Event Camera Simulator in Unity
Computer Vision Project

Andrea Masciulli

University of Trento, Italy
andrea.masciulli@studenti.unitn.it

Abstract. Event cameras are novel sensors that, thanks to their peculiar properties (such as asynchronous sensing, no motion blur and high dynamic range), are showing exciting results in robotics, computer vision and related fields. Although, research on these sensors proceeds slowly due to their high cost and still-in-prototype stage. These issues led to the need of event camera simulators for cheap and high-quality synthetic event data to use for algorithm prototyping and deep learning. One of the first proposed simulators to fulfill these needs reliably is ESIM, an Open Event Camera Simulator. In this paper I propose a streamlined workflow for the use of ESIM along with the Unity game engine. The User will be able - after a brief setup process - to record a scene and to automatically convert it into event camera footage, all with the press of one button.

GitHub page: `https://github.com/AndreaMas/esim-in-unity.git`

1 Introduction

An event camera is an imaging sensor that responds to changes in brightness. Event cameras do not capture images using a shutter as conventional cameras do. Instead, each pixel independently and in continuous time quantizes local relative intensity changes to generate spike events. These events appear at the output of the sensor as an asynchronous stream of digital pixel addresses. These address-events signify scene reflectance change and have sub-millisecond timing precision [1].

Given their asynchronous sensing capability, the absence of motion blur, low latency (usually microseconds) and high dynamic range, these sensors are showing exciting results in robotics, computer vision and related fields. Although, their potential is now limited, since research is slowed down by the high cost and other practical limitations, such as low resolution, poor signal-to-noise ratio and complex sensor configuration, which requires expert knowledge [2]. In parallel, the rise of deep learning has led to an unprecedented demand for data.

To give to the community an efficient and reliable way to generate event camera data without the need to use a physical event camera, the Open Event Camera Simulator ESIM has been proposed by Rebecq et al. [2].

I propose, in this paper, a streamlined workflow for the use of ESIM along with the Unity game engine, so to ease the generation of synthetic event camera data inside of Unity.

2 General Workflow to Generate Event Data from Unity

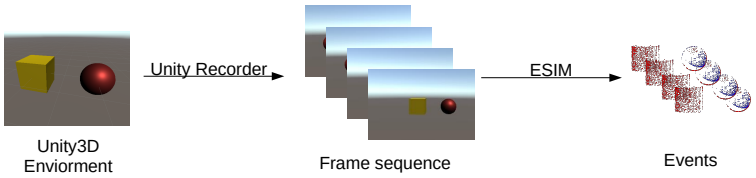

Fig. 1. General pipeline for the generation of synthetic event camera data from the Unity3D environment. In this particular example, the yellow cube and red sphere are moving (slowly wrt the frame-rate), therefore generating a stream of events.

To the best of the author's knowledge, the most straightforward way to start generating synthetic event data from Unity is to capture the Unity scene trough the Unity Recorder, and then feeding the footage to the ESIM Open Event Camera Simulator (see Fig. 1). Therefore, the suggested operating system for this project is Ubuntu, since ESIM is based on ROS (a software mainly meant to be used with Ubuntu operating systems).

To summarize, two things will need to be installed on our Ubuntu system:

- the Unity game engine
- ESIM: an Open Event Camera Simulator

Furthermore, to handle the recording (aka to capture frame sequences of the User's Unity scene), the Unity Recorder package is used.

The Unity Recorder package can be easily imported into the Users Unity project through the Unity Package Manager (the package may be in preview). To install ESIM, please refer to its paper [2] or to the relative GitHub page: `https://github.com/uzh-rpg/rpg_esim`.

3 Streamline the Workflow with Scripts

While the proposed workflow is straightforward and functional, for sure it's not seamless. The User, before recording, needs to set numerous parameters in the Unity Recorder window, making sure that frame-rate and destination file of the frame sequence correspond to the ones that will later be given to ESIM. Once the frame sequence has been recorded and saved, ESIM will need to be run by the User through the terminal, using a couple of commands. This process

is both time consuming and error prone. To streamline the workflow, scripts can be used in order to ease the communication between ESIM and Unity. The proposed scripts, to be included inside the Unity project, are two:

1. a C# script, that needs to be attached to a game object (can be any). This script is responsible for:
   - showing a basic UI button in the Game View, used to start and stop the recording,
   - displaying (in the game object's inspector) only the most significant event camera parameters to set,
   - calling the following bash script as soon as the recording is stopped, sending as arguments the parameters specified by the User;
2. a bash script, that will automatically execute all the commands needed to run ESIM (remember to mark as executable).

By adding these two scripts, the User is able to record a scene and to automatically convert it into event camera footage, all with the press of one button. Also, the User can set the most relevant event camera parameters (such as frame-rate, image resolution and contrast threshold) directly from Unity.

A more detailed guide on how to setup the Unity project, alongside the necessary scripts and a video tutorial, can be found in the relative GitHub page: `https://github.com/AndreaMas/esim-in-unity.git`.

4 Possible Improvements

The recording of the frame sequence is quite slow. To speed up the recording, one could choose to degrade the frames resolution, or record compressed video instead of single frames, so to record at run-time (though, video will need to be divided into frames afterwards anyway, and the frame-rate might not be as stable.

While all runs with one button, the setup process could be even more simplified. It would be better if all the necessary code and packages could be compressed in a single plugin.

The relatively new ROS# package may offer better solutions and workflows, especially for Windows Users, but I haven't dug into it.

References

1. Lichtsteiner, P; Posch, C; Delbruck, T . A 128× 128 120 dB 15 microseconds latency asynchronous temporal contrast vision sensor. IEEE Journal of Solid State Circuits, 43(2):566-576 (2008)
2. Rebecq, H., Gehrig, D., Scaramuzza, D.: ESIM: an Open Event Camera Simulator. Conf. on Robotics Learning (CoRL) (2018)