Chair of Spacecraft Systems
TUM School of Engineering and Design
Technical University of Munich

EveViz

A Graphical User Interface visualizing event data

Yael Kaliner, Luca Illig, Pablo Broders, Helena Niethammer

Report for the

Engineering Project

at the TUM School of Engineering and Design of the Technical University of Munich.

Examiner:

Prof. Alessandro Golkar

Supervisor:

Prof. Alessandro Golkar
Dr. Sydney Dolan

Submitted:

Munich, 09.04.2026

We hereby declare that this report is entirely the result of our own work except where otherwise indicated.
We have only used the resources given in the list of references.

Munich, 09.04.2026 Yael Kaliner, Luca Illig, Pablo Broders, Helena Niethammer

Abstract

This report presents the development of EveViz, a graphical user interface for loading, processing, and visualizing event camera data. Event cameras produce asynchronous data streams, which are especially beneficial for in-space applications, but pose challenges for analysis and visualization. Existing solutions are limited in their ability to handle event-based data efficiently and flexibly, motivating the development of this visualization tool.

The objective of this work is to develop a user-friendly and extensible GUI for the efficient handling and analysis of event camera data. EveViz combines data loading, cleaning, visualization, and export capabilities into a single system, enabling users to process and explore event recordings in a structured and intuitive way. The resulting tool enhances the practical usability of event-based data and provides a flexible framework for further research and application development.

Contents

Abstract iii

List of Figures v

List of Tables v

1 Introduction 1
1.1 Problem Description . . . 1
1.2 Objectives . . . 1
1.3 Structure of this Report . . . 2

2 EveViz Tool 3
2.1 Loading . . . 3
2.1.1 Loading RAW . . . 4
2.1.2 Loading CSV . . . 4
2.1.3 Loading TXT . . . 5
2.1.4 Loading HDF5 / H5 . . . 5
2.1.5 Loading BIN . . . 5
2.2 Cleaning . . . 5
2.2.1 Hot Pixel Filter . . . 6
2.2.2 Voxel-Based Filtering Method . . . 7
2.2.3 Other Filters . . . 8
2.3 Visualization . . . 10
2.3.1 Overview . . . 10
2.3.2 Frame Visualization . . . 11
2.3.3 3D Visualization . . . 13
2.3.4 Voxel Visualization . . . 13
2.3.5 Compare Raw vs Clean Data . . . 14
2.4 Additional Features . . . 14
2.4.1 Exporting . . . 14
2.4.2 Statistics . . . 15
2.4.3 Safety Features . . . 15

3 EveViz Tool Validation and Discussion 16
3.1 Workflow . . . 16
3.2 Assessment . . . 21

4 Conclusions 22
4.1 Limitations . . . 22
4.2 Future Work . . . 22
4.3 Final Conclusions . . . 23

Bibliography 24

List of Figures

2.1 Structure Overview . . . 3

3.1 Workflow State Machine . . . 16
3.2 Loading Data Window . . . 17
3.3 Event Visualization Window . . . 17
3.4 Tune Parameters Window . . . 18
3.5 Frame Visualization Window . . . 18
3.6 Lognormal Frame Visualization Window . . . 19
3.7 Lognormal Frame Video Window . . . 19
3.8 Lognormal 3D Visualization Window . . . 20
3.9 Voxel Visualization Window . . . 20

List of Tables

2.1 Supported Data types in the GUI . . . 4
2.2 Filters in the GUI . . . 6
2.3 Polarities in Visualization . . . 11
2.4 Overview of 2D visualization representations . . . 11
2.5 Overview of Export options . . . 14

3.1 Dataset Overview . . . 21
3.2 Results on Tested Dataset . . . 21

1 Introduction

Event cameras are a type of camera that detects events based on pixel-level changes in brightness. When the brightness change exceeds a threshold, the camera records an event. This event is marked with an $x-$ and a $y-$ value indicating the location, a timestamp t, and the polarity p. Polarity is typically encoded as a 1-bit value: brightness increase is 1 (positive), and brightness decrease is 0 (negative).

Compared to frame-based cameras, which capture images by taking a series of frames of the entire scene at specified time intervals, event cameras update pixels asynchronously. This means that each pixel reports events rather than a single frame being taken simultaneously across the full camera. The asynchronous design of event cameras offers additional benefits, including high temporal resolution, low latency, low power consumption, and high dynamic range [3, pp. 3–4]. In space applications, event cameras' primary advantages over conventional cameras are their ability to detect fast-moving objects and handle extreme lighting conditions, both in the light and in the dark.

The EventSat 6U CubeSat mission is a project of the Chair of Spacecraft Systems at the Technical University of Munich (TUM), in which an event camera will be integrated into a 6U CubeSat (1U = 10x10x10 cm) and sent to Low Earth Orbit (LEO) to detect stars and other space objects [2]. After the data is recorded, it will be downlinked to a database on Earth. To aid in mission data postprocessing, this project has focused on data visualization.

1.1 Problem Description

Because the structure of event data differs from that of standard camera data, traditional visualization tools are insufficient. Event data is a continuous stream of data rather than a series of fixed frames. Thus, event data visualizations must account for differences in data representation and timing. Event data is asynchronous, requiring the tool to aggregate events over time into meaningful representations.

While some event visualization tools already exist, current solutions are not specifically designed for space applications. This highlights the need for better-fitting visualization approaches.

1.2 Objectives

The overall goal of the project is to build a graphical user interface (GUI) that visualizes event data. This associated code for the GUI should be a user-friendly repository of functions for representing various event data, along with the documentation required to use and understand them. This includes annotated code, ReadMe files, and a final report. The system should allow the user to load event data from various file formats, clean the data using filters, and provide visualization options, including 2D, 3D, and video representation. After reviewing the data and applying any filters, the user should be able to export the data in the desired file type and receive a report regarding metadata and filtering statistics. In addition, the system should be easy to understand and extensible, allowing users to modify existing functionality and integrate their own filters and visualization methods. A modular structure should ensure that the system can be extended without requiring major changes to the existing codebase.

With the above objectives of the engineering project defined, we summarize the contributions of this engineering project as follows:

- A graphical user interface was developed
- A well-commented code repository was provided
- Extensive ReadMe files were created to explain the structure
- The GUI was rigorously tested to verify its integrity

1.3 Structure of this Report

The remainder of this document is organized as follows. Chapter 2 explains the EveViz Tool and is organized into several subsections. It first explains how and what data is loaded in 2.1. Section 2.2 explains the different filters implemented to clean Event data. Section 2.3 refers to the different ways event data can be visualized. Additional features offered by EveViz are discussed in 2.4.

Chapter 3, EveViz Tool Validation and Discussion, gives an overview of the workflow and insider observations of the system development phase.

Chapter 4 concludes the report. Section 4.1 explains the current limitations of EveViz and provides an outlook on how future work can be implemented in the tool (Section 4.2).

2 EveViz Tool

The fundamental approach for developing the EveViz tool was to stay within the basic capabilities of the Python suite. Therefore, the graphical user interface of the tool is built upon the standard tkinter library, and all visualizations take advantage of the core functionalities of the matplotlib library. There is strict separation between the executable EveViz GUI and the helper functions providing the core functionality. These structural decisions and dependencies are highlighted in Figure 2.1. The helper functions are sorted by topic in external files. Those Python files each have one class that groups all functions, allowing for easy import into the EveViz GUI file.

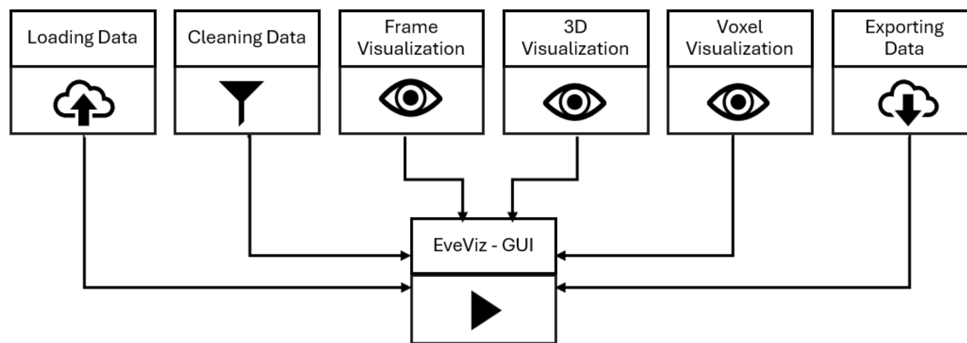

Figure 2.1 Structure Overview

From a data-handling standpoint, the EveViz tool employs NumPy arrays as a unified data storage structure for the event camera data. The class `AppState` stores all global variables and statistics that need to be accessible from different files and therefore lays the foundation for a shared dataset. This class is shared across the full workflow of the GUI, allowing the user to load, clean, visualize, and export event data without switching tools and the need to re-upload initial files.

2.1 Loading

The Loading Data class provides the core functionality of importing data from different data types and storing them in a shared format: the NumPy array. This unified data storage allows all GUI helper functions to focus on array input rather than needing to adapt to varying input formats.

As shown in Table 2.1, the loading capability encompasses a selection of data types typically encountered in the context of event cameras.

Data type	Purpose
RAW	Native event stream format from Prophesee cameras
CSV	Human-readable tabular storage for simple data exchange
TXT	Flexible plain-text storage with minimal structure
HDF5 / H5	Efficient storage and compression for large-scale datasets
BIN	Compact binary compression storage from the Spacecraft System Chair [4]

Table 2.1 Supported Data types in the GUI

Those data types have a significant impact on file sizes and, therefore, strongly differ in usefulness and applicability from use case to use case. An ordered comparison of the discussed file formats on data size after compression is visualized below:

Most compressed $\rightarrow$ BIN $\rightarrow$ HDF5 / H5 $\rightarrow$ RAW $\rightarrow$ CSV $\rightarrow$ TXT $\rightarrow$ Least compressed

In the following sections, the supported data types and extraction of the x, y, t, and p arrays will be discussed in detail.

2.1.1 Loading RAW

The RAW data type is the output format of Prophesee cameras. This file format can be divided into the header part that is written in ASCII, containing metadata associated with the RAW file, and the event data, which is written in binary code [5].

Raw files can be decoded in Python using a custom library accessible only to users of the Metavison SDK or OpenEB software [7]. The RAW data is extracted in event chunks, which are then collected in a list before being concatenated into a structured array. Those arrays are then reordered into four individual x, y, t, and p arrays that are returned by the function. As this file format directly stems from the Prophesee camera, there is no need to double-check those received arrays for their content.

2.1.2 Loading CSV

The CSV data type stores event data in a simple, headerless table, with each event in its own row and columns corresponding to the different classifications. This representation offers human-readable storage but consequently results in a substantially larger file size in comparison to binary encoded formats.

The CSV loading function of the GUI automatically detects the CSV delimiter, differentiating between semicolon and colon to account for differences between German and English default delimiters. As the file layout and, therefore, the column order are not controlled, the extracted columns are checked to ensure correct classification of the x, y, t, and p arrays. First, the time and polarity arrays are found by checking the first 100 elements for respectively monotonic increasing values and only containing $[0, 1, 0.0, 1.0, \texttt{True}, \texttt{False}, -1]$ values. The two remaining arrays, $x-$ and $y-$, are then treated as spatial information and distinguished by checking for the larger maximum value. The maximum x-value of event data is always larger than the maximum y-value since the event camera frame is rectangular and the x coordinate spans the larger portion of the frame. Additionally, the time values are offset to start at zero, and after investigation of the size of the first time step, the time unit is assumed. If the first time step is very small (< 0.001), the values are presumed to be in seconds and converted to microseconds. The validated columns are then returned as NumPy arrays.

2.1.3 Loading TXT

The TXT data type stores the event data in a flexible plain-text format with minimal structure. The TXT format offers many different delimiter options and no clear, agreed-upon dataset structure, resulting in difficulties in finding and classifying the datasets consistently. Additionally, this rudimentary format has a tendency to have comparably long loading durations at higher event counts.

The TXT loading function of the GUI, therefore, only accepts the fixed standard format received from the BIN compression application developed by the Spacecraft Systems Chair [4]. These TXT files are comma-delimited and follow the fixed column order of x, y, p, t. The loader automatically checks whether the first row is a header and skips it if needed. Additionally, it offsets the time array to start at zero and returns the data as NumPy arrays in the new order of x, y, t, and p.

2.1.4 Loading HDF5 / H5

The HDF5 / H5 data type provides efficient storage and compression for large-scale datasets. Its data model utilizes groups and datasets to organize and store data. Every HDF5 / H5 file contains a root group, which can contain other groups, creating a folder structure. Datasets, on the other hand, can be found in said groups and contain the "raw" data values along with additional metadata attributes [8].

The HDF5 loading function in the GUI utilizes the h5py Python library, which is only available to users having the ECF-codec installed [6]. The loader recursively visits all datasets in the group structure, stores them as arrays, and filters them by size. Only arrays with the maximum number of events are considered for classification, since all other arrays with fewer elements are reduced datasets. As already discussed for the CSV loading (see 2.1.2), the file format of HDF5 is not controlled, and therefore the extracted columns are checked to ensure correct classification of the x, y, t and p arrays. First, the time and polarity arrays are found by checking the first 100 elements for a monotonic increase in time and for values in [0,1,0.0,1.0,`True`,`False`]. The two remaining arrays are then treated as spatial information and distinguished by comparing their maximum values. Additionally, the time values are offset to start at zero, and after investigation, the size of the first time step and the time unit are assumed. The validated columns are then returned as NumPy arrays.

2.1.5 Loading BIN

The BIN data type offers a binary compression that drastically reduces the file size. The encoding and decoding algorithm was written by Antonio Junco de Haas as part of a master's thesis at the Spacecraft Systems Chair of TUM [4]. It supports decompressing BIN files to TXT files and can be accessed by running a command prompt after successful installation.

The BIN loading function of the GUI consequently decompresses loaded BIN files to TXT format. Since the decoding algorithm is accessed via an executable named "Main.exe", the user needs to confirm that the BIN converter is correctly installed and that the executable is added to PATH. The loader checks the availability of said "Main.exe" on PATH and then executes the sub-process [Main.exe d <input.bin> <output.txt>]. Once decompressed, the TXT loader is called, and the event data is returned as NumPy arrays.

2.2 Cleaning

The Cleaning Data class provides multiple filtering methods to clean the imported data. Each filter has a different purpose. For example, to remove hot pixels and isolated events or to sparsify high-density

areas, while still preserving meaningful information. Each filter takes the x, y, t, and p arrays as inputs and outputs the cleaned arrays.

Table 2.2 summarizes the available filters and their purposes in the GUI.

Name	Purpose
Hot Pixel Filter	Removes pixels firing excessively
Voxel Activity Filter	Removes isolated events in space and time
Neighbor Activity Filter	Keeps events with nearby neighbors
Subsampling Filter	Probabilistic density-based denoising
Voxel Density Filter	Reduces high-density redundancy
Median Filter	Removes events using a median threshold

Table 2.2 Filters in the GUI

These filters can be used individually or combined in a pipeline. If multiple filters are selected, they are applied in the following fixed order:

1. Hot Pixel Filter
2. Voxel Activity Filter
3. Voxel Density Filter
4. Median Filter

The filters are applied in this order because each subsequent filter generally removes fewer events, which progressively reduces the dataset size. This way, the following filters have to process fewer events, making them more efficient.

The output of each filter is passed as input to the next, and the user cannot select the order, but only change it in the code itself if desired. Additionally, the Neighbor Activity Filter and the Subsampling Filter were removed from the GUI because they were too slow to use effectively, though they may be implemented again manually in the code.

2.2.1 Hot Pixel Filter

The purpose of the Hot Pixel Filter is to detect and remove hot pixels, i.e., pixels that generate an abnormally high number of events consistently over a large span of time. These hot pixels are caused by sensor aging, radiation damage, or thermal noise [2].

To remove these hot pixels, the filter first builds a rate map. This rate map simply counts the number of events fired from each pixel. By taking the 99.999^{th} percentile of these event counts, hot-pixel candidates are identified. The percentile value may be changed, but 99.999 is recommended. Since high temporal permanence is also a key characteristic of hot pixels, and in order to further reduce the hot pixel candidates, a coefficient of variation is computed to measure a pixel's relative variability over time by dividing the standard deviation by the mean, as can be seen in equation 2.1. The hot-pixel candidates are further reduced by applying a coefficient-of-variation threshold. If any hot-pixel candidates remain, all events produced by these pixels are removed.

$$\mathrm{CV} = \frac{\sigma}{\mu} \tag{2.1}$$

The filter can be tuned using the percentile parameter. Increasing the percentile leads to fewer hot pixel candidates, making the filter less aggressive, while decreasing it produces more candidates and therefore a more stringent filter.

2.2.2 Voxel-Based Filtering Method

The following filters all use the same logic of dividing the entire event space into discrete volumes, called voxels, and subsequently counting the number of events per volume. This is done because, unlike a sliding window, it requires much less computing time.

When dividing the event space into voxels, each voxel is assigned a unique ID using bit shifting. Bit shifting means that the discretized x, y, and time coordinates are each allocated a fixed number of bits, shifted into separate regions of a 64-bit integer, and combined into a single uint64. This can be seen in equation 2.2.

$$voxel_{ID} = (t_q \ll 42) \mid (y_q \ll 21) \mid x_q \tag{2.2}$$

To count the number of events per voxel, each event is assigned the voxel ID of the voxel it belongs to. The IDs are then sorted so that identical values are adjacent, allowing the number of events in each voxel to be determined using run-length encoding, i.e., counting consecutive occurrences.

While this method is very beneficial for the cleaning process, due to the aforementioned speed increase, it has a significant drawback for visualization. If the spatial window is increased, thereby enlarging the voxels, they may become visible as squares in the visualization. This is especially notable in the lognormal visualization, where small density differences are more distinct.

This knowledge of how many events are in each voxel will be used in the following filters.

Voxel Activity Filter

The key idea behind both the Neighbor Activity and Voxel Activity Filter is that isolated events are likely noise. Therefore, both filters look at the spatial and temporal neighborhoods of each event to determine whether at least one other event exists. If this is not the case, the event is classified as noise and removed. The methods by which the two filters search the neighborhood, though, are very different.

The Voxel Activity Filter uses the Voxel-Based Filtering Method as described above. With knowledge of the number of events per voxel, every event that occurs alone in a voxel is filtered out as it is considered noise. This can be seen in equation 2.3.

$$retained = occupancy > 1 \tag{2.3}$$

To tune the filter, the spatial and temporal windows can be adjusted. The filter behaves, unlike most other filters, very linearly. Increasing the spatial and temporal window results in a lower percentage of filtered-out events, and vice versa. This is because with larger voxels, the probability of the event being isolated is smaller.

Voxel Density Filter

The purpose of a density filter is to probabilistically thin out dense regions to reduce data size for visualization, without dramatically affecting the output.

The Voxel Density Filter also uses the above-described Voxel-Based Filtering Method, but unlike the Voxel Activity Filter, it does not have a fixed threshold, but instead a variable one. This threshold, C_{max}, is determined by selecting a percentile of the event count per voxel. Events in the voxels exceeding the threshold are kept with a probability of

$$P_{\text{keep}} = \alpha \cdot \frac{C_{\max}}{C} \tag{2.4}$$

where C is the number of events per voxel and $alpha$ corresponds to the strength of the thinning. This results in voxels that substantially exceed the threshold being thinned further, and those that are only slightly above the threshold being thinned less. This is a beneficial feature because it reduces high-density clutter while preserving meaningful information.

Tuning the filter is less linear than the Voxel Activity Filter. Reducing the percentile threshold means a stricter definition of dense voxels, leading to fewer events retained. Moreover, reducing $alpha$ leads to a lower P_{keep} and, therefore, more aggressive filtering. The effect of reducing $alpha$ on the percentage of filtered events rapidly plateaus as $alpha$ approaches zero, since at some point all events in voxels deemed dense are filtered out. The point at which the effect of $alpha$ stagnates is dependent on the dataset.

Median Filter

The purpose of the Median Filter is very similar to that of the Activity Filter. It removes events from sparsely populated voxels by keeping only those above a threshold. While this threshold was fixed to 1 event in the Activity Filter, it is now median-based and adaptive.

Following the Voxel-Based Filtering Method, the median voxel occupancy is computed over the occupied voxels and used to define an adaptive threshold. This threshold is set to the maximum of 2 and the median occupancy, as seen in equation 2.5, ensuring that even in sparse datasets with a low median, fully isolated events are removed, as with the Activity Filter. The events occurring in voxels with occupancy below the threshold will be discarded according to equation 2.6.

$$threshold = max(2, median) \tag{2.5}$$

$$retained = occupancy \geq threshold \tag{2.6}$$

While this does sound very similar to the Voxel Activity Filter, a significant difference has to be highlighted. There are highly populated datasets in which clusters occur, but the Voxel Activity Filter ignores them. Making the threshold variable gives the filter an opportunity to filter out such events as well.

The tuning for the Median Filter is inherently non-linear due to several factors. Increasing the spatial and temporal window does lead to more events per voxel and a higher overall median, but it also causes voxel occupancies to fall below the threshold. Conversely, when using small temporal and spatial windows, the median might fall below 2, at which point the maximum function takes effect. While this is a necessary safety function, it increases the function's nonlinearity.

2.2.3 Other Filters

As mentioned above, there are filters in the code that are not implemented in the GUI at this point. The reason for this is that they use too much computational power to be used effectively, though they may be implemented manually in the code again. Moreover, their purpose partially doubles with the implemented filters, as they were made as a first iteration. Even though they are flawed, the ideas behind them are still valuable, and they will be presented in the following section.

Neighbor Activity Filter

As mentioned before, the Neighbor and Voxel Activity Filter produce very similar results but differ in their underlying mechanisms.

As the key idea of an activity filter is to remove isolated events, the Neighbor Activity Filter uses a KDTree, a space-partitioning data structure that is especially helpful for multidimensional searches. The filter searches within a sphere around each event for another event. If no event, except for the event itself, is found in the vicinity, the event is removed.

The logic is identical to that of the Voxel Activity Filter, as it is still an activity filter. But the difference in their execution does affect the result. The two methods may remove different events, since one uses fixed regions while the other uses local proximity.

The Neighbor Activity Filter was removed from the GUI because it is significantly slower: it performs a neighbor search for every event, whereas the Voxel method relies on faster grouping operations.

The tuning of the Neighbor Activity Filter is very similar to that of the Voxel Activity Filter. Again, the spatial and temporal windows can be adjusted, and increasing either will lead to a lower percentage of filtered-out events, and vice versa.

Subsampling Filter

The Subsampling Filter is based on a paper [1] that introduces a density-based subsampling method. The goal of this subsampling method is to reduce the number of events while still preserving meaningful information. According to the paper, spatio-temporally dense regions are more informative and should therefore be retained more. This method was converted into a usable filter using Numba JIT for increased speed.

Numba JIT compiles the code into instructions the CPU can execute immediately, bypassing the Python interpreter. This is especially helpful with nested loops, which are used here.

The Subsampling Filter uses a combination of an exponential decay function for the temporal component and a Gaussian weight for the spatial component. For the temporal decay, as shown in equation 2.7, at every pixel, the old activity is multiplied by an exponential decay function, which depends on how much time has passed between two events. The goal of this decay function is to achieve a smooth decay in which recent events remain important while older ones fade.

$$A_{\text{new}} = A_{\text{old}} \cdot \exp\left(-\frac{t_{\text{current}} - t_{\text{last}}}{\tau}\right) \tag{2.7}$$

Here τ denotes the temporal decay constant, which controls how quickly past activity diminishes, A_{old} and A_{new} are the past and current activity at that pixel respectively, t_{current} denotes the time at which the current event occurred and t_{last} stands for the last time an event occurred at that pixel.

The spatial component is calculated using equation 2.8, which is a Gaussian weighting function. This is done to emphasize nearby events over those that occurred farther away.

$$G(x, y) = \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right) \tag{2.8}$$

The standard deviation of the Gaussian is defined as

$$\sigma = \frac{\text{KernelWidth}}{5} \tag{2.9}$$

This is simply a rule of thumb, such that most of the Gaussian function fits within the kernel, allowing a smooth decay towards the kernel boundaries.

The temporal and spatial components are combined into a single density value f_v, defined as

$$f_v = \sum A(x', y') \cdot G(x' - x_i,\, y' - y_i) \tag{2.10}$$

The density value f_v is subsequently multiplied by a sampling threshold, as can be seen in equation 2.11. This probability is capped at 0 and 1, in order to then compare it with a randomly drawn number $r \in [0, 1]$ in equation 2.12.

$$p = \min\left(1,\ \max\left(0,\ \text{SamplingThreshold} \cdot f_v\right)\right) \tag{2.11}$$

After drawing a random number for each event, the event is retained if

$$\text{keep} = (r < \text{prob}), \quad r \in [0, 1] \tag{2.12}$$

This stochastic filtering method ensures that events located in regions of high spatio-temporal density are more likely to be preserved, while isolated events are more likely to be discarded.

Regarding filter tuning, several parameters can be adjusted. First, the sampling threshold directly controls the aggressiveness of the filtering, where increasing it results in more events being kept, while decreasing it leads to stronger filtering. By decreasing the τ of the exponential decay function, you get a shorter memory and therefore emphasize recent events. Finally, the filter size determines the spatial neighborhood considered during density estimation, where larger kernels lead to a smoother and more spatially distributed density estimate.

It should be noted that the filter is very aggressive and may remove meaningful data if the parameters are not chosen carefully. Therefore, appropriate tuning is especially necessary to balance data reduction and information preservation.

The filters introduced in this section collectively aim to reduce noise and redundancy in the event data, thereby improving both visualization quality and the performance of downstream applications.

However, this process inherently involves a trade-off between cleaner data and the potential loss of relevant information. Consequently, careful parameter tuning is essential. Furthermore, no single filter is universally optimal. The choice of filtering strategy and its parameters depends strongly on the dataset's characteristics and the desired outcome.

To support further extensions of this project, the GUI was designed to easily integrate additional filters.

2.3 Visualization

2.3.1 Overview

The Visualization module consists of three distinct classes: Frame Visualization, 3D Visualization, and Voxel Visualization. Each class accepts input arrays corresponding to the variables $x, y, t,$ and p, and shows the associated events using different visualization approaches specific to the respective class.

Despite their different representations, the visualization classes share a common structural design. In all three classes, events are displayed using scatter plots. The horizontal axis represents the spatial coordinate x, ranging from 0 to the maximum x-value, while the vertical axis represents the spatial coordinate y, with the maximum y-value mapped to the bottom-left corner of the plot. For the 3D and Voxel Visualization classes, a third dimension is added: the time axis with their respective t values. All visualizations, except the video function, support zooming and panning.

All classes (except the lognormal visualization) follow a consistent syntax for representing event polarity, with different colors used to distinguish between polarity types. The mapping between polarity values and their corresponding color is summarized in Table 2.3.

Polarity	Color
positive (1)	red
negative (0)	blue
total (no distinction between polarities)	grey
both (both polarities 0 and 1 depicted)	red and/or blue

Table 2.3 Polarities in Visualization

For all three classes, the user has the option to select different visual representations in the GUI depending on the polarity. The user can choose between Positive Events, Negative Events, Total Events, or Both Events.

2.3.2 Frame Visualization

The Frame Visualization class shows the events in different two-dimensional representations. An overview of these representations is shown in Table 2.4.

Representation	Description
Linear Visual	two-dimensional static representation in scatter plot
Lognormal Visual	two-dimensional representation with lognormal scale (accumulated visualization)
Frame Video	two-dimensional moving representation in scatter plot

Table 2.4 Overview of 2D visualization representations

The core functionality is based on scatter plots, where each event is visualized as a point in the x-y plane. Temporal filtering is applied using a time window defined by a start time t_{start} and a duration t_{duration} for every function inside the Frame Visualization class (except the Frame Video).

Events are selected according to:

$$t_{\text{start}} \leq t \leq t_{\text{start}} + t_{\text{duration}}$$

Linear Visual

The events in the selected time frame are displayed in a two-dimensional scatter plot, with the x and y values of the event data. According to the selected visualization, additional filtering based on the polarity is performed. The user can choose between representations of positive, negative, total, or both events, shown as points in the respective colors (see Table 2.3).

Lognormal Visual

The lognormal visualization generates a two-dimensional accumulated representation of event data within a user-defined time window. As with all frame-based visualization methods, only events occurring within the specified time window are considered. Due to the logarithmic scaling of the event data, a comparatively larger time interval has to be selected ($\sim$10 s) to be able to display a sufficient number of events for a meaningful representation.

In contrast to the linear visualization method, this approach uses a normalized logarithmic representation of the accumulated event counts rather than a linear one. First, to achieve this representation, each pixel value x is transformed by a logarithmic mapping as described in Equation 2.13. This transformation significantly reduces large differences.

The transformed values are then normalized to the range $[0, 1]$ between a minimum value $v_{\min}$ and a maximum value $v_{\max}$, as described in Equation 2.14. To ensure $\log(0)$ is undefined and to maintain the background at the lowest intensity, $v_{\min}$ is fixed to $v_{\min} = 1$. The upper bound $v_{\max}$ is chosen at a high percentile (here, 99.2^{th} percentile) of the non-background values. Finally, the normalized intensity values I_{log} are mapped to a colormap in which high values are displayed in bright colors and low values in darker colors.

This representation results in an overall more balanced representation of the event data: while small differences among low-count pixels are increased, large differences among high-count pixels are decreased. As a result, both sparse and dense regions remain visible within the frame.

$$x' = \log(x) \tag{2.13}$$

$$I_{\text{log}} = \frac{\log(x) - \log(v_{\min})}{\log(v_{\max}) - \log(v_{\min})} \tag{2.14}$$

The user can choose between positive, negative, or both event data, which are displayed in the two-dimensional frame.

Frame Video

The Frame Video representation provides a dynamic, two-dimensional visualization of event data in the x-y spatial domain. In contrast to the static representations described previously, this method converts asynchronous events into a continuous visual stream, allowing the temporal evolution of the scene to be observed.

The visualization is implemented as a clock-driven animation that follows real elapsed (wall-clock) time. Instead of iterating through discrete frames, the current visualization time t_{current} is determined using the system clock. This ensures accurate temporal behavior and prevents drift between the displayed animation and the original event timestamps.

At each update step, the set of visible events is selected based on the current time and the persistence interval, which retains events from a short time period preceding the current time. This results in a moving visualization in which recent events remain visible for a limited time.

If no persistence duration is manually specified in the code, it is automatically set to the duration of one frame based on the selected frame rate:

$$t_{\text{persistence}} = \frac{10^6}{\text{fps}}.$$

The visualization is based on a single, continuously updated scatter plot. This approach enables efficient rendering by displaying only the currently visible events. The polarity of events is encoded using colors, following the mapping defined in Table 2.3. The user can select between positive, negative, both, or total event representations.

To ensure consistent real-time playback, the animation compensates for rendering delays. If the rendering time exceeds the available frame interval, intermediate frames are skipped automatically. This guarantees that the visualization remains synchronized with real time and preserves the temporal structure of the data.

In addition to the visualization itself, the Frame Video includes interactive controls within the GUI. The user can pause and resume playback while maintaining the current position. Furthermore, a progress bar enables direct navigation through the recording by allowing the user to jump to any chosen time position.

2.3.3 3D Visualization

The class 3D Visualization represents events in a three-dimensional space, with an additional temporal axis in addition to the spatial x and y axes. The time axis is defined so that the starting time is at the origin and scales according to the selected time duration. While the x and y axes are scaled in true geometric proportions, the time axis is scaled relative to the maximum spatial length in the x-direction:

$$s_t = 0.9 \cdot x_{\max}$$

where s_t denotes the scaling factor applied to the time axis and $x_{\max}$ represents the maximum value of the spatial coordinate x. This proportional scaling prevents over-expansion of the temporal dimension and improves the comparison of different visualizations. This visualization class can also be configured to display positive, negative, total events, or both events.

2.3.4 Voxel Visualization

The voxel visualization class is a specialized form of the 3D visualization that represents event data in a discretized spatio-temporal grid. In contrast to the 3D Visualization class, the Voxel Visualization discretizes the time dimension into time bins.

As with all visualization methods, events are first filtered with the user-defined time window. The filtered events are then accumulated into a voxel grid of size $(2, B, H, W)$, where H and W are the spatial height and width, and B is the number of time bins. The first parameter represents the two event polarities: positive and negative. While the spatial resolution and polarity depend on the event data, the user can change the number of time bins.

To construct the voxel representation, the continuous timestamps are normalized to the discrete time domain:

$$\tau_i = \frac{t_i - t_{\text{start}}}{t_{\text{duration}}} \cdot (B - 1). \tag{2.15}$$

This normalization maps the original timestamps to a discrete range between 0 and $B - 1$, corresponding to the temporal bins of the voxel grid. Depending on the configuration, two strategies can be used for binning: The nearest-neighbor binning, where each event is assigned to the closest temporal bin, and linear interpolation, where each event contributes to the two adjacent bins using weights proportional to its fractional position:

$$w_0 = 1 - (\tau_i - \lfloor \tau_i \rfloor), \quad w_1 = \tau_i - \lfloor \tau_i \rfloor. \tag{2.16}$$

The latter approach ensures a smoother distribution of events across the temporal dimension, reduces discretization artifacts, and is therefore used as the default strategy.

For each event, the corresponding voxel entry is stated at position (c, b, y_i, x_i), where c denotes the polarity and b the temporal bin index. Repeated events at the same location accumulate.

For visualization, the voxel grid is converted into a set of 3D points by extracting all voxels whose event count exceeds a predefined threshold (default: 0). Each voxel above the threshold is mapped to a point with coordinates (x, t_b, y), where t_b represents the discrete temporal bin.

2.3.5 Compare Raw vs Clean Data

The compare functionality enables the user to explore the differences between the cleaned dataset and the original raw data. The feature is accessible from all visualization applications except for the frame video. Once opened, two matplotlib plots are displayed side by side, providing a direct way to analyze the events removed by the applied filters. The window supports zooming, panning, and exporting as an image.

2.4 Additional Features

2.4.1 Exporting

The export functionalities allow the user to extract relevant statistics and cleaned event data from the tool for reuse in the future. The visualization of event data can be saved as images and videos. An overview of the different export types the tool offers is given in Table 2.5.

Type	Description
Image	High-resolution exports at 200 DPI for publication quality,
Video	Preserves the exact timing and duration of event recordings
Report	Extracts overall metadata and dataset statistics
Clean Data	Maintains consistent standard file structure

Table 2.5 Overview of Export options

As the reusability of exported files heavily depends on a descriptive filename that stores the context in which the image was taken, or the data was exported, the tool automatically creates a default filename that includes the most important metadata.

The original filename of the input event camera recording is kept, and amended by the source, the applied filters, and the time information. This basic information about the context is combined, if applicable, to create a descriptive filename for each exporting type listed in Table 2.5.

Generated Filename Format:

```
{original_name}_{source}_{filters}_{time_range}.{extension}
```

Example:

```
Berlin15DegMin_120FPS_frame_cleaned_HP_ME_time_range_0ms_500ms.png
```

Image

The image export can be split into two types depending on the visualization context. An individual analysis plot can be saved in all basic visualization applications except for the frame video, where a screenshot of the current visualized frame can be saved with the corresponding timestamp. The images can be exported as PNG (default, highest quality), JPEG (compressed), and PDF (vector format).

Video

The video export preserves the exact original recording duration, and the build-up recording is an exact replica of the visualized frame video in the GUI. The videos can be exported as MP4.

Report

The report export offers a structured report file with dataset metadata and filter statistics from the currently loaded file. The report is formatted as plain text with information organized line by line for clarity, intended for documentation, reproducibility, and quick comparison between different filter configurations. The reports can be exported as OUT or TXT.

Clean Data

The clean data export allows the user to extract the filtered event arrays from the tool. The offered output formats follow the standard structure commonly encountered in research and engineering. The clean data can be exported as HDF5 (most efficient and reusable), BIN (heavily compressed), and TXT (compatible with generic tools).

2.4.2 Statistics

The EveViz tool offers a basic overview of the uploaded event data file. Spanning from the filename, over the recording length to the breakdown of the total amount of events, in positive and negative.

Furthermore, live statistics are offered during the cleaning process. After each filter that terminates, the number of removed events and their percentage of the total events are displayed. Once all selected filters have completed the cleaning process, the GUI outputs the total number of events removed, the percentage of the original events, and the number of remaining events to indicate the effectiveness of the applied filters.

2.4.3 Safety Features

EveViz includes built-in safeguards to prevent invalid operations and make the workflow more robust during interactive use. To achieve this, the GUI validates user input for missing or invalid values (for example, time range and filter parameters), has implemented `AppState` checks that block actions when required data is not loaded yet and additionally includes extensive warning and status feedback in the GUI windows to indicate what the user has done wrong and how to resolve the current issue. There are dependency checks for external components used by specific file formats or export targets, and basic reset and fallback behavior is implemented to return from filtered data to raw data when needed. Overall, the GUI safeguards aim to prevent tool crashes and provide a clear call to action to the end user if issues are detected.

Performance-related GUI slowdowns or crashes can occur when an excessively large time range is selected, resulting in more events being displayed than the tool can handle. As the chosen time input is a highly individual choice that varies by use case, setting a fixed limit was deemed not useful. Hence, the user is advised to keep the visualized data size in mind when choosing time ranges.

3 EveViz Tool Validation and Discussion

3.1 Workflow

The full functionality of the EveViz tool is accessed and controlled by the graphical user interface. It combines data loading, filter configuration and deployment, visualization and export action in a single workflow for event-based camera recordings. The application is organized into a set of windows (see Figure 3.1) that guide the user from loading a file to analyzing and exporting the modified data.

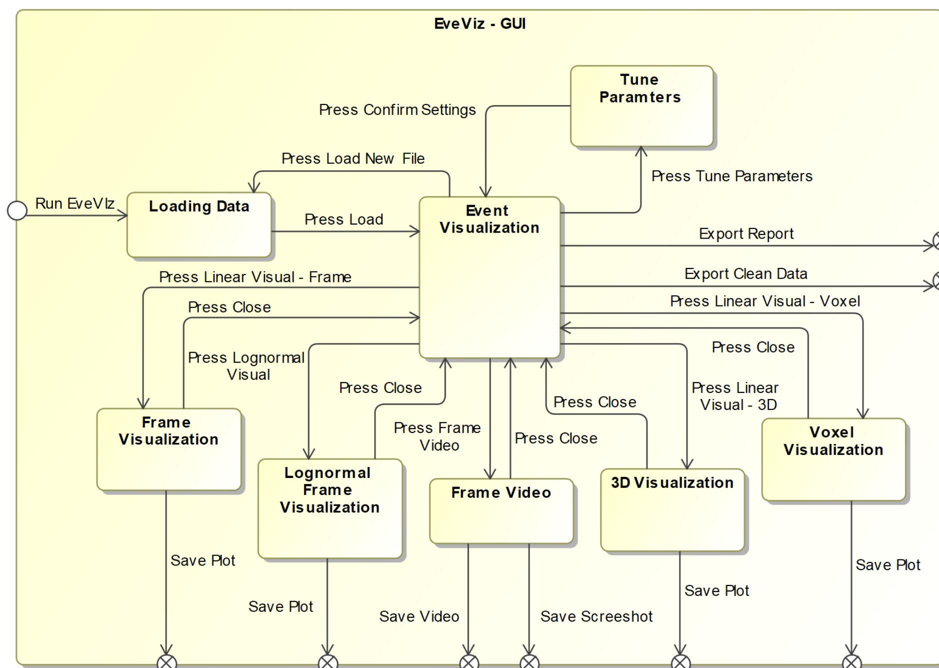

Figure 3.1 Workflow State Machine

The GUI is split into eight individual windows: Loading Data, Event Visualization, Tune Parameters, Frame Visualization, Lognormal Frame Visualization, Frame Video, 3D Visualization, and Voxel Visualization. Those windows are interconnected, as shown in Figure 3.1, allowing the user to navigate from window to window without restarting the tool. The Event Visualization window is the homepage of the GUI, offering several visualizations and additional features, and only terminating if a new file is loaded.

General Workflow

1. Load a supported event file in the Loading Data window.
2. Get an overview of the loaded data and apply cleaning filters in the Event Visualization window.
3. Open one of the visualization windows to inspect the filtered or unfiltered event data.
4. Export reports, plots, videos, or cleaned event data when needed.

Loading Data Window

The Loading Data window (see Figure 3.2) is the entry point of the tool. It allows file selection, detects the file type, and converts input file formats into a unified event storage data type used across EveViz.

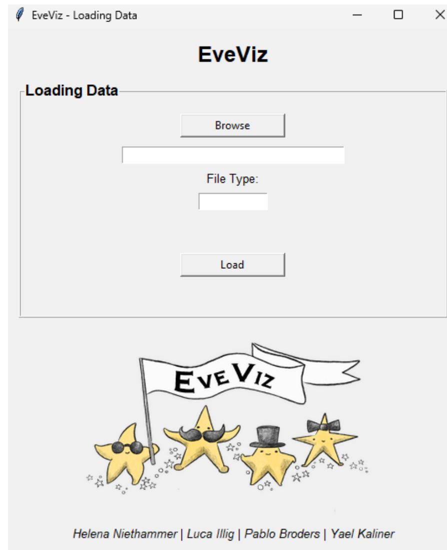

Figure 3.2 Loading Data Window

Event Visualization Window

The Event Visualization window (see Figure 3.3) is the main hub of the GUI after loading. It provides an overview of the loaded dataset and allows selection and execution of the filters. Additionally, live statistics of the cleaning process are displayed, and routing to visualization/export actions is provided.

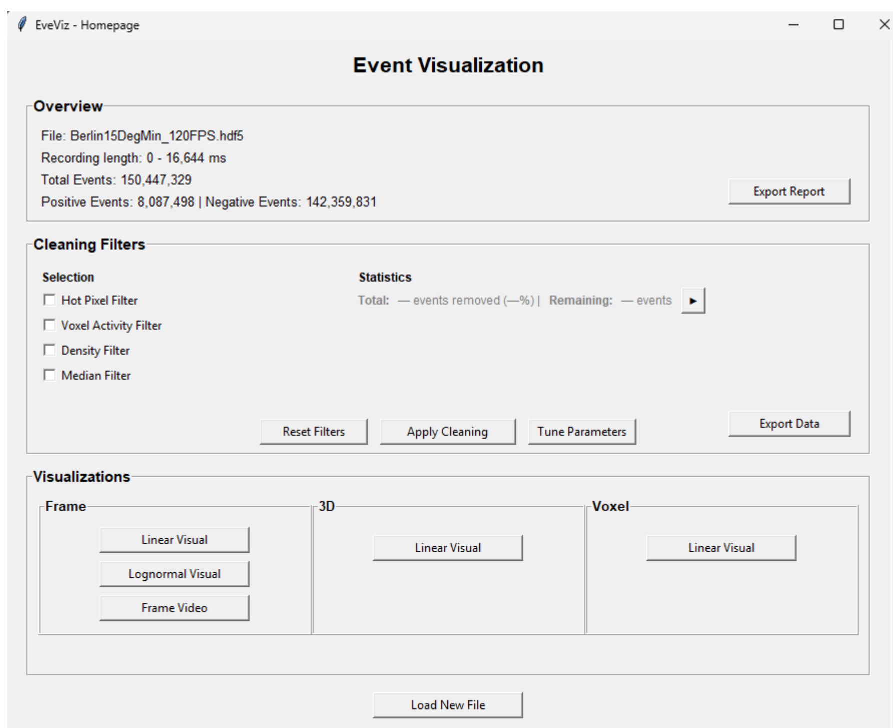

Figure 3.3 Event Visualization Window

Tune Parameters Window

The Tune Parameters window (see Figure 3.4) is used to adjust filter parameters before executing the cleaning process. The thresholds and spatiotemporal windows of the filters can be adjusted to strengthen or weaken the effectiveness and, therefore, the event retention and denoising strength. Furthermore, default values and recommended tuning ranges are displayed in the window.

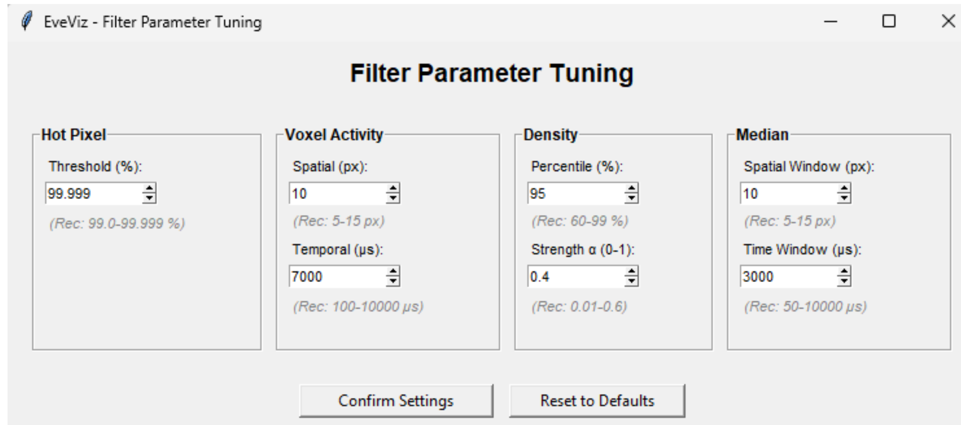

Figure 3.4 Tune Parameters Window

Frame Visualization Window

The Frame Visualization Window (see Figure 3.5) provides linear 2D plots and is intended for quick spatial analysis. It offers different event modes (positive, negative, total, both), time-range selection, comparison, and a plot export functionality.

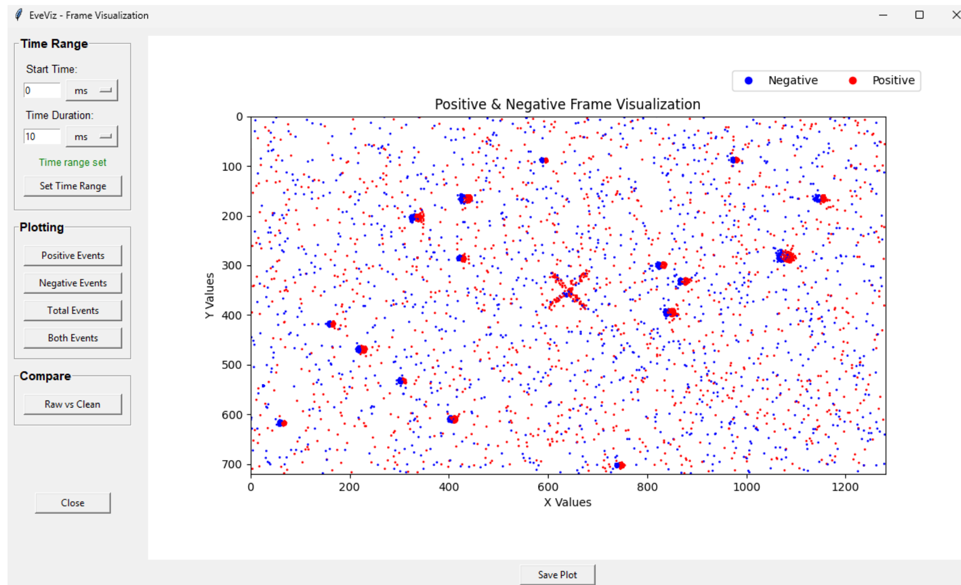

Figure 3.5 Frame Visualization Window

Lognormal Frame Visualization Window

The Lognormal Frame Visualization Window (see Figure 3.6) allows the user to inspect accumulated frame rendering with logarithmic scaling. Such visualizations are useful when event activity is highly non-uniform

and linear scatter views hide lower-density structures. The window also offers different event modes (positive, negative, both), time-range selection, comparison, and plot export functionality.

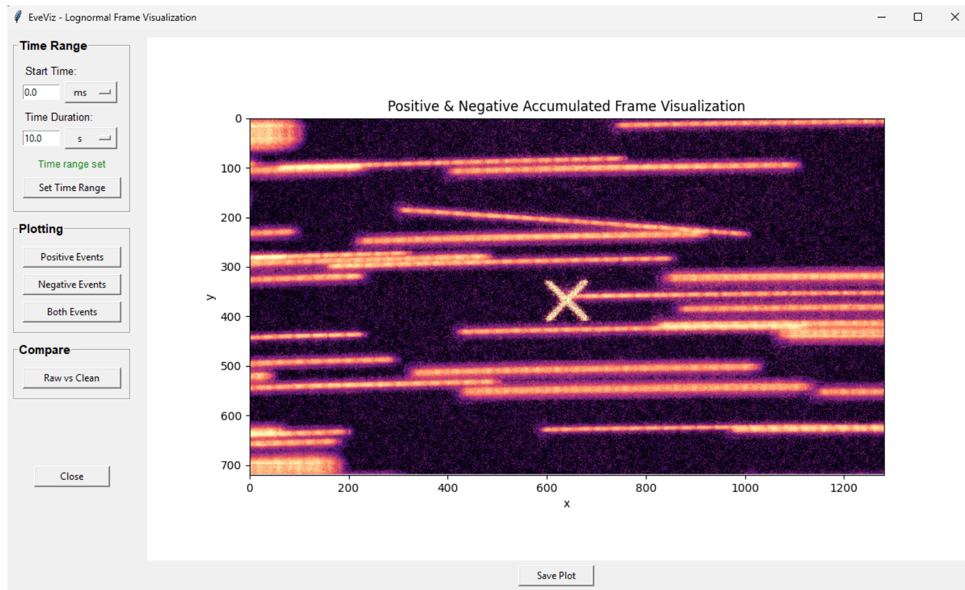

Figure 3.6 Lognormal Frame Visualization Window

Frame Video Window

The Frame Video Window (see Figure 3.7) provides an animated video of event data over time. It includes a progress bar, playback-like analysis, and several event modes (positive, negative, total, both). The animation can be exported as a screenshot or a video.

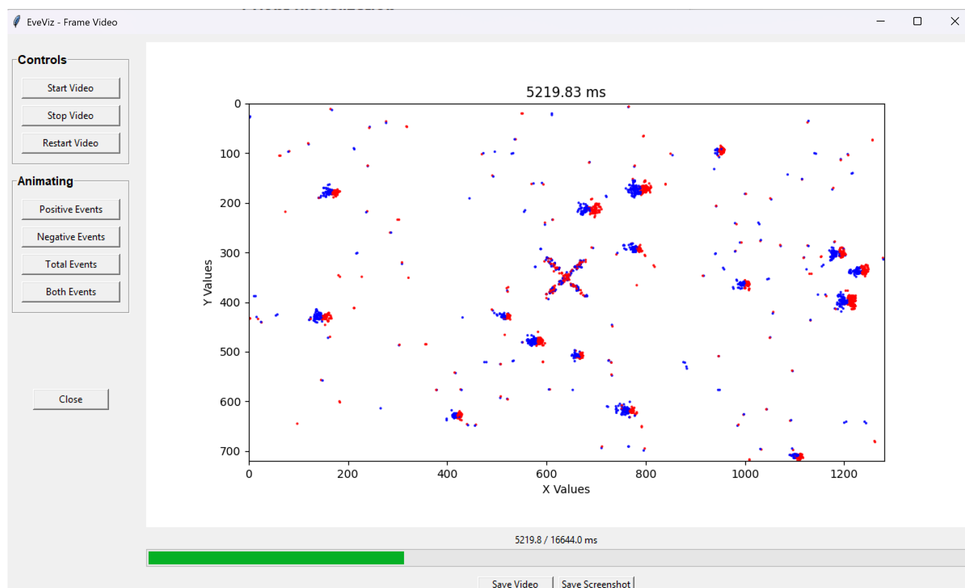

Figure 3.7 Lognormal Frame Video Window

3D Visualization Window

The 3D Visualization Window (see Figure 3.8) displays event data in spatial and temporal axes. This is intended for spatio-temporal inspection of the evolution of event data over time. It also offers different event modes (positive, negative, total, both), time-range selection, comparison, and plot export functionality.

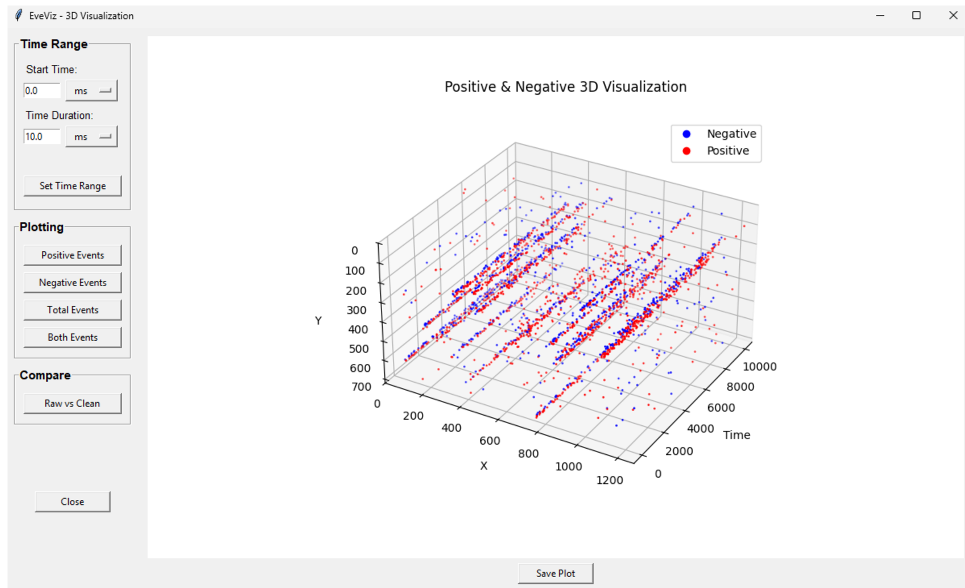

Figure 3.8 Lognormal 3D Visualization Window

Voxel Visualization Window

The Voxel Visualization Window (see Figure 3.9) renders a voxelized 3D representation. Therefore, the event data is discretized into time bins and combined with spatial and temporal structure. Similar to the other visualization windows, it offers different event modes (positive, negative, total, both), time range selection, comparison, and plot export functionality.

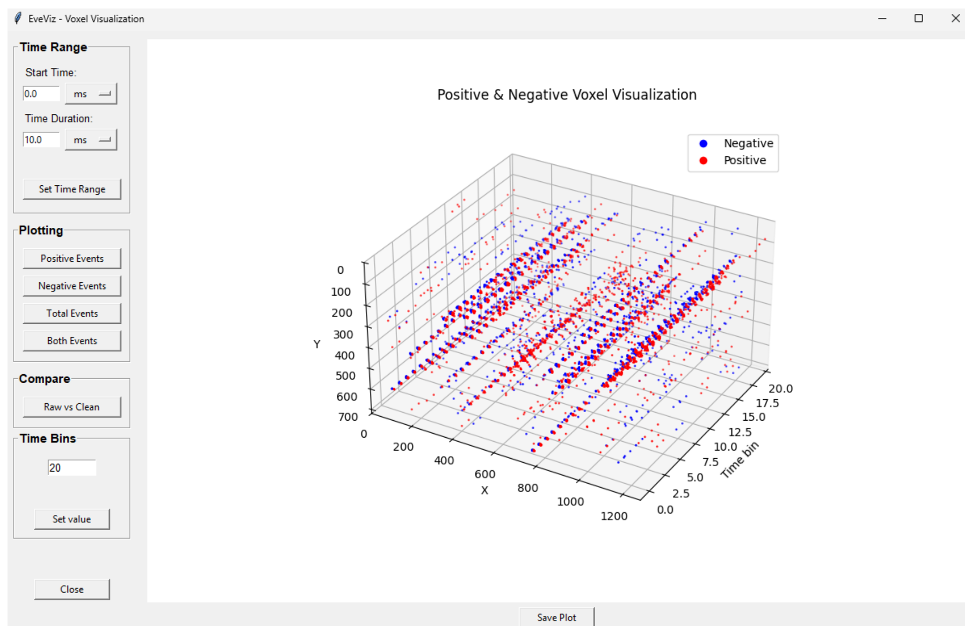

Figure 3.9 Voxel Visualization Window

3.2 Assessment

We evaluated the GUI on a number of event recordings. Table 3.1 summarizes the tested files, including their resolution, number of events, recording duration, and event rate. The performance of EveViz is summarized in Table 3.2.

File Name	Resolution	No. Events	Duration (s)	Event Rate (events/s)	File Size (MB)
bedroomsky.hdf5	1280x720	388,940	17.551	22,159	2.2
Berlin15DegMin_120FPS.hdf5	1280x720	150,447,329	16.643	9,039,416	297
Berlin15DegMin_120FPS.raw	1280x720	150,447,329	16.643	9,039,416	437
Berlin15DegMin_120FPS.bin	1280x720	16,447,160	16.643	988,233	47.6
ID_1_Cleaned_Fixed.hdf5	1280x720	49,824,960	59.865	832,260	177
processed_berlin_60.hdf5	1280x720	6,605,643	18.901	349,482	25
sun_sensing_panning.csv	1280x720	49,824,960	59.865	832,260	924
circle_events_only.h5	240x180	451,396	15.308	29,492	5.8

Table 3.1 Dataset Overview

File Name	Type	Successful loading	Loading Time
bedroomsky.hdf5	stars	✓	0.5
Berlin15DegMin_120FPS.hdf5	stars	✓	15.1
Berlin15DegMin_120FPS.raw	stars	✓	11.7
Berlin15DegMin_120FPS.bin	stars	✓	98.2
ID_1_Cleaned_Fixed.hdf5	sun sensing	✓	6.3
processed_berlin_60.hdf5	stars	✓	1.1
sun_sensing_panning.csv	sun sensing	✓	3.4
circle_events_only.h5	shapes	✓	0.4

Table 3.2 Results on Tested Dataset

The selected files cover all event camera file formats supported by the GUI, including HDF5, RAW, BIN, CSV, and H5. While several files originate from the same recording, they differ significantly in storage format and compression, allowing a direct comparison of their performance and usability.

A clear difference in loading times can be observed across file types. Binary formats, such as BIN, generally require more processing during loading due to decompression, which can result in longer loading times. Conversely, simpler formats, including HDF5, H5, and CSV, are loaded more quickly, with HDF5 and H5 being the fastest. As expected, datasets with fewer events, such as `circle_events_only.h5`, load faster due to the lower data volume.

Despite identical event content, different file formats of the same dataset expose varying performance characteristics. For example, the `Berlin15DegMin_120FPS` dataset shows consistent event counts and durations across formats, while loading times vary. This highlights the difference in data encoding and required processing.

Furthermore, the datasets differ in size and event rate, ranging from small-scale recordings with low event density to large datasets with millions of events per second. This helped us improve the tool for different scenarios. Large datasets primarily posed challenges during video visualization, as playback lagged due to the high number of events that needed to be processed and displayed in real time. Smaller datasets were more challenging to select appropriate cleaning filter parameters for. Since these datasets contain fewer events, overly aggressive filtering can remove meaningful information, leading to visualizations that no longer provide useful insight.

Overall, the assessment demonstrates that EveViz can handle a wide range of data formats and dataset sizes, while highlighting trade-offs among storage format, loading performance, and usability.

4 Conclusions

The goal of this project was to develop a GUI for visualizing event-based data. It should serve as a user-friendly platform for loading, cleaning, visualizing, and exporting data.

Chapter 2 introduces the GUI and its general structure. As was described in section 2.1, the GUI enables users to load event data from various file formats, namely CSV, HDF5, BIN, RAW, and TXT. Section 2.2 presents several filtering methods aimed at reducing noise and redundancy in the event data while preserving meaningful information. Furthermore, the visualization techniques described in Section 2.3 enable users to analyze the data in more detail. In addition to visualization, Section 2.4 introduces functionalities such as exporting images, videos, and cleaned data.

The project documentation includes annotated code, several ReadMe files, and this final report. This was done to ensure that the tool is easy to understand and extensible. The user should be able to modify the existing functionality, for example, by adding their own filters or integrating custom visualization methods. For this, a modular structure was used to allow said extensions.

4.1 Limitations

While the GUI serves as a valuable tool for event data processing, it has several limitations. One major constraint is the computational cost associated with large event datasets, which can lead to longer processing times. In addition, the GUI imposes a limit on the number of events that can be used for the video generation. While technically, slightly larger datasets can be processed, the video playback becomes noticeably less smooth. While there is no fixed limit on the time range for the other static visualizations, the GUI might crash if there are too many events in that time range. Setting a maximum limit was deemed unreasonable, as datasets differ in temporal event density.

If a limit is required for a user application, filtering is recommended to reduce the number of events. However, if filtering is not properly tuned, these filters might lead to a loss of information. While the recommended filter tuning values in EveViz work well for the test data used in this project, event datasets can vary significantly and may require a unique tuning.

Moreover, the code is partially based on external software, such as OpenEB, Metavision SDK, and ECF-codec. Some of these can be difficult to obtain, and if they are not installed, the tool's functionalities are significantly limited.

4.2 Future Work

Future work could further enhance this event visualization tool's capabilities. One potential extension is integrating live streaming, enabling the tool to visualize events in real time rather than relying solely on pre-recorded data. Moreover, adding a wider range of filters could be beneficial, enabling more flexible and effective cleaning. Finally, incorporating application-specific tools, such as star trackers, would expand the tools' usability in real-world scenarios.

4.3 Final Conclusions

Overall, this work highlights the usefulness of good visualization tools for working with event data, and the presented GUI provides a solid foundation for further developments in event data applications.

Bibliography

[1] Hesam Araghi, Jan van Gemert, and Nergis Tomen. Making every event count: Balancing data efficiency and accuracy in event camera subsampling. 2025.

[2] Sydney Dolan, Lara Schuberth, Rugved Arge, R. M. G. Alarcia, Vincenzo Messina, C. J. J. Oliver, Francesco Salmaso, Jaspar Sindermann, Federico Sofio, and Alessandro Golkar. Design and analysis of an event camera payload for space-based object detection on the eventsat 6u cubesat mission. In *Proceedings of the 15th IAA Symposium on Small Satellites for Earth System Observation*, 2025.

[3] Guillermo Gallego, Tobi Delbruck, Garrick Orchard, Chiara Bartolozzi, Brian Taba, Andrea Censi, Stefan Leutenegger, Andrew J. Davison, Jorg Conradt, Kostas Daniilidis, and Davide Scaramuzza. Event-based vision: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020.

[4] Antonio Junco de Haas, Sydney Dolan, and Alessandro Golkar. Optimized lossless event data compression for spaceborne event cameras. *Acta Astronautica*, 2026. Under review.

[5] Prophesee. Raw file format. `https://docs.prophesee.ai/stable/data/file_formats/raw.html`. Accessed: 2026-04-05.

[6] Prophesee. hdf5_ecf: Hdf5 event compression format implementation. `https://github.com/prophesee-ai/hdf5_ecf`, 2024. Accessed: 2026-04-08.

[7] Prophesee AI. Openeb. `https://github.com/prophesee-ai/openeb`, 2024. Accessed: 2026-04-07.

[8] The HDF Group. Hdf5 documentation: Introduction to hdf5. `https://support.hdfgroup.org/documentation/hdf5/latest/_intro_h_d_f5.html`. Accessed: 2026-04-05.