A Real-time Dynamic Vision Sensor Emulator using Off-the-shelf Hardware

Pineda García, Garibaldi Camilleri, Patrick Liu, Qian Furber, Steve

March 12, 2016

Abstract

Vision is one of our most important senses, a vast amount of information is perceived through our eyes. Neuroscientists have performed several studies using vision as input to their experiments. However, computational neuroscience has typically used Poisson-encoded images as spike-based visual sources. Recently neuromorphic Dynamic Vision Sensors have surfaced, while they have excellent capabilities, they remain scarce and difficult to use.

We propose a visual input system inspired by the behaviour of a DVS, but using a digital camera as a sensor. By using readily-available components, we believe, most scientist would have access to a spiking visual input source. While the primary goal was to use the system as a real-time input, it is also able to transcode well established images and video databases into spike train representations. Our main contributions are adding locally inhibitory behaviour, adaptive thresholds and proposing time-based encoding of the output spikes.

1 Introduction

In recent years the performance of computer processors has been advancing in smaller increments than it used to a few years ago. This is mainly because manufacturing technologies are reaching their limits. One way to improve performance is to use many processors in parallel, which has been successfully applied to parallel-friendly applications like computer graphics. Task like pattern recognition are still a hard task for computers even with these technological advances.

Our brains are particularly good at learning and recognizing visual patterns (e.g. letters, dogs, houses, etc.). In order to achieve better performance for similar tasks on computers, scientists have looked into biology for inspiration. This has lead to the rise of brain-like (neuromorphic) hardware, which looks to mimic functional aspects of the nervous system. We can divide neuromorphic hardware into sensors (providing input) and computing devices (make use of information from sensors). Visual input has been traditionally obtained from images that are rate-encoded using Poisson processes, while this might be a biologically-plausible encoding in the first phase of a "visual pipeline" it is unlikely that eyes transmit as much information into later stages. Furthermore, if we think of it in terms of digital networks, having each pixel represented by a Poisson process incurs in high bandwidth requirements.

In 1998, Mead proposed a silicon retina consisting of individual photoreceptors and a resistor mesh that allowed nearby receptors to influence on the output of a pixel [1]. Later, researchers developed frame-free Dynamic Vision Sensors (DVSs) [2], [3]. They feature independent pixels that emit a signal when its intensity value changes above a certain threshold. These sensors have μ-second response time and excellent dynamic range properties, although they are still not as commercially available as regular cameras.

In this work, we propose to emulate the behaviour of a DVS using a conventional digital camera as a sensor. Basing the emulator on widely available hardware, would allow most computational neuroscientists to include video as a spike-based input.

Katz, Nikolic, and Delbruck developed a DVS emu-

lator in order to test behaviours for new sensor models [4]. In their work, they transform the image provided a commercial camera into a spike stream at 125 frames per second (fps). In simple terms, the emulation is done by differencing video input with a reference frame; if this difference is larger than a threshold it produces an output and updates the reference. The number of spikes produced per pixel are proportional to how many times the difference would pass the threshold. This emulator has been merged into the jAER project, a Java-based Address-Event Representation software framework that specializes on processing DVS output in real time.

2 Work

Pixels in DVSs are independent and transform light input into a logarithmic representation. Most commercial cameras produce gamma-encoded images [5] to better utilize bits and, in older days, to be compliant with cathode ray tube (CRT) monitors. Figure 1 shows the response for the encoding process (crosses), CRT monitors (hexagons) and decoding process (dots). Since the encoding response is similar to the logarithmic used in the DVS, we skip this step of the emulation.

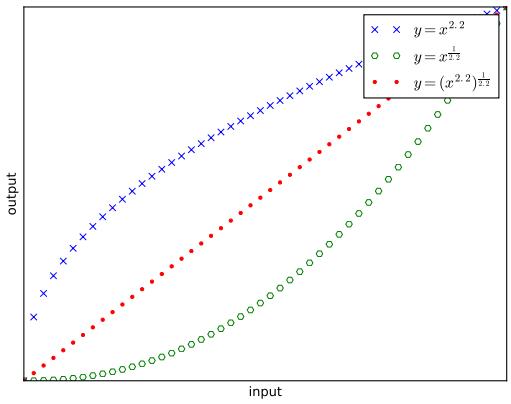

Figure 1: Gamma encoding and decoding functions, $\gamma = 2.2$

Asynchronous pixel behaviour is approximated via differencing the current image obtained by the camera and a reference frame; whenever a pixel's difference is larger than a certain threshold, we mark that position as "spiked". Each spike carries a flag, *up* if there was a positive change in intensity or *down* if a decrease was registered, this can also be thought of as a spike's *sign*. Depending on the selected type of output we simulate a receiver and update the reference accordingly. This is the basic DVS emulation and it's illustrated in Figure 2.

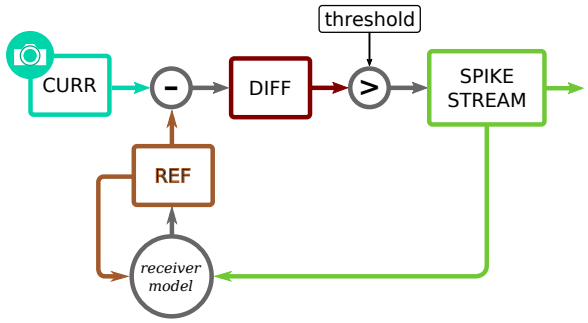

Figure 2: DVS emulation diagram. Circles indicate operations and rectangles states of data

2.1 Output modes

2.1.1 Rate-based

As in previous emulators[4], the standard output format is rate-based. In this mode each spike can be interpreted as an increment or decrement in intensity. To calculate the number of spikes that represents this change in intensity we use the following expression

$$N_H = \left\lfloor \min\left(T, \ \frac{\Delta I}{H}\right) \right\rfloor \tag{1}$$

where N_H is the number of spikes needed to represent the change in intensity ΔI in terms of the threshold H. Notice that the maximum time to transmit all the spikes for one frame is bound by the frame rate of the camera (fps). If it's only possible to send one spike per millisecond, we can send as many

spikes as milliseconds in one frame-capture period ($T = 1000/fps$). At this stage we model a perfect receiver, so the update rule for the reference is

$$R_{now} = R_{old} + N_s \times H \tag{2}$$

2.1.2 Time-based

In its worst case rate-based encoding can send a spike per millisecond per pixel, which can potentially saturate communication channels. One way to prevent this is to encode the value that each spike represents in the time each it is sent.

First we propose to linearly encode the number of thresholds exceeded. To do this the result of Equation 1 would in, a bin width (W_b) of 1 ms was used for time discretization. As in the previous case the time to send all spikes for the current frame is at most $N_b = T/W_b$, which means the maximum difference possible is $T \cdot H$.

Figure 3 shows the value-to-spike-time relation, in our proposal earlier spikes represent larger changes in intensity. The main advantage of this encoding is that a single spike could represent multiple rate-based spikes, though the encoded values are limited by time resolution and the frame rate of the camera.

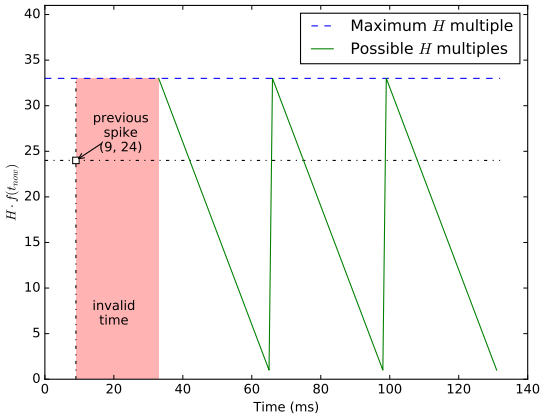

Figure 3: Linear time-based encoding of thresholds (H) exceeded

To decode on the receiver end, we must keep track of the time and value from the last collected spiked. Let Δt be the difference in arrival time between the previous and current spike (Eq. 3)

$$\Delta t = t_{now} - \left(t_{last} - \left[N_b - N_H^{last} W_b\right]\right) \tag{3}$$

where the subtraction of $N_b - N_H^{last} W_b$ sets the time reference to a multiple of time period T. We now divide the time difference by the time-bin width to obtain the current bin

$$B = \frac{\mathrm{mod}\,(\Delta t,\ T)}{W_b} \tag{4}$$

here, mod calculates the arguments' division remainder. Now that the time bin B is known, all it takes to compute the number of thresholds exceeded is

$$N_H^{now} = \lfloor N_b - B \rfloor \tag{5}$$

To overcome the limitation on maximum encoded values we propose *binary encoding* of either the absolute difference in intensity or the number of thresholds exceeded. The main advantage of this technique would be to achieve large values with fewer time bins as the growth is exponential (Fig. 4). We propose to use this encoding in two ways: *shoot-and-refine* and *full value*.

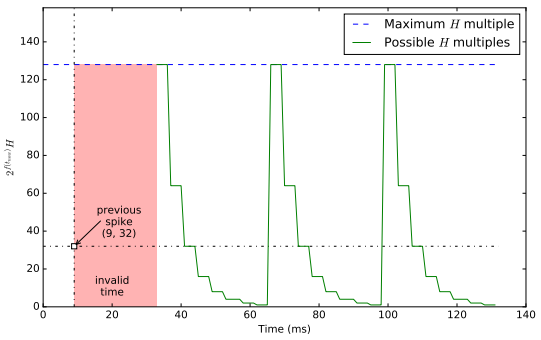

Figure 4: Exponential time-based encoding of thresholds (H) exceeded

In the *shoot-and-refine* mode, a single spike is sent with an over- or underestimate of the desired value (e.g. the next power of two), and in the following

frames we send at most one spike to refine the received value towards the desired one. Decoding can be done in a similar way as the linear case, but we divide the available time in N_b bins of width is $W_b = T/N_b$. Now to compute the time difference

$$\Delta t = t_{now} - \left(t_{last} - 2 * \left[N_b - log_2(N_H^{last})W_b + 1\right]\right) \tag{6}$$

Since we are using time bins that are larger than the system's time resolution, the term $N_b - N_H^{last}W_b$ of Eq. 3 into

$$2 * \left[N_b - log_2(N_H^{last})W_b + 1\right] \tag{7}$$

The correct bin is calculated using Eq. 4 as in the linear case; finally for the decoded value

$$N_H^{now} = 2^{\lfloor N_b - B \rfloor} \tag{8}$$

For the *full value* mode many spikes would be sent per pixel each frame, thus providing better resolution to the sent value, the downside to this is that an accumulation buffer is needed in addition to the previous spike time and value buffers. Decoding multiple spikes per frame has to be split in two cases: first if the new spike arrives before the current period is finished, then we accumulate its value to the current decoded value; otherwise we replace the contents of the accumulation buffer with the newly decoded value.

2.2 Additional behaviours

In this segment we describe modifications done to the basic DVS emulator, these changes are rendered in Figure 5. In the lower left a history decay mechanism was added to the receiver model, constant threshold has been exchanged by an adaptive version, and an inhibitory block (*max*) completes the list of changes.

2.2.1 History decay - What if spikes get lost?

Initially we described a system where every spike that is sent will be captured on the receiver end, but this is not always the case. To cope with the latter cases, we now introduce a history decay mechanism which will allow the receiver to, in the long term, recover from missing spikes. Let $D = \{0, 1\}$ be the weight

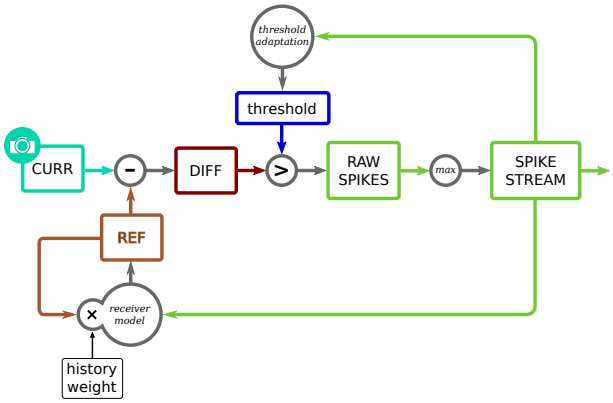

Figure 5: DVS emulation with adaptive thresholds.

history has to calculate the new reference value, then the reference's update rule becomes

$$R_{now} = D \times R_{old} + N_s \times H \tag{9}$$

2.2.2 Adaptive threshold

A subtle detail that other emulators have not captured is the slow-charging pixels, cameras would capture these as similar values in each frame thus the difference would never be enough to generate a spike. Meanwhile in a real DVS pixels receiving light that is not enough to immediately trigger a spike gain some charge and, after some time, will generate a spike event. We propose to mimic this behaviour by adapting the threshold in a per-pixel basis, that is reduce it if a pixel did not create a spike. Since thresholds value may lowered, they also have to be increased if a spike was generated. This increment also adds a "protection" against fast changing pixels which could be the result of hardware malfunction or bring no information to the image. Figure **??** shows how threshold adaptation fits into the DVS emulator diagram.

2.2.3 Local inhibition

In mammalian retinas inhibitory circuits play an important role, some researchers have suggested that it reduces the number of spikes needed to represent

what the eye is seeing{ref}. Our inhibition mechanism follows the same idea, since neighbouring pixels have similar values, we suppose that they are transmitting redundant information. The inhibitory behaviour is simply a local MAX-like operation{ref} of pixel areas. An example is shown in Figure 6, the maximum value (in green; 77) will generate a spike, while other values (in red; 0, 31, and 15) are blocked.

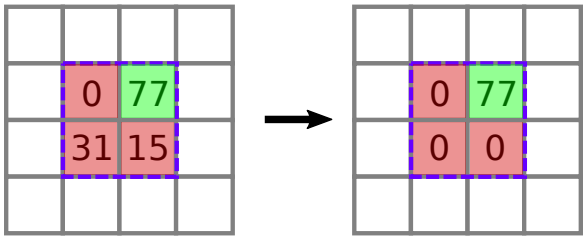

Figure 6: Local inhibition mechanism, quantities represent the absolute value of the difference between an image and reference.

Open source, and available in github.com

3 Results

Real-time DVS emulation on consumer hardware (Intel i5, 8GB ram) sPyNNaker wrapper for videos and images.

4 Conclusion

GPU computing version would allow for higher resolutions.

Working on ganglion cell/Difference of Gaussian encoding.

References

[1] C. Mead, *Analog VLSI Implementation of Neural Systems*, C. Mead and M. Ismail, Eds. Boston, MA: Springer US, 1998, ch. Adaptive Retina, pp. 239–246, ISBN: 978-1-4613-1639-8.

[2] P. Lichtsteiner, C. Posch, and T. Delbruck, "A 128x128 120 db 15 us latency asynchronous temporal contrast vision sensor," *IEEE Journal of Solid-State Circuits*, vol. 43, no. 2, pp. 566–576, 2008, ISSN: 0018-9200. DOI: 10.1109/JSSC.2007.914337.

[3] J. A. Lenero-Bardallo, T. Serrano-Gotarredona, and B. Linares-Barranco, "A 3.6 us latency asynchronous frame-free event-driven dynamic-vision-sensor," *IEEE Journal of Solid-State Circuits*, vol. 46, no. 6, pp. 1443–1455, 2011, ISSN: 0018-9200. DOI: 10.1109/JSSC.2011.2118490.

[4] M. L. Katz, K. Nikolic, and T. Delbruck, "Live demonstration: Behavioural emulation of event-based vision sensors," in *Circuits and Systems (ISCAS), 2012 IEEE International Symposium on*, 2012, pp. 736–740. DOI: 10.1109/ISCAS.2012.6272143.

[5] C. Poynton, *Digital Video and HDTV Algorithms and Interfaces*, 1st ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2003, ISBN: 1558607927.