2009 Second International Conference on Computer and Electrical Engineering

Simulation of Triple Buffer Scheme

(Comparison with double buffering scheme)

Shujjat Khan, Donald Bailey, Gourab Sen Gupta
School of Engineering and Advanced Technology
Massey University
Palmerston North, New Zealand
shujjaatkhan@gmail.com

***Abstract*—Buffer management is a key design issue in the development of any machine vision system. Triple buffering is a memory economical buffer management scheme which minimizes the processor idle time while frames are being acquired and ensures the processing of intact frames. Matlab simulation results of the triple buffer scheme clearly prove the significance of this scheme in contrast to other buffering schemes.**

Keywords-Buffer management schemes, single buffer, double buffer, triple buffer, Matlab simulation

I. INTRODUCTION

In the modern automation industry, there is a need for efficient data acquisition to enable robust control of real-time processes. Vision can provide a large amount of data very quickly, making it the sensor of choice in many control applications. It is also very economical due to its relatively low complexity and the ready availability of off-the-shelf components. On the other hand, digital images contain a large volume of data, hence as the size of an image grows, the time taken to transmit and process an image also grows proportionally [1]. This situation becomes even more complex in multiple-view applications [2, 3], where a number of visual sensors need to be integrated with centralised or distributed processing units. In a real-time application, irrespective of the number of incoming visual information streams, a digital image has to be processed once it is received in memory. Processing and acquisition stages are synchronised by either a single shared buffer or two "ping-pong" buffers but they are not able to overcome the delay and under-filled buffer swap problems respectively. Therefore, there is need for a self-synchronising technique (a triple buffer scheme) that ensures the integrity and validity of the frame being processed using minimum memory.

Section II briefly reviews existing buffer management techniques. In section III, a triple buffer scheme is described and simulated and its results are discussed, while conclusions are presented in sections IV.

II. BUFFER MANAGEMENT SCHEMES

On a software platform image data captured from a camera needs to be stored in memory before it is processed. The image should be processed as soon as possible after it becomes available. To fulfil this requirement, following are the buffer management schemes are normally adopted in vision applications.

A. FIFO Queues

In this scheme, the frame-buffer is no more than an array of *n* image slots arranged in First-In-First-Out (FIFO) order. Incoming frames are temporarily stored in a predefined frame-buffer of the length specified at development time. Processing is done on all acquired frames one by one in a round-robin fashion and the processed slots are filled again. The size of the queue is a vital factor which determines the performance of system. If the size is too small, some frames may be over-written resulting in lost of indispensable information. On the other hand, increasing the number of the buffer slots unnecessarily not only incurs heavy burden on system's memory but also reduces system performance.

This scheme is very simple and easy to implementation [4, 5] but not suitable for real-time applications because the processor has to process each frame in the queue before it can access the most recent one.

B. LIFO Queues

In this scheme, frames are processed according to Last-In-First-Out (LIFO) order. A camera sends video frames to the frame-buffer and the most recent buffer is detached from the list and fed to processing unit. This scheme overcomes the shortcoming of FIFO queues by readily processing the most recent frame. A disadvantage, however, is that the frames are not always processed in their natural order. This scheme suits software vision applications where memory is not a big issue but it is not an optimal solution for embedded applications where memory is the most expensive resource.

C. Single buffer scheme

The single buffer scheme is a form of FIFO or LIFO frame-buffer with *n=1*. In this scheme, a single buffer is shared between acquisition and processing modules. Image data from the source is captured in a buffer of the same size as that of image being acquired, and the processing module shares the same buffer for the image processing operations. It is up to the designer to switch the control of the buffer between acquisition and processing. Fig. 1 illustrates one common method, where the buffer is synchronised by two

978-0-7695-3925-6/09 $26.00 © 2009 IEEE
DOI 10.1109/ICCEE.2009.226

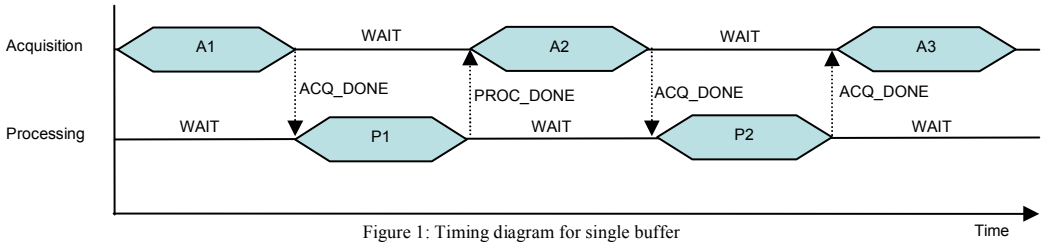


Figure 1: Timing diagram for single buffer

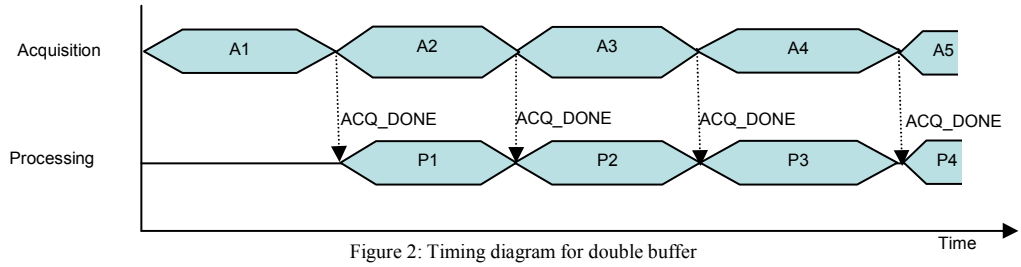


Figure 2: Timing diagram for double buffer

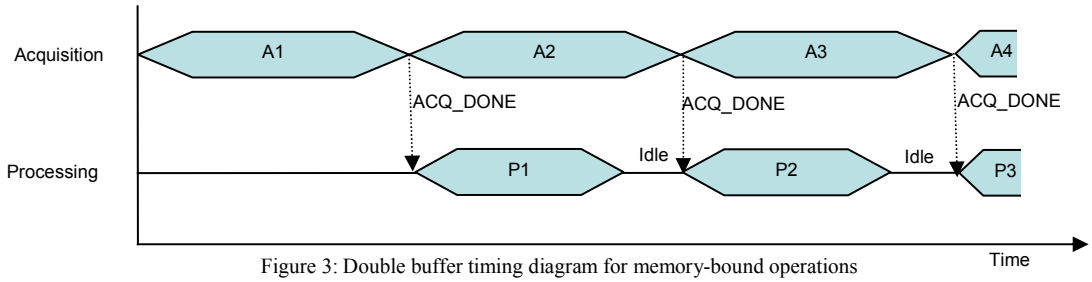


Figure 3: Double buffer timing diagram for memory-bound operations

events: ACQ_DONE (acquisition complete) and PROC_DONE (processor completion).

The main advantage of the single buffer scheme is its low memory requirement due to sharing of single slot frame-buffer shared for both acquisition and processing. This scheme is suitable for storage efficient devices, for example in hardware frame-grabbers, because the cost of this kind of device is directly proportional to the amount of available on-board memory.

The main disadvantage of this method is that while one stage is using the buffer, the other has to wait. A consequence is that the application may drop important observations due to prolonged processing. Another significant problem is with image capture, the acquisition timing is often controlled externally, so delaying the start may result in only part of the frame being captured. If the acquisition pre-empts the processing, only part of the buffer may be processed before the data is over-written. The resulting discontinuities can lead to processing artefacts in the output image.

D. Double buffer scheme

Many of the problems associated with a single buffer may be overcome by double buffering. This scheme uses two distinct buffers (ping-pong buffers) in such a way that each task has exclusive access to its corresponding buffer [6]. While the data is being processed in buffer 1, the next frame is being captured into buffer 2. On the completion of acquisition or processing, the buffers are swapped [7]. An ideal case for such scheme is demonstrated in Fig.2, where acquisition rate is equal to the processing rate. Once each frame has been acquired, an ACQ_DONE signal is generated and used to switch buffers. From the timing diagram, it is apparent that both acquisition and processing are able to work in parallel.

Unfortunately, the situation is often not as ideal as illustrated in Fig. 2. There is often a disparity between the acquisition rate and the processing rate. Generally the timing (hence the buffer switching) is controlled by the acquisition module. This can cause a number of problems.

The simplest case is where the processing takes less time than acquisition. The system is speed is limited by the acquisition speed, and hence the system is “memory-bound” because most of the time is consumed in image movement to memory. The timing for this is shown in Fig. 3. After processing a frame, the processor must sit idle until the next frame is ready.

Conversely, if the image processing takes longer than the acquisition time, the system is termed as “computational-bound”. Several different timing scenarios for such a system are compared in Fig. 4. With computational-bound processes, swapping the buffers before processing is complete can have unexpected consequences, such as severe jitter resulting from discontinuities in the data as described in

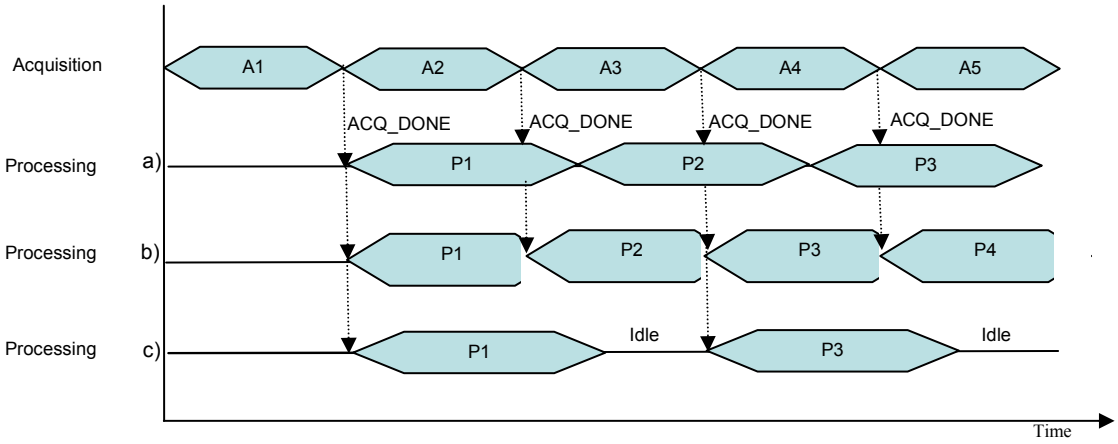
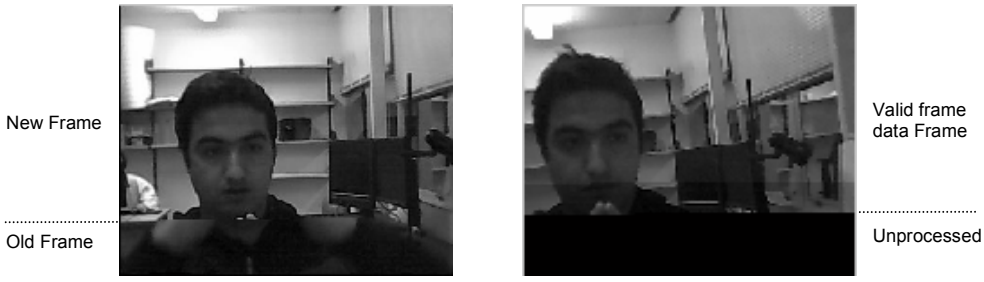


Figure 4: Double buffer timing diagram for computational-bound Operations: (a) Continuing processing even with swapped buffers; (b) Preemptively aborting processing; (c) Delaying buffer swap until processing completes



(a) Overlap (b) Tearing

Figure 5: Output image from simulated double buffer

Fig. 4 (a). Fig. 5 (a) clearly shows the tearing effect due to swapping of buffers while processing is underway. Note that it is not valid to restart the processing when the new frame is acquired, because the bottom of the image would never be processed and any activity in that region can not be registered. Fig. 4 (b) best describes the timing diagram of such scheme and it simulation results are shown in Fig 5 (b) where a gesticulation for New Zealand Sign Language word "House" is formed but all critical information is lost . The usual solution is to prevent swapping the buffers until after the processing is completed. This is usually achieved by overwriting the frame in the acquisition buffer, and swapping at the end of the next frame (Fig. 4(c)). The resulting idle time of the processor makes this a non-optimal solution.

Double buffering is unable to manage the case where both the acquisition and output are controlled by external events, for example in scan-rate conversion. The problem is that either the buffers are swapped in the middle of processing or in the middle of acquisition.

E. Triple buffer scheme

Triple buffering is an efficient and advanced buffer management technique in which a third buffer (WAIT) is introduced to hold the data between acquisition and processing [8]. In this scheme, while data is being processed in one buffer other two can hold incoming data, eliminating the problem of erroneously swapping of partially processed buffer. Hines et al [9] highlighted the use of this scheme to avoid the delay caused by waiting for a new buffer in a retinex based image enhancement system. Weir et al [10] also mentioned the significance of using a triple buffer scheme in an integrated framework of controllers and sensors.

III. SIMULATION AND RESULTS

The working strategy of such a triple buffer scheme can be controlled by one of two models depending in the type of synchronising events. These models, a 3 state model and an 8 state model, are depicted in Fig. 6 and Fig. 7 respectively. In this section, these synchronising events and the working strategy of the triple buffer scheme are described. The results are compared with other techniques through simulation.

A. 3 State Model

A frame-buffer with 3 buffer slots (Acquisition, Wait, and Processing) is shown with its simple state diagram Fig. 6, comprising only 3 states. The system is controlled by two synchronising events: Acquisition complete (ACQ_DONE) and Processing complete (PROC_DONE). Assuming that acquisition starts automatically from the system state UEE where the Acquisition buffer is being used (U) for acquisition and the others are empty (E). When acquisition completes, the buffer is time stamped and the Acquisition and Processing buffers (A and P) are swapped and

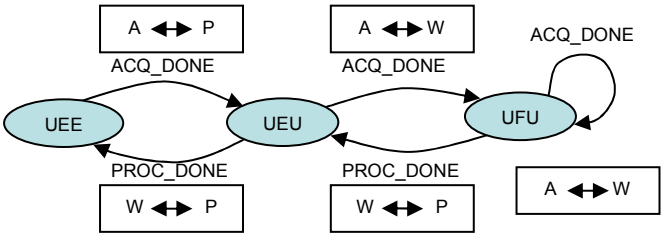


Figure 6: Triple buffer (3 state model)

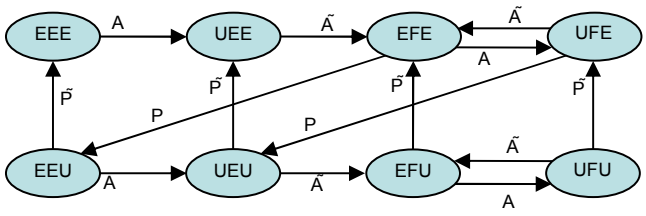


Figure 7: Triple buffer scheme (8 state model)

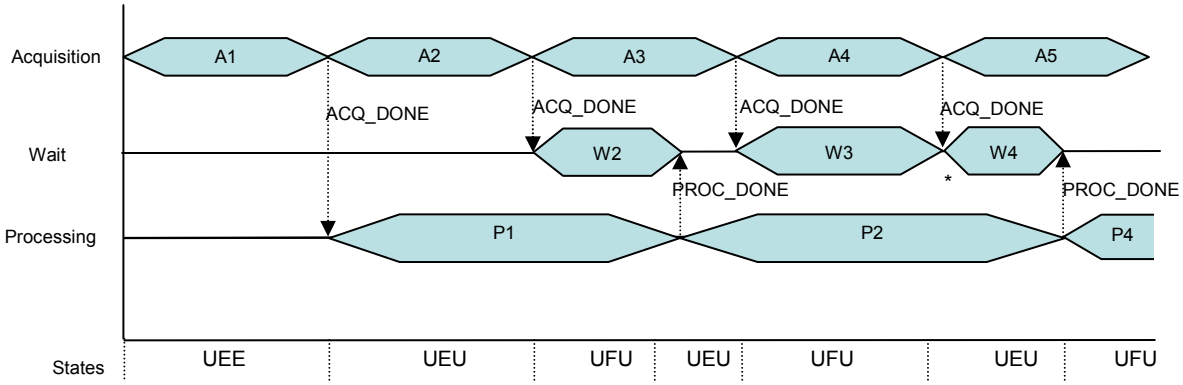


Figure 8: Triple buffer timing diagram for computational-bound operation (* indicates when a stale buffer is dropped)

processing is automatically started on the recently acquired frame. As both Acquisition and Processing are underway, their corresponding buffers are in use, the system's state is now UEU. Another ACQ_DONE event in this state means that while the image is being processed, a new acquisition has been finished. Here due to availability of a third (WAIT) buffer, the recently filled Acquisition buffer can be swapped with the WAIT buffer and acquisition restarted in the empty frame (UFU state).

If a further frame acquisition is completed when in this state, the stale frame (identified by time stamp) in the Wait buffer is automatically dropped by swapping it with the newly acquired frame. When processing is completed in the UFU state, the processing buffer is swapped with the currently waiting buffer, allowing processing to begin immediately on the waiting frame. The Wait buffer is then marked as empty (state UEU). If processing is completed in this state, then there is no waiting frame, so the system transitions back to state UEE, and the processor is idle until the next frame is acquired.

B. 8 State Model

The 3 state mode of operation assumes that the acquisition is continuously free-running, and that processing is started automatically when a frame is available. In many machine vision applications, both the acquisition and processing can be triggered by external events. There are then four events: Start_Acquisition (**A**), Acquisition_Complete (**Ã**), Start_Processing (**P**) and Processing_Complete (**P̃**). The three states of Fig. 6 must be expanded to an eight state system as shown in Fig. 7.

Initially the system begins in the all-empty state (EEE) and can be initiated by an external (Start_Acquisition) event "**A**" which turns system's state into UEE by allocating Acquisition buffer to be used for image acquisition. On Acquisition_Complete event "**Ã**", the recently filled buffer is swapped with WAIT buffer while the Process buffer is still empty (EFE). This filled buffer can then be processed by receiving the triggering event "**P**" which shifts the state to EEU where processing is started after swapping the contents of WAIT and Process buffer. Conversely if another "**A**" occurs in the EFE state, it moves to UFE in which empty Acquisition buffer starts filling. If the WAIT buffer is full when acquisition is complete, then the stale frame is dropped, and replaced with the most recently acquired frame.

C. Simulation

In order to analyse the performance of triple buffer scheme, simulations for both double and triple buffer scheme are modelled in MATLAB®. Images from a USB webcam are acquired at different frame rates (15, 25, 30 fps) while the processing frame rate is set constant and the swapping of the buffers is triggered accordingly i.e. in double buffering, they are swapped on the occurrence of new frame and in triple buffering incoming frame is kept in a WAIT buffer. Theoretically, the timing diagram of Fig. 8 states that if both acquisition and processing is being performed on their designated buffers, we still have another dedicated WAIT buffer to hold a frame. Pictorial results obtained by Matlab simulation are compared in Fig. 9, which highlights the

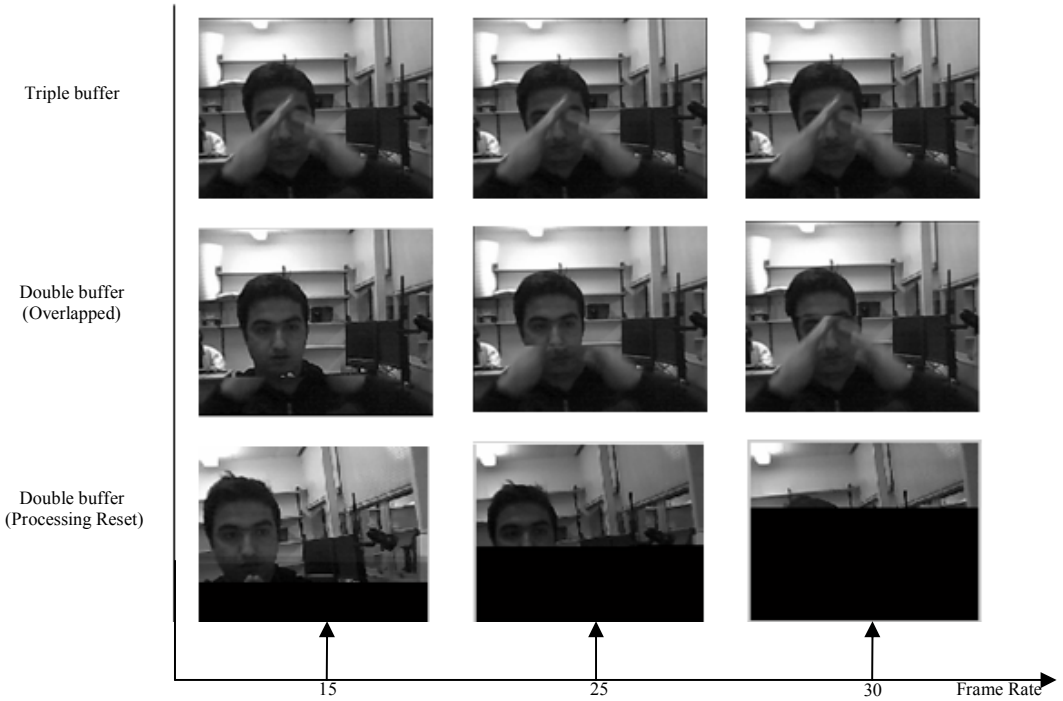


Figure 9: Simulation results comparison

effectiveness of triple buffer scheme against the mentioned problems of double buffering.

IV. CONCLUSIONS

We briefly discussed different buffer management schemes and implemented MATLAB simulations of double and triple buffering methods as per their timing diagrams. By analysing the simulation results, the output of the double buffering scheme, resulted as severely distorted by frames overlap and tearing at higher frequencies which is not suitable for real-time vision applications but the output images from the triple buffer scheme were free of such artefacts.

REFERENCES

[1] E. R. Davies, *Machine Vision Theory, Algorithms, Practicalities*, Elsevier, 2005.

[2] W. K. For, K. Leman, H. L. Eng, B. F. Chew, and K. W. Wan, "A multi-camera collaboration framework for real-time vehicle detection and license plate recognition on highways," *IEEE Intelligent Vehicles Symposium* Eindhoven, The Netherlands, 4-6 June 2008 pp. 192-197.

[3] J. B. Hayet, T. Mathes, J. Czyz, J. Piater, J. Verly, and B. Macq, "A modular multi-camera framework for team sports tracking," *IEEE Conference on Advanced Video and Signal Based Surveillance(AVSS) 2005* Como, Italy, 15-16 Sept 2005 pp. 493-498.

[4] A. Rowe, C. Rosenberg, and I. Nourbakhsh, "A Low Cost Embedded Color Vision System," *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Vol 1, EPFL, Switzerland, 30 Sept- 5 Oct 2002, pp. 208-213.

[5] D. L. Knierim, Image frame buffer access speedup by providing multiple buffer controllers, US Patent, 5109520, 1992.

[6] W. E. Wright, "Single Versus Double Buffering in Constrained Merging," *The Computer Journal*, vol. 25, no. 2, 1982, pp. 227-230.

[7] R. B. Sheeparamatti, B. G. Sheeparamatti, M. Bharamagoudar, and N. Ambali, "Simulink Model for Double Buffering," *32nd Annual IEEE Conference on Industrial Electronics (IECON)*, Paris France, 6-10 Nov 2006, pp. 4593-4597.

[8] L. Lumelsky, Triple field buffer for television image storage and visualization on raster, US Patent, 5291275, 1990.

[9] G. Hines, Z.-u. Rahman, D. Jobson, and G. Woodell, "DSP Implementation of the Retinex Image Enhancement Algorithm," *Visual Information Processing XIII* Vol SPIE 5438, Orlando, Florida USA, 15-16 April 2004 pp. 13-24.

[10] R. D. Weir, G. Sen Gupta, and D. G. Bailey, "Implementation of a framework to integrate sensors and controllers," *International Journal of Intelligent Systems Technologies & Applications*, vol. 3, 2007, pp. 4 -19.