sensors

MDPI

Article

Low-Complexity Lossless Coding of Asynchronous Event Sequences for Low-Power Chip Integration

Ionut Schiopu * and Radu Ciprian Bilcu

Tampere Handset Camera Innovation Lab, Huawei Technologies Oy (Finland) Co., Ltd., 33720 Tampere, Finland
* Correspondence: ionut.schiopu@huawei.com

Abstract: The event sensor provides high temporal resolution and generates large amounts of raw event data. Efficient low-complexity coding solutions are required for integration into low-power event-processing chips with limited memory. In this paper, a novel lossless compression method is proposed for encoding the event data represented as asynchronous event sequences. The proposed method employs only low-complexity coding techniques so that it is suitable for hardware implementation into low-power event-processing chips. A first, novel, contribution consists of a low-complexity coding scheme which uses a decision tree to reduce the representation range of the residual error. The decision tree is formed by using a triplet threshold parameter which divides the input data range into several coding ranges arranged at concentric distances from an initial prediction, so that the residual error of the true value information is represented by using a reduced number of bits. Another novel contribution consists of an improved representation, which divides the input sequence into same-timestamp subsequences, wherein each subsequence collects the same timestamp events in ascending order of the largest dimension of the event spatial information. The proposed same-timestamp representation replaces the event timestamp information with the same-timestamp subsequence length and encodes it together with the event spatial and polarity information into a different bitstream. Another novel contribution is the random access to any time window by using additional header information. The experimental evaluation on a highly variable event density dataset demonstrates that the proposed low-complexity lossless coding method provides an average improvement of 5.49%, 11.45%, and 35.57% compared with the state-of-the-art performance-oriented lossless data compression codecs Bzip2, LZMA, and ZLIB, respectively. To our knowledge, the paper proposes the first low-complexity lossless compression method for encoding asynchronous event sequences that are suitable for hardware implementation into low-power chips.

Keywords: low-power electronics; low-complexity codec; lossless compressio; event camera

check for updates

Citation: Schiopu, I.; Bilcu, R.C. Low-Complexity Lossless Coding of Asynchronous Event Sequences for Low-Power Chip Integration. *Sensors* **2022**, *22*, 10014. https://doi.org/10.3390/s222410014

Academic Editors: Ittetsu Taniguchi, Jinjia Zhou and Xin Jin

Received: 21 November 2022
Accepted: 16 December 2022
Published: 19 December 2022

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Copyright: © 2022 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

The recent research breakthroughs in the neuromorphic engineering domain have made possible the development of a new type of sensor, called the event camera, which is bioinspired by the human brain, as each pixel operates individually and mimics the behaviour of a separate nerve cell. In contrast to the conventional camera, in which all pixels are designed to capture the intensity of the incoming light at the same time, the event camera sensor reports only the changes of the incoming light intensity above a threshold, at any timestamp, and at any pixel position by triggering a sequence of asynchronous events (sometimes called spikes); otherwise it remains silent. Because each pixel detects and reports independently only the change in brightness, the event camera sensor proposes a new paradigm shift for capturing visual data.

The event camera provides a series of important technological advantages, such as a high temporal resolution as the asynchronous events can be triggered at a minimum timestamp distance of only 1 μs (10^{-6} s), i.e., the event sensor can achieve a frame rate of up to 1 million (M) frames per second (fps). This is made possible thanks to the

remarkable novel event camera feature of capturing all dynamic information without unnecessary static information (e.g., background), which is an extremely useful feature for capturing high-speed motion scenes for which the conventional camera usually fails to provide a good performance. Two types of sensors are currently available on the market: (i) the dynamic vision sensor (DVS) [1], which captures only the event modality; and (ii) the dynamic and active-pixel vision sensor (DAVIS) [2], which is comprised of a DVS sensor and an active pixel sensor (APS), i.e., it captures a sequence of conventional camera frames and their corresponding event data. The event camera sensors are now widely used in the computer vision domain, wherein the RGB and event-based solutions already provide an improved performance compared with state-of-the-art RGB-based solutions for applications such as deblurring [3], feature detection and tracking [4,5], optic flow estimation [6], 3D estimation [7], superresolution [8], interpolation [9], visual odometry [10], and many others. For more details regarding event-based applications in computer vision, please see the comprehensive literature review presented in [11]. To achieve high frame rates, the captured asynchronous event sequences reach high bit-rate levels when stored using the raw event representation of 8 bytes (B) per event provided by the event camera. Therefore, for better preprocessing of event data on low-power event-processing chips, novel low-complexity and efficient event coding solutions are required to be able to store without any information loss the acquired raw event data. In this paper, a novel low-complexity lossless compression method is proposed for efficient-memory representation of the asynchronous event sequences by employing a novel low-complexity coding scheme so that the proposed codec is suitable for hardware implementation into low-cost event signal processing (ESP) chips.

The event data compression domain is understudied whereas the sensor's popularity continues to grow thanks to improved technical specifications offered by the latest class of event sensors. The problem was tackled in only a few articles that propose to either encode the raw asynchronous event sequences generated by the sensor with or without any information loss [12–14], or to first preprocess the event data from a sequence of synchronous event frames (EFs) that are finally encoded by employing a video coding standard [15,16]. The EF sequences are formed by using an event-accumulation process that consists of splitting the asynchronous event sequence into spatiotemporal neighbourhoods of time intervals, processing the events triggered in a single time interval, and then generating a single event for each pixel position in the EF. These performance-oriented coding solutions are too complex for hardware implementation in the ESP chip designed with limited memory, and may be integrated only in a system on a chip (SoC) wherein enough computation power and memory is available.

In our prior work [17,18], we proposed employing an event-accumulation process which first splits each asynchronous event sequence into spatiotemporal neighbourhoods by using different time-window values, and then generates the EF sequence by using a sum-accumulation process, whereby the events triggered in a time window are represented by a single event that is set as the sign of the event polarity sum and stored at the corresponding pixel position. In [17], we proposed a performance-oriented, context-based lossless image codec for encoding the sequence of event camera frames, in which the event spatial information and the event polarity are encoded separately by using the event map image (EMI) and the concatenated polarity vector (CPV). One can note that the lossless compression codec proposed in [17] is suitable for hardware implementation in SoC chips. In [18], we proposed a low-complexity lossless coding framework for encoding event camera frames by adapting the run-length encoding scheme and Elias coding [19] for EF coding. One can note that the low-complexity lossless compression codec proposed in [18] is suitable for hardware implementation in ESP chips. The goal of this work is to propose a novel low complexity-oriented lossless compression codec for encoding asynchronous event sequences, suitable for hardware implementation in ESP chips.

In summary, the novel contributions of this work are summarized as follows.

(1) A novel low-complexity lossless compression method for encoding raw event data represented as asynchronous event sequences, which is suitable for hardware implementation into ESP chips.
(2) A novel low-complexity coding scheme for encoding residual errors by dividing the input range into several coding ranges arranged at concentric distances from an initial prediction.
(3) A novel event sequence representation that removes the event timestamp information by dividing the input sequence into ordered same-timestamp event subsequences that can be encoded in separated bit streams.
(4) A lossless event data codec that provides random access (RA) to any time window by using additional header information.

The remainder of this paper is organized as follows. Section 2 presents an overview of state-of-the-art methods. Section 3 describes the proposed low-complexity lossless coding framework. Section 4 presents the experimental evaluation of the proposed codecs. Section 5 draws the conclusions of this work.

2. State-of-the-Art Methods

To achieve an efficient representation of the large amount of event data, a first approach was proposed to losslessly (without any information loss) encode the asynchronous event representation. In [12], a lossless compression method is proposed by removing the redundancy of the spatial and temporal information by using three strategies: adaptive macrocube partitioning structure, the address-prior mode, and the time-prior mode. The method was extended in [13] by introducing an event sequence octree-based cube partition and a flexible intercube prediction method based on motion estimation and motion compensation. However, the coding performance of these methods (based on the spike coding strategy) remains limited.

In another approach, the asynchronous event representation is compressed by employing traditional lossless data compression methods. In [14], the authors present a coding performance comparison study of different traditionally based lossless data compression strategies when employed to encode raw event data. The study shows that traditional dictionary-based methods for data compression provide the best performance. The dictionary-based approach consists of searching for matches of data between the data to be compressed and a set of strings stored as a dictionary, in which the goal is to find the best match between the information maintained in the dictionary and the data to be compressed. One of the most well-known algorithms for lossless data compression is the Lempel-Ziv 77 (LZ77) algorithm [20], which was created by Lempel and Ziv in 1977. LZ77 iterates sequentially through the input string and stores any new match into a search buffer. The Zeta Library (ZLIB) [21], an LZ77 variant called deflation, proposed a strategy whereby the input data is divided into a sequence of blocks. The Lempel–Ziv–Markov chain algorithm (LZMA) [22] is an advanced dictionary-based codec developed by Igor Pavlov for lossless data compression, which was first used in the 7-Zip open source code. The Bzip2 algorithm is based on the well-known Burrows–Wheeler transform [23] for block sorting, which operates by applying a reversible transformation to a block of input data.

In a more recent approach [24], the authors propose to treat the asynchronous event sequence as a point cloud representation and to employ a lossless compression method based on a point cloud compression strategy. One can note that the coding performance of such a method depends on the performance of the geometry-based point cloud compression (G-PCC) algorithm used in the algorithm design.

Many of the upper-level applications prefer to consume the event data as an "intensity-like" image rather than asynchronous events sequence, wherein several event-accumulation processes are proposed [25–30] to form the EF sequence. Hence, in another approach, several methods are proposed to losslessly encode the generated EF sequence. The study in [14] was extended in [15] by proposing a time aggregation-based lossless video encoding method based on the strategy of accumulating events over a time interval by creating

two event frames that count the number positive and negative polarity events, which are concatenated and encoded by the high-efficiency video coding (HEVC) standard [31]. Similarly, the coding performance depends on the performance of the video coding standard employed to encode the concatenated frames.

To further improve event data representation, another approach was proposed to encode the asynchronous event sequences by relaxing the lossless compression constraint problem and accepting information loss. In [32], the authors propose a macrocuboids partition of the raw event data, and they employ a novel spike coding framework, inspired by video coding, to encode spike segments. In [16], the authors propose a lossy coding method based on a quad-tree segmentation map derived from the adjacent intensity images. One can note that the information loss introduced by such methods might affect the performance of the upper-level applications.

3. Proposed Low-Complexity Lossless Coding Framework

Let us consider an event camera having a $W \times H$ pixel resolution. Any change of the incoming light intensity triggers an asynchronous event, $e_i = (x_i, y_i, p_i, t_i)$, which stores (based on the sensors representation) the following information in 8 B of memory:

- spatial information $(x_i, y_i), \forall x_i \in [1, H], y_i \in [1, W]$, i.e., the pixel positions where the event was triggered;
- polarity information $p_i \in \{-1, 1\}$,where the symbol "-1" signals a decrease and symbol "1" signals an increase in the light intensity; and
- timestamp t_i, the time when the event was triggered.

Hence, an asynchronous event sequence, denoted as $\mathcal{S}_\mathcal{T} = \{e_i\}_{i=1,2,\ldots,N_e}$, collects N_e events triggered over a time period of $\mathcal{T}$ μs. The goal of this paper is to encode $\mathcal{S}_\mathcal{T}$ by employing a novel, low-complexity lossless compression algorithm.

Figure 1 depicts the proposed low-complexity lossless coding framework scheme for encoding asynchronous event sequences. A novel sequence representation groups the same-timestamp events in subsequences and reorders them. Each same-timestamp subsequence is encoded in turn by the proposed method, called low-complexity lossless compression of asynchronous event sequences (LLC-ARES). LLC-ARES is built based on a novel coding scheme, called the triple threshold-based range partition (TTP).

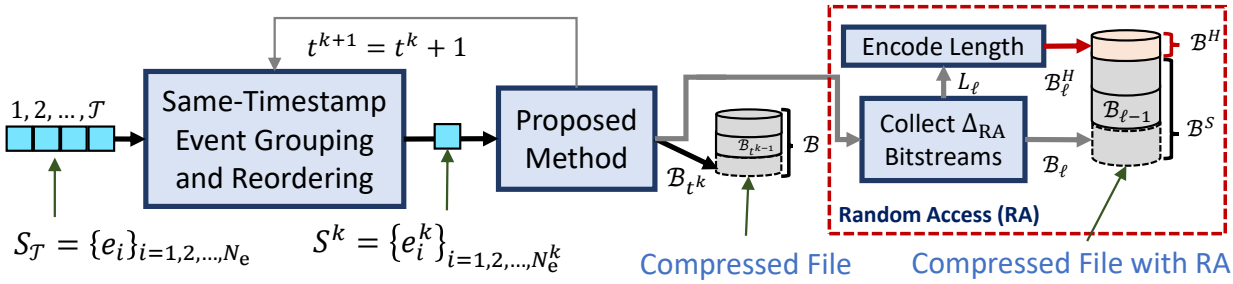


Figure 1. The proposed low-complexity lossless coding framework. The input asynchronous event sequence, $\mathcal{S}_\mathcal{T}$, is first represented by using the proposed event representation as a set of same-timestamp subsequences, $\mathcal{S}^k$, having same-timestamp t^k, and then encoded losslessly by employing the proposed method. The output bitstream of each same-timestamp subsequence can be stored in memory as a compressed file. Moreover, it can also be collected as a package bitstream for all the timestamps found in a time period Δ_{RA} and then stored in memory together with bitstream-length information stored as a header as a compressed file with RA, so that the proposed codec can provide RA to any time window of size Δ_{RA}.

Section 3.1 presents the proposed sequence representation. Section 3.2 presents the proposed low-complexity coding scheme. Section 3.3 presents the proposed method.

3.1. Proposed Sequence Representation

An input asynchronous event sequence, $\mathcal{S}_{\mathcal{T}}$, is arranged as a set of same-timestamp subsequences, $\mathcal{S}_{\mathcal{T}} = \{S^k\}_{k=0,1,\ldots,\mathcal{T}-1}$, where each same-timestamp subsequence $S^k = \{e_i^k\}_{i=1,2,\ldots,N_e^k} = \{(x_i^k, y_i^k, p_i^k)\}_{i=1,2,\ldots,N_e^k}$ collects all N_e^k events in $\mathcal{S}_{\mathcal{T}}$ triggered at the same timestamp t^k. One can note that at the decoder side the timestamp information is recovered based on the subsequence length information, $\{N_e^k\}_{k=0,1,\ldots,\mathcal{T}-1}$, i.e., $t^k = k$ is set to all N_e^k events. Each S^k is ordered in the ascending order of the largest spatial information dimension, e.g., $y_i^k < y_{i+1}^k$. However, if $y_i^k = y_{i+1}^k$, then S^k is further ordered in the ascending order of the remaining dimension, i.e., $x_i^k < x_{i+1}^k$.

Figure 2 depicts the proposed sequence representation and highlights the difference between the sensor's event-by-event (EE) order, depicted on the left side, and the same-timestamp (ST) order, depicted on the right side. Note that the EE order proposes to write to file, in turn, each event e_i. Although the proposed ST order proposes to write to file the number of events of each same-timestamp subsequence, N_e^k having the same-timestamp t^k, and, if $N_e^k > 0$, it is followed by the spatial and the event information of all same-timestamp events, i.e., $\{x_i\}_{i=1:N_e^k}, \{y_i\}_{i=1:N_e^k}, \{p_i\}_{i=1:N_e^k}$. Section 4 shows that the state-of-the-art dictionary-based data compression methods provide an improved performance when the proposed ST order is employed to represent the input data compared with the EE order.

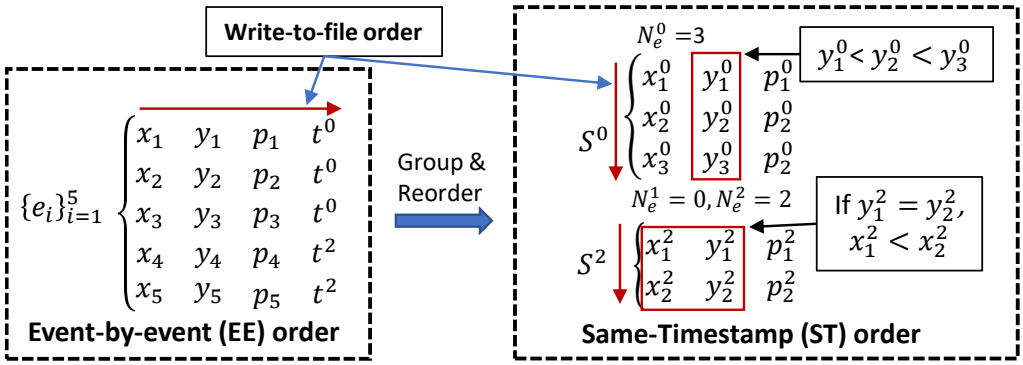


Figure 2. The proposed representation based the proposed same-timestamp (ST) order (on the right) in comparison with the sensor's event-by-event (EE) order (on the left). The red arrow shows the write-to-file order used to generate the input data files feed to the traditional methods.

3.2. Proposed Triple Threshold-Based Range Partition (TTP)

For hardware implementation of the proposed event data codec into low-power event-processing chips, a novel low-complexity coding scheme is proposed. The binary representation range of the residual error is partitioned into smaller intervals selected by using a short-depth decision tree designed based on a triple threshold, $\mathbf{\Delta} = (\delta_1, \delta_2, \delta_3)$. Hence, the input range is partitioned into several smaller coding ranges arranged at concentric distances from the initial prediction.

Let us consider the case of encoding $x \in [1, H]$, i.e., a finite range, by using the prediction $\hat{x}$ by writing the binary representation of the residual error $\epsilon = x - \hat{x}$ on exactly n_ϵ bits. Because on the decoder side n_ϵ is unknown, the triple threshold $\mathbf{\Delta}$ is used to create a decision tree having the role of partitioning the input range $[1, H]$ into five types of coding ranges (see Figure 3a), where either the binary representation of ϵ is represented by using a different number of bits or the binary representation of x is written by using a different number of bits.

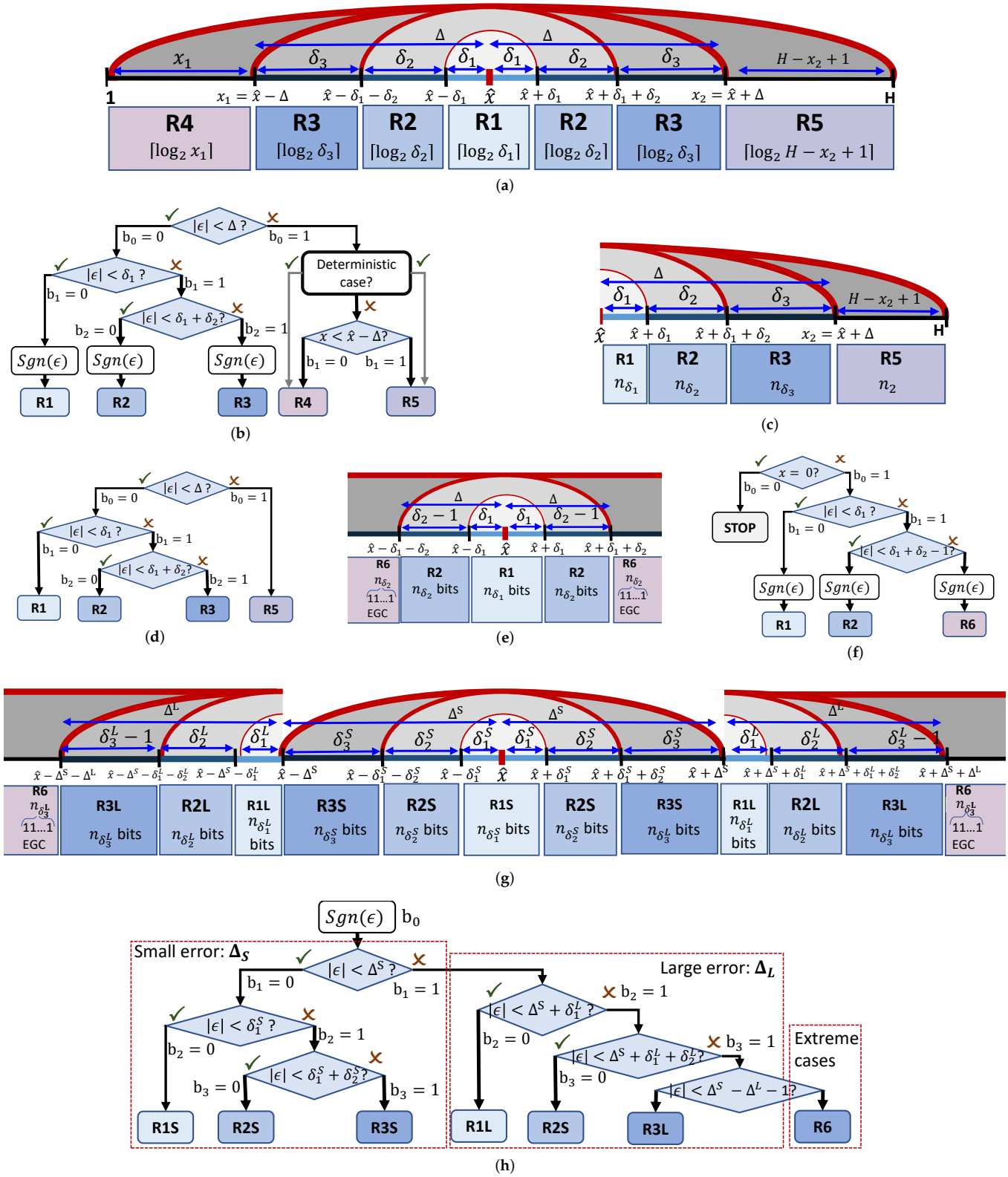


Figure 3. The proposed low-complexity coding scheme, triple threshold-based range partition (TTP). (**a**) TTP range partition. (**b**) TTP decision tree. (**c**) TTP$_y$ range partition. (**d**) TTP$_y$ decision tree. (**e**) TTP$_e$ range partition. (**f**) TTP$_e$ decision tree. (**g**) TTP$_L$ range partition. (**h**) TTP$_L$ range partition.

Let us denote $\Delta = \delta_1 + \delta_2 + \delta_3, x_1 = \hat{x} - \Delta$, $x_2 = \hat{x} + \Delta$, $n_{\delta_j} = \lceil \log_2 \delta_j \rceil, \forall j = 1,2,3$, $n_1 = \lceil \log_2 x_1 \rceil$, and $n_2 = \lceil \log_2 (H - x_2 - 1) \rceil$. The 1st range, R1, is defined by using δ_1 as $(\hat{x} - \delta_1, \hat{x} + \delta_1)$ to represent any residual error $|\epsilon| < \delta_1$ on n_{δ_1} bits plus an additional bit for $sign(\epsilon)$. The 2nd range, R2, is defined by using δ_2 to represent any residual error $|\epsilon| - \delta_1 < \delta_2$ on n_{δ_2} bits plus a sign bit, i.e., $x \in (\hat{x} - \delta_1 - \delta_2, \hat{x} - \delta_1]$ for $\epsilon < 0$ and $x \in [\hat{x} + \delta_1, \hat{x} + \delta_1 + \delta_2)$ for $\epsilon \geq 0$. Similarly, the 3rd range, R3, is defined by using δ_3 to represent any residual error $|\epsilon| - \delta_1 - \delta_2 < \delta_3$ on n_{δ_3} bits plus a sign bit. The 4th (R4) and 5th (R5) ranges are defined for $|\epsilon| \geq \Delta$ and used to represent $x - 1$ on n_1 bits and $H - x$ on n_2 bits, respectively.

Figure 3b depicts the decision tree defined by checking the following four constraints:

(c1) b_0 is set by checking $|\epsilon| < \Delta$. If true then $b_0 = 0$; otherwise, $b_0 = 1$.
(c2) If $b_0 = 0$, then b_1 is set by checking $|\epsilon| < \delta_1$. If true, then $b_1 = 0$ and R1 is employed to represent ϵ on $n_\epsilon = n_{\delta_1} + 1$ bits; otherwise $b_1 = 1$.
(c3) If $b_1 = 1$, then b_2 is set by checking $|\epsilon| < \delta_1 + \delta_2$. If true then $b_2 = 0$ and R2 is employed to represent ϵ on $n_\epsilon = n_{\delta_2} + 1$ bits. Otherwise, $b_1 = 1$ and R3 is used to represent ϵ on $n_\epsilon = n_{\delta_3} + 1$ bits.
(c4) If $b_0 = 1$, then b_1 is set by checking $x \leq x_1$. If true, then $b_1 = 0$ and R4 is employed to represent $x - 1$ on n_1 bits. Otherwise, $b_1 = 1$ and R5 is used to represent $H - x$ on n_2 bits.

Note that the range $[1, x_1]$ contains x_1 possible values. To fully utilize the entire set of code words (i.e., including $00 \cdots 0$ having n_1 bits length), $x - 1$ is represented on n_1 bits.

Algorithm 1 presents the pseudocode of the basic implementation of the TTP encoding algorithm. It is employed to represent a general value x by using the prediction $\hat{x}$, the support range $[1, H]$, and the triple threshold parameter, $\boldsymbol{\Delta}$, as output bitstream B, which contains the decision tree bits, followed by the binary representation of the required additional information for the corresponding coding range. Algorithm 2 presents the pseudocode of the basic implementation of the corresponding TTP decoding algorithm.

Algorithm 1: Encode a general x by using TTP

Data: True value x, prediction $\hat{x}$, range $[1, H]$, and triple threshold $\boldsymbol{\Delta}$;
Result: Output bitstream B;
$B(0) \leftarrow 0$; $B(1) \leftarrow 0$; $\epsilon = x - \hat{x}$; $\Delta = \delta_1 + \delta_2 + \delta_3$;
if $|\epsilon| < \Delta$ **then**
  if $|\epsilon| < \delta_1$ **then** // R1 Range
    $B(2 : \lceil \log_2 \delta_1 \rceil + 2) \leftarrow [sign(\epsilon)$; Write $|\epsilon|$ on $\lceil \log_2 \delta_1 \rceil$ bits];
  else
    if $|\epsilon| < \delta_1 + \delta_2$ **then** // R2 Range
      $B(1 : \lceil \log_2 \delta_2 \rceil + 3) \leftarrow [1; 0; sign(\epsilon)$; Write $|\epsilon| - \delta_1$ on $\lceil \log_2 \delta_2 \rceil$ bits];
    else // R3 Range
      $B(1 : \lceil \log_2 \delta_3 \rceil + 3) \leftarrow [1; 1; sign(\epsilon)$; Write $|\epsilon| - \delta_1 - \delta_2$ on $\lceil \log_2 \delta_3 \rceil$ b];
else
  if $x \leq \hat{x} - \Delta$, **then** // R4 Range
    $B(2 : \lceil \log_2 (\hat{x} - \Delta) \rceil + 1) \leftarrow [$Write $x - 1$ on $\lceil \log_2 (\hat{x} - \Delta) \rceil$ bits];
  else // R5 Range
    $B(1 : \lceil \log_2 (H - \hat{x} - \Delta - 1) \rceil + 1) \leftarrow [1; H - x$ on $\lceil \log_2 (H - \hat{x} - \Delta - 1) \rceil]$;
Return B;

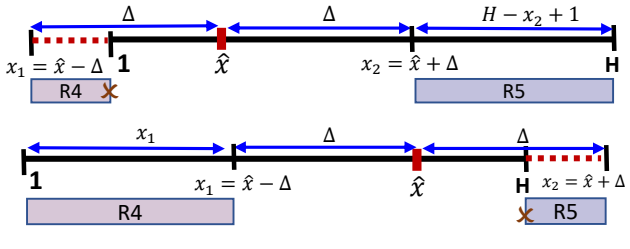
Algorithm 2: Decode a general x by using TTP

Data: Bitstream B; prediction $\hat{x}$, range $[1, H]$, and triple threshold $\boldsymbol{\Delta}$;
Result: True value x;
1 **if** $B(0) = 0$ **then**
2 **if** $B(1) = 0$ **then** // R1 Range
3 $sign_\epsilon \leftarrow B(2)$; $\epsilon_{abs} \leftarrow Dec2bin(B(3 : \lceil \log_2 \delta_1 \rceil + 2)$;
4 **else**
5 **if** $B(2) = 0$ **then** // R2 Range
6 $sign_\epsilon \leftarrow B(3)$; $\epsilon_{abs} \leftarrow \delta_1 + Dec2bin(B(4 : \lceil \log_2 \delta_2 \rceil + 3)$;
7 **else** // R3 Range
8 $sign_\epsilon \leftarrow B(3)$; $\epsilon_{abs} \leftarrow \delta_1 + \delta_2 + Dec2bin(B(4 : \lceil \log_2 \delta_3 \rceil + 3)$;
9 $x \leftarrow sign_\epsilon \cdot \epsilon_{abs}$;
10 **else**
11 **if** $B(1) = 0$ **then** // R4 Range
12 $x \leftarrow 1 + Dec2bin(B(2 : \lceil \log_2 (\hat{x} - \Delta) \rceil + 2)$;
13 **else** // R5 Range
14 $x \leftarrow H - Dec2bin(B(2 : \lceil \log_2 (H - \hat{x} - \Delta - 1) \rceil + 2)$;
15 **Return** x;

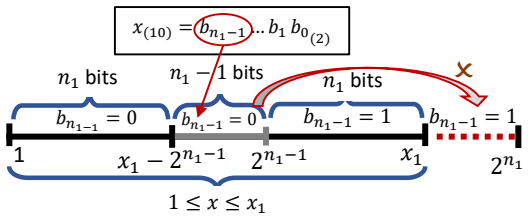
Section 3.2.1 presents the deterministic cases that may occur. Section 3.2.2 analyses the different algorithmic variations proposed to encode the data structures in the proposed event representation that have different properties.

3.2.1. Deterministic Cases

In some special cases, some part of the information can be directly determined from the current coding context. For example, if x_1 or x_2 is outside the finite range (see Figure 4a), then R4 or R5 does not exist and the context tree is built without checking condition (c4), i.e., in such case one bit is saved. More exactly, steps 11–14 in Algorithms 1 and 2 are replaced with either step 12 (encode/decode using R4) or step 14 (encode/decode using R5).



(**a**)



(**b**)

Figure 4. Deterministic cases: (**a**) if $x_1 < 1$ or $x_2 > H$, then condition (c4) is not checked when building the context tree and one bit is saved. (**b**) If $x \in (x_1 - 2^{n_1-1}, 2^{n_1-1}]$, then x is represented by using one bit less than in the case when $x \in [1, x_1 - 2^{n_1-1}]$ or $x \in (2^{n_1-1}, x_1]$.

Moreover, because x_1 and $x_2' = H - x_2 + 1$ are not power-2 numbers, the most significant bit of x, b_{n_1-1}, is 0, thanks to the constraint $1 \le x \le x_1$ and $1 \le x \le x_2'$, respectively. Figure 4b shows that if $x \in (x_1 - 2^{n_1-1}, 2^{n_1-1}]$ and b_{n_1-1} would be set as

1, then $x > x_1$ and the constraint would be violated. Hence, b_{n_1-1} is always set 0 if $x \in (x_1 - 2^{n_1-1}, 2^{n_1-1}]$, (or similarly when $x \in (x'_2 - 2^{n'_2-1}, 2^{n'_2-1}]$).

3.2.2. Algorithm Variations

The basic implementation of the TTP algorithm was modified for encoding different types of data. Let us denote $\epsilon_{x_i^k} = x_i^k - \hat{x}_i^k$ and $\epsilon_{y_i^k} = y_i^k - \hat{y}_i^k$. Then the sequence $\{x_i^k\}_{i=1,2,\ldots,N_e^k}$ is encoded by using version TTP_x, where $\epsilon_{y_i^k}$ is used to detect another deterministic case: if $\epsilon_{y_i^k} = 0$, then $\hat{x}_i^k = x_{i-1}^k$ and the sign bit is saved (see Figure 2 (ST order)). The sequence $\{y_i^k\}_{i=2,3,\ldots,N_e^k}$ having $\epsilon_{y_i^k} \geq 0$ (thanks to ST order) is encoded by using version TTP_y, which is designed to encode a general value x found in range $[\hat{x}, H]$. Figure 3c,d show the TTP_y range partitioning and decision tree, respectively.

Some data types have a very large or infinite support range. The sequence of number of events of each timestamp, $\{N_e^k\}_{k=0,1,\ldots,\mathcal{T}-1}$, is encoded by using version TTP_e. Note that $N_e^k \in [0, HW]$; however, there is a very low probability of having a large majority of pixels triggered with the same timestamp. Therefore, because N_e is usually very small, TTP_e is designed to use the doublet threshold $\boldsymbol{\Delta}_e = (\delta_1, \delta_2)$, as experiments show that a triplet threshold does not improve the coding performance. Figure 3e shows the TTP_e range partitioning, where the values $0, 1, \ldots, \delta_2 - 2$ are encoded by R2 as the last value, $\delta_2 - 1$ (having the binary representation as n_{δ_2} bits of 1, i.e., $\underbrace{11\ldots1}_{n_{\delta_2}}$), signals the use of R6 to encode $|\epsilon| - \Delta - 2$ by using a simple coding technique, the Elias gamma coding (EGC) [19]. Figure 3f shows the decision tree, where $N_e^k = 0$ (i.e., $S^k = \emptyset$) is encoded by the first bit of the decision tree.

Finally, TTP_L is designed to encode the length of the package bitstream $\mathcal{B}_\ell$, denoted as L_ℓ (see Section 3.3.3). TTP_L defines seven partition intervals by using two triple thresholds: $\boldsymbol{\Delta}^S = (\delta_1^S, \delta_2^S, \delta_3^S)$ is used for encoding small errors using R1S, R2S, and R3S, and $\boldsymbol{\Delta}^L = (\delta_1^L, \delta_2^L, \delta_3^L)$ is used for encoding large errors using R1L, R2L, and R3L. Similar to TTP_e, R6 is signalled in R3L by using the last value $\delta_3^L - 1$ and $|\epsilon| - \Delta^S - \Delta^L - 2$ is encoded by employing EGC [19].

3.3. Proposed Method

The proposed method, LLC-ARES, employs the proposed representation to generate the set of same-timestamp subsequences, $\{S^k\}_{k=0,1,\ldots,\mathcal{T}-1}$ (see Section 3.1). Subsequence S^k is encoded as bitstream $\mathcal{B}_{tk}$ by using Algorithm 3, which employs the proposed coding scheme, TTP (see Section 3.2). The compressed file collects these bitstreams as $\mathcal{B} = [\mathcal{B}_{t0}\, \mathcal{B}_{t1} \cdots \mathcal{B}_{t\mathcal{T}-1}]$.

Algorithm 3: Encode the subsequence of ordered events

Data: $S_k = \{(x_i^k, y_i^k, p_i^k)\}_{i=1:N_e^k}$ of same timestamp t^k, $\{N_e^j\}_{j=k-3:k-1}$, H, W;
Result: Output bitstream B_{t^k};
$\hat{N}_e^k \leftarrow$ **Predict** N_e^k using $\{N_e^j\}_{j=k-3:k-1}$;
$B_{t^k} \leftarrow$ **Encode** N_e^k using $\text{TTP}_e(\hat{N}_e^k, \mathbf{\Delta}_e)$;
if $N_e^k > 0$ **then**
  $B_{t^k} \leftarrow$ **Encode** y_1^k using $\text{TTP}_x(\hat{y}_r^k, \epsilon_{y_1^k} > 0, [1, W], \mathbf{\Delta}^{e1})$;
  $B_{t^k} \leftarrow$ **Encode** x_1^k using $\text{TTP}_x(\hat{x}_r^k, \epsilon_{y_1^k} > 0, [1, H], \mathbf{\Delta}^{e1})$;
  $B_{t^k} \leftarrow$ **Encode** p_1^k as 0 for $p_1^k = -1$ and 1 for $p_1^k = 1$;
  for $i = 2, 3, \ldots, N_e^k$ **do**
    $B_{t^k} \leftarrow$ **Encode** y_i^k using $\text{TTP}_y(y_{i-1}^k, [\hat{y}_i^k, W], \mathbf{\Delta}_W^k)$;
    $\hat{x}_i^k \leftarrow$ **Predict** x_i^k using $\{x_j^k\}_{j=1,2,\ldots,i-1}$;
    $B_{t^k} \leftarrow$ **Encode** x_i^k using $\text{TTP}_x(\hat{x}_i^k, \epsilon_{y_i^k}, [1, H], \mathbf{\Delta}_H^k)$;
    $B_{t^k} \leftarrow$ **Encode** p_i^k as 0 for $p_i^k = -1$ and 1 for $p_i^k = 1$;
  end
  $\mathbf{\Delta}_H^{k+1} \leftarrow$ **Update** $\mathbf{\Delta}_H^k$ using $\epsilon^k = y_{N_e^k}^k - y_1^k$;
  $\mathbf{\Delta}_W^{k+1} \leftarrow$ **Update** $\mathbf{\Delta}_W^k$ using $\epsilon^k = y_{N_e^k}^k - y_1^k$;
end
Return B_{t^k};

Algorithm 3 encodes the following data structures:

(i) Encode N_e^k by employing TTP_e using $\hat{N}_e^k$, computed by (1), and $\mathbf{\Delta}_e$;

(ii) Encode e_1^k as follows:

(ii.1) y_1^k by employing TTP_x using $\hat{y}_r^k$ computed by (2), range $[1, W]$, and $\mathbf{\Delta}^{e1}$;
(ii.2) x_1^k by employing TTP_x using $\hat{x}_r^k$ computed by (2), range $[1, H]$, and $\mathbf{\Delta}^{e1}$; and
(ii.3) p_1^k using binarization;

(iii) The remaining events are encoded as follows:

(iii.1) y_i^k by employing TTP_y using $\hat{y}_i^k = y_{i-1}^k$, range $[\hat{y}_i^k, W]$, and $\mathbf{\Delta}_W^k$;
(iii.2) x_i^k by employing TTP_x using $\hat{x}_i^k$ computed by (3), $\epsilon_{y_i^k}$, range $[1, H]$, and $\mathbf{\Delta}_H^k$; and
(iii.3) p_i^k using binarization.

(iv) Update the triple thresholds $\mathbf{\Delta}_H^k$ and $\mathbf{\Delta}_W^k$.

The decoding algorithm can be simply deducted by replacing the TTP encoding algorithm in Algorithm 3 with the corresponding decoding algorithm.

Section 3.3.1 describes the prediction of each type of data used in the proposed event representation. Section 3.3.2 provides information about the setting of the triple thresholds used in the proposed method. Section 3.3.3 describes the variation of LLC-ARES algorithm to provide RA to any time window Δ_{RA}. Finally, Section 3.3.4 presents a coding example.

3.3.1. Prediction

To be able to employ each one of the four algorithm variations, TTP_x, TTP_y, TTP_e, and TTP_L, four types of predictions, $\hat{N}_e^k$, $(\hat{x}_r^k, \hat{y}_r^k)$, $\hat{x}_i^k$, $\hat{L}_\ell$, are computed by using the following set of equations:

$$\hat{N}_e^k = \begin{cases} \tau_e & \text{if } k = 0, \\ N_e^1 & \text{if } k = 1, \\ \frac{N_e^1 + N_e^1}{2} & \text{if } k = 2, \\ \frac{N_e^{k-3} + N_e^{k-2} + 2N_e^{k-1}}{4} & \text{if } k \geq 3. \end{cases} \quad (1)$$

$$(\hat{x}_r^k,\, \hat{y}_r^k) = \begin{cases} (\frac{H}{2},\, \frac{W}{2}) & \text{if } k = 0, \\ (x_1^\kappa,\, y_1^\kappa + \tau_y) & \text{if } k > \kappa > 0, N_e^\kappa > 0. \end{cases}, \tag{2}$$

$$\hat{x}_i^k = \begin{cases} x_{i-1}^k & \text{if } i = 1 \text{ or } \epsilon_{y_i^k} = 0, \\ \frac{x_{i-1}^k + x_{i-2}^k}{2} & \text{if } i = 2, \\ med(\{x_{i-j}\}_{j=1:w_1}) & \text{if } i > 2 \text{ and } |\epsilon_{y_i^k}| < \tau_x, \\ med(\{x_{i-j}\}_{j=1:w_2}) & \text{if } i > 2 \text{ and } |\epsilon_{y_i^k}| \geq \tau_x. \end{cases}, \tag{3}$$

$$\hat{L}_\ell = \begin{cases} 2^{7+\lceil log_2 \Delta_{RA} \rceil} & \text{if } \ell = 1, \\ L_{\ell-1} & \text{otherwise.} \end{cases}. \tag{4}$$

In (2), the prediction for the spatial information of the first event, e_1^0, in the same-timestamp subsequence S^k, is set as the sensor's centre $(\frac{H}{2}, \frac{W}{2})$, whereas the rest of the values depend on the first event e_1^κ of the previously nonempty same-timestamp subsequence S^κ. In (3), if $\epsilon_{y_i^k}$ is small, $\hat{x}_i^k$ is set as the median of a small prediction window of size w_1; otherwise it is of a larger prediction window of size w_2. In our work, we set the parameters as follows: $\tau_e = 10$, $\tau_x = 2^3 + 2^4$, $\tau_y = 3$, $w_1 = 5, w_2 = 15$.

3.3.2. Threshold Setting

In this paper, the triple threshold parameters, $\mathbf{\Delta}_e, \mathbf{\Delta}^{e1}, \mathbf{\Delta}^S, \mathbf{\Delta}_H^{k+1}, \mathbf{\Delta}_W^{k+1}$, and $\mathbf{\Delta}^L$ are selected as power-2 numbers, and are set as follows:

$$\mathbf{\Delta}_e = (2^2, 2^2), \tag{5}$$

$$\mathbf{\Delta}^{e1} = (2^3, 2^4, 2^5), \tag{6}$$

$$\mathbf{\Delta}_H^{k+1} = \begin{cases} \mathbf{\Delta}^{e1} & \text{if } k = 0 \\ (2^5, 2^5, 2^6) & \text{if } k > 0 \;\&\; \epsilon^k < 8, \\ (2^4, 2^4, 2^5) & \text{otherwise} \end{cases} \tag{7}$$

$$\mathbf{\Delta}_W^{k+1} = \begin{cases} (2^2, 2^3, 2^4) & \text{if } k = 0 \\ (2^1, 2^1, 2^2) & \text{if } k > 0 \;\&\; \epsilon^k < 4 \\ (2^1, 2^2, 2^3) & \text{if } k > 0 \;\&\; \epsilon^k < 8 \\ (2^2, 2^2, 2^3) & \text{if } k > 0 \;\&\; \epsilon^k < 16 \\ (2^2, 2^3, 2^4) & \text{otherwise} \end{cases}, \tag{8}$$

$$\mathbf{\Delta}^S = (2^8, 2^{10}, 2^{12}), \tag{9}$$

$$\mathbf{\Delta}^L = (2^{5+\lceil log_2 \Delta_{RA} \rceil}, 2^{7+\lceil log_2 \Delta_{RA} \rceil}, 2^{9+\lceil log_2 \Delta_{RA} \rceil}). \tag{10}$$

3.3.3. Random Access Functionality

LLC-ARES-RA is an LLC-ARES version which provides RA to any time window of size Δ_{RA}. Hence, $\mathcal{S}_\mathcal{T}$ is now divided into $\mathcal{P} = \lceil \frac{\mathcal{T}}{\Delta_{RA}} \rceil$ packages of Δ_{RA} time-length, denoted $\mathcal{S}_\mathcal{T} = \{\mathcal{S}_\ell\}_{\ell=1,2,\ldots,\mathcal{P}}$. The proposed LLC-ARES is employed to encode each package $\mathcal{S}_\ell$ as the bitstream set $\{\mathcal{B}_{t^k}\}_{k=0,1,\cdots,\Delta_{RA}-1}$, which is collected as the package ℓ bitstream, $\mathcal{B}_\ell = [\mathcal{B}_{t^0}\, \mathcal{B}_{t^1} \cdots \mathcal{B}_{t^{\Delta_{RA}}}]$, having L_ℓ bit length. The TTP_L version is employed to encode L_ℓ using the prediction $\hat{L}_\ell$, computed using (4), and the two triple threshold $\mathbf{\Delta}^S$ and $\mathbf{\Delta}^L$, and to generate the header bitstream, $\mathcal{B}_\ell^H$, as depicted in Figure 1. Hence, the bitstreams of the set $\{L_\ell\}_{\ell=1,2,\ldots,\mathcal{P}}$ are collected by the header bitstream, denoted as $\mathcal{B}^H = [\mathcal{B}_1^H\, \mathcal{B}_2^H \cdots \mathcal{B}_\mathcal{P}^H]$, whereas all package bitstreams are collected by the sequence bitstream, denoted as $\mathcal{B}^\mathcal{S} = [\mathcal{B}_1\, \mathcal{B}_2 \cdots \mathcal{B}_\mathcal{P}]$. Finally, the compressed file with RA collects the $\mathcal{B}^H$ and $\mathcal{B}^\mathcal{S}$ bitstreams in this order.

3.3.4. A Coding Example

Figure 5 presents in detail the workflow of encoding by using the proposed LLC-ARES method an asynchronous event sequence of 2 μs time-length, containing 23 triggered events. The input sequence received from the event sensor is initially represented by using the EE order. The proposed sequence representation is employed by first grouping and then rearranging the asynchronous event sequence by using the ST order. Because the input sequence contains two timestamps, the ST order consist of the same-timestamp subsequence S^0 of 10 events and the same-timestamp subsequence S^1 or 13 events. LLC-ARES encodes each data structure by using different TTP variations as described in Algorithm 3.

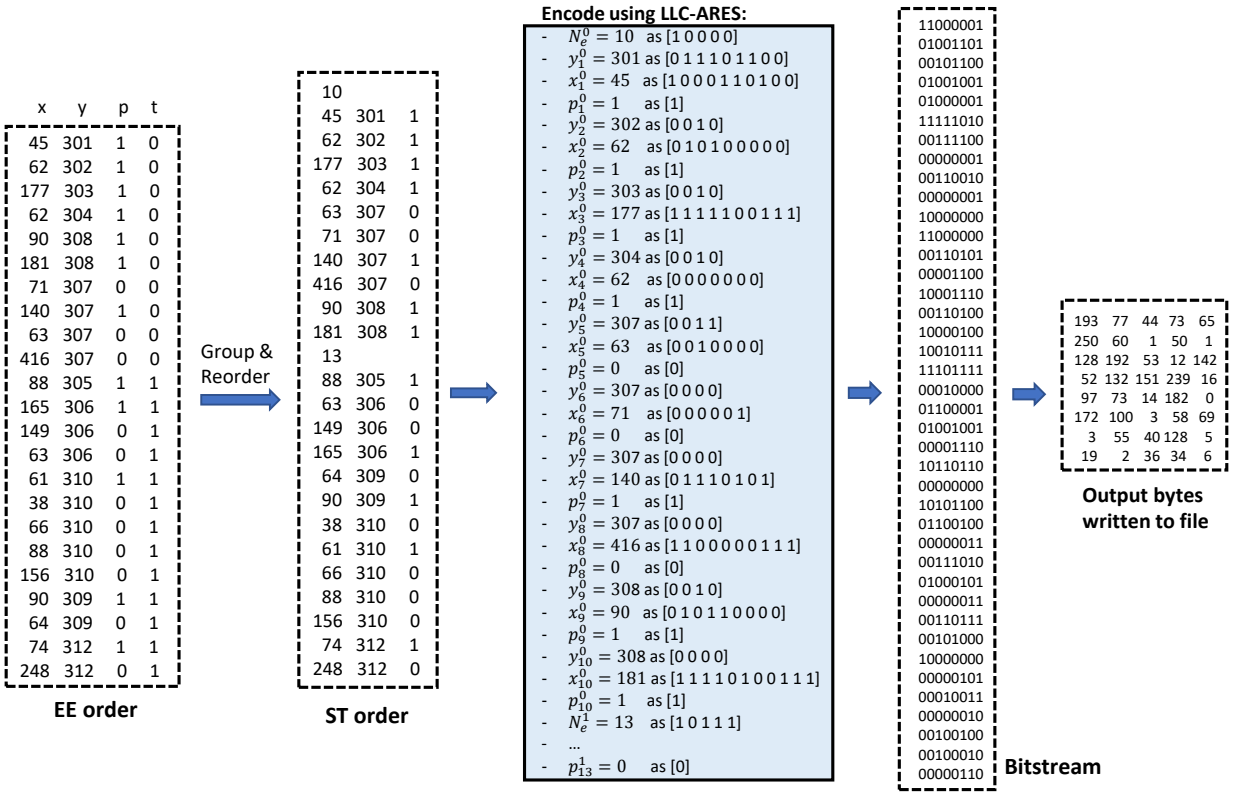


Figure 5. The encoding workflow using the proposed LLC-ARES method as an asynchronous event sequence of 2 µs time-length, containing 23 events. The input sequence, represented by using the EE order, is first grouped and rearranged by using the proposed ST order. LLC-ARES encodes each data structure by using different TTP variations as an output bitstream of 316 bits stored by using 40 bytes, i.e., 40 numbers having an 8-bit representation.

4. Experimental Evaluation

4.1. Experimental Setup

In our work, the experimental evaluation is carried out on large-scale outdoor stereo event camera datasets [33], called DSEC. They contain 82 asynchronous event sequences captured for network training (training data) by using the Prophesee Gen3.1 event sensor placed on top of a moving car, having a $W \times H = 640 \times 480$ pixel resolution. All results reported in this paper use the DSEC asynchronous event sequences sorted in the ascending order of their event acquisition density. By driving at different speeds and in different outdoor scenarios, the DSEC sequences provide a highly variable density of events (see Figure 5a, in which one can see that the event density variates between 5 and 30 Mevps). Figure 6b depicts the cumulated number of events over the first 10 s of the DSEC sequences having the lowest, medium, and highest acquired event density shown in Figure 6a. To limit the runtime of state-of-the-art codecs, for each event sequence, only the first $\mathcal{T} = 10^8$ µs

(100 s) of captured event data are encoded in this work. The DSEC dataset is made publicly available online [34].

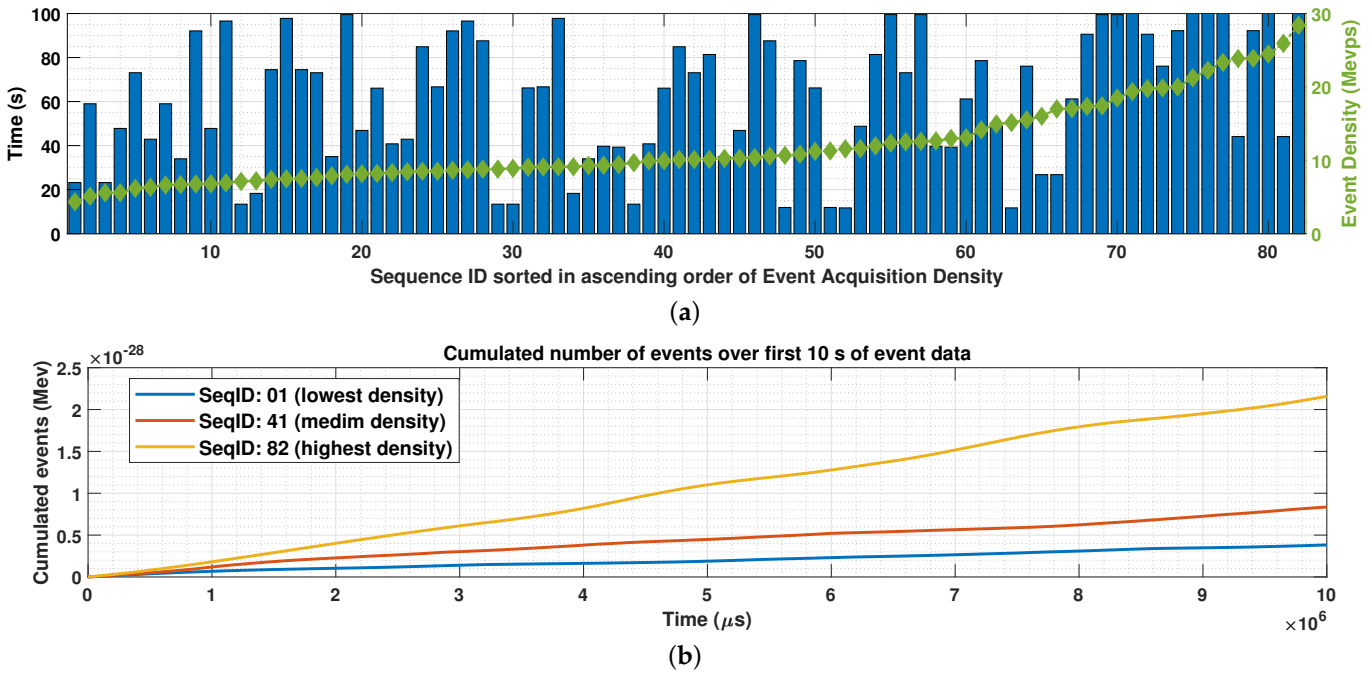


Figure 6. (**a**) The DSEC sequence time length (s) and event density (Mevps), where the asynchronous event sequences are sorted in ascending order of the sequence acquisition density and the sequence time length was constrained to contain only the first $\mathcal{T} = 10^8$ µs (100 s) of the captured event data. (**b**) The cumulated number of events (Mev) over the first 10 s of the DSEC sequences having the lowest (SeqID: 01), medium (SeqID: 41), and highest (SeqID: 82) acquired event density.

The proposed method, LLC-ARES, is implemented in the C programming language. The LLC-ARES-RA version is tested by using a time window of Δ_{RA} of 10^2 µs, 10^3 µs, and 10^4 µs, where for each event sequence only the first $\mathcal{T} = 10^7$ µs of captured event data are encoded. The raw data size is computed by using the sensor specifications of 8 B per event.

The compression results are compared by using the following metrics:

(c1) Compression ratio (CR), defined as the ratio between the raw data size and the compressed file size;
(c2) Relative compression (RC), defined as the ratio between the compressed file size of a target codec and the compressed file size of LLC-ARES; and
(c3) Bit rate (BR), defined as the ratio between the compressed file size in bits and the number of events in the asynchronous event sequence, measured in bits per event (bpev), e.g., raw data has 64 bpev.

The runtime results are compared by using the following metrics:

(t1) Event density (ρ_E), defined as the ratio between the number of events in the asynchronous event sequence and the encoding/acquisition time, measured in millions of events per second (Mevps);
(t2) Time ratio (TR), defined as the ratio between the data acquisition time and the codec encoding time; and
(t3) Runtime, defined as the ratio between the encoding/decoding time (µs) and the number of events.

The LLC-ARES performance is compared with the following state-of-the-art traditional data compression codecs:

(a) ZLIB [21] (version 1.2.3 available online [35]);
(b) LZMA [22]; and
(c) Bzip2 (version 1.0.5 available online [36]).

One can note that the comparison with [12] was not possible, as the codec is not publicly available and the dataset is made available only for academic research purposes.

4.2. Compression Results

Figure 7 shows the CR results and Figure 8 shows the BR results over DSEC [34]. One can note that, for state-of-the-art methods, the proposed ST order provides an improved performance of up to 96% compared with the sensor's EE order. LLC-ARES (designed for low-power chip integration) provides an improved performance compared with all state-of-the-art codecs (designed for SoC integration) over the sequences having a small and medium event density, and a close performance over the sequences having a high event density as more complex coding techniques are employed by the traditional lossless data compression methods.

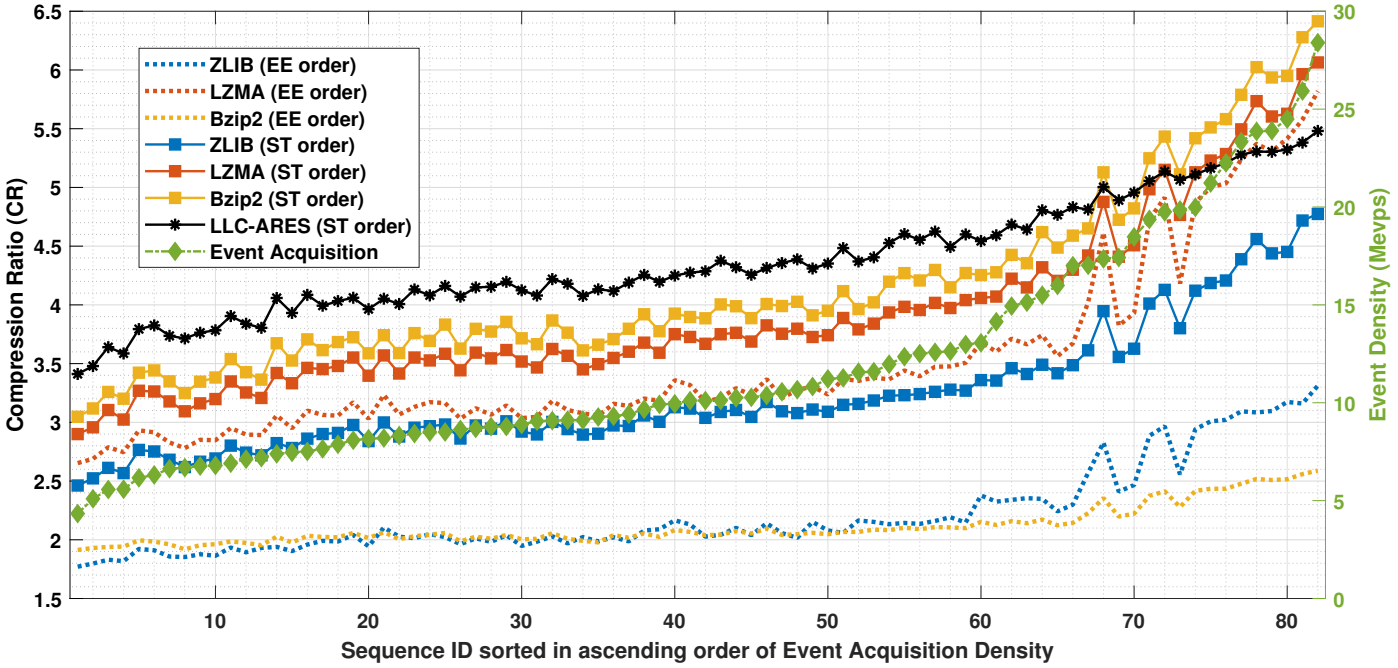


Figure 7. The compression ratio (CR) results over the DSEC dataset [34], where the asynchronous event sequences are sorted in ascending order of the sequence acquisition density.

Table 1 shows the average CR and BR results over DSEC [34].One can note that, compared with the state-of-the-art performance-oriented lossless data compression codecs, Bzip2, LZMA, and ZLIB, the proposed LLC-ARES codec provides the following:

(i) an average CR improvement of 5.49%, 11.45%, and 35.57%, respectively;
(ii) an average BR improvement of 7.37%, 13.40%, and 37.12%, respectively; and
(iii) an average bitsavings of 1.09 bpev, 1.99 bpev, and 5.50 bpev, respectively.

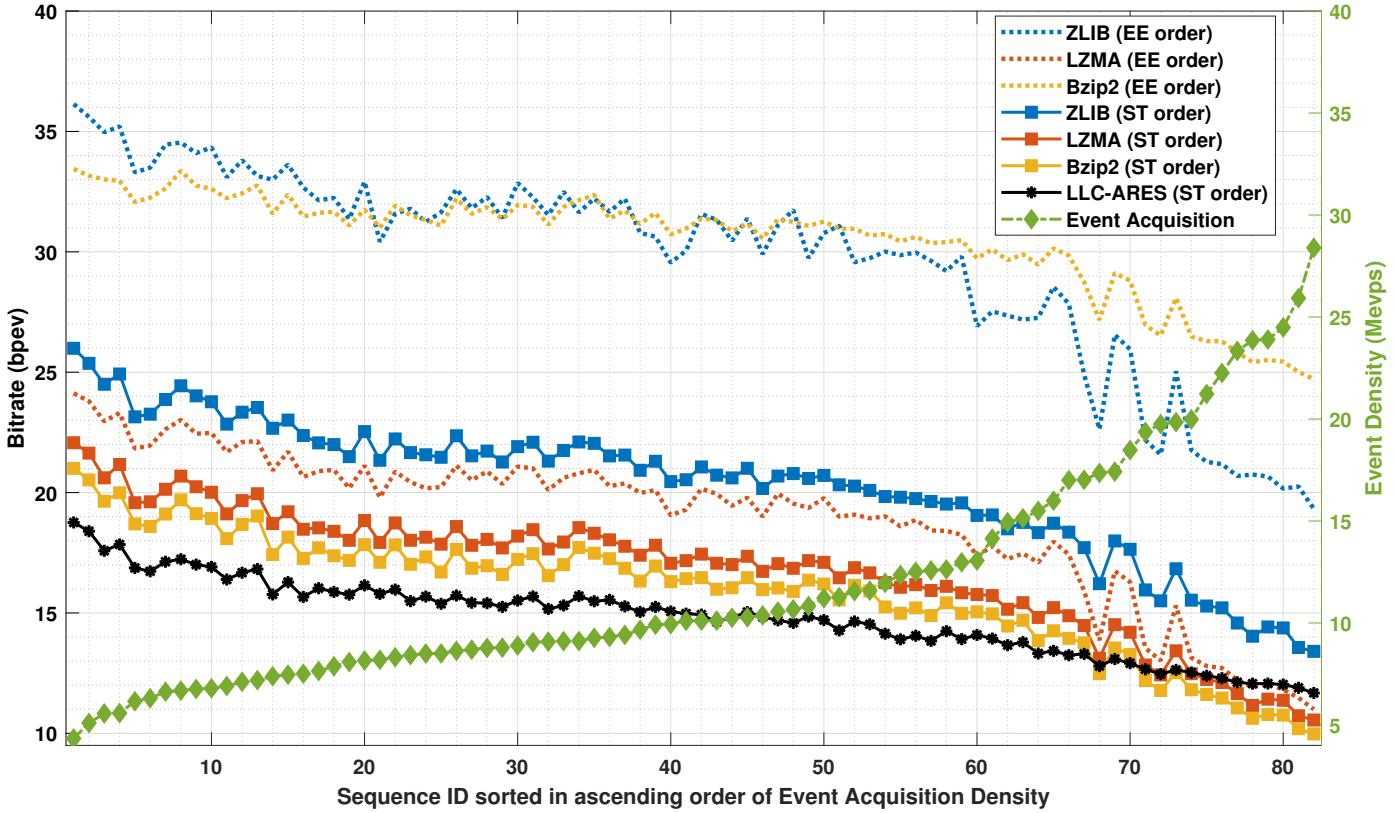


Figure 8. The bitrate (BR) results over DSEC [34], where the asynchronous event sequences are sorted in ascending order of the sequence acquisition density.

Table 1. Average performance over DSEC by using the EE and ST order.

Method		ZLIB [35]	LZMA [22]	bzip2 [36]	Proposed LLC-ARES
CR	EE order	2.21	3.51	2.11	–
	ST order	3.22	3.92	4.14	**4.3**
EBR (bpev)	EE order	29.65	18.91	30.50	–
	ST order	20.32	16.80	15.91	**14.8**
ρ_E (Mevps)	ST order	1.392	0.275	2.453	**5.736**
TR	ST order	0.133	0.027	0.246	**0.531**

4.3. Runtime Results

Figure 9 shows the event density results and Figure 10 shows the TR results over DSEC. One can note that compared with runtime performance of state-of-the-art codecs, LLC-ARES provides a performance much closer to real time for all sequences, and an outstanding performance for the sequences having a high event density. More exactly, LLC-ARES provides a much faster coding speed than the state of the art for the case of high event acquisition density. Whereas the asynchronous event sequences have a very low event acquisition density, LLC-ARES provides an encoding speed as close as approximately 90% of the real-time performance (see Figure 10). Moreover, the software implementation was not optimized, as it can be further improved by a software developer expert to provide an improved runtime performance when deployed on an ESP chip.

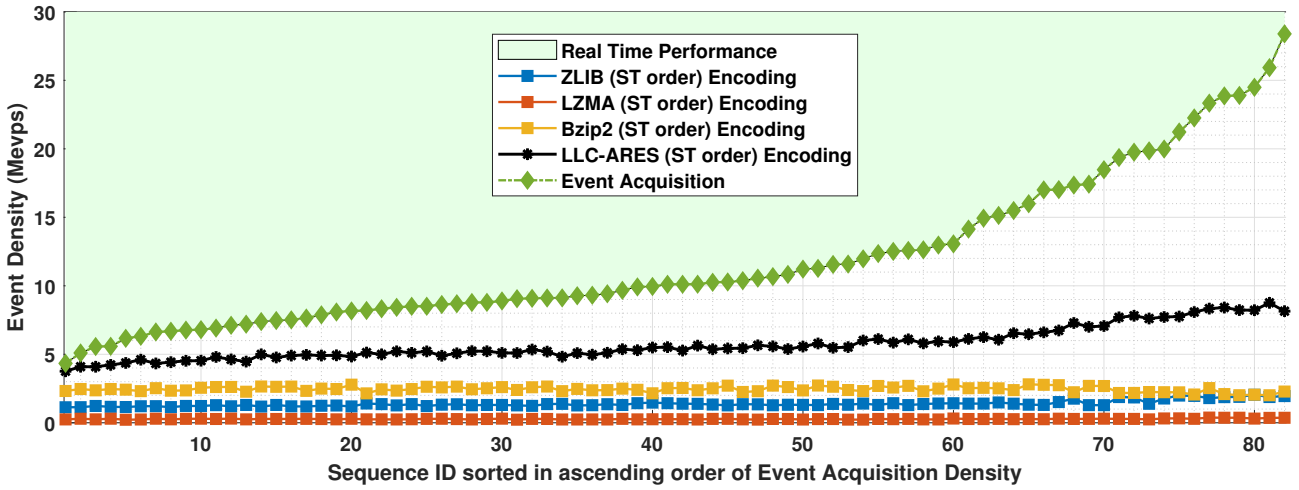


Figure 9. The encoded event density results over the DSEC dataset [34], where the asynchronous event sequences are sorted in ascending order of the sequence acquisition density.

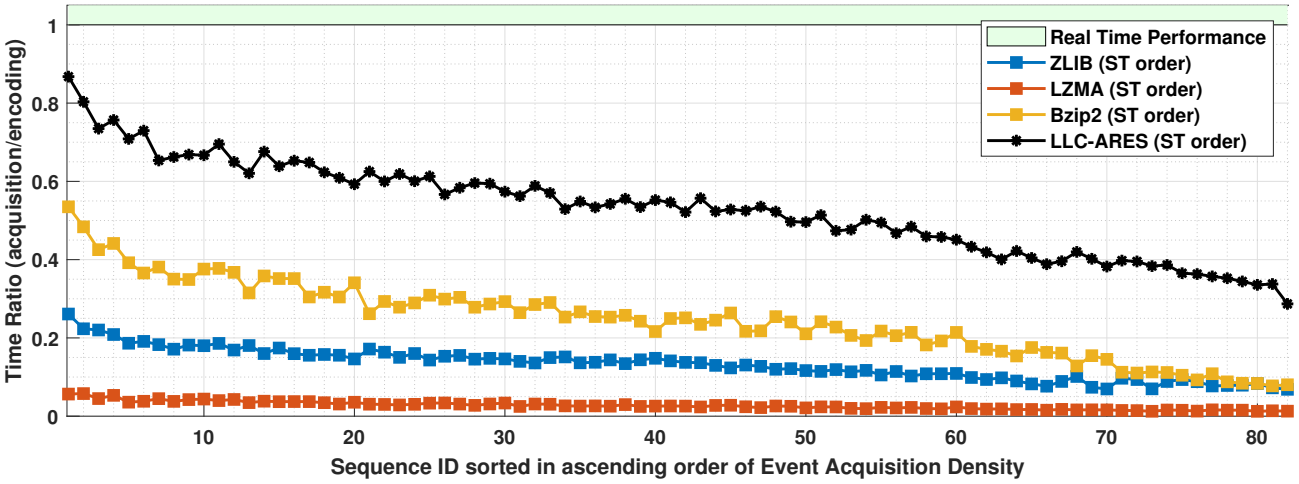


Figure 10. The time ratio (TR) results over the DSEC dataset [34], where the asynchronous event sequences are sorted in ascending order of the sequence acquisition density.

Table 1 shows the average event density and TR results over DSEC. One can note that, compared with the state-of-the-art lossless data compression codecs, Bzip2, LZMA, and ZLIB, the proposed LLC-ARES codec provides the following:

(i) an average event density improvement of 234×, 412×, and 2086×, respectively; and
(ii) an average TR improvement of 216×, 401×, and 1969×, respectively.

Figures 11 and 12 show the encoding and decoding runtime over DSEC, respectively. Note that LLC-ARES is a symmetric codec, wherein the encoder and decoder have similar complexity and runtime, whereas the traditional state-of-the-art lossless data compression methods are asymmetric codecs, as the encoder is much more complex than the decoder. Table 2 presents the average results over DSEC by using the EE order and the proposed ST order. Note that the LLC-ARES performance is approximately 10 µs/ev for both encoding and decoding, while the traditional state-of-the-art lossless data compression methods achieve an encoding time between 135% and 515% higher than LLC-ARES and a decoding time between 92% lower and 58% higher than LLC-ARES.

The implementation of LLC-ARES was not optimized, as the implemented method must be redesigned for integration into low-power chips. These experimental results show that a proof-of-concept implementation of the algorithm on a CPU machine provides an improved performance compared with the state-of-the-art methods when tested on

the same experimental setup. Please note that only LLC-ARES employs simple coding techniques so that it is suitable for hardware implementation into low-power ESP chips.

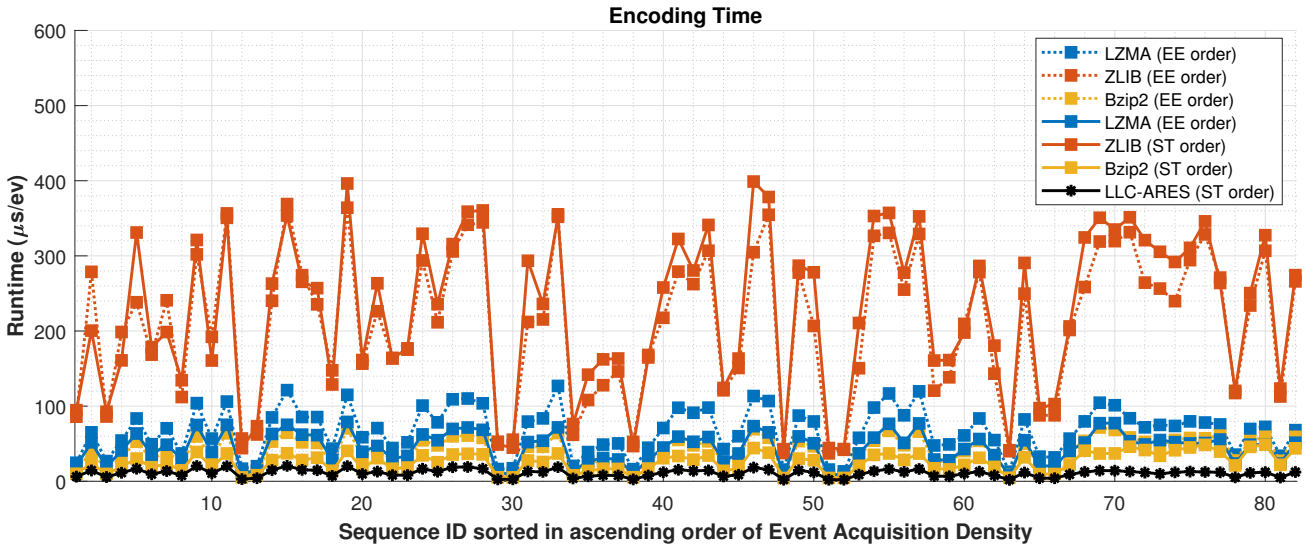


Figure 11. Encoding runtime results over the DSEC dataset [34], where the asynchronous event sequences are sorted in ascending order of the sequence acquisition density.

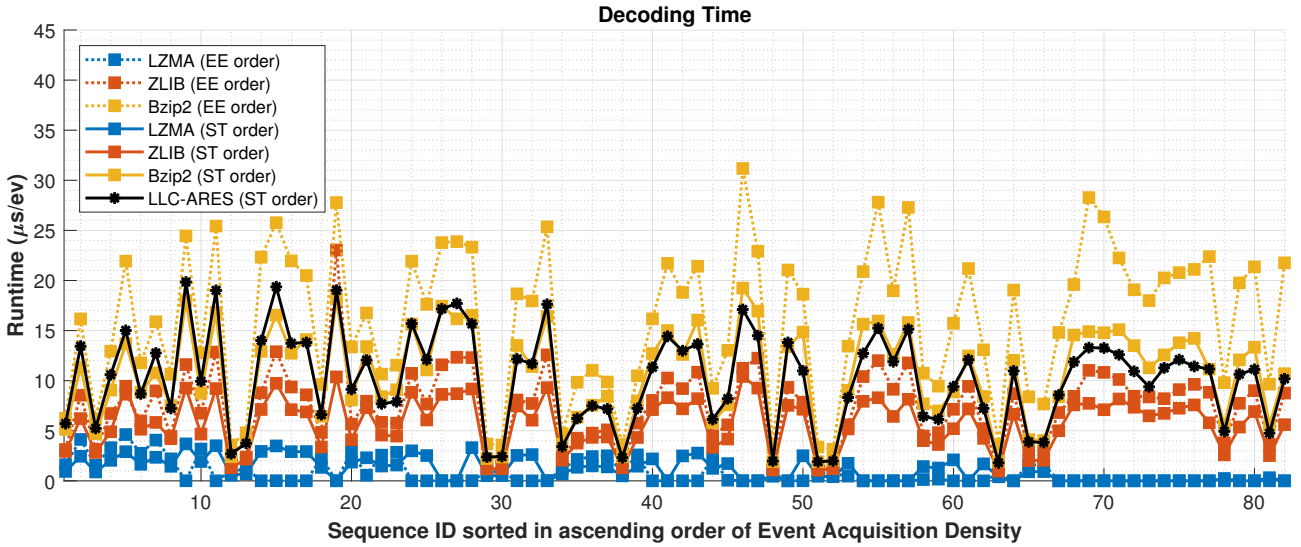


Figure 12. Decoding runtime results over the DSEC dataset [34], where the asynchronous event sequences are sorted in ascending order of the sequence acquisition density.

Table 2. Average runtime results over DSEC using EE and ST order.

Method		**ZLIB** [35]	**LZMA** [22]	**bzip2** [36]	**Proposed LLC-ARES**
Encoding Runtime	EE order	67.20 µs/ev	210.39 µs/ev	40.91 µs/ev	–
	ST order	44.70 µs/ev	227.27 µs/ev	25.75 µs/ev	**10.92 µs/ev**
Decoding Runtime	EE order	**0.78 µs/ev**	7.46 µs/ev	16.09 µs/ev	–
	ST order	1.14 µs/ev	5.71 µs/ev	10.58 µs/ev	10.21 µs/ev

4.4. RA Results

Figure 13 shows the RC results over DSEC. One can note that the RC results are quite similar, as the size of the header bitstream is neglectable compared with the time-window sequence bitstream. When providing RA to the smallest tested time window of

$\Delta_{RA} = 100\,\mu s$, compared with LLC-ARES, the coding performance of the proposed LLC-ARES-RA method decreases with less than 0.19% when the encoded header information is stored in memory and less than 0.35% when the decoded header information is stored in memory, denoted here as memory usage (MU) results.

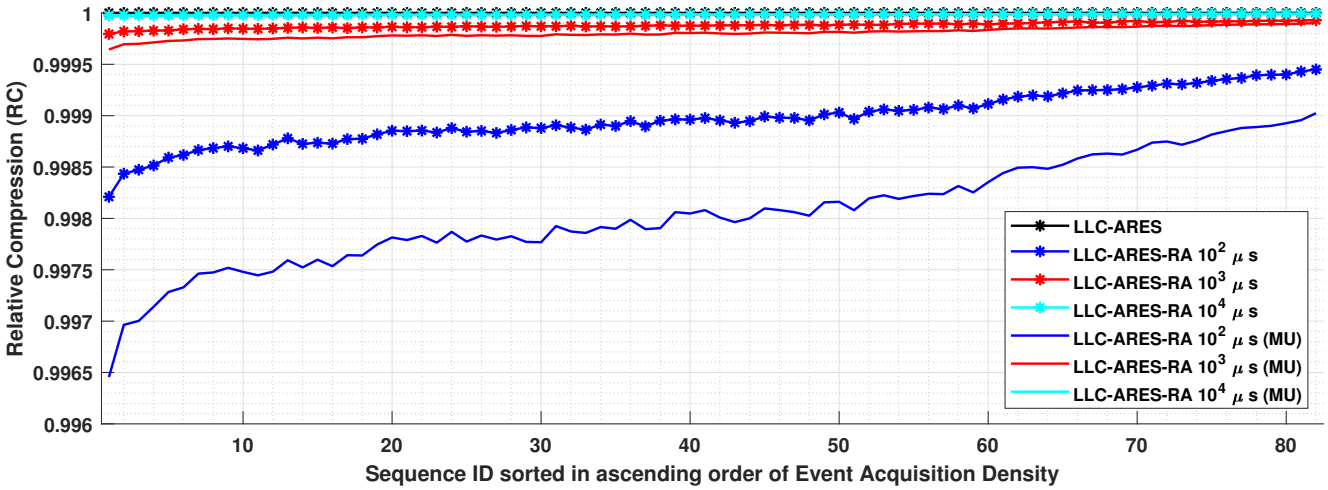


Figure 13. The relative compression (RC) results for RA results over the DSEC dataset [34], wherein the asynchronous event sequences are sorted in ascending order of the sequence acquisition density.

5. Conclusions

In this paper, we proposed a novel lossless compression method for encoding the event data acquired by the new event sensor and represented as an asynchronous event sequence. The proposed LLC-ARES method is built based on a novel low-complexity coding technique so that it is suitable for hardware implementation into low-power ESP chips. The proposed low-complexity coding scheme, TTP, creates short-depth decision trees to reduce either the binary representation of the residual error computed based on a simple prediction, or the binary representation of the true value. The proposed event representation employs the novel ST order, whereby same-timestamp events are first grouped into same-timestamp subsequences, and then reordered to improve the coding performance. The proposed LLC-ARES-RA method provides RA to any time window by employing a header structure to store the length of the bitstream packages.

The experimental results demonstrate that the proposed LLC-ARES codec provides an improved coding performance and a closer to real-time runtime performance compared with state-of-the-art lossless data compression codecs. More exactly, compared with Bzip2 [36], LZMA [22], and ZLIB [35], respectively, the proposed method provides:

(1) an average CR improvement of 5.49%, 11.45%, and 35.57%;
(2) an average BR improvement of 7.37%, 13.40%, and 37.12%;
(3) an average bitsavings of 1.09 bpev, 1.99 bpev, and 5.50 bpev;
(4) an average event density improvement of 234×, 412×, and 2086×; and
(5) an average TR improvement of 216×, 401×, and 1969×.

To our knowledge, the paper proposes the first low-complexity lossless compression method for encoding asynchronous event sequences that is suitable for hardware implementation into low-power chips.

Author Contributions: Conceptualization, I.S.; methodology, I.S.; software, I.S.; validation, I.S.; formal analysis, I.S.; investigation, I.S.; writing—original draft preparation, I.S.; writing—review and editing, I.S. and R.C.B.; visualization, I.S.; supervision, R.C.B. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

DVS	Dynamic Vision Sensor
APS	Active Pixel Sensor
DAVIS	Dynamic and Active-pixel VIsion Sensor
EF	Event Frame
RA	Random Access
TALVEN	Time Aggregation-based Lossless Video Encoding for Neuromorphic sensor
ESP	Event Signal Processing
SoC	System-on-a-chip
EMI	Event Map Image
CPV	Concatenated Polarity Vector
HEVC	High Efficiency Video Coding
SNN	Spike Neural Network
EGC	Elias-Gamma-Coding
LLC-ARES	Low-Complexity Lossless AsynchRonous Event Sequences
LLC-ARES-RA	LLC-ARES with RA
ZLIB	Zeta Library
LZMA	Lempel–Ziv–Markov chain Algorithm
G-PCC	Geometry-based Point Cloud Compression
CR	Compression Ratio
BR	Bitrate
TR	Time Ratio

References

1. Lichtsteiner, P.; Posch, C.; Delbruck, T. A 128× 128 120 dB 15 µs Latency Asynchronous Temporal Contrast Vision Sensor. *IEEE J. Solid State Circ.* **2008**, *43*, 566–576. [CrossRef]
2. Brandli, C.; Berner, R.; Yang, M.; Liu, S.C.; Delbruck, T. A 240 × 180 130 dB 3 µs Latency Global Shutter Spatiotemporal Vision Sensor. *IEEE J. Solid State Circ.* **2014**, *49*, 2333–2341. [CrossRef]
3. Pan, L.; Scheerlinck, C.; Yu, X.; Hartley, R.; Liu, M.; Dai, Y. Bringing a Blurry Frame Alive at High Frame-Rate With an Event Camera. In Proceedings of the 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Long Beach, CA, USA, 15–20 June 2019; pp. 6813–6822. [CrossRef]
4. Gehrig, D.; Rebecq, H.; Gallego, G.; Scaramuzza, D. EKLT: Asynchronous Photometric Feature Tracking using Events and Frames. *Int. J. Comput. Vis.* **2020**, *128*, 601–618. [CrossRef]
5. Iaboni, C.; Lobo, D.; Choi, J.W.; Abichandani, P. Event-Based Motion Capture System for Online Multi-Quadrotor Localization and Tracking. *Sensors* **2022**, *22*, 3240. [CrossRef] [PubMed]
6. Zhu, A.; Yuan, L.; Chaney, K.; Daniilidis, K. EV-FlowNet: Self-Supervised Optical Flow Estimation for Event-based Cameras. In Proceedings of the Robotics: Science and Systems, Pittsburgh, PA, USA, 26–30 June 2018. [CrossRef]
7. Brandli, C.; Mantel, T.; Hutter, M.; Höpflinger, M.; Berner, R.; Siegwart, R.; Delbruck, T. Adaptive pulsed laser line extraction for terrain reconstruction using a dynamic vision sensor. *Front. Neurosci.* **2014**, *7*, 1–9. [CrossRef]
8. Li, S.; Feng, Y.; Li, Y.; Jiang, Y.; Zou, C.; Gao, Y. Event Stream Super-Resolution via Spatiotemporal Constraint Learning. In Proceedings of the 2021 IEEE/CVF International Conference on Computer Vision (ICCV), Montreal, QC, Canada, 11 October 2021; pp. 4460–4469. [CrossRef]

9. Yu, Z.; Zhang, Y.; Liu, D.; Zou, D.; Chen, X.; Liu, Y.; Ren, J. Training Weakly Supervised Video Frame Interpolation with Events. In Proceedings of the 2021 IEEE/CVF International Conference on Computer Vision (ICCV), Montreal, QC, Canada, 10–17 October 2021; pp. 14569–14578. [CrossRef]
10. Wang, Y.; Yang, J.; Peng, X.; Wu, P.; Gao, L.; Huang, K.; Chen, J.; Kneip, L. Visual Odometry with an Event Camera Using Continuous Ray Warping and Volumetric Contrast Maximization. *Sensors* **2022**, *22*, 5687. [CrossRef]
11. Gallego, G.; Delbrück, T.; Orchard, G.; Bartolozzi, C.; Taba, B.; Censi, A; Leutenegger, S.; Davison, A.; Conradt, J.; Daniilidis, K.; et al. Event-Based Vision: A Survey. *IEEE Trans. Pattern Anal. Mach. Intell.* **2022**, *44*, 154–180. [CrossRef]
12. Bi, Z.; Dong, S.; Tian, Y.; Huang, T. Spike Coding for Dynamic Vision Sensors. In Proceedings of the Data Compression Conf., Snowbird, UT, USA, 27–30 March 2018; pp. 117–126. [CrossRef]
13. Dong, S.; Bi, Z.; Tian, Y.; Huang, T. Spike Coding for Dynamic Vision Sensor in Intelligent Driving. *IEEE Internet Things J.* **2019**, *6*, 60–71. [CrossRef]
14. Khan, N.; Iqbal, K.; Martini, M.G. Lossless Compression of Data From Static and Mobile Dynamic Vision Sensors-Performance and Trade-Offs. *IEEE Access* **2020**, *8*, 103149–103163. [CrossRef]
15. Khan, N.; Iqbal, K.; Martini, M.G. Time-Aggregation-Based Lossless Video Encoding for Neuromorphic Vision Sensor Data. *IEEE Internet Things J.* **2021**, *8*, 596–609. [CrossRef]
16. Banerjee, S.; Wang, Z.W.; Chopp, H.H.; Cossairt, O.; Katsaggelos, A.K. Lossy Event Compression Based On Image-Derived Quad Trees And Poisson Disk Sampling. In Proceedings of the IEEE Int. Conf. Image Process., Imaging Without Borders, Anchorage, AK, USA, 19–22 September 2021; pp. 2154–2158. [CrossRef]
17. Schiopu, I.; Bilcu, R.C. Lossless Compression of Event Camera Frames. *IEEE Signal Process. Lett.* **2022**, *29*, 1779–1783. [CrossRef]
18. Schiopu, I.; Bilcu, R.C. Low-Complexity Lossless Coding for Memory-Efficient Representation of Event Camera Frames. *IEEE Sens. Lett.* **2022**, *6*, 1–4. [CrossRef]
19. Elias, P. Universal codeword sets and representations of the integers. *IEEE Trans. Inf. Theory* **1975**, *21*, 194–203. [CrossRef]
20. Ziv, J.; Lempel, A. A universal algorithm for sequential data compression. *IEEE Trans. Inf. Theory* **1977**, *23*, 337–343. [CrossRef]
21. Deutsch, P.; Gailly, J.L. Zlib Compressed Data Format Specification; Version 3.3; RFC: 1950; IETF. 1996. Available online: https://www.ietf.org/ (accessed on 19 July 2021).
22. Pavlov, I. LZMA SDK (Software Development Kit). Available online: https://www.7-zip.org/ (accessed on 19 July 2021).
23. Burrows, M.; Wheeler, D.J. *A Block-Sorting Lossless Data Compression Algorithm*; IEEE: Piscataway, NJ, USA, 1994.
24. Martini, M.G.; Adhuran, J.; Khan, N. Lossless Compression of Neuromorphic Vision Sensor Data based on Point Cloud Representation. *IEEE Access* **2022**, *10* 121352–121364. [CrossRef]
25. Henri Rebecq, T.H.; Scaramuzza, D. Real-time Visual-Inertial Odometry for Event Cameras using Keyframe-based Nonlinear Optimization. In Proceedings of the British Machine Vision Conference (BMVC), London, UK, 21–24 November 2017; Tae-Kyun, K., Stefanos Zafeiriou, G.B., Mikolajczyk, K., Eds.; BMVA Press: Durham, UK; pp. 16.1–16.12. [CrossRef]
26. Maqueda, A.I.; Loquercio, A.; Gallego, G.; Garcia, N.; Scaramuzza, D. Event-Based Vision Meets Deep Learning on Steering Prediction for Self-Driving Cars. In Proceedings of the 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–23 June 2018; IEEE: Piscataway, NJ, USA. [CrossRef]
27. Almatrafi, M.; Baldwin, R.; Aizawa, K.; Hirakawa, K. Distance Surface for Event-Based Optical Flow. *IEEE Trans. Pattern Anal. Mach. Intell.* **2020**, *42*, 1547–1556. [CrossRef]
28. Benosman, R.; Clercq, C.; Lagorce, X.; Ieng, S.H.; Bartolozzi, C. Event-Based Visual Flow. *IEEE Trans. Neural Netw. Learn. Syst.* **2014**, *25*, 407–417. [CrossRef]
29. Zhu, A.; Yuan, L.; Chaney, K.; Daniilidis, K. Unsupervised Event-Based Learning of Optical Flow, Depth, and Egomotion. In Proceedings of the 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Los Alamitos, CA, USA, 15–20 June 2019; IEEE Computer Society: Washington, DC, USA, 2019; pp. 989–997. [CrossRef]
30. Baldwin, R.; Liu, R.; Almatrafi, M.M.; Asari, V.K.; Hirakawa, K. Time-Ordered Recent Event (TORE) Volumes for Event Cameras. *IEEE Trans. Pattern Anal. Mach. Intell.* 2022, *Early Access*. [CrossRef]
31. Sullivan, G.J.; Ohm, J.R.; Han, W.J.; Wiegand, T. Overview of the High Efficiency Video Coding (HEVC) Standard. *IEEE Trans. Circ. Syst. Video Technol.* **2012**, *22*, 1649–1668. [CrossRef]
32. Zhu, L.; Dong, S.; Huang, T.; Tian, Y. Hybrid Coding of Spatiotemporal Spike Data for a Bio-Inspired Camera. *IEEE Trans. Circ. Syst. Video Technol.* **2021**, *31*, 2837–2851. [CrossRef]
33. Gehrig, M.; Aarents, W.; Gehrig, D.; Scaramuzza, D. DSEC: A Stereo Event Camera Dataset for Driving Scenarios. *IEEE Robot. Autom. Lett.* **2021**, *6*, 4947–4954. [CrossRef]
34. DSEC Dataset. Available online: https://dsec.ifi.uzh.ch/dsec-datasets/download/ (accessed on 1 October 2021).
35. Vollan, G. ZLIB Pre-Build DLL. Available online: http://www.winimage.com/zLibDll/ (accessed on 19 July 2021).
36. Seward, J. bzip2 Pre-Build Binaries. Available online: http://gnuwin32.sourceforge.net/packages/bzip2.htm (accessed on 19 July 2021).