, 14, 16, 20, 23, 30, 38, 46, 47], no previous work has studied the application of event-graph neural networks to optical flow estimation.

### 3. Method

In this study we propose that, in order to simplify the continuous updating of event-graphs, and eliminate event-level prediction latency, event-graphs should be updated using a hemi-spherical search volume where directed (past to future) graph edges are formed only from past events to a newly generated event (Fig.1(b)).

In previous work [5, 37], each event in an event-graph is connected not only to past events, but also to events that will arrive in the future. The required steps for such an algorithm are denoted in the pseudo-code of Algorithm.1. For a newly generated event $ev_i$, at time $t_i$, the first step is to find the subset of $EV = \{x, y, t\} \in \mathcal{R}^3$ within the radii $r_t$ and $r_{xy}$ (in respective temporal and spatial dimensions) of $ev_i$, using a function $B()$. For each of the $M$ events concerned, a K-nearest neighbour search function, $knn()$, must be performed. This defines what edges, $E_m$, are formed between the existing events and $ev_i$ as well as updating the edge configurations of existing events in the case where $ev_i$ becomes one of their own K-nearest neighbours. The distance $r_t$ is typically on the order of tens of milliseconds whilst $r_{xy}$ can be a small number of pixels (here between 6 and 9). The nodes, $EV$, and edges, $E$, of the event-graph, $G$, are then updated using a function $R()$ which takes as input the previous graph state, the new event $ev_i$, and the new and re-calculated edges $E_m$ and returns an updated event-graph. Next, the effect of the new event, $ev_i$, on the existing event-graph node embeddings must be evaluated for all $L$ event-graph neural network layers. To avoid updating the node embeddings of the entire event-graph, a recursive k-hop graph search function $hop()$ can be used to find the sparse subset of nodes impacted in each of the $l$ layers [37]. This function simply finds which events are connected to other events in the graph over an integer number, $K$, of hops through immediately connected nodes (see Fig.1(a)). The affected node embeddings in a given layer, $Z_{j,l}$, are then updated using a graph convolutional function $\phi()$. A final function, $\sigma()$, is applied to the node embeddings of $ev_i$, $Z_l$, which outputs the optical flow prediction $V_i$ for the node. However, this cannot be done instantaneously. The event $ev_i$ may be influenced by future events arriving within a time equal to the time search radius $r_t$ multiplied by the
number of graph neural network layers and therefore it is required to store all generated events and their edges during this time. A prediction must therefore be scheduled in the future accordingly - typically this delay may be on the order of hundreds of milliseconds.

**Algorithm 1** Sparse fully-spherical update

```
Input : \( ev_i = \{x_i, y_i, t_i, p_i\} \), \( G = \{EV, E\}, r_t, r_{xy}, L \)
Output : \( V_i \)
\( M = EV \cap B(ev_i, r_{xy}, r_t) \)
for \( ev_m \in M \)
do
\( E_m = \text{knn}(ev_m, EV, r_t, r_{xy}) \)
end for
\( G = \pi(G, E_m, ev_i) \)
for \( i \) in range(\( L \)) do
  for \( ev_j \in \text{hop}(M, G, i) \) do
    \( z_{j,i} = \phi(ev_j, G, i) \)
  end for
end for
if \( t > t_i + (r_t \times L) \) then
  \( V_i = \sigma(Z_i) \)
end if
```

### 3.1. Hemi-spherical event-graph updates

Evidently updating an event-graph where each event can make edges to past and future events is highly complicated. The resulting per-event latency risks to be prohibitive and incompatible with the continuous stream of data generated by an event camera. In order to reduce this latency, we propose to constrain graph connectivity such that (i) new edges are only formed from events in the past to a new event \( ev_i \), and (ii) that these are directed edges such that information always flows from the past to the future.

Our approach, HUGNet, is detailed in Algorithm 2. Given a newly generated event \( ev_i \), a K-nearest neighbours search is performed to find up to \( K \) nearest neighbouring events within a hemi-spherical search volume that extends into the past only. Relative to the fully-spherical method, the temporal search radius, \( r_t \), is increased such that the search volume is equivalent. At this point our Algorithm 2 diverges from, and greatly simplifies upon, the previous Algorithm 1. First, a tiny sub-graph containing \( ev_i \) and its neighbours, \( G_i \), is created. This sub-graph is then used to calculate the \( L \) node embeddings for the new event \( ev_i \) using a graph convolution function \( \phi'() \). Crucially, the only node embeddings that are calculated are those of the newly arrived event - it is not required to re-apply graph convolutions to update past events. These embeddings, \( Z_i \), can then be immediately used (neither the edges nor the node embeddings of \( ev_i \) will be updated by future events) in the calculation of output optical flow, \( V_i \), using a function \( \sigma'() \). Note that, relative to Algorithm 1, only the events generated within a past temporal window of \( r_t \) need to be stored. Furthermore the edges, and in fact the entire event-graph structure, do not need to be stored explicitly. They are implicitly represented within the node embeddings of the past events \( EV \). This will permit the total system memory requirements of HUGNet to be greatly reduced relative to fully-spherical approaches, which is of great importance in edge computer vision systems where memory may be severely limited.

In our implementation of these algorithms we use the KD-tree K-nearest neighbour search algorithm [44] implemented in the Open3D library [45]. The graph convolution functions are implemented using the framework PyTorch geometric [9]. Specific codes were developed in order to measure the graph update latencies of Algorithms 1 (GNN-sparse) and 2 (HUGNet) as well as a third algorithm, identical to Algorithm 1, but where a k-hop search is not used and all nodes, \( EV \), within a certain time window are updated (GNN-full).

**Algorithm 2** Hemi-spherical update

```
Input : \( ev_i = \{x_i, y_i, t_i, p_i\} \), \( EV, r_t, r_{xy}, L \)
Output : \( V_i \)
\( K_i, E_i = \text{knn}(ev_i, EV, r_t, r_{xy}) \)
\( G_i = \{K_i, E_i\} \)
for \( l \) in range(\( L \)) do
  \( z_{i,l} = \phi(G_i, l) \)
end for
\( V_i = \sigma'(Z_i) \)
```

### 3.2. Event-graph neural network architecture

Our architecture for event-graph optical flow prediction, depicted in Fig.2, takes as input the tiny sub-graph \( G_i \) (as described in Algorithm 2) built upon the generation of each new event \( ev_i \). It is composed of five successive graph convolutional layers. A node embedding, \( Z_{i,l} \), is calculated for the new event in each of the \( l \) layers using the previously calculated node embeddings of the past events in the sub-graph. Node embeddings are 64-size vectors in each of the layers which are concatenated together into a single vector \( Z_i \) which is then processed by a multi-layer perceptron (MLP) (Fig.2(b)). In the first layer of this MLP, an instance normalisation [41] is applied to produce a 128-size vector representation for the event. Instance normalisation is an appealing alternative to batch normalisation in this step since feature normalisation can be performed using node-level, instead of graph-level, statistics. Finally, to predict the optical flow of \( ev_i \), we apply three further layers (\( 128 \times 128, 128 \times 64, 64 \times 2 \)) resulting in a final two-size vector \( V_i \) whose components correspond to the x and y flow \( V_x \) and \( V_y \). The architecture is agnostic to convolution method.
and could be applied to any event-level prediction task - not only optical flow.

3.3. Supervised event-graph optical flow learning

For a given event $ev_i$, the objective is to output a vector $V_i$ describing the optical flow $I_{flow}(x, y, t)$:

$$\frac{dI_{flow}}{dx} V_x + \frac{dI_{flow}}{dy} V_y + \frac{dI_{flow}}{dt} = 0,$$

(1)

of the object that generated the event. We minimise a smooth-L1 loss, $L_{sl1}$, for a event-graph of $N_{ev}$ vertices,

$$L_{sl1} = \frac{1}{N_{ev}} \sum_{i} \frac{1}{2} (V_i - \hat{V}_i)^2 \begin{cases} \|V_i - \hat{V}_i\|_2^2 & \text{if } \|V_i - \hat{V}_i\|_2 < \beta, \\ \|V_i - \hat{V}_i\|_2 - \frac{1}{2} \beta & \text{otherwise} \end{cases}$$

(2)

given a ground-truth flow $\hat{V}_i$. The threshold $\beta$ is set to 0.025.

Additionally, to enforce smoothness in the spatiotemporal evolution of predicted flow, we add a constraint based on the Charbonnier loss: $\rho(x) = (x^2 + \epsilon^2)^\alpha$ [40]. Instead of iterating over events within a certain radius (that could be computationally costly to find), we propose a graph-Charbonnier loss that enforces smoothness between immediately connected nodes:

$$L_{smooth} = \frac{1}{N_{ev}} \sum_{i} \frac{1}{2} ((V_i - A_i \cdot I_d \cdot V)^2 + \epsilon^2)^\alpha,$$

(3)

where $A_i$ is the normalised adjacency matrix for a graph $G_i$, the symbol $I_d$ is the identity matrix of a dimension two and $V$ is the matrix of predicted flow for the past events, $EV$. Alpha is set to 0.5 and epsilon to 0.001. This smoothness loss is weighted by 0.1 and summed with $L_{sl1}$ (the supplementary material reports on an ablation study of this smoothness loss).

Event-graphs are processed in single batches, after which the gradient is calculated using the the AdamW optimisation method [24]. A plateau scheduling is applied to the learning rate where, if no relative change of 0.05 is observed in the loss during ten successive epochs, the learning rate is divided by two. All event-graph neural networks are trained over 100 epochs on a single NVIDIA GeForce RTX 2080 GPU.

4. Experiments

4.1. Metrics

Average Endpoint Error (AEE) is a commonly used as a metric in optical flow prediction tasks [46]. We adapt this metric for the event-based setting, simply by computing end-point error over generated events instead of over all active pixels in a frame:

$$AEE = \frac{1}{N_{ev}} \sum_{i} \|V_i - \hat{V}_i\|_2$$

(4)

where $V_i$ and $\hat{V}_i$ are respectively the predicted and the ground-truth optical flow vectors and $N_{ev}$ is the number of events. Similarly to [46], we also report the percentage of outlying event predictions - defined as the fraction of events with an end-point error greater than a given number of pixels and in excess of 5% of the magnitude of the ground-truth flow vector.

While these metrics allow for a concise means of quickly comparing different approaches, AEE is not normalised. Large flows contribute more than smaller ones. We therefore introduce an additional metric - event flow accuracy - which we find to be more informative since it assesses the
ability of a model to correctly predict optical flow regardless of its magnitude:

\[ F_{\zeta} = \frac{1}{N_{ev}} \sum_{i}^{N_{ev}} \left\{ \begin{array}{ll} 1 & \text{if } \frac{||V_i - 0||_2}{||V_i||_2} < \zeta \\ 0 & \text{otherwise} \end{array} \right. \] \tag{5}

where \( \zeta \) is a number between zero and one corresponding to a tolerated percentage error of the ground truth magnitude.

### 4.2. Datasets

We compare HUGNet to existing methods using two event-based optical flow datasets: the commonly used MVSEC [46] and Rock Scenes (RS) - which we introduce in this paper (refer to the supplementary material for examples). In our experience, the construction of graph data-structures and the development of graph neural network models is time intensive - in particular for the large 3D-pointclouds generated by high-resolution event-camera recordings. For this reason, it can be difficult to perform extensive early-exploration in new applications and event-graph architectures. We have therefore developed a manageable yet challenging synthetic motion segmentation and optical flow dataset that will be made available on request. The 100 x 100 resolution sequences from RS last between 3 and 6 seconds and feature a classic rock album moving in front of a natural background. Both the album cover and the background scene (simulating ego-motion) exhibit non-periodic and uncorrelated sharp direction and speed changes which are often not present in many available automotive datasets - which are largely characterised by smooth changes in flow. This will allow us to test the ability of models to exploit the high-temporal resolution of event-cameras. Ground truth flow and an object masks are provided at a rate of 5kHz. The training split is composed of one hundred sequences (ten objects combined with ten scenes) while the test set has a further four sequences with two new unseen objects and scenes.

We use RS to select the graph convolution method and the event and edge features that will be used to benchmark against other approaches in Section 4.3. MVSEC is composed of longer event-camera recordings with ground truth optical flow masks provided at 20Hz: four indoor drone, and two outdoor vehicle sequences. We split MVSEC into two tasks - indoor and outdoor optical flow prediction due to the difference in event density in each scenario. For the indoor task we train on the first, third and fourth sequences and test on the second. For the outdoor task, we train on the second outdoor sequence and test on the first. Both datasets are pre-filtered using an event-based spatiotemporal contrast filter [2], already built-in to state-of-the-art event-cameras [1], that operates by removing redundant contrast information encoded in the number of events emitted by a pixel as it is crossed by an edge. This filtering strategy is preferable to the uniform downsampling typically used in previous works, since it preserves the original fine spatiotemporal structure of motion. The event-graphs of full sequences are chopped into smaller temporal slices of a fixed duration and with a certain temporal overlap. Our slices are 400ms long for both RS and MVSEC and we use an overlap equal to half of this. In order to mitigate over-fitting, slices undergo a temporal warping, whereby event timestamps are multiplied by a sample from a uniform random variable between 0.5 and 1.5. We also randomly xy-flip the graph. The optical flow ground-truths are updated accordingly. We additionally perform edge dropout [36] where event-graph edges are removed with a probability of 0.25 during the forward pass. At test time, graph slices do not undergo these augmentations and the metrics are evaluated from half of the temporal search radius in each slice.

### 4.3. Optical-flow results

We first investigate, using Rock scenes, what combination of input and edge features are most pertinent for optical flow estimation. In addition to the pixel coordinates \( XY \) (which are always used) we study the effect of the event timestamp \( T \), the polarity \( P \), and an approximation to the event normal-vector \( N \) - calculated as described in [28]. The edge features, \( XY \) and \( T \), correspond to the normalised spatial and temporal differences between events as described in [10]. We also compare three graph convolution approaches: \textit{GCNConv} [18], \textit{B-spline} [10] (with a kernel size of five and linear basis functions) and \textit{Hybrid} which combines an input B-spline layer with four subsequent GCNConv layers.

The results are summarised in Table 1 where each cell contains two metrics - on top the flow accuracy at 25% error \( F_{25\%} \) (Eq.5), and below the average end-point error \( AEE \) (Eq.4). For all models, but in particular for GCNConv, we note that the polarity and the event timestamps are not adequate to accurately predict optical flow from event-graphs. In all cases, the normal-vector features greatly improve performance, and the addition of polarity and timestamp as features bring about only a small improvement in performance. Across the three methods, the use of all feature results in the highest \( F_{25\%} \). It also results in the lowest \( AEE \) for the GCNConv and Hybrid methods, whilst the B-Spline method achieves a marginally improved \( AEE \) without the timestamp. The impact of temporal difference between events as edge features are studied in the final three rows of Table 1. Relative to the first three rows of the table, temporal differences in edges appear to be an important feature, bringing an improvement in all cases. Table 1 also underlines the utility of the flow accuracy metric (Eq.5) whereby models that have an almost equivalent \( AEE \) often exhibit important differences in \( F_{25\%} \). This highlights the ability of some models to remain accurate over a wide range of optical flow - not only large flows.
In light of this, we elect to use the Hybrid method with all available event and edge features as our reference model in two benchmarking studies: RS and MVSEC. We compare our hemi-spherical update approach, HUGNet, against two event-graph neural networks based on fully-spherical updates which we refer to as GNN-full [5] and GNN-sparse [37] as described in Sec 3.1. It should be noted that the actual graph neural network architectures for HUGNet, GNN-sparse and GNN-full are identical. It is only the graph update method and, therefore, the structure of the resulting event-graph that differ. We also compare to two dense-frame CNN approaches: Seq-flownet and EV-flownet [46]. Since EV-flownet was designed using the MVSEC dataset, we only apply it to the MVSEC benchmark and re-train it in a self-supervised fashion using the publicly available code on our MVSEC splits. Seq-flownet is a convolutional architecture very similar to EV-flownet but trained in a fully-supervised fashion and applied to both RS and MVSEC - more details can be found in the supplementary material.

Table 3 presents the benchmarking results on RS. We report the AEE and the $F_{25\%}$ to quantify the performance of each approach in the tasks. In addition, we add to the Table the average latency required to update an event-graph due to a newly generated event (Graph update), the number of model parameters (#params) and the number of multiply-and-accumulations per second (MAC/s) to update the event-graph and output a prediction. As expected, HUGNet greatly improves upon the update latency achieved by the fully-spherical event graph neural networks. While the measured graph update latency for GNN-full and GNN-sparse is on the order of hundreds of milliseconds, HUGNet is capable of rapidly incorporating events with a latency of only some tens of microseconds - a striking improvement spanning four orders of magnitude. Similarly, the number of MAC operations per second required by HUGNet is reduced by almost two orders of magnitude. This reduction is due to the fact that each time an event is incorporated into GNN-sparse or GNN-full, all of the node embeddings impacted by the new event (typically several thousand), must be updated by re-applying graph convolutions. In the case of HUGNet, only the newly arrived event requires MAC operations. While we only report the latency incurred due to graph building, it is important to note that the fully-spherical approaches will incur a further delay equal to the temporal search radius multiplied by the number of layers (HUGNet requires no such delay) that further increases the total latency of these approaches. More surprisingly however, is the observation that, given the constraints imposed upon the event-graph structure in order to achieve these impressive efficiency gains, HUGNet

<table>
<thead>
<tr>
<th>Features</th>
<th>Edges</th>
<th>GCNConv</th>
<th>B-Spline</th>
<th>Hybrid</th>
</tr>
</thead>
<tbody>
<tr>
<td>XY,T,P,N</td>
<td>XY,T</td>
<td>0.398</td>
<td>0.612</td>
<td>0.557</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.0446</td>
<td>0.0343</td>
<td>0.0371</td>
</tr>
<tr>
<td>XY,P,N</td>
<td>XY,T</td>
<td>0.378</td>
<td>0.563</td>
<td>0.549</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.0466</td>
<td>0.0380</td>
<td>0.0380</td>
</tr>
<tr>
<td>XY,T</td>
<td>XY,T</td>
<td>0.061</td>
<td>0.526</td>
<td>0.294</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.0753</td>
<td>0.0384</td>
<td>0.0498</td>
</tr>
<tr>
<td>XY,N</td>
<td>XY,T</td>
<td>0.374</td>
<td>0.587</td>
<td>0.544</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.0476</td>
<td>0.0358</td>
<td>0.0385</td>
</tr>
<tr>
<td>XYP</td>
<td>XY,T</td>
<td>0.029</td>
<td>0.530</td>
<td>0.346</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.0850</td>
<td>0.0390</td>
<td>0.0484</td>
</tr>
<tr>
<td>XY,T,P,N</td>
<td>XY</td>
<td>-</td>
<td>0.561</td>
<td>0.526</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.0367</td>
<td>0.0397</td>
</tr>
<tr>
<td>XYP</td>
<td>XY</td>
<td>-</td>
<td>0.580</td>
<td>0.536</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.0364</td>
<td>0.0391</td>
</tr>
<tr>
<td>XY,T,P</td>
<td>XY</td>
<td>-</td>
<td>0.510</td>
<td>0.362</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.0391</td>
<td>0.0465</td>
</tr>
</tbody>
</table>

Table 1. Comparison of input node features and the graph convolutional type applied. $F_{25\%}$ (top, higher is better) and AEE (bottom, lower is better) are given for each model. The best result per column is highlighted by a bold font. Features: $(X, Y, T)$ event coordinates, $P$ polarity, $N$ approximate normal vector to local event surface. Edges: $(X, Y, T)$ edge coordinates between events. Note that GCNConv does not support edge features and that all results are obtained using the a hemi-spherical event-graph.

The event-graph neural networks based solely on GCNConv layers consistently exhibit lower performance than the others. They have access only to absolute coordinates as features, and the data aggregation from neighboring events is effectively an averaging. On the contrary, B-spline convolutions exploit relative spatiotemporal differences with respect to neighbouring events which seems to extract more useful representations for the downstream optical flow prediction. However, B-spline convolutions are also more complex than those of GCNConv, since a basis-function weighted combination of products from a 3D-kernel of matrices is required. To better understand the trade-off in computational complexity between the methods, we count the number of parameters, the total number of multiply-and-accumulate operations per second (MAC/s) and the latency of one training epoch in Table 2. We note that, whilst the B-spline method does perform better than GCNConv, it requires an order of magnitude more parameters and double the MAC/s while being several times slower to train. The advantage of the Hybrid approach, where B-spline graph convolution is used only in the first layer, can be clearly seen whereby it greatly improves upon the performance of GCNConv for a relatively modest increase in the number of parameters and MAC/s.

![Table 2. Comparison of model and calculation complexity between the three graph neural network architectures computed on the RS dataset. All results use a hemi-spherical event-graph.](image-url)
Table 3. Benchmarking on the RS dataset. RS has an average event-rate of 18keV/s and Seq-flownet uses a framerate of 100Hz.

Table 4. Benchmarking on the MVSEC dataset. GNN-full is removed from this table - it requires more calculations to update the graph than GNN-sparse, but the underlying graph structure is identical. Indoor and outdoor have respective average event rates of 122kev/s and 198kev/s and key performance indicators (KPIs) are reported for indoor only. *MAC/s due to bi-linear upsampling in the decoder are not computed.

also outperforms the fully-spherical approaches on Rock Scenes. HUGNet achieves a smaller average end-point error and a greatly improved flow accuracy at 25%. The reason for this is not yet well understood and warrants a deeper exploration in future work. Finally, we also observe that both HUGNet and the fully-spherical event-graph neural networks outperform the fully-supervised convolutional approach Seq-flownet, despite requiring an order of magnitude fewer parameters. This underlines the potential of the event-graph approach to become a powerful method in event-based optical flow prediction.

The benchmarking study on the two MVSEC tasks (indoor and outdoor) is presented in Table 4. The full-GNN results have been omitted due to the exaggerated number of MAC/s relative to Sparse-GNN. We additionally report the metric percentage outliers, %Out. As was the case for RS, HUGNet outperforms the fully-spherical event-graph method over all metrics - obtaining an even greater reduction in the graph update latency and MAC/s than observed in RS. While HUGNet outperforms Seq-flownet, in particular on the indoor task, we note that this is not the case (besides $F_{25\%}$ in the outdoor task) compared to EV-flownet - the only dense-frame CNN benchmark trained in a self-supervised fashion [46] (although it should be noted that EV-flownet does require a significantly greater number of MAC/s). Given that Seq-flownet and EV-flownet are largely identical, this echoes observations that self-supervised learning of optical flow can greatly improve upon supervised methods [38], and indicates that a promising future avenue will be to understand how event-graph neural networks can be adapted for self-supervised optical flow learning. Finally, it is indicative to note that while Seq-flownet requires $21 \times$ more MAC/s than HUGNet in RS (at a $100 \times 100$ resolution), in MVSEC ($260 \times 346$) the difference increased to $27 \times$. In fact, as we scale to higher resolution event-cameras, this efficiency gap from CNNs to event-graph neural networks will only increase, and in a quadratic fashion.

5. Conclusion

We have presented HUGNet, an event-graph neural network approach that, with respect to state of the art event-graph methods, reduces graph update latency and the required number multiply-and-accumulate operations per second by up to four orders of magnitude and by $70 \times$ respectively. On two optical flow benchmarks we observed that the hemi-spherical updates of HUGNet allowed for a better accuracy in all cases relative to existing fully-spherical methods. HUGNet was also found to perform favourably relative to dense-frame convolutional approaches, in particular with respect to MAC/s, whereby event-graphs will become increasingly more efficient than CNNs as camera resolution increases. By forming directed (past to future) edges with previously generated events only, we have slashed the latency inherent to continuous graph adaptation and make it possible to apply event-graph neural networks to a stream of event-camera data efficiently, with a low latency and with high fidelity.

Acknowledgements: We would like to thank Z. Hua (Univ. Maryland) for his help during the early phase of this study.
References


