8-9-2024

Event Sensor Simulator with Hardware Accelerated Ray Tracer and Image Space Photon Mapping

Chengyi Ma
cxm3593@rit.edu

Event Sensor Simulator with Hardware Accelerated Ray Tracer and Image Space Photon Mapping

by

Chengyi Ma

A thesis submitted in partial fulfillment of the
requirements for the degree of
Master of Science
in Computing and Information Sciences

B. Thomas Golisano College of Computing and
Information Sciences
Rochester Institute of Technology

August 9th, 2024

Signature of the Author _____

Certified by _____
M.S. Program Director Date

B. Thomas Golisano College of Computing and Information Sciences

ROCHESTER INSTITUTE OF TECHNOLOGY

ROCHESTER, NEW YORK

CERTIFICATE OF APPROVAL

MS DEGREE THESIS

The MS degree proposal of Chengyi Ma
has been examined and approved by the
thesis committee as satisfactory for the
thesis required for the
MS degree in Computing and Information Sciences

Dr. Reynold Bailey, Thesis Advisor

Dr. Gabriel Diaz, Co-Advisor, Reader

Dr. Joe Geigel, Observer

Date

Abstract

In this study, we present an advanced event sensor simulator leveraging NVIDIA OptiX to address and overcome the limitations of existing event sensor simulators. Our simulator demonstrates significant improvements in both processing speed and temporal resolution compared to traditional methods such as v2e. These advancements are achieved by programming a real-time ray tracer with asynchronous event pixels using Nvidia OptiX's sophisticated motion system, which provides hardware motion interpolation on GPU threads. This enables sampling dynamic scenes to generate a series of events independent from frame rates. Additionally, we introduce an image space photon mapping method utilizing OptiX for enhanced global illumination, providing fast and realistic lighting effects in the simulated environment to meet the performance required to generate high frequency event streams. Our solution simulates real event cameras more accurately, expanding the potential applications of simulators in research.

Acknowledgments

I would like to express my deepest gratitude to all those who supported this research project. I extend my sincere thanks to my supervisors, Dr. Reynold Bailey and Dr. Gabriel Diaz. Your patience, guidance, and expertise have been crucial to the completion of this project. And I wish to appreciate my thesis committee member, Dr. Joe Geigel, for providing insightful suggestions throughout the exploration.

I am grateful to my fellow students and colleagues in the PerForm lab. Working with you over the past two years has been a tremendous learning experience; your creativity and talent have greatly impacted me.

Lastly, I want to say thank you to my family for all their support throughout my journey at RIT. This has been a long journey away from home, but your understanding and encouragement have been invaluable, helping me overcome challenges along the way.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Event Camera and Event based Vision

Event sensors, also known as neuromorphic cameras, represent a significant advancement in computer vision by capturing visual information in a unique way. Unlike traditional frame-based cameras that capture full images at fixed intervals, event cameras detect and transmit changes in brightness at each pixel asynchronously. Each transmitted change is referred to as an "event" and each event includes the x and y coordinates of the pixel, a timestamp, and a polarity value indicating whether the change was an increase or a decrease in brightness. Each event in an event camera is triggered by a change in logarithmic intensity. This logarithmic scaling allows the event camera to function effectively in environments with varying brightness levels. [6]

Event sensors operate using various bias settings. Each event pixel detects changes in brightness and compares them to a set bias value. An event is triggered if the change in brightness exceeds either the ON comparator bias or the OFF comparator bias. The refractory period bias then determines how long the pixel remains inactive before it can generate a new event. The overall bandwidth is controlled by the photoreceptor bias and source-follower bias. All these biases are generated by the chip's bias generator at the pixel level [15].

The ability to operate asynchronously at high frequencies allows event cameras to function with much lower bandwidth and latency than more conventional sensors, making them ideal for applications requiring real-time responsiveness and precise motion detection. Their capability to capture rapid

movements and adapt to varying lighting conditions is crucial in fields such as image and video restoration [11], object detection [17], simultaneous localization and mapping (SLAM) [5] and eye tracking [13].

1.2 Event Sensor Simulators

A common feature among all the mentioned applications is their reliance on machine learning methods to analyze input event data. Traditional computer vision algorithms are typically designed to work on 2D arrays of RGB values, making it challenging to process event streams effectively. Consequently, learning-based methods such as convolutional neural networks (CNNs) and spiking neural networks (SNNs) have become popular solutions for handling event streams. These learning based methods require training with annotated event datasets to function properly, making the scale and quality of the training dataset crucial to the performance of the machine learning model. However, the availability of large-scale annotated event datasets is limited. Event data, while advantageous, are sparse both temporally and spatially, making them significantly more challenging to label compared to traditional image-based datasets. Additionally, event cameras are not always available to researchers as most of these cameras are much more expensive than conventional frame-based cameras. As a result, there is a scarcity of annotated event datasets available for various research purposes [20]. Event camera simulators, such as v2e [18] and ESIM [7], offer a promising solution to these challenges. Firstly, simulators can generate labels directly, simplifying the otherwise difficult and time-consuming process of data annotation. Secondly, simulator software is easier to set up and use compared to physical event cameras, improving accessibility and reducing costs. Additionally, certain scenes and motions, such as rapid movements and structured lighting, are easier to create and reproduce virtually, making it more efficient to prototype solutions. Consequently, event camera simulators not only streamline the research process but also make advanced event-based vision technology more accessible.

While existing simulation solutions have effectively addressed the fundamental functionality of event sensors, several critical issues remain unresolved. Both V2E and ESIM generate events by comparing image frames, and produce an event stream characterized by large packets of events occurring at relatively low temporal frequencies. This differs from the asynchronous nature of event

cameras in which each event pixel operates independently from another. This discrepancy causes current solutions to produce event streams that are limited by the input video's frame rate, which is significantly lower than the frequency of an event camera. This substantial difference can negatively impact performance. Alternatively, to achieve higher temporal resolution, a significantly greater number of frames must be generated to ensure each pixel updates correctly over time, resulting in an extremely lengthy generation process. Both scenarios are suboptimal for research purposes.

1.3 Objective

This project aims to overcome these challenges by designing a new event sensor simulator with asynchronous event pixels that generate event streams efficiently. We employ computer graphics techniques to build the simulator for rendering events within a 3D virtual scene. This approach enables precise control over the behavior of each event pixel, effectively simulating an event camera with adjustable parameters. We utilize Nvidia OptiX, leveraging its motion system and rays with timestamps to sample a scene asynchronously. The rendering pipeline is simplified to compute intensity only, as event sensors output events independently of frequency and wavelength. Additionally, to efficiently compute indirect illumination and prevent the global illumination feature from becoming a computational bottleneck and slowing down the entire system, we introduce a novel hardware-accelerated photon mapping method to compute global illumination. This method simulates how light interacts within the 3D scene representation efficiently.

Chapter 2

Background

The development of event camera simulators has become crucial for advancing research and applications in event-based vision. Current simulation methods primarily focus on two approaches: converting video frames into event streams and generating events from virtual 3D environments. Each method has its own set of techniques, benefits, and challenges, which are critical to understand in order to appreciate the innovations introduced by this project. In this chapter, we will first delve into the advantages and challenges of these existing methods. Following that, we will discuss our approach to overcoming these challenges, paving the way for more efficient and accurate event camera simulations.

2.1 Current Simulators

This section provides an overview of the current state-of-the-art simulators used for event cameras. These simulators play a crucial role in the development and testing of event-based algorithms and systems. We will examine two prominent simulators, V2E and ESIM, discussing their features, capabilities, and limitations.

2.1.1 V2E

V2E [18], or Video to Events, is a simulator that converts standard frame-based videos into event streams, leveraging existing video data to simulate the behavior of an event camera. One of the primary advantages of V2E is its

ability to utilize neural networks, specifically the SuperSlowmo network [18], to generate interpolated frames. This interpolation improves the temporal resolution of the video, allowing for a more detailed representation of motion. By using readily available video footage, V2E provides a convenient and accessible method for generating event data without the need for specialized hardware.

However, despite its innovative approach, V2E faces several challenges that limit its effectiveness in accurately simulating event cameras. One significant limitation is the inability of traditional video images to capture the high dynamic range (HDR) signal in real-world scenes. Event cameras are capable of detecting a wide range of light intensities, which is essential for accurately representing dynamic environments. Standard videos, cannot replicate this HDR capability, resulting in a loss of detail and accuracy in the simulated event stream.

Another challenge is that the frame rate of videos, even after interpolation by SuperSlowmo, remains much lower than the frequency at which event cameras operate, because creating a sequence of images with periods of only a few microseconds is impractical. It is also inefficient to generate interpolated images since this requires rendering full images, which consumes significant resources, even though only a few pixels changes. Such methods creates unnecessary computational overhead without substantial benefits, making it an inefficient use of computational resources. In comparison, event cameras can detect changes in light intensity at extremely high speeds, often much faster than the highest frame rates achievable by conventional video cameras. This discrepancy means that V2E-generated event streams cannot fully capture the rapid dynamics that event cameras are designed to detect, leading to a less precise simulation.

Furthermore, the nature of event pixels in V2E simulations is not truly asynchronous. Event cameras operate by detecting changes in light intensity at each pixel independently and asynchronously, providing a continuous and highly responsive stream of data. In contrast, V2E generates events based on interpolated video frames, meaning the event pixels are synchronized to the frame rate of the video. This synchronization fails to replicate the true asynchronous nature of event cameras, reducing the fidelity of the simulation.

Lastly, the SuperSlowMo neural network used in V2E is not specifically trained for certain types of images, such as those used in eye-tracking applications. Eye-tracking images often have unique characteristics and require

specialized processing methods to accurately simulate event data. The general-purpose nature of SuperSlowmo means it may not perform optimally for these specialized scenarios, further limiting the applicability of V2E in diverse research and development contexts.

In summary, while V2E offers a practical solution for generating event data from existing videos, it faces significant challenges in accurately replicating the capabilities of event cameras, especially in special cases such as eye movements. These challenges include the inability to capture high dynamic range, lower frequency compared to event cameras, lack of continuous asynchronous event pixel generation, and limitations in handling specialized image types. Addressing these challenges is crucial for advancing the fidelity and utility of event sensor simulators.

2.1.2 ESIM

ESIM [7], or Event-based Simulator, is another prominent tool used for generating synthetic event data from 3D models and scenes. One of the notable features of ESIM is its ability to estimate the time delta (dt) based on the motion within the scene. This approach allows ESIM to create event streams that closely mimic the temporal dynamics of real-world environments, enhancing the realism and accuracy of the simulation. By simulating the motion of objects and estimating the corresponding event data, ESIM provides a valuable platform for testing and developing event-based vision algorithms.

However, despite its advanced capabilities, ESIM also faces several limitations that impact its effectiveness. A significant drawback is the lack of global illumination, as ESIM does not employ path tracing techniques to simulate realistic lighting conditions. Global illumination is essential for accurately rendering how light interacts with surfaces, especially in complex environments. Without it, the simulated scenes in ESIM may lack the depth and realism needed for high-fidelity simulations. While ESIM offers a "photo realistic" option by integrating with Unreal Engine. The provided Unreal Engine renderer is very slow, as described in ESIM's official documentation [7], it could take many hours to render one minute of trajectory. Moreover, the system is built for simulating camera motion in 3D scenes. It does not support object animation. This makes it suitable for SLAM applications, but not for other object and motion detection purposes. Another limitation is that it also uses a

frame-based generation method that events are only generated by comparing complete frames, leading to the same issue that v2e has.

Moreover, the integration of specific renderers is required to achieve the desired visual quality in ESIM. This dependence on external rendering engines adds complexity to the simulation setup and can introduce inconsistencies in the event data. Each renderer may have different capabilities and limitations, affecting the overall quality of the simulation. Researchers and developers must carefully select and configure these renderers to ensure the best possible approximation of real-world conditions. And the strong coupling between renderers and simulators makes it difficult to write custom components for a specific tasks.

In conclusion, while ESIM provides an framework for simulating event data based on estimated motion time, it faces several challenges that limit its effectiveness. The absence of global illumination and object animation, reliance on traditional renderers, and the lack of asynchronous pixel generation are significant drawbacks. These limitations must be addressed to enhance the realism and accuracy of event-based simulations.

2.2 Overcoming Challenges in Existing Solutions

One of the major challenges in the development of event sensor simulators is achieving truly asynchronous event pixels and obtaining better temporal resolution, particularly for applications involving rapid movements, such as eye tracking. These challenges can be addressed by focusing on two key areas: making events asynchronous and rendering with better performance.

Creating truly asynchronous events involves designing algorithms that can independently monitor and respond to changes in light intensity at each pixel, without waiting for a global frame update. This can be achieved by leveraging advancements in parallel processing and GPU computing. By mapping each pixel's event detection process to individual threads on a GPU, it is possible to simulate the independent and continuous nature of real event cameras. This approach ensures that each pixel operates asynchronously, allowing for a more accurate and responsive simulation of rapid movements, such as those observed in eye tracking.

Achieving better temporal resolution requires not only asynchronous event generation but also the ability to render scenes efficiently and accurately. In

computer graphics, multiple rendering techniques have been developed over the past years. The main the methods that are widely used nowadays are the rasterization and ray tracing methods. They each have their own advantages and disadvantages for different tasks. Therefore we need to make a comparison between these two methods to find out the best solution for our program.

Rasterization is a widely used rendering technique known for its speed and efficiency. It converts 3D objects into 2D images by mapping vertices to the screen space and filling in the pixels. This approach is highly efficient and suitable for real-time applications, such as video games and interactive graphics, where rendering speed is crucial. However, rasterization has significant limitations when it comes to rendering complex optical effects such as refraction and reflection.

One of the primary drawbacks of rasterization is its inability to accurately simulate the way light interacts with surfaces in real-world environments. Refraction, which occurs when light passes through transparent materials and bends, and reflection, which involves light bouncing off reflective surfaces, are challenging to render accurately using rasterization. These effects require tracing the path of light as it interacts with surfaces, a task that rasterization cannot perform efficiently due to its inherent design. Consequently, rasterization often produces approximations of these effects, which can lead to less realistic images.

To address some of these limitations, rasterization techniques often incorporate software-based global illumination [19]. Global illumination models how light bounces around a scene, contributing to indirect lighting and shadows. While software global illumination can enhance the realism of rasterized images, it is heavily reliant on precomputed parameters and simplifications. These approximations make the lighting less accurate and dependent on fine-tuning parameters to achieve acceptable results. The reliance on such parameters can also limit the adaptability of rasterization to dynamic scenes with varying lighting conditions.

In contrast, ray tracing offers a more accurate and flexible approach to rendering. By simulating the actual paths of rays of light as they travel through the scene, ray tracing can naturally handle complex interactions like refraction and reflection. Each ray can interact with surfaces in a realistic manner, producing high-fidelity images with accurate lighting and optical effects. Additionally, ray tracing inherently supports global illumination, as it can trace

the multiple bounces of light required to capture indirect lighting accurately.

While ray tracing is computationally intensive and traditionally slower than rasterization, advancements in GPU technology and frameworks like NVIDIA OptiX have made real-time ray tracing feasible. The ability to leverage GPU acceleration allows ray tracing to perform efficiently, closing the performance gap with rasterization while delivering superior visual quality.

Given the significant advantages in rendering accuracy and realism, ray tracing was chosen for our event sensor simulator. Ray tracing's ability to model intricate light interactions, such as global illumination, which includes both direct and indirect lighting, makes it an ideal choice for our needs.

2.3 Utilizing Nvidia OptiX to Address Simulation Challenges

To overcome the challenges associated with existing event camera simulators, we employ Nvidia OptiX, a powerful framework for real-time ray tracing. Nvidia OptiX provides a flexible and efficient platform for rendering complex scenes, enabling us to create more accurate and realistic simulations of event cameras. One of the primary advantages of using Nvidia OptiX is its ability to handle asynchronous event generation. By leveraging OptiX's motion system and ray tracing capabilities, we can sample scenes at different timestamps for each pixel, reflecting the asynchronous nature of event cameras. This allows us to generate event streams that are both temporally precise and computationally efficient. Additionally, the flexibility of Nvidia OptiX allows for the integration of custom shaders and algorithms, enabling precise control over the behavior of each event pixel. This customization is essential for tailoring the simulator to specific applications and research needs.

Chapter 3

System Design and Implementation

In this chapter, we present the design and implementation of our event sensor simulator utilizing Nvidia OptiX. The goal of this project is to address the limitations of existing simulators by achieving truly asynchronous event pixel generation and improving temporal resolution, particularly for applications involving rapid movements such as eye tracking. This chapter will provide a comprehensive overview of the system architecture, the key components and algorithms used, and the implementation details.

3.1 Architecture Overview

The architecture of our event sensor simulator is designed to efficiently simulate asynchronous event generation and high temporal resolution using Nvidia OptiX. In Figure 3.1, we provide an overview of the system's architecture. This architecture integrates various components and subsystems to achieve our goals.

The system begins with input parameters and a 3D asset file. The parameters encompass various settings required for the simulation, such as scene configuration, event sensor parameters, camera settings, and other relevant details. These parameters are either used to configure the pipeline or uploaded to a GPU buffer for use by shaders at runtime. The 3D asset file contains

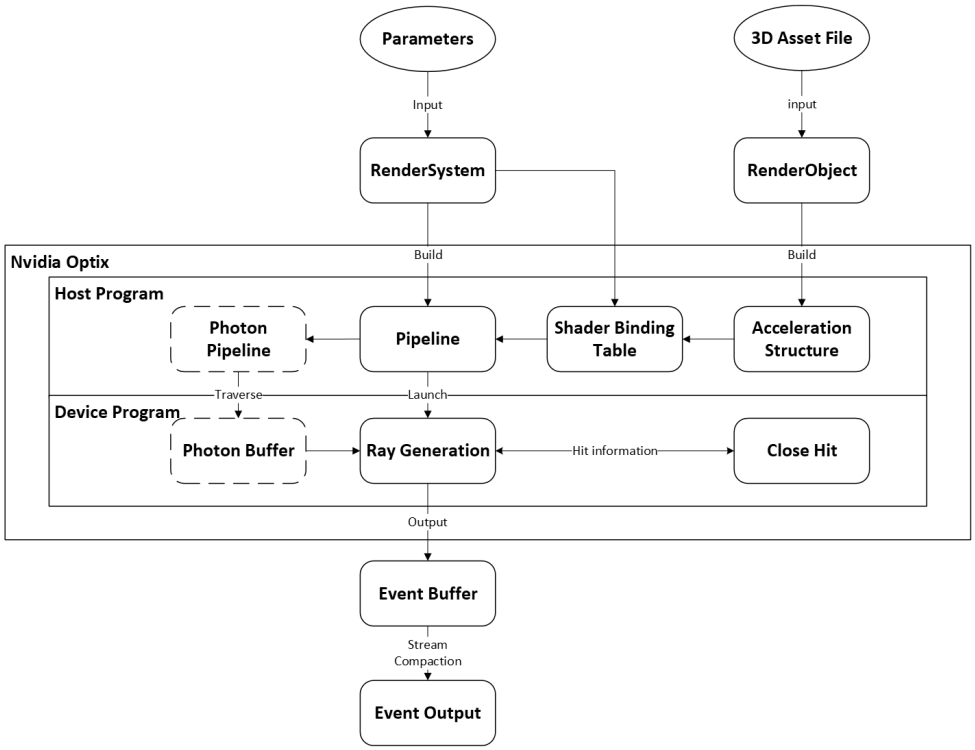


Figure 3.1: An overview of the architecture. This figure illustrates how input data and parameters are processed in the application program to generate synthetic events. The system could be divided into three stages: preprocessing stage, simulation stage and output stage. In the first preprocessing stage, input data such as 3D models, animations and textures are organized in RenderObjects and parameters are used to configure the RenderSystem properties. In the second stage, Nvidia OptiX's components are built, such as pipelines (photon mapping pipeline is optional), a shader binding table and acceleration structures in the host CPU program. Then the pipeline is launched to run GPU device programs to perform the simulation. In the final stage, the events will be saved in an event buffer and exported with a stream compaction algorithm.

the model of the scene to be rendered, including material data and animation

data for motion.

RenderObject is the structure responsible for recursively processing the input 3D assets. RenderObjects are nested in a tree structure, similar to the structure used in GL Transmission Format (glTF) models, to efficiently manage the hierarchical nature of the scene data. Material data and animation keyframes are stored within the corresponding objects.

RenderSystem holds all the parameters and states of the rendering pipelines. It is responsible for compiling and building several different types of components required by an OptiX program, including contexts, modules, program groups, pipelines, and shader binding tables, which are defined as follows:

- The **context** in OptiX is the primary object that manages all the states and resources for the ray tracing operations. It encapsulates the device memory, execution configuration, and overall state needed for rendering.
- **Modules** are program files of OptiX programs written in CUDA or PTX that define the behavior of the rays as they traverse the scene. Module types include including ray generation, closest-hit, any-hit, and miss shaders.
- **Program groups** in Optix define specific stages of the ray tracing pipeline. Each group can contain different types of programs, such as ray generation, intersection, and hit programs, allowing for modular and organized shader management.
- **Optix pipelines** are a sequence of operations that define the execution flow of the ray tracing process. A pipeline connects the various program groups and ensures that rays are processed correctly through the different stages of shading and intersection tests.
- The **Shader binding table (SBT)** is a data structure that maps ray types to their corresponding shaders. It provides the necessary linkage between the ray tracing pipeline and the specific programs that should be executed for different ray interactions [2]

Both *RenderObject* and *RenderSystem* help to build the OptiX program. The essential components required to launch an OptiX application is the linked pipeline, *shader binding table* (SBT) and an *acceleration structure* (AS). The

acceleration structure built by *RenderObject* is a spatial data structure required for efficient ray tracing. It optimizes intersection tests of rays with the geometric data in the scene, significantly speeding up the rendering process [2].

The core of our event sensor simulator is built upon the powerful ray tracing capabilities provided by NVIDIA OptiX 8.0. NVIDIA OptiX provides a flexible and programmable ray tracing pipeline, which allows us to implement custom algorithms tailored to our specific needs. The OptiX application is divided into two main categories: host programs and device programs. Host programs manage the overall state of the application, set up the rendering pipeline, and coordinate tasks between the CPU and GPU. Device programs run on the GPU and handle the actual ray tracing computations by executing a series of linked shaders, including ray generation shaders, closest-hit shaders, any-hit shaders, and miss shaders. These components work together to trace rays through the scene, determine intersections with objects, and compute the resulting shading and lighting effects.

When the OptiX application is launched, the device program first executes ray generation shaders for each event pixel. These shaders generate rays with different ray time values to sample the acceleration structure at different times. The intersection data is then reported by the close hit shaders. Unlike conventional ray tracing programs that uses this information to compute pixel colors, this intersection information is used to generate event data, which will be saved in a CUDA object called the *event buffer*.

Finally, the event buffer copies the event data back to the host CPU program, eliminating empty events in the buffer through a stream compaction process.

An additional rendering pipeline, the photon pipeline, is optionally used to improve the simulation capabilities. This pipeline utilizes a hardware-accelerated photon mapping technique to model global illumination within the 3D environment. By tracing the paths of photons and their interactions with various surfaces, it provides a more efficient method to compute indirect illumination in the scene.

3.2 Data Input

The system starts with importing necessary data such as 3D representations and animations. The data input stage is a foundational component of our

event sensor simulator, and we utilize the Open Asset Import Library (i.e., *ASSIMP* [3]) to import and process 3D asset files.

Assimp is an open-source library designed to import and export various 3D model formats. It supports over 40 different file formats, including widely-used ones such as OBJ, FBX, and glTF, making it a versatile tool for handling a wide range of 3D assets created using different modeling software [1].

One of the key features of Assimp is its ability to manage hierarchical structures inherent in complex 3D models. These structures include nested objects and transformations, which are represented as nodes in a scene graph. Each node can have multiple children, forming a tree-like hierarchy. our RenderObject class mirrors this arrangement to maintain the relationships and transformations between different parts of the model.

Additionally, Assimp can read material data and animation keyframes bound to these hierarchical meshes. This comprehensive data extraction helps us bind Shader Binding Table (SBT) records when building acceleration structures for each object. The SBT links ray tracing operations with the appropriate shaders and materials, ensuring that the rendering pipeline could access necessary data for each object.

3.3 Optix Ray Tracing Application Components

Once the input data is processed properly, they will be loaded to the main ray tracing program. As shown in Figure 3.1, the simulator's ray tracing application is built around three major components: acceleration structures, shader binding tables, and pipelines. In this section, we will detail the design and implementation of these components.

3.3.1 Acceleration Structure

As mentioned above, the acceleration structure is crucial for optimizing the ray tracing process. We designed a hierarchical acceleration structure based on the input data's hierarchical organization. This design leverages the inherent hierarchy to efficiently manage motion systems and temporal rays, ensuring that dynamic changes and asynchronous sampling are handled seamlessly.

As shown in Figure 3.2, we have designed a three-level acceleration structure to represent the hierarchical data structure of 3D objects in the scene.

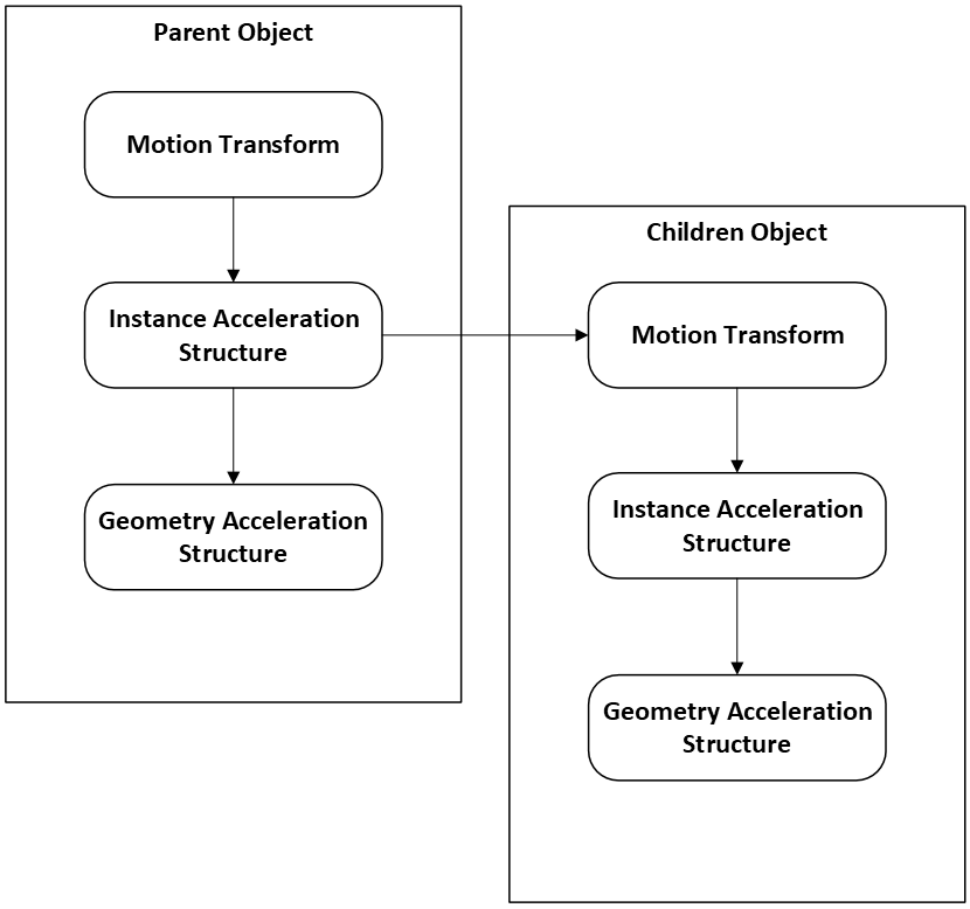


Figure 3.2: The architecture of acceleration structures. The motion transform nodes hold the motion keys to do interpolations. Instance acceleration structure can hold an object's own geometry or handles to other children objects.

Each object is represented by a *motion transform*, an *instance acceleration structure* (IAS), and a *geometry acceleration structure* (GAS). This hierarchical approach optimizes the ray tracing process, ensuring efficient handling of both static and dynamic objects.

The motion transform component manages the dynamic aspects of the

object, such as translations, rotations, and scaling over time. This ensures that any movement or transformation of the object is accurately reflected in the acceleration structure. This node is important for sampling with asynchronous event rays because it represent the motion between current two key frames with their own timestamps that the system uses to generate interpolated transformation matrices.

The *instance acceleration structure* (IAS) is the intermediate level that connects the motion transform to the GAS. It organizes instances of objects and their respective transformations. In our system, the IAS is primarily responsible for holding all the children of a parent object and maintaining their hierarchical transformations, as well as the static transformations of each object for their initial states. This design ensures that any transformations applied to the parent object are correctly propagated to its children, preserving the integrity of the scene's spatial relationships.

The geometry acceleration structure (GAS), is responsible for holding the actual mesh data. This structure includes all the vertices and indices that define the geometry of the objects.

3.3.2 Pipeline

In NVIDIA OptiX, the pipeline defines the flow of execution, connecting various shader programs and managing the stages of the ray tracing process. It ensures that rays are generated, traced, and shaded correctly, facilitating the integration of complex rendering techniques. The pipeline consists of multiple stages, including ray generation, intersection, closest hit, any-hit, and miss shaders. This is illustrated with Figure 3.3

In our program, the key shaders include the ray generation shader, the miss shader, and the closest hit shader. Each ray generation shader acts like an event pixel, generating multiple rays over time. Once the rays are generated, the OptiX framework performs ray traversal. By default, this traversal tests intersections against triangles using the default intersection shader and invokes the closest hit shader when a ray intersects an object at a specific ray time. The corresponding Shader Binding Table (SBT) record is passed to the closest hit shader for evaluation. The closest hit shader then computes the intensity value and returns the result to the ray generation shader through a payload, which is a special memory with very limited size to pass data between shaders

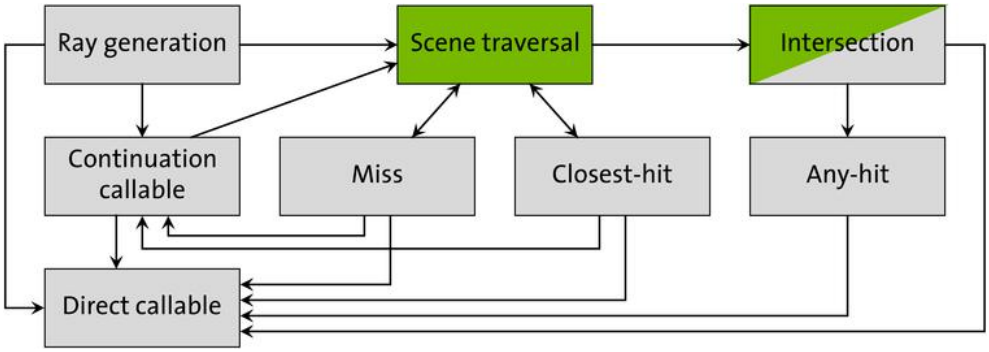


Figure 3.3: The relationship of Nvidia OptiX programs. Image from the Nvida OptiX programming guide [4]

efficiently [2]. Finally, the ray generation shader compares the intensity values and generates event data, which is saved to an event buffer.

3.3.3 Shader Binding Table (SBT)

In those shader programs, the program needs to get access to rendering data such as materials, vertex normals or texture coordinates, but they do not know which object they will be rendering for until the intersection tests are finished. Unlike traditional shaders, which typically access material and other relevant data directly through texture maps and uniform buffers, hardware-accelerated ray tracing frameworks require a Shader Binding Table (SBT) to acquire object material and other relevant data for each shader program.

In NVIDIA OptiX, shader data is stored in Shader Binding Table (SBT) records for each shader. The main types of SBT records include ray generation records, miss records, and hit group records. Among these, the hit group record is the most crucial. Each acceleration structure is indexed to an SBT record to provide the necessary rendering data for closest-hit shaders. Therefore, the SBT records need to be organized based on the acceleration structure's hierarchy.

3.4 Asynchronous Temporal Event Pixel

Unlike traditional image renderers, our goal is to simulate the asynchronous pixels of an event sensor. To achieve asynchronous temporal event pixel generation, we leverage the OptiX Motion System and the ray time property to sample the scene over different moments in time. This approach allows each ray to capture unique temporal information.

To sample the dynamic scene with different ray times, we need to provide two keyframes for the motion transform node in the object's acceleration structure. The first keyframe represents the starting frame, while the second keyframe represents the ending frame. Additionally, we must specify a starting time and an ending time for these keyframes. To have rotation transformations interpolated correctly, translation, rotation and scaling transforms are set separately. The rotation inputs are in quaternion format so that the system could use spherical linear interpolation(SLERP) for generating undistorted transformations.

When a ray traverses the scene, the OptiX framework interpolates the motion over time for each ray individually based on its ray time property and current two keyframes and timestamps. This interpolation allows the framework to return results that accurately reflect the object's state at the specific time the ray intersects it.

The ray time property in OptiX is crucial for capturing dynamic changes in the scene. Determining the appropriate ray time for ray generation is a key aspect of this process. By varying the ray time for each ray, we can simulate the passage of time and effectively track motion. To implement this, we use a strategy for computing ray times. The ray times are calculated by taking the difference between the current frame time and the last frame time, divided by N, where N represents the number of samples taken. Additionally, we can apply an indexed value or random offset to each ray time, ensuring that different event pixels operate on different initial timestamps. This method ensures that we can generate N events between two frames for each pixel with an unique set of ray times, providing a high temporal resolution and capturing rapid scene changes effectively.

Figure 3.4 illustrates the process of temporal multi-sampling. t1 and t2 represent two frames of the object's motion. The simulated event camera has an event pixel casting rays between these two frames. Normally, the ray R

will not intersect with the object in motion because it does not hit the object at either t1 or t2. However, by sampling the scene at t1.5, the ray can detect the object mid-motion.

Furthermore, by casting multiple rays at different times between t1 and t2, such as t1.49, t1.50 and t1.51, we can obtain multiple samples. This approach increases the likelihood of capturing the object during its motion, providing a more accurate and detailed representation of the dynamic scene. This is particularly effective for simulating the asynchronous behavior of event cameras, which need to detect rapid changes in the environment accurately.

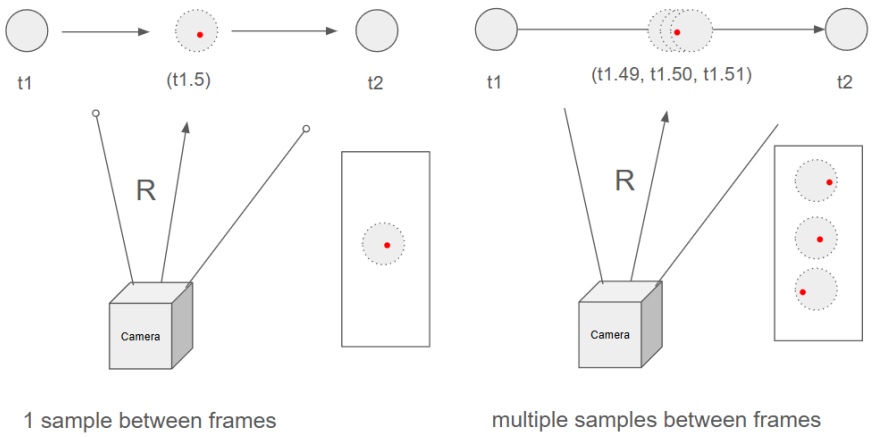


Figure 3.4: An illustration of temporal multi-sampling for an event pixel. A synthetic camera could generate multiple sampling rays to sample any moment between two frames individually, capturing the motion with better temporal resolution.

This method is advantageous because it leverages the parallel processing capabilities of the GPU, resulting in a highly efficient simulation process. Typically, interpolation is computed on the CPU, which means the process is handled sequentially and inefficient. Furthermore, the constant need to transfer data between the CPU and GPU further degrades performance, making real-time processing impractical. Interpolating motion frames on the GPU enables each ray to be processed in parallel, significantly accelerating the calculation

process.

3.4.1 Motion System and Key-frame Strategy

Now that we have the motion system in place, but it takes only two key frames for interpolation. To render a sequence of animations, we need to continuously update the motion transform node of the acceleration structure. For accurate interpolation of rotation transformations, we utilize the SRT (Scale, Rotation, and Translate) structure, with rotation represented in quaternion format.

The host program's animation loop maintains the current time and extracts new frames from the imported sequence, creating a new motion transform node to update the existing one in real-time. This ensures smooth transitions and accurate motion representation. The sequence of frames comes from the output of Blender, which provides frame-by-frame animation. Notice that even though the animation could be generated with an arbitrary number of key frames, Blender will interpolate key-framed motions to a sequence of frames with constant frame rate when exporting the output file.

3.5 Illumination

For each sampling ray, an illumination model is needed to evaluate the intensity value for the hit point. Illumination models describe how light interacts with surfaces in a scene, influencing the appearance of materials. These models are essential for simulating realistic lighting effects in computer graphics.

Unlike traditional renderers that compute the color for each pixel, our method focuses solely on computing the intensity. This approach is tailored to the requirements of event sensors, which detect changes in light intensity rather than color information. By computing intensity only, we streamline the rendering process and ensure that the generated event data accurately reflects the variations in light intensity within the scene.

For this project, we use a simplified Bidirectional Scattering Distribution Function (BSDF), specifically employing a Lambertian model. The Lambertian model assumes that light is scattered equally in all directions from a surface, resulting in a matte appearance. This model is computationally efficient and provides a basic yet effective approximation of diffuse reflection.

We chose this model because we need further investigation to determine which BSDF is most suitable for our simulator. The Lambertian model offers a starting point with efficiency, allowing us to set up the fundamental aspects of the simulator. This simplified approach enables us to establish a baseline for performance and functionality, from which we can explore more complex and precise models in the future.

3.5.1 Indirect Illumination: Photon Mapping vs Path Tracing

Having established the basics of direct illumination using a simplified Lambertian model, we now turn our attention to indirect illumination. Indirect illumination involves the complex interactions of light as it bounces off multiple surfaces before reaching the observer. Capturing these interactions is crucial for producing high-fidelity simulations, particularly in dynamic scenes where light and shadow are in constant flux.

Two common techniques for simulating indirect illumination are Path Tracing and Photon Mapping.

Path Tracing [10] is a more straightforward and popular method. It traces the paths of individual light rays as they bounce through the scene, simulating the way light naturally propagates. Each path is traced until it either leaves the scene or is absorbed. Path tracing can produce highly realistic images by accurately simulating the interactions of light with surfaces, but it is computationally intensive and can be slow to converge to a noise-free image.

Photon Mapping [8], on the other hand, is a two-pass global illumination algorithm. The energy emitted from a light source are divided into many energy packs called photons. Since modern computers cannot generate and trace a realistic amount of photons efficiently, photons represent distribution of energy in the 3D scene. In the first pass, photons are emitted from the light sources in rays and traced through the scene, where they may interact with surfaces and scatter. These interactions are stored in a photon map. In the second pass, the photon map is used to estimate the light intensity at various points in the scene, providing an approximation of indirect illumination. Photon mapping is relatively efficient and can handle complex lighting effects such as caustics, but it often requires careful tuning of parameters and can suffer from artifacts if not adequately sampled.

It is difficult to make a direct quantitative comparison between path tracing

and photon mapping due to the inherent challenges in evaluating realism from result images and the different sets of parameters each method uses. Few research studies have comprehensively evaluated these methods side by side. The subjective nature of image quality assessment and the dependency on specific scene parameters complicate direct comparisons.

Quality-wise, both methods can provide excellent results. Path tracing excels in producing highly realistic images with accurate global illumination, reflections, and refractions. Photon mapping, on the other hand, is particularly adept at simulating complex light interactions such as caustics and diffuse inter-reflections, which can be challenging for path tracing [14].

Performance-wise, path tracing generally requires more rays and samples to produce a noise-free image, making it computationally intensive. It relies heavily on modern hardware acceleration and denoising techniques to be feasible for real-time applications. Photon mapping emits a predefined number of photons, with the number depending on the scene's complexity. It also requires a spatial data structure like a kd-tree to efficiently store and retrieve photon interactions, which can add to the computational overhead but is generally more predictable in performance than path tracing.

Usage-wise, path tracing is commonly used for industrial offline rendering where the highest quality and realism are prioritized, such as in film production and high-end visual effects. Photon mapping is frequently employed in research settings and simulations, where specific lighting effects and scenarios are studied in detail [9].

For real-time interactive applications, path tracing can be used with AI-based denoisers to approximate real-time performance. However, the performance remains limited and can struggle with highly complex scenes. Photon mapping offers methods like image space photon mapping and photon splatting, which are adapted for interactive applications, providing a balance between quality and performance suitable for real-time scenarios [12].

Considering the high frequency and real-time requirements of event cameras, achieving better performance is crucial. Therefore, we have chosen photon mapping for its efficiency in handling complex lighting interactions. By leveraging the hardware-accelerated OptiX pipeline, we can efficiently manage photons and utilize a screen space photon mapping method to meet the performance needs of our simulation. This approach ensures that we can achieve high-fidelity lighting effects while maintaining the necessary computational

efficiency for real-time applications.

3.5.2 Image Space Photon Mapping with Hardware acceleration

Traditionally, photon mapping involves storing photons in a 3D data structure, which can be computationally intensive and memory-demanding. Today, such complicated data structures, such as kd-trees, still do not work well on GPU. To address these challenges, we utilize a 2D buffer that maps the photons directly into image space. This buffer allows for efficient storage and retrieval of photon data, leveraging the GPU's parallel processing capabilities to accelerate the process.

In our event sensor simulator, we implement a method called Image Space Photon Mapping, enhanced with hardware acceleration. This approach stores photons in a 2D buffer in image space, optimizing the search and evaluation process for indirect illumination.

This method is inspired by McGuire's Image Space Photon Mapping (ISPM) [12]. In ISPM, it is noted that the final bounce of photons is the most computationally expensive, so it introduces the concept of using photon volumes for rasterization rendering to handle this last step efficiently. Since our approach does not rely on rasterization, we map photons directly to an image space buffer instead, and we can perform a 2D search for each pixel from the photon buffer to gather nearby photons efficiently. This process is illustrated with Figure 3.5

Photon Emission and Surface Interaction

To fully utilize the hardware acceleration features of OptiX, we create an additional pipeline specifically for photon processing. This new pipeline operates alongside the main event pipeline, benefiting from the same acceleration structure and Shader Binding Table (SBT) resources, thus optimizing resource usage and maintaining efficiency.

Photons are emitted and traced from a separate ray generation shader, originating from the light sources within the scene. The total flux of each light source is divided into N photons, which are then cast into the scene using rays. These photons interact with surfaces, capturing essential light interactions that contribute to the indirect illumination within the scene.

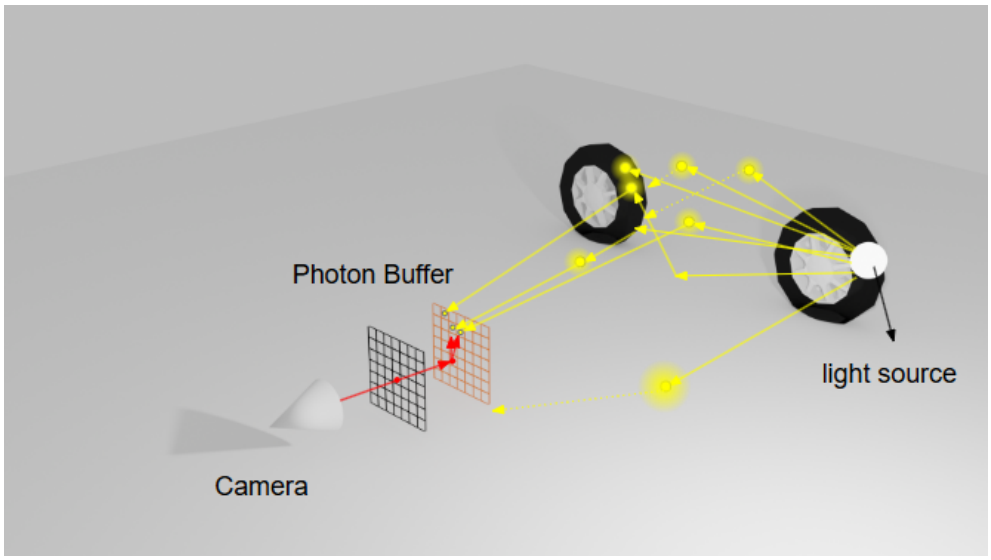


Figure 3.5: An illustration of Image Space Photon Mapping (ISPM). Photons are emitted and traced from the light source in the first pass. Once it stays on a surface, it will be mapped to a photon buffer (orange). In the rendering pass, camera pixels will search in the photon buffer for closest photons and make an estimation of intensity value for indirect illumination.

The interaction of a photon with a surface can result in three possible outcomes: the photon remains on the surface, undergoes diffuse reflection, or undergoes specular reflection. To control this behavior, we set two parameters for materials in the SBT record:

$$0 \leq t_{\text{diffuse}} \leq t_{\text{specular}} \leq 1$$

Once a photon hits a surface, a random number is generated. If the number is smaller than t_{diffuse} the photon remains on the surface. If the number is larger than t_{diffuse} but smaller than t_{specular}, the photon undergoes diffuse reflection. If the number is above t_{specular} the photon reflects in a mirror-like manner.

Usually, photon mapping utilizes BRDF parameters to dictate the photon's behavior upon interacting with surfaces. However, since we are currently employing a simplified Lambertian model, we have introduced this system to

control photon interactions for now. More complex behaviors will be implemented as we integrate more advanced shading models in the future. This initial approach allows us to lay a solid foundation, ensuring that our system can be easily extended and refined with more sophisticated models as our work progresses.

Image Space Photon Map

When a photon interacts with a surface and comes to rest, it must be determined if the photon is visible to the camera before saving it. This is achieved by tracing an occlusion ray from the photon's position to the camera. If the photon is not obstructed and is visible to the camera, it is then saved; otherwise, it is discarded.

To compute the pixel coordinate from hit position, we first convert the 3D world position to normalized device coordinates(ndc) and then use the ndc coordiantes with camera's resolution to determine which pixel position it falls into. The equations are listed below:

$$\text{hit\_vec4} = \begin{pmatrix} \text{hit\_position.x} \\ \text{hit\_position.y} \\ \text{hit\_position.z} \\ 1.0 \end{pmatrix}$$

$$\text{ndc} = \text{camera\_matrix} \times \text{hit\_vec4}$$

$$\text{pixel.x} = \left\lfloor \left(\frac{\text{ndc.x}}{\text{ndc.w}} + 1.0 \right) \times 0.5 \times \text{dim.x} \right\rfloor$$

$$\text{pixel.y} = \left\lfloor \left(\frac{\text{ndc.y}}{\text{ndc.w}} + 1.0 \right) \times 0.5 \times \text{dim.y} \right\rfloor$$

Ideally, a 3D photon buffer is used to store multiple photons that fall into the same pixel bucket, ensuring comprehensive illumination data for each pixel. However, this introduces significant complexity due to the asynchronous nature and sheer volume of photons generated. Managing millions of photons asynchronously poses substantial challenges. The synchronization required to insert each photon into the buffer across all threads is computationally prohibitive and inefficient.

To address this, we employ an asynchronous insertion method. While this approach can lead to overwriting and race conditions, it is generally acceptable

because photons are typically sparse and photons that are very close to each other tend to have very similar properties. This method allows for efficient photon handling despite potential conflicts.

Future improvements could include advanced synchronization techniques to locally coordinate threads, minimizing data conflicts and enhancing the accuracy of photon mapping. This will allow for a more robust and precise simulation of indirect illumination.

Photon Collection and Radiance Estimation

Once the event pipeline is initiated, it first computes the direct illumination for each pixel. After this initial computation, the pipeline proceeds to gather photons from the photon buffer near the pixel's position to estimate the indirect illumination. The search for relevant photons is conducted within a defined square boundary in the 2D buffer. This boundary size is set by a parameter to ensure that the search does not extend too far in areas with low photon density, maintaining computational efficiency. The size of the search boundary is scaled by the pixel's distance from the view position, ensuring that the search area is appropriately sized relative to the pixel's position in the scene.

The total number of photons considered is limited by a parameter. This helps manage the computational load and ensures that the estimation process remains efficient. By limiting the number of photons, we prevent the system from being overwhelmed in the regions with photons in very high density, which could slow down the rendering process by a lot.

Additionally, to prevent extreme cases where distant photons might be incorrectly considered, a distance filter r is applied. If a photon is farther than this specified distance r from the evaluation point in image space, it is skipped. This ensures that only relevant photons contribute to the radiance estimation.

For efficient photon gathering, the search starts from the center and expands outward in square steps. This methodical approach ensures that photons closest to the pixel are considered first, optimizing both the accuracy and speed of the indirect illumination estimation. By prioritizing nearby photons, we can more accurately simulate the diffuse lighting effects that contribute to the overall realism of the scene.

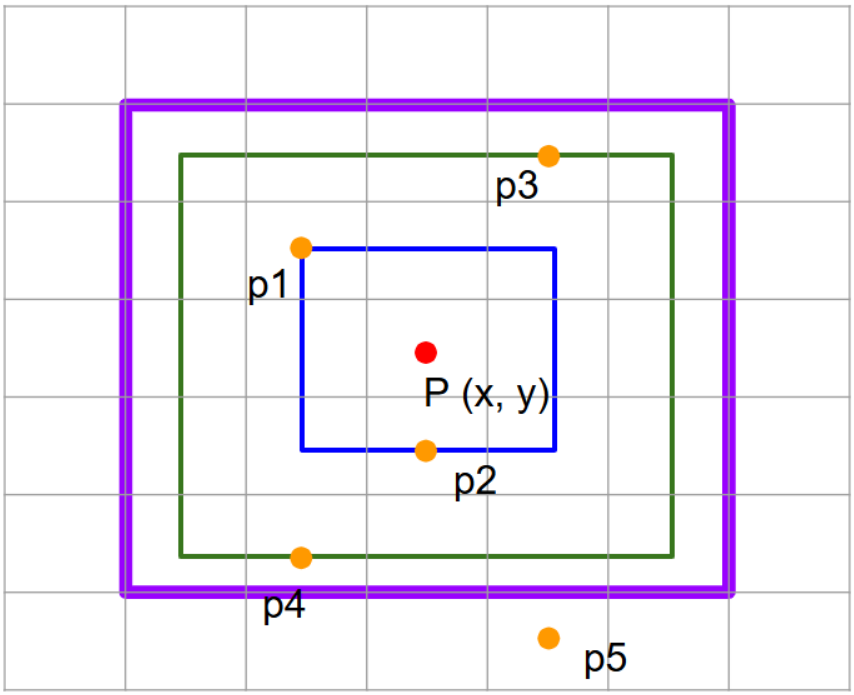


Figure 3.6: An example of gathering process from photon buffer. Given a photon buffer with 5 photons saved (p1 to p5), to search photons within window boundary(purple). The system will first search the positions closest to the evaluation pixel(blue box) and then expand to a larger area(green box).

The Figure 3.6 illustrates an example of the photon collection process. We are evaluating a pixel at coordinates x, y, indicated by $P(x, y)$, with a search boundary size of 5, represented by the purple square. In the photon buffer, there are five photons, labeled from p_1 to p_5. The searching process begins at $P(x, y)$ and iterates through the positions covered by the blue square, followed by the green square, moving left to right and top to bottom.

If we are gathering only three photons, the photons p_1, p_2 and p_3 will be collected, while p_4 and p_5 will not be included in this iteration.

To estimate the radiance from the collected photons, we use a general

equation commonly found in existing photon mapping methods [14]. The equation is given by:

$$L(x, \omega) \approx \frac{1}{N} \sum_{i=1}^{N} k_r(x, x_i) \tau_i$$

In this equation, $L(x, \omega)$ represents the estimated radiance at point x in direction of ω. N is the total number of photons considered in the estimation. $k_r(x, x_i)$ s the kernel function that determines the contribution of the i-th photon to the radiance estimation. In our project, we use a Gaussian distribution kernel for the kernel function. However, other common kernels such as cone kernels and other distance-based functions could also be used to weight the contributions of photons based on their distance from the evaluation point. This function evaluates the distance between the evaluation point x and the photon position x_i. τ_i is the flux of the i-th photon times BRDF.

3.6 Bias and Event Stream Output

Once the intensity values are calculated for each pixel, the next step involves converting these values into event streams. This process begins by applying a logarithmic transformation to the intensity values, which compresses the dynamic range and simulates the response characteristics of real event sensors. Subsequently, a bias threshold is applied to the log-transformed intensities to determine when an event should be generated. Events are triggered when the change in intensity exceeds the bias, effectively capturing significant variations in light intensity. Once an event is fired from a pixel, the base intensity level will be reset to the new value for future comparisons. The resulting event streams, which indicate the pixel location, timestamp, and polarity, provide a high temporal resolution and asynchronous output that accurately mimics the behavior of real event cameras.

3.6.1 Stream Compaction

The returned event buffer contains many empty events that need to be filtered out before the event data is loaded back to the CPU host program. Because the CPU program is not efficient with iterating through such a large 3D data

structure and eliminating all the empty spaces to output events in a compact format. This is known as a stream compaction problem [16]. To do this efficiently, we perform stream compaction using the Thrust library provided by CUDA. Thrust provides efficient parallel algorithms on GPU that allow us to compact the event stream, removing any empty or invalid events and ensuring that only valid events are retained for further processing.

3.6.2 Tone Mapping

For frame visualization, we employ the Extended Reinhard tone mapping method to handle the higher dynamic range values produced by our simulator. This method effectively compresses the wide range of light intensities into a format that can be displayed on standard screens, preserving detail and contrast.

3.7 Evaluation Setup and Methodology

We will conduct a preliminary evaluation focusing on processing speed and temporal resolution, which are the primary challenges we aim to address with our event sensor simulator. These aspects are crucial for ensuring that the simulator can operate efficiently and accurately in real-time scenarios, particularly for applications involving rapid movements and dynamic scenes.

At this stage, we are unable to perform a systematic evaluation of the accuracy of our simulator due to the lack of a ground truth dataset. Creating a ground truth dataset requires measurements of light intensities and object materials in a real scene, which would then be replicated in a similar virtual environment for testing. This process is essential for validating the accuracy of the simulator's output against real-world data.

Additionally, we do not yet have an advanced and measured shading model to evaluate the accuracy comprehensively. The current shading model used in our simulator is a simplified Lambertian model, which is suitable for initial testing but may not provide the level of detail necessary for precise accuracy assessments. Future work will involve integrating more sophisticated shading models that can better replicate the complexities of real-world lighting interactions.

To set up the testing scene, we use GIW data to animate eye movement

with an simplified RITEyes eye model. This updated model has been refined to remove colliding polygons, which previously hindered performance by complicating intersection tests for hardware accelerated raytracing frameworks.

We render a 30-second image sequence in Blender and use v2e (video to events) to generate corresponding event data. This provides a reference dataset for comparison.

The animation and model set up in Blender are exported to a glTF 2.0 file, which is then imported into our event sensor simulator to render 30 seconds of event data. We carefully tune the biases in the simulator to ensure a similar number of events are generated over the same animation sequence, allowing for a fair comparison.

Chapter 4

Result and Discussion

In this chapter, we will discuss the results of our preliminary tests conducted using the OptiX-based event sensor simulator. The tests were performed on a platform equipped with an Intel i9-10900 CPU, an NVIDIA 2080TI GPU, and 64 GB of memory.

4.1 V2E Dataset

To create a simulated event dataset using v2e, we began by rendering a sequence of eye animation in Blender. The animation lasted 30 seconds and was rendered at a resolution of 1280 by 720 pixels with a frame rate of 30 fps, resulting in a total of 900 frames. This rendering process took approximately 40 minutes.

Once the image sequence was rendered, we loaded it into v2e to generate the event data. For this, we set the positive and negative event thresholds to 0.15 and the timestamp resolution to 3ms, ensuring the output resolution remained consistent with the input resolution of 1270 by 720 pixels. We use a 3ms resolution because any value lower than it would result in much longer time to process.

The first step in v2e involved interpolating the image sequence to a higher frame rate of 360Hz, which corresponds to a time resolution of 2.78ms per frame. This interpolation process was computationally intensive, taking about 1 hour and 20 minutes to complete.

After interpolation, v2e proceeded to generate the event data from the

interpolated image sequence. This final step took an additional 20 minutes, resulting in a comprehensive event dataset that was ready for further analysis. Figure 4.1 shows an example images from this dataset.

4.2 OptiX Dataset

To prepare the dataset using our OptiX event simulator, we first exported the animated 3D scene from Blender into the GLTF 2.0 format. This format was chosen for its compatibility and efficiency in handling complex 3D scenes. Once exported, the scene was read into our OptiX event simulator.

To ensure that the dataset generated by OptiX was comprehensive and comparable to the v2e dataset, we manually fine-tuned the bias (threshold) to 0.05. This adjustment was necessary to match the magnitude of events generated by v2e, considering that our simulator uses a different illumination model than Blender.

For the simulation, we set the system to take 10 samples between each frame. This setup ensured a high temporal resolution, with an average time interval of 1ms between samples. By doing so, we aimed to capture the dynamic changes in the scene more accurately, similar to the high-frequency sampling performed by v2e. Notice that the temporal resolution here refers to the time periods between samples for a single pixel, not time difference between a sequence of events in a larger scope.

The animation playback rate was set to 1.0, meaning the simulation ran in real-time and completed in 30 seconds, matching the duration of the rendered sequence from Blender. Example images from this dataset are displayed in

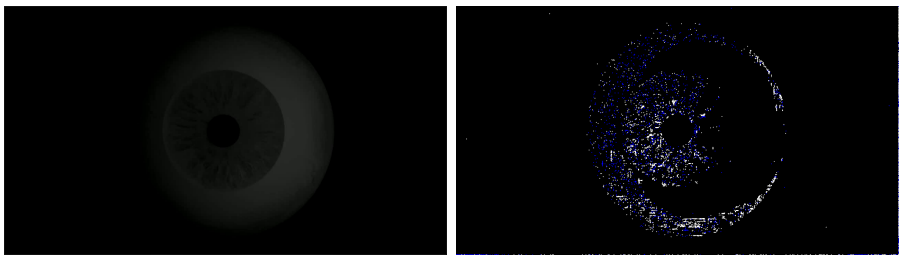
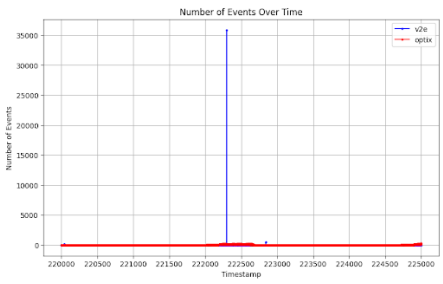


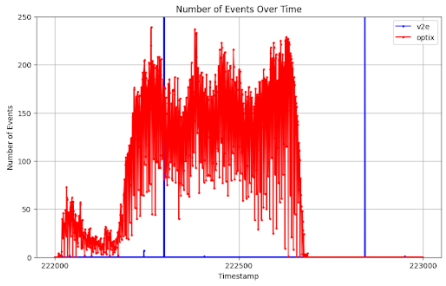
Figure 4.1: Images of V2E data: extracted frame image(left), accumulated event frame(right)

Figure 4.2

Figure 4.2: Images from OptiX event dataset: extracted frame image(left), accumulated event frame(right)



(a) The blue line represents v2e. The spike shows that most of the events generated have the same timestamp.



(b) A portion of data from 4.3a, but with the Y axes rescaled to reveal variation in the red trace that was not previously visible. Unlike the event data from V2c, events generated with our Optix-based algorithm can occur independently.

Figure 4.3: Comparison of event numbers over timestamps

4.3 Evaluation

Table 4.1 provides a comparison between v2e and our method's results.

The comparison between the v2e and OptiX event datasets reveals several key insights. The v2e event dataset comprises 169,825,482 events, while the

OptiX event dataset contains 186,636,537 events. This close match in the number of events indicates that both datasets are comparable in terms of the volume of data they provide.

In terms of processing efficiency, the OptiX simulator significantly outperforms the v2e system. The OptiX simulator operates in real-time, completing the event generation process in just 30 seconds. In contrast, generating the event data with v2e takes approximately 140 minutes. This substantial difference underscores the efficiency of the OptiX simulator in producing event data.

Temporal resolution also highlights a notable advantage of the OptiX simulator. The frame-to-frame interval, or temporal resolution, of the OptiX dataset is 1 millisecond, compared to 3 milliseconds for the v2e dataset. This finer temporal resolution allows the OptiX simulator to capture more precise and rapid changes in the scene.

Additionally, the asynchronous pixel feature of the OptiX simulator contributes to a higher number of unique timestamps. Specifically, the OptiX dataset includes 8,011,167 unique timestamps, whereas the v2e dataset contains only 75,675 unique timestamps. This greater number of unique timestamps in the OptiX dataset reflects its enhanced ability to capture a wider range of temporal variations within the scene. It also reflects the ability for Optix to better capture the asynchronous nature of pixel level events transmitted by the event sensor. The provided diagram in Figure 4.3 illustrates the distribution of events over time for a section of the data. When plotting the number of events against their timestamps, it becomes evident that the v2e dataset exhibits spikes at several timestamps. These spikes occur because v2e operates on a frame-by-frame basis, generating a burst of events at each frame interval. In contrast, our OptiX-based method produces events that are spread continuously over time. This continuous distribution of events ensures a more consistent and accurate representation of dynamic changes within the scene that is more consistent with the an event stream from a true sensor. The asynchronous event generation allows the OptiX simulator to capture subtle variations and movements more effectively than the frame-based approach used by v2e.

Table 4.1: Comparison between v2e and our method in processing time and performance

	V2E	OptiX Events
Total Events	169,825,482	186,636,537
Processing Time	140 minutes	30 seconds
Temporal Resolution	3ms	1ms
Unique Timestamps	75,675	8,011,167

Table 4.2: Comparison between various parameter settings in processing time and performance

	OptiX Standard	OptiX Slow	OptiX Extra	OptiX Photon
Total Events	186,636,537	264,307,541	293,080,056	112,302,293
Processing Time	30s	300s	30s	30s
Temporal Resolution	1ms	0.1ms	0.6ms	8ms
Unique Timestamps	8,011,167	18,547,501	15,772,339	1,872,819

4.4 Additional Comparisons and Verifications

To further verify the parameters and features of our simulation program, we conducted additional comparisons using datasets generated with different settings. These datasets were compared against the standard dataset used in our comparison with v2e. Table 4.2 presents this comparison.

Firstly, the *Optix Slow* method modified the playback rate of the animation to 0.1, significantly slowing down the simulation. This change extended the rendering time by a factor of ten, providing a temporal resolution of 0.1ms compared to the standard 1ms. The higher temporal resolution could break down a single change in intensity into several steps, so more than one event could be fired from a stimuli that previously generate only one event, producing many more events and unique timestamps.

Additionally, the *Optix Extra* method involved increasing the sampling rate to 100 times per frame, up from the standard 10 times per frame. This adjustment yielded a temporal resolution of 0.6ms. The performance improved notably due to the reduction in CPU iterations required, highlighting the efficiency gains achievable through higher sampling rates. This method pro-

duced more events and unique timestamps than the standard setting, but fewer unique timestamps than *Optix Slow*. This is because it has lower temporal resolution.

Optix Photon tested the use of photon mapping for global illumination. Despite the potential for improved lighting accuracy, this feature had minimal impact on the results in this specific case due to the insufficient number of polygons for photons to bounce off. Moreover, enabling photon mapping slowed down the performance, resulting in a temporal resolution of 8ms and much smaller amount of generated events and unique timestamp.

These additional comparisons underscore the flexibility and adaptability of our simulation program. By adjusting parameters such as playback rate, sampling rate, and global illumination settings, we can tailor the simulator to meet specific needs and constraints. This capability is crucial for optimizing performance and accuracy in various application scenarios.

Chapter 5

Conclusion

In this study, we have demonstrated the effectiveness and efficiency of our OptiX-based event sensor simulator compared to the existing v2e system. Our method leverages NVIDIA OptiX to program asynchronous event pixels using its motion system and ray time features. This approach allows for precise temporal sampling of dynamic scenes. Additionally, we introduced an image space photon mapping method to simulate global illumination. This method, combined with OptiX's powerful ray tracing capabilities, enables our simulator to produce event data efficiently.

Our results show that the OptiX simulator can generate a comparable number of events while significantly reducing processing time. Specifically, the OptiX simulator processes events in real-time, completing the task in just 30 seconds compared to v2e's 140 minutes.

Furthermore, the OptiX simulator achieves a finer temporal resolution. This higher resolution allows for more precise capture of dynamic changes within the scene, which is crucial for applications requiring high temporal fidelity. The asynchronous nature of the OptiX simulator results in a more continuous and evenly distributed event stream over time. This characteristic makes our generated event stream similar to event data captured by real event cameras, as evidenced by the higher number of unique timestamps in the OptiX dataset compared to v2e.

Chapter 6

Future work

The development of our event sensor simulator not only showcases significant advancements but also lays a solid foundation for addressing the challenges we've identified. The current implementation highlights key areas for further exploration and improvement, opening up numerous opportunities in this field. This chapter outlines the major limitations of the current project and suggests directions for future work to enhance the simulator's capabilities and performance.

First, our simulator currently employs a simple illumination model, specifically a Lambertian model, which is efficient but limited in its ability to simulate complex lighting interactions. In future iterations, we plan to incorporate more advanced illumination models such as OpenPBR, the same system used in Blender. we will also support physically measured materials, enhancing the realism and accuracy of the simulated scenes.

Second, the current system does not yet integrate the complete RITEyes system, which includes head models and other components essential for comprehensive eye-tracking simulations. Future work will focus on adding these elements, ensuring that the simulator can provide a fully functional and realistic eye-tracking solution.

Third, the performance of our simulator, while already optimized for GPU acceleration, can be further improved. By employing techniques such as compressed acceleration structures, Shader Execution Reordering and more efficient photon buffer for parallel reading, we could reduce computation time and enhance the overall efficiency of the system. These optimizations will enable

the simulator to handle more complex scenes and higher workloads with even better performance.

Additionally, to validate the accuracy of our simulation, we plan to construct a measured real scene and compare the output of our simulator against data from actual event cameras. This validation step is crucial for ensuring that our simulator produces precise and reliable event data, which is essential for its application in research and development.

Finally, for better workflow and integration into existing pipelines, we consider incorporating our simulator into platforms like Unity or Blender. This integration would provide a seamless user experience, leveraging the powerful features of these platforms for scene creation, animation, and rendering. It would also facilitate broader adoption of our simulator in various applications, from game development to scientific research.

In conclusion, the future work on our event sensor simulator will focus on enhancing the illumination model, integrating the full RITEyes system, optimizing performance, validating accuracy against real-world data, and improving workflow integration. These improvements will significantly advance the capabilities of our simulator, making it a more robust and versatile tool for event-based vision research and applications.

Bibliography

[1] The asset-importer-lib documentation. Available at: urlhttps://assimp-docs.readthedocs.io/en/latest/.

[2] Nvidia optix 8.0 – programming guide. Available at: https://raytracing-docs.nvidia.com/optix8/guide/index.html.

[3] The open asset importer library.

[4] Relationship of nvidia optix programs. Available at: urlhttps://raytracing-docs.nvidia.com/optix8/guide/index.html.

[5] Zhu Alex, Zihao, Yuan Liangzhe, Chaney Kenneth, and Daniilidis Kostas. Unsupervised event-based learning of optical flow, depth, and egomotion. *European Conference on Computer Vision*, 2018.

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

[7] Rebecq Henri, Gehrig Daniel, and Scaramuzza Davide. Esim: an open event camera simulator. *The 2nd Conference on Robot Learning*, 2018.

[8] Jensen Henrik, Wann. Global illumination using photon maps. *Eurographics Workshop*, 1996.

[9] Jensen Henrik, Wann and ChristensenAuthors Per, H. Efficient simulation of light transport in scenes with participating media using photon maps. *Seminal Graphics Papers: Pushing the Boundaries*, 1998.

[10] Kajiya James, T. The rendering equation. *SIGGRAPH '86*, 1986.

[11] Shijie Lin, Xu Fang, Wang Xuhong, Yang Wen, and Lei Yu. Efficient spatial-temporal normalization of sae representation for event camera. *IEEE Robotics and Automation Letters*, 2020.

[12] McGuire Morgan and Luebke David. Hardware-accelerated global illumination by image space photon mapping. *HPG '09: Proceedings of the Conference on High Performance Graphics*, 2009.

[13] Bonazzi Pietro, Bian Sizhen, Lippolis Giovanni, Li Yawei, Sheik Sadique, and Magno Michele. Retina : Low-power eye tracking with event camera and spiking hardware. *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, 2024.

[14] Zhu Shilin, Xu Zexiang, Jensen Henrik, Wann, Su Hao, and Ramamoorthi Ravi. Deep photon mapping. *Computer Graphics Forum: Volume*, 2020.

[15] Delbruck Tobi, Graca Rui, and Paluch Marcin. Feedback control of event cameras. *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2020.

[16] Rego Vernon, Sang Janche, and Yu Chansu. A fast hybrid approach for stream compaction on gpus. *2016 Fourth International Symposium on Computing and Networking (CANDAR)*, 2016.

[17] Deng Yongjian, Li Youfu, and Chen Hao. Amae: Adaptive motion-agnostic encoder for event-based object classification. *IEEE Robotics and Automation Letters*, 2020.

[18] Hu Yuhuang, Liu Shih-Chii, and Tobi Delbruck. v2e: From video frames to realistic dvs events. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021.

[19] Majercik Zander, Marrs Adam, Spjut Josef, and McGuire Morgan. Scaling probe-based real-time dynamic global illumination for production. *Journal of Computer Graphics Techniques*, 2009.

[20] Zhang Zhongyang, Cui Shuyang, Chai Kaidong, Yu Haowen, Dasgupta Subhasis, Mahbub Upal, and Rahman Tauhidur. V2ce: Video to continuous events simulator. *IEEE International Conference on Robotics and Automation*, 2024.