Eventor: An Efficient <u>Event</u>-Based Monocular Multi-View Stereo <u>Accelerator</u> on FPGA Platform *

Mingjun Li[1],   Jianlei Yang[1]*,   Yingjie Qi[1],   Meng Dong[2],   Yuhao Yang[1],
Runze Liu[3],   Weitao Pan[2],   Bei Yu[4],   Weisheng Zhao[1]

[1] Beihang University, Beijing, China   [2] Xidian University, Xi'an, Shaanxi, China
[3] Beijing Real Imaging Medical Technology co., Ltd.   [4] The Chinese University of Hong Kong, Hong Kong
* Corresponding author's Email: jianlei@buaa.edu.cn

Abstract

Event cameras are bio-inspired vision sensors that asynchronously represent pixel-level brightness changes as event streams. Event-based monocular multi-view stereo (EMVS) is a technique that exploits the event streams to estimate semi-dense 3D structure with known trajectory. It is a critical task for event-based monocular SLAM. However, the required intensive computation workloads make it challenging for real-time deployment on embedded platforms. In this paper, Eventor is proposed as a fast and efficient EMVS accelerator by realizing the most critical and time-consuming stages including event back-projection and volumetric ray-counting on FPGA. Highly paralleled and fully pipelined processing elements are specially designed via FPGA and integrated with the embedded ARM as a heterogeneous system to improve the throughput and reduce the memory footprint. Meanwhile, the EMVS algorithm is reformulated to a more hardware-friendly manner by rescheduling, approximate computing and hybrid data quantization. Evaluation results on DAVIS dataset show that Eventor achieves up to 24× improvement in energy efficiency compared with Intel i5 CPU platform.

Keywords

Event-based Vision, Multi-View Stereo, FPGA, Acceleration

1 Introduction

Event cameras are bio-inspired vision sensors developed in recent years [1]. Different from traditional frame-based cameras which capture a scene as a synchronous sequence of 2D images, event cameras asynchronously measure brightness changes on each pixel and output *event* streams. An event encodes the timestamp, pixel coordinates and polarity of brightness changes. Compared with traditional cameras, event cameras have numerous advantages: extremely high event rate ($> 10^6$ events per second, event/s) and dynamic rage (up to 130 dB) while traditional cameras usually obtain ~ 30 FPS and 65 dB, respectively [2]. Additionally, event cameras only require a very low data rate (KB vs. MB) by removing an amount of the inherent redundancy of standard cameras, thus making it quite efficient.

The unique properties of event cameras make them as ideal sensors for running visual SLAM systems on low-power embedded platforms, such as robots and drones, for real-time applications. The event-based monocular visual SLAM systems involve event-based 3D reconstruction which aims to estimate the depth information and the structure of the scene from event cameras. Unlike the multi-view stereo methods, the monocular methods only require a single event camera which

do not pursue instantaneous depth estimation, but rather depth estimation for SLAM [3]. Recently, the event-based monocular multi-view stereo (EMVS) technique has received particular attention, since its performance will greatly affect the overall performance of visual SLAM systems [4]. However, it is very challenging to unlock the benefits of event cameras for monocular multi-view stereo applications on embedded platforms for real-time purpose. This is due to the fact that event cameras represent a paradigm shift in acquisition of visual information, thus requiring novel algorithms and specified hardware design [5]. Previous accelerators designed for traditional intensity-frame-based multi-view stereo algorithms cannot be directly applied for the event-based algorithms.

Several previous algorithms have been proposed for EMVS implementations [6][7][8] but all of them could only run on relatively powerful CPU or GPU platforms. Aiming to improve the computational efficiency of EMVS, an event-based space-sweep method [6] is proposed by back-projecting events to create a ray density volume [9], and then find local maxima of ray density to estimate the scene structure. Such an efficient EMVS implementation integrated with an event-based visual odometry (EVO) system [10] could process 1.2 million event/s when running with a single core of Intel x86 CPU, and 4.7 million event/s with 4 cores [6]. However, running the EMVS algorithms on multi-core x86 CPUs is not practical for embedded EVO applications. Another event processing pipeline is proposed in [7] by utilizing three filters running in parallel to jointly estimate the motion of the event camera and 3D map. Such an approach only runs on GPUs for real-time performance and cannot process high event rate input (up to 1M event/s). A unified event processing framework is proposed in [8] focusing on motion estimation, depth estimation and optical flow estimation. However, such a framework is only evaluated on a desktop CPU and no quantitative results are provided. Overall, all of these implementations are insufficient to fully unlock the potential advantages of event cameras for EMVS systems.

This motivates us to explore more efficient EMVS algorithm-hardware co-design approach for real-time target on low-power embedded platforms. From comparative analysis, we observed that the event-based space-sweep procedures in EMVS have significant advantages including relatively high parallelism, low data dependency and low computational redundancy. These advantages make it very suitable for customized hardware acceleration, which is adopted as the basic framework for our algorithm-hardware co-design and optimizations.

In this paper, Eventor is proposed as an FPGA/ARM heterogeneous accelerator for EMVS systems. The most time-consuming tasks of event back-projection and volumetric ray-counting are performed on FPGA. The main contributions are listed below:
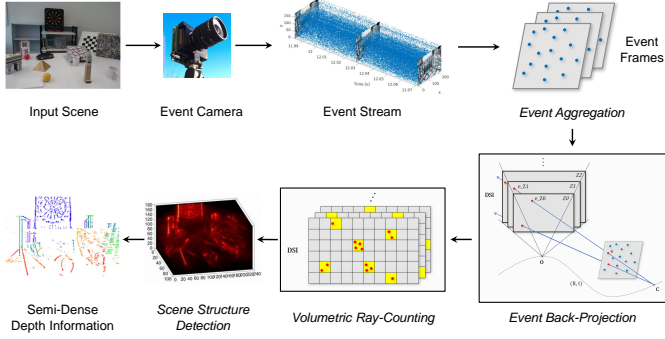
Figure 1: A typical EMVS workflow. This paper focuses on building the semi-dense depth information from the event streams.

✓ A novel efficient EMVS accelerator, Eventor, is proposed for real-time applications on embedded FPGA platform via algorithm-architecture co-design approaches.

✓ The involved EMVS algorithm is redesigned and customized in a hardware-friendly manner, which makes the accelerator much more efficient.

✓ Highly paralleled and fully pipelined architecture is designed and integrated with the heterogeneous execution model to improve the throughput and reduce the memory footprint.

The remainder of the paper is organized as follows. Section 2 demonstrates some comprehensive analysis of EMVS algorithm for potential optimization. Section 3 illustrates the detailed architecture of the proposed Eventor. Evaluation results are provided in Section 4. Finally the conclusions are given in Section 5.

2 EMVS System

In this section, typical EMVS algorithm is analyzed for computational patterns evaluation and reformulated for hardware-friendly targeting. Meanwhile, data quantization and compression strategies are further exploited to improve the computational efficiency.

2.1 Algorithm Analysis

EMVS algorithm aims to address the problem of estimating 3D structure from the event stream acquired by a moving event camera with a known trajectory [6]. A typical EMVS system is depicted in Fig. 1. It mainly consists of four procedures: event *aggregation* ($\mathcal{A}$), event back-*projection* ($\mathcal{P}$), volumetric *ray*-counting ($\mathcal{R}$) and scene structure *detection* ($\mathcal{D}$). The system receives the input event stream and corresponding camera trajectory, and reconstructs the semi-dense depth information of the viewing scene by event-based space-sweep method. The complete workflow of EMVS algorithm is illustrated in Fig. 2, and each stage is described as follows.

Event Aggregation. Specifically, when the logarithmic brightness at a certain pixel (x_k, y_k) reaches a threshold, event camera generates an event $e_k \doteq \langle x_k, y_k, t_k, p_k \rangle$, where x_k and y_k is the corresponding pixel's coordinates of k-th event, t_k is the timestamp of the triggered event and p_k is the polarity of the brightness change. *Aggregation* (denoted as $\mathcal{A}$) divides the generated event stream to event frames (i.e. event packets) which will be processed together.

Event Back-Projection. Event back-*projection* (denoted as $\mathcal{P}$) is the first stage of event-based space-sweep method. Each event in an event frame is back-projected to the viewing space according to the camera pose of the frame. Usually a ray density volume is created to
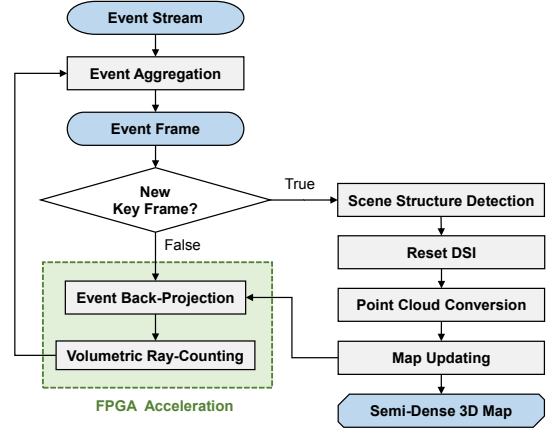

Figure 2: EMVS algorithm framework.

record the distribution of back-projected rays. A disparity space image (DSI) is interchangeably used to describe the discretized space volume and the scores stored in each voxel (i.e., the number of back-projected viewing rays passing through each voxel) [6].

The DSI is defined by dividing the viewing space to N_z slices along the depth and discretizing each slice to $w \times h$ cuboid voxels, where w and h are the horizontal and vertical resolution of the event camera. So the DSI size is $w \times h \times N_z$. Assuming the center of a voxel is $\mathcal{X} = (X, Y, Z)^T$, then back projecting events to the DSI can be discretized to the execution of mapping events to all the depth planes $\{\mathbb{Z}_i\}_{i=1}^{N_z}$ located in the middle of the slices.

By creating a virtual camera located at a reference viewpoint, a DSI could be defined for its view recording. The event back-projection is performed by two steps: ❶ Each event is firstly mapped from the current camera to the virtual camera via a canonical plane $\mathbb{Z}_0$ using homography matrix $\mathcal{H}_{\mathbb{Z}_0}$, which are denoted as $\mathcal{P}(\mathbb{Z}_0)$. The coordinates of events back-projected to $\mathbb{Z}_0$ are denoted as $\{x_k(\mathbb{Z}_0), y_k(\mathbb{Z}_0)\}$. ❷ The other depth planes $\mathbb{Z}_i$ could be obtained by mapping the points from $\mathbb{Z}_0$, which are denoted as $\mathcal{P}(\mathbb{Z}_0 \rightsquigarrow \mathbb{Z}_i)$. The coordinates of events back-projected to $\mathbb{Z}_i$ are denoted as $\{x_k(\mathbb{Z}_i), y_k(\mathbb{Z}_i)\}$.

Volumetric Ray-Counting. After back-projecting events to DSI volume, the second stage of event-based space-sweep method is counting the number of back-projection rays that pass through each voxel (denoted as $\mathcal{R}$). In the previous stage, the ray-voxel intersections are discretized to back projecting events to depth planes $\{\mathbb{Z}_i\}_{i=1}^{N_z}$. Then accumulating votes in the DSI can be done by voting DSI voxels at positions of $\{x_k(\mathbb{Z}_i), y_k(\mathbb{Z}_i), \mathbb{Z}_i\}$.

Key Frame Selection. The EMVS algorithm selects several key reference views along the trajectory of the event camera and constructs local DSI. After setting the original reference viewpoint, a new event frame could be only selected as a new key frame ($\mathcal{K}$) if the distance between the current event camera pose and the previous key reference view exceeds a threshold. All of the events between two key frames will be utilized to estimate the local depth information.

Scene Structure Detection. Scene structure detection ($\mathcal{D}$) is the last stage of event-based space-sweep method. A semi-dense depth map at the reference viewpoint is extracted from the DSI by determining whether a 3D point is present in each DSI voxel. Based on the theory that the regions where multiple back-projection rays nearly intersect are likely to possess scene points, the algorithm determine
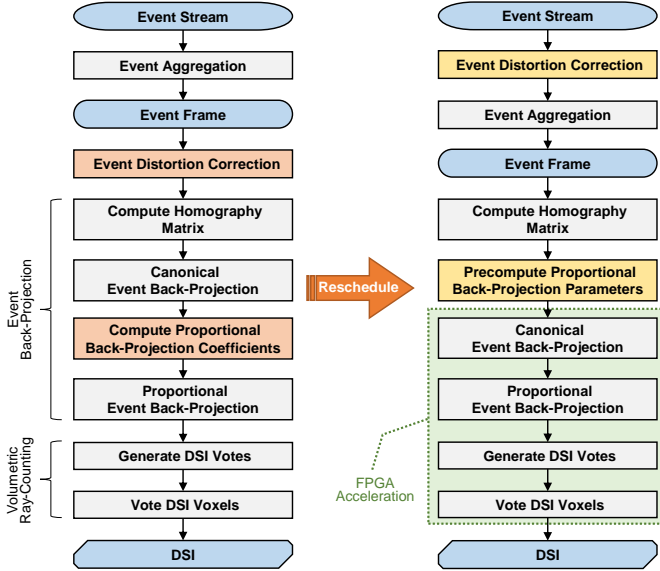
Figure 3: Details of original EMVS framework (left) and rescheduled for hardware-friendly optimization in our Eventor (right).

3D points by finding DSI voxels whose ray density scores are at local maximum of the ray density function.

Merging Depth Information. After getting the semi-dense depth map of the previous reference view, the old local DSI is abandoned and a new local DSI is set in the viewing space of the new reference viewpoint, after the scene structure detection procedure. Then the depth map is converted to a local point cloud and merged into the global point cloud ($\mathcal{M}$). Hence, it includes three steps: point cloud conversion, reset DSI and map updating.

Computational Evaluation. According to our observations, the most computational intensive and time-consuming tasks in the whole algorithm is *event back-projection* ($\mathcal{P}$) and *volumetric ray-counting* ($\mathcal{R}$). When evaluating the EMVS algorithm on the DAVIS event camera dataset [11], the runtime of these two tasks accounts for over 80% of total runtime. To execute EMVS efficiently in real-time on a low-power embedded system, optimizations for these two tasks are obviously required, from both algorithm and hardware perspectives. Hence, the procedures of $\mathcal{P}$ and $\mathcal{R}$ are accelerated by FPGA in our proposed Eventor.

2.2 Hardware-Friendly Reformulation

Aiming to relieve the computational bottleneck ($\mathcal{P}$ and $\mathcal{R}$) of EMVS algorithm, an algorithm-hardware co-optimization approach is proposed where the original algorithm is rescheduled in a hardware-friendly manner as shown in Fig. 3. The event back-projection ($\mathcal{P}$) is divided into four sub-tasks: ❶ *Compute Homography Matrix* aims to compute the homography matrix $\mathcal{H}_{\mathbb{Z}_0}$, ❷ *Canonical Event Back-Projection* corresponds to $\mathcal{P}(\mathbb{Z}_0)$, ❸ *Compute Proportional Back-Projection Parameters* determines the parameters ϕ required in $\mathcal{P}(\mathbb{Z}_0 \rightsquigarrow \mathbb{Z}_i)$, ❹ *Proportional Event Back-Projection* conducts the actual $\mathcal{P}(\mathbb{Z}_0 \rightsquigarrow \mathbb{Z}_i)$. And the volumetric ray-counting ($\mathcal{R}$) is divided into two sub-tasks: *Generate DSI Votes* ($\mathcal{G}$) and *Vote DSI Voxels* ($\mathcal{V}$).

Workload Evaluation. We further evaluate the computational workload of each sub-task for the above $\mathcal{P}$ and $\mathcal{R}$ procedures. Among
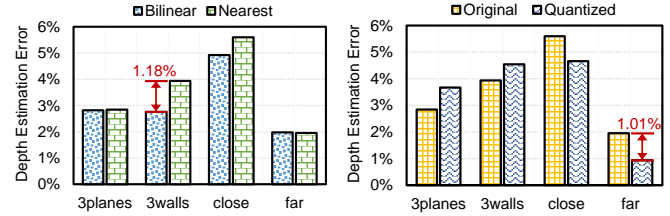
(a) Different voting approaches.   (b) w/ or w/o quantization.

Figure 4: Depth estimation error (AbsRel) comparison between different approaches evaluated on different datasets. The maximum AbsRel difference between Nearest Voting and original Bilinear Voting is about 1.18%. The maximum AbsRel difference before and after quantization is about 1.01%.

all the sub-tasks, *Canonical Event Back-Projection* ($\mathcal{P}(\mathbb{Z}_0)$), *Proportional Event Back-Projection* ($\mathcal{P}(\mathbb{Z}_0 \rightsquigarrow \mathbb{Z}_i)$), *Generate DSI Votes* ($\mathcal{G}$) and *Vote DSI Voxels* ($\mathcal{V}$) will take up most of the runtime, because the required executions are proportional to the number of input events, while the *Homography Matrix* ($\mathcal{H}_{\mathbb{Z}_0}$) and *Proportional Back-Projection Parameters* (ϕ) are only updated once when a new event frame is received. Validation results on the DAVIS dataset show that the four sub-tasks above are responsible for over 90% execution time of $\mathcal{P}$ and $\mathcal{R}$ procedures.

Computation Parallelism Analysis. The above $\mathcal{P}$ and $\mathcal{R}$ procedures could be found with high parallel availability. According to the mechanism of event-based space-sweep method, there are mainly three types of parallelism in workloads:

▷ *Operator-Level Parallelism.* For the involved matrix and vector calculations in the procedure of $\mathcal{P}$, multiple arithmetic logic units (ALUs) could be deployed for fine-grained parallelism.

▷ *Event-Level Parallelism.* The procedure $\mathcal{P}$ requires to back-project each input event to the viewing space separately and extract scene structure from the ray density volume, which does not require simultaneous event observations or event matching. Hence, different events can be processed in parallel and the computation stages involved can be fully pipelined.

▷ *DSI-Level Parallelism.* Due to the discretized structure of DSI and depth planes $\{\mathbb{Z}_i\}_{i=1}^{N_z}$, the procedure $\mathcal{P}$ for different depth planes can be executed in parallel, so can voting for different DSI voxels.

Dataflow Reformulation. According to the evaluation and analysis above, there are four tasks accelerated on FPGA: $\mathcal{P}(\mathbb{Z}_0)$, $\mathcal{P}(\mathbb{Z}_0 \rightsquigarrow \mathbb{Z}_i)$, $\mathcal{G}$ and $\mathcal{V}$. The high parallelism makes accelerating these tasks on FPGA rewarding. However, the dataflow of the original EMVS framework shown in Fig. 3 (left) is not hardware-friendly enough. Rescheduling the original algorithm to a streaming and hardware-friendly manner is proven to be an effective strategy in previous software-hardware co-optimization designs for traditional visual SLAM, such as the ORB-SLAM accelerator in [12]. Therefore, we perform reformulation to the EMVS algorithm for sufficient acceleration on heterogeneous systems. As illustrated in Fig. 3 (right), the reformulation is mainly performed in the aspects of *Rescheduling* and *Approximate Computing*:

▶ *Rescheduling* includes the stages of *Event Distortion Correction* and *Compute Proportional Back-Projection Coefficients*. ❶ *Event Distortion Correction* execution is originally performed after the events aggregated to a whole frame. We set this stage before *Event Aggregation* so that the correction is executed for each event in a streaming

Table 1: Detailed quantization strategies for procedure $\mathcal{P}$ and $\mathcal{R}$. Original floating-point data are quantized by fix-point data.

Quantized Data Type	Total #bit	#bit of Integer	#bit of Decimal
(x_k, y_k)	16	9	7
$\{x_k(\mathbb{Z}_0), y_k(\mathbb{Z}_0)\}$	16	9	7
$\{x_k(\mathbb{Z}_i), y_k(\mathbb{Z}_i)\}$	8	8	0
$\mathcal{H}_{\mathbb{Z}_0}$	32	11	21
ϕ	32	11	21
DSI Scores	16	16	0

manner. Streaming corrections could improve memory access efficiency during the aggregation stage. ❷ *Proportional Back-Projection Coefficients* ϕ is pre-computed before performing $\mathcal{P}(\mathbb{Z}_0)$. With the pre-computed ϕ, the subsequent stages $\mathcal{P}(\mathbb{Z}_0)$, $\mathcal{P}(\mathbb{Z}_0 \rightsquigarrow \mathbb{Z}_i)$, $\mathcal{G}$ and $\mathcal{V}$ could be efficiently accelerated on FPGA in parallel and fully pipelined. Meanwhile, the originally required data transfer of ϕ could be significantly reduced.

▶ *Approximate Computing* is adopted to improve the execution efficiency of procedure $\mathcal{R}$. ❶ A standard DSI voting approach is named *bilinear voting*, which is similar to bilinear interpolation. Bilinear voting adopts a point $\{x_k(\mathbb{Z}_i), y_k(\mathbb{Z}_i), \mathbb{Z}_i\}$ to vote for the corresponding four nearest voxels on depth plane $\mathbb{Z}_i$ by splitting its contribution according to the distance between this point to each voxel. ❷ Another approximate approach is called *nearest voting*, which simply adopts each point to vote for its nearest neighboring voxels. Nearest voting approach is less accurate than bilinear voting. However, the computation complexity and memory access characteristics of nearest voting are much more hardware-friendly than bilinear voting. The depth estimation accuracy comparison between Bilinear Voting and Nearest Voting is illustrated in Fig. 4a by absolute relative error (AbsRel) across different datasets. Fig. 4a shows that the accuracy loss is acceptable when adopting nearest voting. Considering the requirement of hardware-friendly manner, nearest voting is exploited in our dataflow.

2.3 Hybrid Data Quantization

Since most data involved in EMVS dataflow are represented by long floating-point format, we consider converting them as short fixed-point representations to reduce the memory footprint and data transferring bandwidth requirements. Linear quantization method is utilized both for event coordinates and related parameters during the procedure of $\mathcal{P}$ and $\mathcal{R}$. Detailed quantization strategies are illustrated in Table 1.

Event Coordinates Quantization. For event coordinates, we adopt a hybrid quantization strategy. Considering the byte-aligned bit width limitation and the 32-bit data bus width between DRAM and FPGA, we utilize 16-bit data to store the coordinates of the original input events (x_k, y_k). In this way, the coordinates of an event are quantized as a pair of 16-bit data and concatenated to a 32-bit data to be saved in memory. For events generated by DAVIS event camera with resolution of 240×180, 9-bit is enough for integer part of fixed-point coordinates, and remaining 7-bit is exploited for decimal part. Coordinates of $\{x_k(\mathbb{Z}_0), y_k(\mathbb{Z}_0)\}$ are quantized by using the same strategy. As for coordinates of $\{x_k(\mathbb{Z}_i), y_k(\mathbb{Z}_i)\}$, due to the mechanism of nearest voting method adopted in procedure $\mathcal{R}$, finding the nearest voxel to the projected point could be done by rounding the precise floating coordinates to integers. Therefore, their coordinates can be quantized as 8-bit integers.

Parameters Quantization. Since the homography matrix $\mathcal{H}_{\mathbb{Z}_0}$ and pre-computed parameters ϕ are usually invoked repeatedly during
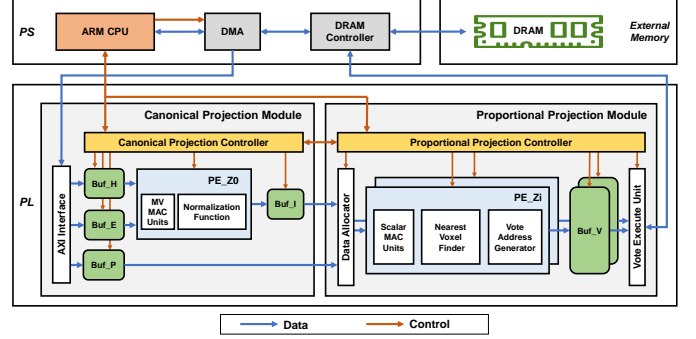
Figure 5: Overall hardware architecture of our proposed Eventor.

the procedures, their precision settings will have larger impact on the whole algorithm. On the other hand, the required memory of these parameters are essentially much less than event coordinates and DSI scores. As an appropriate strategy, they are quantized as 32-bit data with 11-bit integer part and 21-bit decimal part. As our observations, the sufficient integer bit width avoids data overflow, and continuing to increase the decimal bit width will not bring significant improvement to the depth estimation accuracy.

DSI Scores Quantization. For the scores stored in DSI voxels, they are quantized from 32-bit float to 16-bit integer. Benefiting from nearest voting method, the increments (i.e. votes) of the scores are integer so that no decimal part is required. Since the entire DSI structure are usually required to be stored in memory, such a quantization strategy can significantly reduce the memory footprint.

In summary, our hybrid data quantization strategy can save up to 50% of the memory requirement and data transferring bandwidth. Meanwhile, the depth estimation errors resulted from quantization are also evaluated across different datasets and illustrated in Fig. 4b. Evaluation results indicate that the accuracy of our quantized framework is comparable to the original full-precision framework.

3 Eventor Architecture

Base on the reformulated dataflow, overall hardware architecture of Eventor is designed on Zynq FPGA platform as shown in Fig. 5. Eventor is partially implemented with programmable logic (PL) of FPGA and hosted by an ARM CPU as the processing system (PS). *Canonical Projection Module* and *Proportional Projection Module* are exploited to compute $\mathcal{P}(\mathbb{Z}_0)$, $\mathcal{P}(\mathbb{Z}_0 \rightsquigarrow \mathbb{Z}_i)$ and $\mathcal{R}$. For processing each input event frame, ARM configures DMA to transfer input event coordinates (x_k, y_k) and parameters to input buffers. Then ARM sends instructions to start the computational modules. Overall, Eventor receives the input event frames streaming and updates the DSI data stored in DRAM.

3.1 Canonical Projection Module

Canonical Projection Module aims to compute $\mathcal{P}(\mathbb{Z}_0)$. It receives the input event frames, $\mathcal{H}_{\mathbb{Z}_0}$, and outputs $\{x_k(\mathbb{Z}_0), y_k(\mathbb{Z}_0)\}$. It also temporarily stores the proportional back-projection parameters and provides them together with intermediate event coordinates.

AXI Interface supports DMA to transfer input data and parameters via AXI bus. Quantized 16-bit coordinates (x_k, y_k) are concatenated as 32-bit data which are transferred via AXI bus and stored in buffer.

Buffers in Canonical Projection Module include: ❶ Buf_H for storing $\mathcal{H}_{\mathbb{Z}_0}$, ❷ Event Buffer Buf_E for storing input event coordinates

(x_k, y_k), ❸ Proportional Back-Projection Parameter Buffer Buf_P for storing parameters ϕ required in $\mathcal{P}(\mathbb{Z}_0 \rightsquigarrow \mathbb{Z}_i)$, ❹ Intermediate Buffer Buf_I for storing $\{x_k(\mathbb{Z}_0), y_k(\mathbb{Z}_0)\}$. Among them, Buf_H is composed of registers since only one 3×3 homography matrix is required for each input event frame. And the others are built with on-chip BRAM. All of these buffers (including the Vote Buffer Buf_V illustrated in Subsection 3.2 are realized by the manner of *double-buffering*. Many dataflow-driven accelerator designs have adopted this strategy to guarantee continuous loading and output streaming [13]. In this way, the transferring and processing of streaming data can be executed simultaneously, thus avoiding pipeline halt due to wait for input data.

PE_Z0 is the processing element (PE) deployed in Canonical Projection Module for computing $\mathcal{P}(\mathbb{Z}_0)$. It is equipped with a set of matrix-vector multiply-accumulate (MV MAC) units and a normalization function unit. $\mathcal{P}(\mathbb{Z}_0)$ is accelerated by multiple ALUs deployed in PE_Z0, which are fully pipelined. PE_Z0 loads $\mathcal{H}_{\mathbb{Z}_0}$ from Buf_H, then receives streaming (x_k, y_k) from Buf_E and outputs $\{x_k(\mathbb{Z}_0), y_k(\mathbb{Z}_0)\}$ to Buf_I. Since the workload of $\mathcal{P}(\mathbb{Z}_0)$ is less than $\mathcal{P}(\mathbb{Z}_0 \rightsquigarrow \mathbb{Z}_i)$ and $\mathcal{R}$, only one PE_Z0 is deployed. Besides, the latency of computing $\mathcal{P}(\mathbb{Z}_0)$ is not the critical path for normal frames in the pipelined workflow which will be demonstrated in Subsection 3.3.

Controller in Canonical Projection Module mainly receives the starting instructions and configurations, then initializes PE_Z0 and buffers. The Canonical Projection Controller is built as a finite-state machine (FSM), which has a specially designed synchronization state to synchronize the double-buffering state of Buf_E together with the Proportional Projection Controller. This synchronization mechanism ensures two modules to work in a pipelined mode.

3.2 Proportional Projection Module

Proportional Projection Module is responsible for $\mathcal{P}(\mathbb{Z}_0 \rightsquigarrow \mathbb{Z}_i)$ and $\mathcal{R}$. It receives $\{x_k(\mathbb{Z}_0), y_k(\mathbb{Z}_0)\}$ and ϕ from Canonical Projection Module, and updates the DSI voxels scores.

PE_Zi: Canonical Projection Module has multiple PE_Zi to execute $\mathcal{P}(\mathbb{Z}_0 \rightsquigarrow \mathbb{Z}_i)$ and $\mathcal{G}$. PE_Zi receives $\{x_k(\mathbb{Z}_0), y_k(\mathbb{Z}_0)\}$ and ϕ from Data Allocator, and generates the addresses of DSI voxels which are required for Buf_V. PE_Zi include: Scalar MAC Units, Nearest Voxel Finder and Vote Address Generator. Scalar MAC Units execute $\mathcal{P}(\mathbb{Z}_0 \rightsquigarrow \mathbb{Z}_i)$. Nearest Voxel Finder computes the nearest DSI voxel to $\{x_k(\mathbb{Z}_i), y_k(\mathbb{Z}_i), \mathbb{Z}_i\}$ and conducts projection missing judgement. Vote Address Generator generates the vote addresses, which are directly utilized for updating DSI scores. Usually different PEs (multiple PE_Zi) could share a same event input and operate simultaneously in parallel for different depth planes.

Data Allocator fetches input data and parameters required by PE_Zi and allocates them to PEs. Different PEs need different parameters while sharing a same event input. The dataflow between Buf_I and PE_Zi is managed by this allocator.

Vote Execute Unit exploits the DSI vote addresses stored in Buf_V to vote the corresponding voxels. It is equipped with two AXI-HP ports and data transfer logic to directly access the DRAM via the DRAM controller, no need for ARM intervention. The old scores stored in DSI voxels are fetched from DRAM, added by a vote value (typically 1) and wrote back to DRAM.

3.3 Accelerator Workflow

The overall execution model of Eventor is shown in Figure 6. Canonical Projection Module and the Proportional Projection Module
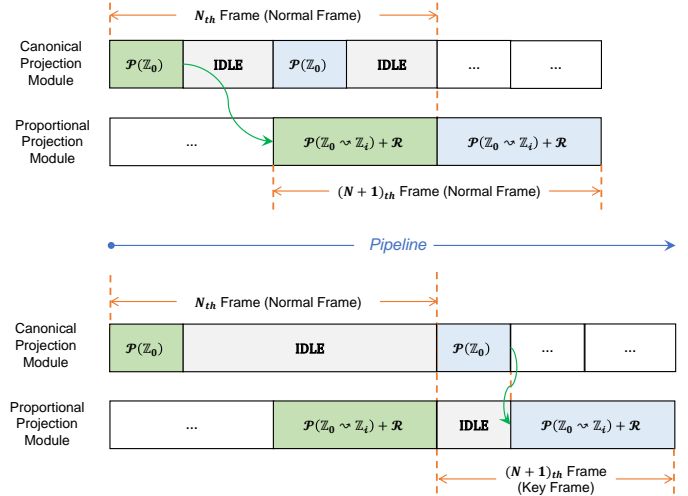

Figure 6: The pipelined workflow of normal event frame (upper) and key event frame (lower).

work in a pipelined order while Eventor receives the streaming input event frames.

For normal event frames, the two modules work simultaneously. Canonical Projection Module starts working as soon as Buf_I is ready for new input so that Proportional Projection Module can operate continuously. In this way, the actual execution time for each frame is equal to the sum of the execution time of $\mathcal{P}(\mathbb{Z}_0 \rightsquigarrow \mathbb{Z}_i)$ and $\mathcal{R}$, and the execution time of $\mathcal{P}(\mathbb{Z}_0)$ is overlapped.

Things are different when a new key event frame is selected. Because a new key frame means a new reference view, the DSI will be reset and the following events will be back projected and vote for the new DSI. So the Canonical Projection Module will wait until the Proportional Projection Module finishes processing the previous event frame, then start processing the key event frame if it is fired up. The Proportional Projection Module then starts to work once receiving $\{x_k(\mathbb{Z}_0), y_k(\mathbb{Z}_0)\}$. Therefore, the execution time for a key frame is equal to the sum of the execution time of $\mathcal{P}(\mathbb{Z}_0)$, $\mathcal{P}(\mathbb{Z}_0 \rightsquigarrow \mathbb{Z}_i)$ and $\mathcal{R}$.

## 3.4 Parallelization Mechanism

According to the computation parallelism analysis carried out in Section 2.2, three levels of parallelism are involved: operator-level, event-level and DSI-level. Eventor aims to fully utilize these parallelism. For operator-level parallelism, we deploy multiple ALUs in PE_Z0 to accelerate matrix and vector calculation. For event-level parallelism, the workflow and datapath of Eventor is designed as a fully-pipelined scheme to process events without data dependency. For DSI-level parallelism, multiple PE_Zi are implemented inside the Proportional Projection Module to back-project an event to multiple depth planes and generate vote addresses simultaneously. Benefiting from the exploration of parallelism, our Eventor is able to achieve a relatively high event processing rate.

4 Experimental Results

This section first introduces our experimental setup. Then, we evaluate the effectiveness of our hardware-friendly dataflow reformulation and the proposed Eventor accelerator.

Table 2: The FPGA resources utilization of Eventor.

	# LUT	# FF	BRAM
Utilization	17538 (32.97%)	22830 (21.46%)	64 KB (11.43%)
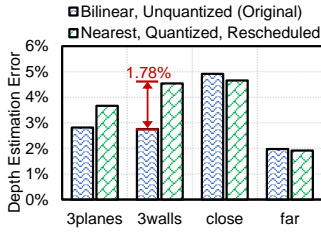
(a) The depth estimation error (AbsREL) of our reformulated hardware-friendly EMVS when compared with original EMVS.
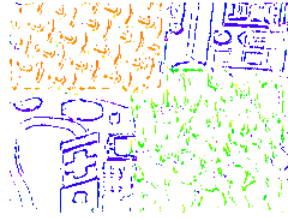
(b) A sample demonstration of reconstructed scene structure from the sequence of *simulation_3planes*.

Figure 7: Accuracy of depth estimation comparison and reconstructed scene structure demonstration.

4.1 Experimental Setup

Hardware Implementation: The Eventor is implemented and evaluated on Xilinx Zynq XC7Z020 SoC. Its PL is with 4.9 Mb BRAM as on-chip memory and 1 GB, 32-bit DDR3 DRAM as external memory. The clock frequency of Eventor is 130 MHz, and the DDR clock is 533 MHz. The prototype of Eventor is equipped with two PE_Zi and corresponding Buf_I in Proportional Projection Module. The resources utilization of Eventor are shown in Table 2. It can be seen that Eventor uses quite few resources.

Dataset: The reformulated EMVS framework and Eventor are evaluated on DAVIS event camera dataset and simulator [11]. It contains event streams captured with a DAVIS event camera in a variety of simulated and real environments, along with ground-truth camera trajectories. The resolution of a DAVIS event camera is 240×180. Four different sequences are used for evaluation: *simulation_3planes* and *simulation_3walls* are simulated sequences, *slider_close* and *slider_far* are captured in real scene.

4.2 Accuracy Analysis

The accuracy of EMVS is measured by depth estimation error (absolute relative error, AbsREL), which means the difference between the depth of reconstructed scene structure and the groundtruth. Fig. 7a shows the comparison of average depth estimation error between original EMVS and our reformulated framework. For *simulation_3planes* and *simulation_3walls*, the original EMVS has a better accuracy than our reformulated framework, but the maximum difference is less than 1.78%. For *slider_close* and *slider_far*, our framework even has a better accuracy than the original EMVS. Overall, the results indicate that the accuracy of our reformulated framework is comparable to original EMVS. A sample reconstructed scene structure from the sequence of *simulation_3planes* is also demonstrated in Fig. 7b for 3D view.

4.3 Performance Evaluation

The performance of Eventor is compared with the EMVS run on Intel i5-7300HQ CPU. Comparison results of computation speed and power consumption are illustrated in Table 3, including detailed runtime breakdown, average runtime per event frame and the event processing rate. Each event frame consists of 1024 events, which is determined according to the sensor's event rate and storage.

Table 3: Performance comparison between Eventor and original EMVS run on Intel i5 CPU.

		Intel CPU	Eventor
Runtime per Event Frame (μs / task)	$\mathcal{P}\left(\mathbb{Z}_0\right)$	22.40	8.24
	$\mathcal{P}\left(\mathbb{Z}_0 \rightsquigarrow \mathbb{Z}_i\right)$ & $\mathcal{R}$	559.55	551.58
Runtime per Event Frame (μs / task)	Normal frame	581.95	551.58
	Key frame	581.95	559.82
Event Processing Rate (10^6 event / second)	Normal frame	1.76	1.86
	Keyframe	1.76	1.83
Power (W)		45	1.86

Compared with the Intel i5 CPU, the event processing rate of Eventor is slightly higher, without obvious advantage. However, in terms of power consumption, Eventor shows great advantage over the Intel CPU. As shown in Table 3, the power consumption can be reduced by 24×. Eventor is able to achieve significant energy reduction with no loss of performance.

5 Conclusions

In this paper, an efficient EMVS accelerator, Eventor, is proposed for real-time applications and evaluated on Zynq FPGA platform. The EMVS algorithm is partly reformulated to a more hardware-friendly manner, and hybrid data quantization strategies are adopted to improve the computational efficiency. Meanwhile, the most time-consuming stages, i.e., event back-projection and volumetric ray-counting are accelerated on FPGA with different parallelism. Evaluation results show that Eventor could achieve 24× improvement in energy efficiency compared with Intel i5 CPU. The overall performance of Eventor could satisfy the requirements of real-time reconstruction on power-limited embedded platforms.

References

[1] Christian Brandli, Raphael Berner, Minhao Yang, Shih-Chii Liu, and Tobi Delbruck. A 240×180 130dB 3µs latency global shutter spatiotemporal vision sensor. *IEEE JSSC*, 49(10):2333–2341, 2014.

[2] Bongki Son, Yunjae Suh, Sungho Kim, et al. A 640× 480 dynamic vision sensor with a 9µm pixel and 300meps address-event representation. In *IEEE ISSCC*, pages 66–67. IEEE, 2017.

[3] Cesar Cadena, Luca Carlone, et al. Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age. *IEEE Transactions on Robotics*, 32(6):1309–1332, 2016.

[4] Yi Zhou, Guillermo Gallego, and Shaojie Shen. Event-based stereo visual odometry. *IEEE Transactions on Robotics*, 37(5):1433–1450, 2021.

[5] Guillermo Gallego, Tobi Delbrück, et al. Event-based vision: A survey. *IEEE TPAMI*, 44(1):154–180, 2020.

[6] Henri Rebecq, Guillermo Gallego, et al. EMVS: Event-based multi-view stereo—3D reconstruction with an event camera in real-time. *IJCV*, 126(12):1394–1414, 2018.

[7] Hanme Kim et al. Real-time 3D reconstruction and 6-DOF tracking with an event camera. In *Proceedings of ECCV*, pages 349–364. Springer, 2016.

[8] Guillermo Gallego, Henri Rebecq, and Davide Scaramuzza. A unifying contrast maximization framework for event cameras, with applications to motion, depth, and optical flow estimation. In *Proceedings of CVPR*, pages 3867–3876, 2018.

[9] Robert T Collins. A space-sweep approach to true multi-image matching. In *Proceedings of CVPR*, pages 358–363. IEEE, 1996.

[10] Henri Rebecq, Timo Horstschäfer, et al. EVO: A geometric approach to event-based 6-DOF parallel tracking and mapping in real time. *IEEE RAL*, 2(2):593–600, 2016.

[11] Elias Mueggler, Henri Rebecq, Guillermo Gallego, Tobi Delbruck, and Davide Scaramuzza. The event-camera dataset and simulator: Event-based data for pose estimation, visual odometry, and SLAM. *IJRR*, 36(2):142–149, 2017.

[12] Runze Liu, Jianlei Yang, Yiran Chen, and Weisheng Zhao. eSLAM: An energy-efficient accelerator for real-time ORB-SLAM on FPGA platform. In *Proceedings of DAC*, pages 1–6, 2019.

[13] Wenzhi Fu, Jianlei Yang, Pengcheng Dai, Yiran Chen, and Weisheng Zhao. A scalable pipelined dataflow accelerator for object region proposals on FPGA platform. In *Proceedings of FPT*, pages 346–349. IEEE, 2018.