Real-time exposure fusion on a mobile computer

Asheer Kasar Bachoo

Signal Processing Research Group
Optronics Sensor Systems
Council for Scientific and Industrial Research (CSIR)
Pretoria, South Africa
abachoo@csir.co.za

Abstract

Within the context of video surveillance, object and incident detection by an operator is minimized in over- and under-exposed pixels. Multiple different exposures of the same scene can be fused to create a well-balanced and detailed image that is meaningful to a human operator or surveillance system. We present an implementation for real-time video enhancement using videos of varying exposures. The algorithm increases the details in over- or under-exposed areas that are present in a single frame of fixed exposure. The processing power of a mobile computer and its graphics processing unit (GPU) are able to fuse three greyscale videos of resolution 1600×1200 at 20 frames-per-second.

1. Introduction

Surveillance systems make extensive use of video cameras for the monitoring of people, objects and high risk areas. Environments monitored by the military and navy are generally composed of harsh lighting conditions that introduce a large number of shadows and highlights into a captured frame. These great variations in scene radiance tend to over-expose or under-expose certain areas in a captured image. This is due to the camera sensor's dynamic range being much lower than that of the scene. Advanced digital image processing will not reveal any hidden details since the camera's sensor system would have clipped the pixel digital values in the over- and under-exposed regions. The sequential capture of multiple images of the same scene using different exposures is able to sufficiently capture the required information in these scenarios. An image captured using a short exposure time will not saturate bright image regions while an image captured with a long exposure time will show more detail in the dark regions.

The pixel depth provided by most camera sensors range from 8-bit to 14-bit i.e. 256 to 16384 digital values, while the range of radiance values in a natural scene, from a digital perspective, are several orders larger than that. Details present in high dynamic range scenes can be captured through the use of the following two techniques:

- **High Dynamic Range (HDR) imaging:** HDR imaging methods allow a larger dynamic range of luminance values that traditional imaging techniques do not offer [1, 2]. These techniques aim to closely model the intensities in the real world, such as strong direct light sources. An HDR image will contain the full range of luminances found in a scene. Tone mapping techniques are used to render the HDR data to low dynamic range hardware such as screens or common data files such as JPEG images. HDR images are created using low dynamic range data.
- **Exposure fusion:** Exposure fusion selects the best regions from each frame in a sequence of different exposures of the same scene and seamlessly blends the regions together to created a fused image [3, 4]. Regions are selected based on user defined criteria such as maximum entropy, high contrast and saturation. Unlike HDR imaging, tone mapping for the final rendered image is not required since the output is the same bit depth as the input. Exposure fusion also produces more natural looking images and can create a large depth of field by fusing together frames with different degrees of sharpness.

In this paper, we discuss real-time exposure fusion for creating detailed video output that can assist with surveillance. The proposed implementation is a new contribution to the field of real-time image enhancement. In addition, the hardware and software platform is a mobile computer that can be deployed quite easily into the field for testing. We begin by discussing the background material and thereafter describe the algorithm and implementation in detail. This is followed by the experimental results and discussion. Concluding remarks are then presented.

2. Background

Goshtasby proposes an algorithm for exposure fusion that increases the final image entropy [3]. The method proceeds by splitting the image up into blocks and, for each block, the exposure that has the highest entropy is selected. The selected blocks of varying exposures are thereafter blended together to create the final scene. It is an iterative algorithm that computes the entropy of a fused image for different block sizes. The fused image with the highest entropy is the final result. Goshtasby uses rational Gaussian surfaces for image blending. Experimental results are presented for still images.

Mertens *et al.* fuse multiple exposures of a scene through pyramid blending [4]. In contrast to Goshtasby's method, they perform exposure selection at a pixel level (rather than block level). Three measures are proposed for pixel selection: i) contrast, ii) saturation and iii) well-exposedness. These measures are used to generate a scalar weight map that guides the fusion process. A Gaussian pyramid of the weight map and a Laplacian pyramid for each exposure are then used to perform the final image blending. Similar to Goshtasby's experimental setup, results are presented for still images.

Real-time video enhancement can be achieved using different types of hardware [5]. A Field Programmable Gate Array (FPGA) is a device with a large number of logic gates.

It is widely used in digital signal processing (DSP) for real-time applications. FPGAs can be reconfigured in real-time to form different circuits. This provides them with the ability to have a wide range of applications with precise outputs. Custom circuit and memory configurations can be used to exploit the data layout and the algorithm for high performance. Specialized DSP chips also exist that process data captured by cameras and video recorders. They have dedicated low level accelerators and higher level general purpose processors for more complex tasks. Common tasks include LCD display, auto white balance and auto focus on cameras.

Desktop computers are used by most scientists and researchers due to their low cost and the large number of programming languages and software available for DSP. The central processing unit (CPU) of desktop computers currently have multiple cores for high performance computing. Recent advances include 64-bit platforms, up to 8 megabytes of level 2 cache memory and Single Instruction, Multiple Data (SIMD) operations in the instruction sets. A disadvantage of the desktop processor is its large power consumption.

The graphics processing unit (GPU) is a common hardware component in desktop and laptop computers. It is used primarily for rendering graphics in modern computer games. In recent years, the rendering pipeline of the GPU has been made accessible to programmers through the introduction of GPU programming languages, making it capable of doing general purpose computing for a wide range of scientific applications [6]. It employs the SIMD technique to achieve data and computing parallelism. This technique provides huge performance gains in certain parallel computing problems.

In the next section, we present a detailed description of the fusion algorithm that uses the GPU and CPU on a laptop computer to achieve real-time processing. This is a low cost solution for real-time processing that evades the substantial development time required by specialized solutions such as FPGAs. In addition, a variety of applications can be executed on a laptop making it a highly flexible analysis tool.

3. Algorithm Details

A variation of Goshtasby's algorithm is used for exposure fusion. Our algorithm selects the image block with the highest entropy and performs blending between blocks using bilinear interpolation [7]. The block size is the only input parameter required and is user defined. The processing frame rate achieved implies that the block size can be adjusted by the operator during operation without any impact on performance. The exposure fusion algorithm can be described as follows:

1. A block size $w \times w$ is specified for the computation of entropy measures. The Shannon entropy measure for information can be found in the literature [8].
2. Given a sequence of n frames captured with different exposures e_k where $k \in \{0, \ldots, n-1\}$, each frame f_k, starting at the top left hand corner, is tiled into blocks of size $w \times w$. Blocks at the left and bottom edges may be smaller in size than $w \times w$. We will denote a block as $b_{k_{ij}}$ where ij, the center of a block, is a particular block location in the frame. Figure 1 shows blocks at locations ij in an image.
3. For each frame f_k, corresponding to an exposure e_k, each block $b_{k_{ij}}$ is selected and its entropy is computed.
4. For each block location ij, we now have n entropy measures. The block $b_{k_{ij}}$ with the highest entropy is selected for fusion into the final image at location ij.
5. All the blocks selected for each block location ij (in the above step) are blended together, using bilinear interpolation, for the final image.

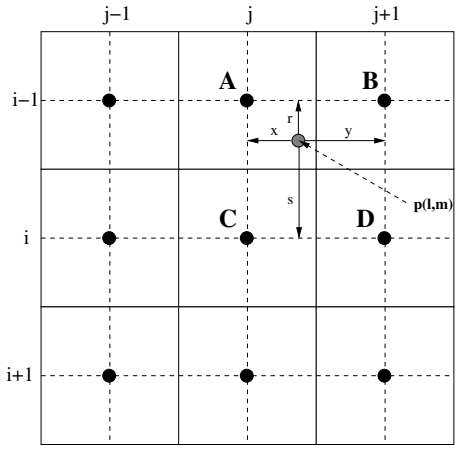


Figure 1: *Bilinear interpolation for pixel blending.*

Image blending at an interior pixel location (l, m) - a pixel location surrounded by four blocks - is achieved by using the pixel values at location (l, m) in the frames having the exposures of the surrounding blocks. In Figure 1, these four blocks correspond to the ones with their centers labelled A, B, C and D. Let A, B, C and D denote a particular exposure index(k) for a block. Then the mapping for an interior pixel is:

$$p(l,m) = \frac{s}{r+s}\left(\frac{y}{x+y}f_A(l,m) + \frac{x}{x+y}f_B(l,m)\right) + \frac{r}{r+s}\left(\frac{y}{x+y}f_C(l,m) + \frac{x}{x+y}f_D(l,m)\right)$$

where $f_k(l,m)$, $k \in \{A, B, C, D\}$, is a pixel value at location (l, m) in an acquired frame having exposure k. Figure 1 shows how the parameters r, s, x and y are computed. The weighting is derived from the horizontal or vertical distance to the block centers in the image. Edge pixels with only two blocks in their proximity are mapped using a linear combination of the two image functions. Pixels at the corner of the image have only a single mapping function i.e. the image function of the closest block. In the next section, the real-time implementation of the algorithm is presented.

4. Implementation for the CPU and GPU

The implementation of the exposure fusion algorithm utilizes the processing power of the CPU and GPU. Computing hardware are designed to have particular types of operation. For example, CPUs are good for sequential operations while parallel operations are suitable for the GPUs. Although GPUs have a fixed functionality shader programs can be executed in certain stages of the pipeline. We utilize the fragment shader pipeline to process image pixels. More information regarding GPU programming can be found in the literature [9].

Certain image processing algorithms cannot be ported to the GPU without reducing overall performance or drastically increasing complexity of the program. This constraint arises from the fact that pixels are processed in a random order and

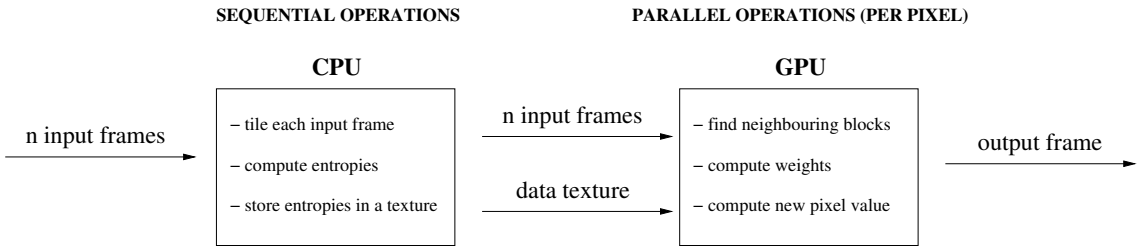


Figure 2: *Flow of execution for the exposure fusion algorithm.*

independently of each other on the GPU. The output at a particular pixel location cannot utilize outputs at other pixel locations. Hence, it will be more costly to system performance if certain algorithms executed only the GPU. Algorithms must be structured such that the full power of the GPU and CPU are utilized. We structure the flow of execution as follows (shown in Figure 2):

1. n input frames representing the different exposures for a scene are first transferred to the CPU for sequential processing. At this stage, the entropies for each block in the n frames are computed. The entropy data, together with block information such as center locations, is stored in a floating point texture.
2. Once the CPU processing has been completed, the n input frames and the texture storing the entropy data is transferred to the GPU. The GPU then executes a shader program that performs operations on pixels in parallel - neigbouring block locations are computed and the blending is performed. The data required for each pixel is extracted from the textures storing the entropy information and the input data. The new pixel values are then stored in an output frame and rendered to screen.

When using the GPU for general purpose computing, copying of image data between the CPU and GPU consumes a large number of clock cycles when using texture fetches. This problem can be averted by using the pixel buffer object (PBO) provided by the OpenGL driver [10]. OpenGL is a low level graphics rendering library that provides an interface to the GPU. The PBO provides regions of GPU memory that are directly accessible through identifiers. It achieves fast data transfer across the CPU/GPU bus by using direct memory access (DMA). Our initial tests showed an increase in processing frame rate by a factor of 6.

The texture that is used to store the entropy data is one with format RGBA (4 channels i.e. Red Green Blue Alpha) and data type 32-bit single precision floating point. It has a constant size and this ensures that the DMA data transfer will have a fixed execution time. The amount of data stored in the texture is dependent on the block size for processing; smaller blocks produce more data than larger ones. In our implementation, a minimum block size of 32×32 is possible for video frame sizes of 1600×1200 or less. Each row in the data texture corresponds to a particular image block and contains 3 columns; each column is used to store data for a particular exposure. The 4 channels provided by the RGBA format ensure sufficient memory for image block data. The pixel format (RGBA) was not changed to 1 or 2 channels for lower memory bandwidth consumption since it was felt that future work on the algorithm would require additional image information.

The CPU and GPU code was optimized by reducing the number of arithmetic and boolean operations. For example, replacing certain division operations by multiplications (which execute much faster than division operations). Other operations were reduced by pre-computing frequently used variables and by using built-in hardware functions. The most computationally intensive portion of the algorithm is executed by the GPU i.e. the pixel blending. The FPGA implementation of bilinear blending is able to exploit particular properties of the algorithm and the data layout [11]. Hence, the number of clock cycles required by the processor are significantly lower than a GPU. This is not possible on the GPU due to the random order of pixel processing. As a result, a large number of if-then statements are present in the GPU shader code. This is essential for processing particular image blocks e.g., interior or corner blocks. Future optimizations will consider creating different shaders (pipelines) for particular image blocks and pixel locations. In this case, the execution speedup will be significant.

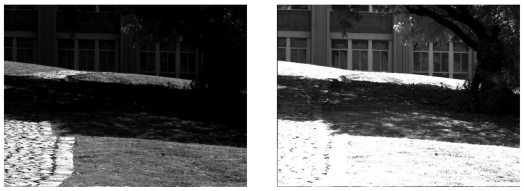
5. Experimental Results and Discussion

The exposure fusion algorithm was tested on a Dell Latitude D830 Laptop. The hardware and software specifications of the laptop were:

- Intel Core2 Duo 2.2 GHz processor.
- 4GB memory.
- Nvidia Quadro 140M NVS graphics card.
- Linux operating system (Gentoo).
- OpenSceneGraph for GPU processing using C++ and OpenGL Shading Language.

The Nvidia Quadro 140M provides entry level computing performance on a mobile platform. It has 16 shaders (pipelines), a core speed of 400 MHz, memory speed of 700MHz and operates with 10W power consumption. The memory capacity is 256MB with a bus width of 64-bits. We fuse together three videos recorded using a Prosilica GE1660 video camera with resolution 1600×1200 and 8-bit pixel depth. 10 scenes were captured for testing the fusion algorithm. An example of an HDR scene with multiple exposures and the final fusion are shown in Figure 3(a) - 3(d). The 10 test scenes are shown in Figure 4.

In Figure 3, a camouflaged dummy that is placed next to the tree is not clearly visible using the auto exposure function provided by the camera. The auto exposure also creates strong blown-out highlights in the foreground (the grass patch). The short shutter time (Exposure 1) correctly exposes the grass while the long shutter time (Exposure 3) is able to correctly expose the camouflaged dummy next to the tree. All of these details are successively fused into the final image (Figure 3(d)).



(a) *Exposure 1.* (b) *Exposure 2 (auto exposure).*

(c) *Exposure 3.* (d) *Fused frame.*

Figure 3: *A typical scene with a high dynamic range.*

The normalized entropy of the input scenes and the fused scenes are presented in Table 1. The entropy for the input scene is computed using the second of the three exposures, which is generally the auto-exposure provided by the camera. Block sizes were selected based on human visual inspection of the output provided by the algorithm. In most of the test scenes, the global image entropy is increased by the algorithm. However, fused scenes with a slight decrease in entropy do show an increase in image detail on closer inspection. Some fused results are shown in Figure 5. The performance of the algorithm, in frames-per-second (fps), is shown in Table 2. The captured video scenes were scaled for the non-native video resolutions. Videos of resolution 1360×1024 refer to the frame size provided by older Prosilica cameras. The processing frame rate tends to increase monotonically as block size increases. Across all mentioned video resolutions, a minimum frame rate of 20 is achievable when using a minimum block size of 64×64. The algorithm execution time is independent of the scene content. This means that the processing times for different scenes of the same resolution are almost the same.

Table 1: *Entropy of original image and fused image.*

Image	Block Size	Original Entropy	Fused Image Entropy
BIKE	196	0.64	0.67
CARPARK	128	0.51	0.66
SHADOW	128	0.56	0.66
INTERIOR	196	0.65	0.67
MIDDAY	128	0.65	0.64
HIDDEN	128	0.51	0.64
INDOOR1	128	0.60	0.66
INDOOR2	128	0.65	0.67
VAN	128	0.50	0.64
TREE	128	0.64	0.66

Table 2: *Exposure fusion performance (in frames-per-second).*

Video Size	Block Size			
	32	64	128	256
256×256	195.37	196.56	198.23	na
512×512	101.75	103.55	103.83	104.61
1024×1024	28.14	34.09	35.84	35.95
1360×1024	19.75	26.59	27.87	27.65
1600×1200	19.92	20.21	20.25	20.31

The algorithm presented will maintain real-time execution when the cumalative exposure times are less than the capture time required for a particular frame rate. Day time scenes with sufficient lighting easily satisfy these requirements. However, there are limitations when lighting is extremely poor. In these cases, low light, thermal or infrared sensors will provide more information than sensors for the visible spectrum. Another important consideration is the selection of the different exposure times. The exposures must be selected so that all or most of the important scene details are captured. Currently, we use 3 exposures which may be insufficient. Inclusion of more exposures will introduce a bottleneck to the fusion process but will also provide better pixel details.

A practical problem associated with the described algorithm is the acquisition process for multiple exposures. This can be achieved by using multiple video cameras that have been calibrated. The calibration information can be used to associate acquired pixel values with points in space and a cropped frame of interest can be generated [12]. In this way, each pixel location will now have multiple digital values corresponding to the different exposures used. Alternatively, there are digital video cameras available that allow a user to set a different exposure for every frame acquisition in real-time. Emerging technologies also show that new sensors being developed can provide separate readouts (multiple exposures) for each pixel for a single acquisition [2].

The quality of the fused image is dependent on a number of factors and there is a trade-off between performance and quality. Blending artifacts can be seen in Figure 5(f). Small block sizes, for entropy computation, can produce finer details in the fused image. However, sharp grey scale transitions between edge blocks are not blended smoothly and this presents a low quality fused image. Large block sizes enable the best blending but local details are sometimes lost. These problems can be addressed by using a multiscale feature extraction and image fusion/blending process or adaptive window size. A learning process may also be incorporated every few frames to provide information for an automatic block size selection process. However, the methods mentioned above may not satisfy the requirements for real-time execution.

Future work will consider looking at optimization techniques on multi-core CPUs and improved shader code. More complex image partitioning, automatic block size selection and multiscale blending of image regions will also be considered. New measures for exposure selection will also be examined.

6. Conclusion

In this paper, we have presented a real-time algorithm for exposure fusion. Only a single input parameter is required, namely the block size for entropy computation and blending. The test results show that the algorithm increases the overall entropy in

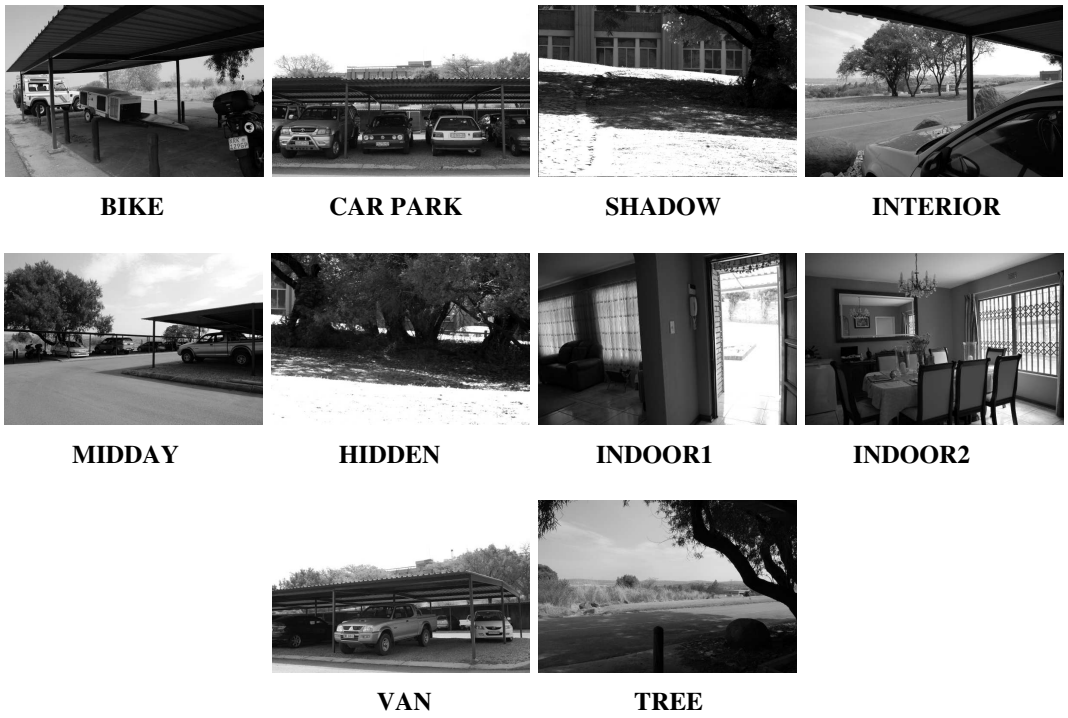


Figure 4: *The 10 scenes used in the experiments.*

the final fused image and restores lost image details in over- and under-exposed regions. The high processing frame rate achieved for large video resolutions, using a laptop computer, introduces new possibilities in terms of mobility and testing in the real-world.

7. References

[1] P. E. Debevec and J. Malik, "Recovering high dynamic range radiance maps from photographs," in *SIGGRAPH 97, Computer Graphics Proceedings, Annual Conference Series*, 1997, pp. 369–378.

[2] S. Nayar and T. Mitsunaga, "High dynamic range imaging: Spatially varying pixel exposure," vol. 1, 2000, pp. 472–479.

[3] A. Goshtasby, "Fusion of multi-exposure images," *Image and Vision Computing*, vol. 23, pp. 611–618, 2005.

[4] T. Mertens, J. Kautz, and F. van Reeth, "Exposure fusion," in *Pacific Graphics*, 2007.

[5] N. Kehtarnavaz and M. Gamadia, *Real-Time Image and Video Processing: From Research to Reality*. Morgan and Claypool, 2006.

[6] J. Owens, M. Houston, D. Luebke, S. Green, J. Stone, and J. Philips, "GPU Computing," *Proceedings of the IEEE*, vol. 96, no. 5, pp. 879–899, 2008.

[7] S. Pizer, E. Amburn, J. Austin, R. Cromartie, A. Geselowitz, T. Greer, B. ter Haar Romeny, J. Zimmerman, and K. Zuiderveld, "Adaptive histogram equalization and its variations," *Computer Vision, Graphics and Image Processing*, vol. 39, pp. 355–368, 1987.

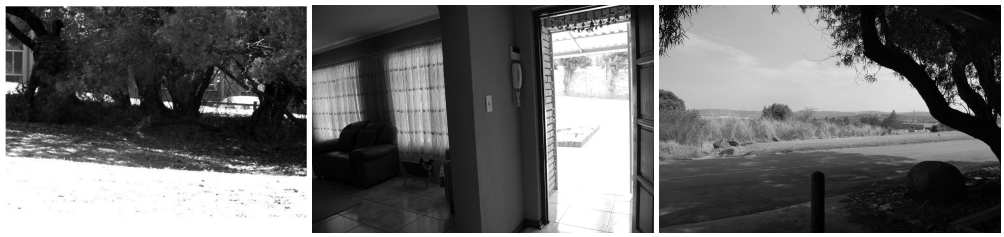
[8] R. Gonzalez and R. Woods, *Digital image processing*. Addison-Wesley Publishing Company, 2002.

[9] R. Rost, *OpenGL(R) Shading Language (2nd Edition)*. Addison-Wesley Professional, 2006.

[10] D. Shreiner, M. Woo, J. Neider, and T. Davis, *OpenGL Programming Guide*. Addison-Wesley Professional, 2007.

[11] A. Reza, "Realization of the Contrast Limited Adaptive Histogram Equalization (CLAHE) for real-time image enhancement," *Journal of VLSI Signal Processing*, vol. 38, pp. 35–44, 2004.

[12] J. de Villiers, "Real-time stitching of high resolution video on COTS hardware," in *Proceedings of the 2009 International Symposium on Optomechatronic Technologies*, ser. ISOT2009, vol. 9, 2009, pp. 46–51.



(a) *HIDDEN.* (b) *INDOOR1.* (c) *TREE.*

(d) *HIDDEN fused.* (e) *INDOOR1 fused.* (f) *TREE fused.*

Figure 5: *Fusion results.*