Reducing Processing Time for a LCD Motion Blur Reduction Algorithm

Table of Contents

Table of Figures

Abstract

This paper presents a solution to implement a LCD screen motion blur reduction algorithm on a real-time system by vastly decreasing the processing time. LCD screens inherit motion blur due to their sample-and-hold nature. One way to reduce the motion blur is to pre-process an image using deconvolution to invert the LCD's sample-and-hold nature and the Human Visual System so that the perceived image is sharp. A filter bank approach comes close to the inversion. This algorithm is implemented in C/C++ utilizing OpenCV making it up to 80x faster than processing in Matlab.

Introduction

Liquid Crystal Displays (LCD) are popular consumer devices due to their low cost, low power output, high contrast, and thin profile. They are used in multiple applications such as computer monitors, laptop screens, televisions., and mobile devices. Also, they are increasingly popular for hospital and military applications. Unfortunately, LCD screens suffer from motion blur due to their sample-and-hold phenomenon.

This paper describes a process to reduce the processing time for an LCD motion blur reduction algorithm. This phenomenon is described in the background section along with multiple solutions to the problem. Next, the signal processing approach is described. Third, this paper explains using C/C++ with OpenCV instead of Matlab, which led to a vast decrease in processing time in order to implement the system in real-time. Finally, conclusions about the effects of a low processing time and future work are discussed.

Background

On an LCD screen, an frame is displayed, and it stays on the screen until the next frame replaces it, usually at 60 Hz. Cathode-Ray Tube (CRT) screens, on the other hand, flash an image on the screen for a couple milliseconds, and then the screen turns black. CRT screens project images as impulse functions while LCD screens use step functions, as shown in Figure 1.

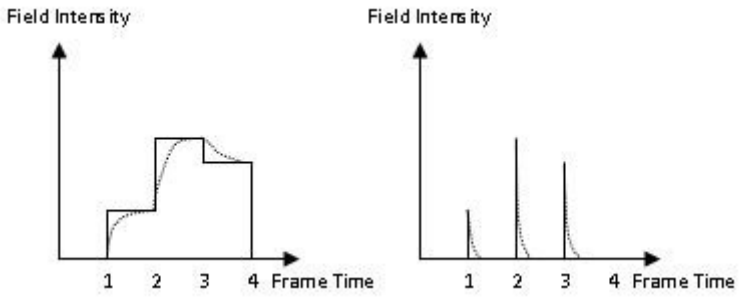

Figure 1: LCD and CRT outputs assuming 0 ms response time.

The LCD's sample-and-hold nature creates motion blur. Images from both screens seem identical when the image is still or there is little motion, but the LCD screen is perceived as more blurry when motion is present. As shown in Fig. 2, the viewer expects to see continuous motion throughout the scene while the LCD image output is in discrete values; thus, motion blur is due to the difference between expected and actual motion.

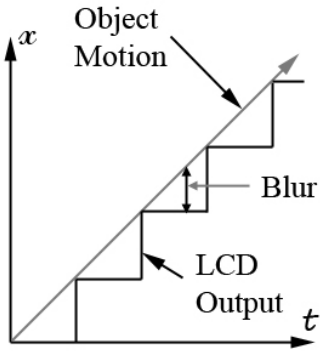

Figure 2: Typical LCD Output assuming 0 ms response time.

Although LCD's output image is not actually blurry, it is perceived as one when it passes through the human visual system (HVS) [1].The HVS is an attempt to model human nature. It is modeled as a low pass filter and a motion tracker, as shown in Figure 3.

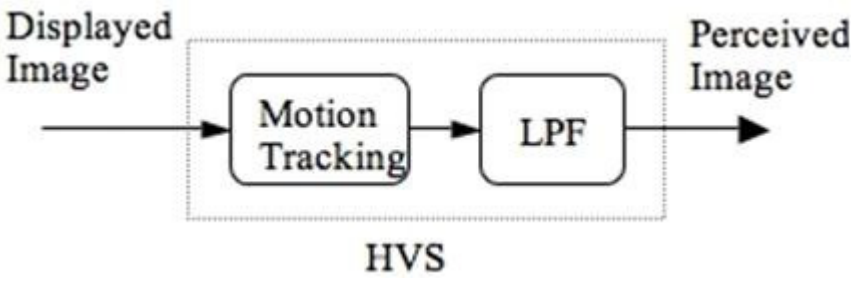

Figure 3: Human Visual System

Since humans notice fast moving objects more than slow ones, the HVS includes a motion tracker. These fast moving objects appear blurry while in motion to create a low pass filter. For instance, one can look left and right very quickly to blur the environment. Since the goal for the motion blur reduction algorithm is for humans to perceive a clear, sharp image, the HVS is an important filter to the process.

Previous Attempts

A signal processing approach is certainly not the only way to reduce LCD motion blur. Previous attempts include a flashing backlight, inserting a black frame, or interpolating a frame [2]. CRT TVs do not create motion blur since they do not have the sample-and-hold nature, engineers have mimicked the CRT response on the LCD screen in an attempt to combine the positive characteristics of each system.

One way to mimic the CRT screens is to duplicate the impulse response from each image. To do this, the LCD screen can flash the blacklight whenever a new image is displayed [3]. This way, the viewer will likely notice the new image instead of paying attention to the image throughout the whole frame. This method has reduced motion blur, but the screen quality is not as clear.

Another method to mimic a CRT screen is to insert a black frame in between each frame on a 120Hz LCD, since this screen can display twice as many images than a normal LCD screen [4]. Inserting a black frame has reduced motion blur, but it has increased viewer eye strain. Another attempt using 120Hz LCD is to interpolate a frame [5]. This principle is shown in Figure 4:

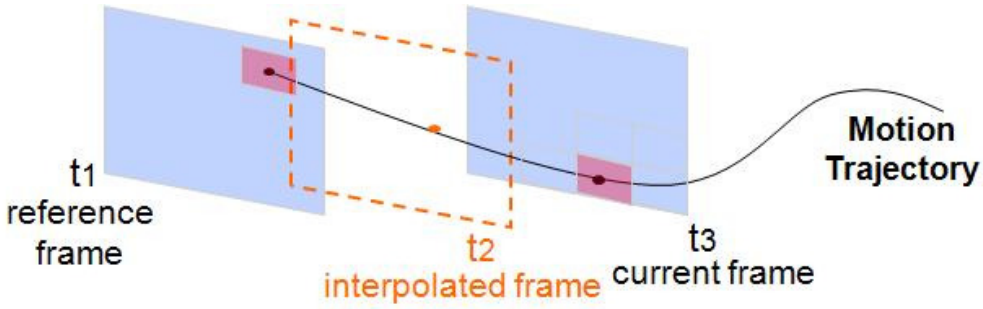

Figure 4: Interpolated frame on 120 Hz LCD screen

An interpolated frame guesses an objects motion based on past and/or future information. If implemented correctly, motion will appear smoother than normal LCD screen response since each image is held on the screen half the amount. Like the other attempts, this method reduces the motion blur, but the blur will not be completely removed since the LCD exhibits the sample-and-hold nature, and these attempts do not account for the HVS.

Signal Processing Approach

The goal for the signal processing approach is to invert LCD response and human visual system (HVS) so that the resulting image appears sharp [6]. The ideal approach is shown below in Figure 5.

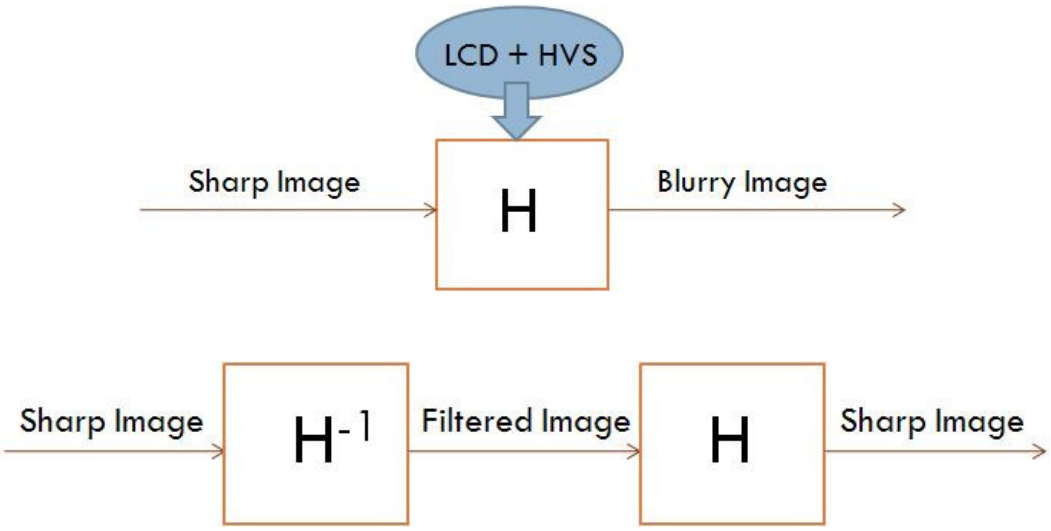

Figure 5: Ideal Conditions for Signal Processing Approach

The LCD plus HVS can be grouped together as filter *H*. Ideally, the system would invert *H* to form H^{-1}, and the resulting image from the LCD screen would appear to be sharp. This way, the viewer will perceive crisp, clear images on a LCD screen. Since the original blurry output image is a convolution of the images and *H* filter, the inverted filter H^{-1} will have to be a deconvolution of the original filter *H*. Unfortunately, this ideal case cannot be implemented in a system due to the LCD and HVS models.

These LCD and HVS models are modeled as sinc functions. Since a sinc functions have zeros present in the frequency domain, it is non-invertible. Therefore, the ideal case is unattainable. In response, this signal processing approach tries to come as close to the inverted case as possible. A filter bank approach decreases the computation time, and it provides better results than other deconvolution techniques such as the Richardson-Lucy Algorithm [7].

Filter Banks

Filter banks process different frequencies separately and then recombine the processed signals together. The filter bank approach is shown in Figure 6.

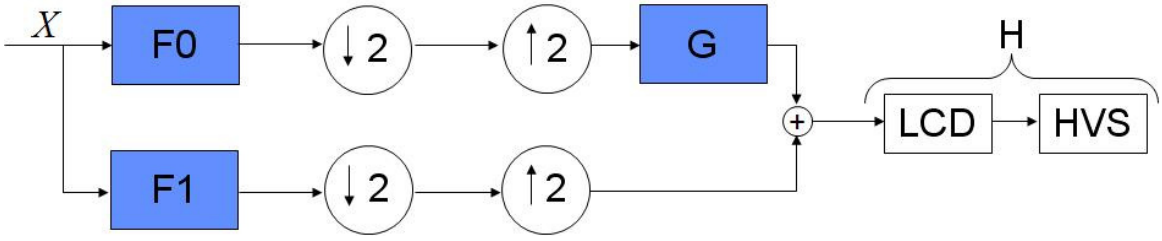

Figure 6: Filter Bank Approach

The filter *F0* is a high pass filter, and *F1* is a low pass filter. In order to achieve perfect reconstruction, the following conditions must be met.

- F0(z) = -H(-z)
- F1(z) = G(-z)H(-z)
- P0(z) = -H(-z)H(z)G(z) must be halfband

Halfband filters have alternating zeros when converted to the spatial domain, and the stopband and passband ripples are the same. In theory, this should achieve ideal results with an accurate filter *H*. Unfortunately, if G(z) is a finite impulse response filter, perfect reconstruction is not possible for the arbitrary filter H(z) since the P0(z) filter becomes an infinite impulse response filter. To get close to the ideal conditions, the filter coefficients for *G* are determined through convex optimization.

Convex optimization minimizes the error of the filter to make the filter more ideal. Convex optimization algorithms are computationally expensive, especially when the size of *G* increases for a better filter. Since the filter coefficients depend on the motion vector,

values for each motion vector length can be stored offline in a table. This helps decrease the processing time since the program reads coefficients from a table instead of compute convex optimization equations every frame.

The filter banks account for the LCD's sample-and-hold nature and the low pass filter part of the HVS. To account for the motion tracker, this deblurring algorithm includes a scaled gradient magnitude equation. Humans notice areas of high contrast when a scene is in motion, so the SGM equation applies the inversion to these areas on a weighted scale. Also, smooth areas do not exhibit as much motion blur as high contrast areas. In addition, these smooth regions usually have poor motion estimation vectors, so the inversion may output artifacts. The SGM equation limits artifacts and improves performance.

Signal Processing Algorithm Results

The following pictures are from a scene displaying Stockholm, England, as shown in Figure 7. The camera pans from left to right fairly quickly. The first image is one frame from the scene. The second image is simulated motion blur, which is the typical LCD output without any processing. The third image is the inverted scene. This image is displayed on the LCD screen after processing. The fourth image is the perceived image after passing through the LCD output and HVS.

Figure 7: Stockholm scene before and after processing. Top-Left: Original Image. Top-Right: Simulated Motion Blur. Bottom-Left: Preprocessed Image. Bottom-Right: Simulated Perceived Image

Although the final output image is not as good as the original image, it is sharper than the output image without any processing. The SGM equation is apparent in areas with high contrast when compared to smooth regions. For instance, the truck has been inverted more than the low contrast building right above it, and humans are more likely to notice the truck than smooth regions like the sky.

Processing a Frame using C/C++

Matlab is useful for processing images because it has all the toolboxes and libraries for implementing this signal processing approach. In addition, it is user friendly and easy to debug when compared to other languages since it is a high level language with multiple functions available. Utilizing the convex optimization toolbos, it is used to compute the inverting filter coefficients for storage offline. Unfortunately, it is slow for processing images.

Using Matlab, processing one frame takes about 12 seconds. On a typical video scene with 60 frames per second, a minute of video takes 12 hours to process. This is way too long, especially when this deblurring algorithm is meant to be used on a real time system. In order to decrease the processing time, the algorithm is implemented in C/C++ with OpenCV.

OpenCV is a computer vision library created by Intel for use with C/C++. It has many functions for manipulating and displaying images. Since OpenCV was created by Intel, it takes advantage of the Intel chip designs to dramatically decrease the processing time through parallel processing. Unfortunately, C/C++ programming is not as user friendly as Matlab.

Matlab functions were replicated in C/C++. This way, they can be optimized to reduce the amount of computations and take advantage of OpenCV's quick matrix operations. When referencing memory in these functions, debugging and memory errors become much more frequent.

Results

Processing an image using C/C++ utilizing OpenCV is 80 times faster than processing the same image using Matlab. This was achieved through reading and writing directly to memory, processing filter coefficients before the input image stream, reading coefficients from a table, and reducing unnecessary computations. This program takes advantage of C/C++ with OpenCV by implementing the algorithm in a low level language.

Processing a frame using OpenCV takes approximately 0.15 seconds, and a minute of video at 60 Hz takes about 9 minutes. This result is from using an inverting filter length of 16 coefficients, which gives good results. Inverting filter lengths of 64 and 128 give better results, but their processing times are 2-3 times longer using OpenCV. About half the time is for convolving the entire 480x640 image with the inverting filter, and the other half is for applying the scaled gradient magnitude equation.

This program uses pointers instead of loops to convolve the image. This saves time because the processor does not have to read a value from memory to a variable, perform computations with those variables, and then reassign the variable's value back to memory. Instead, the processor simply applies OpenCV's quick matrix cross product calculation at one memory spot and then it moves on to the next memory spot. This decreases time tremendously, but it is much harder to debug since output values are locations in memory for a 2-D array instead of variable values in debug mode.

This program first calculates the filter coefficients so that it does not recalculate the same coefficients for each frame. For example, allocating memory for variables on the first frame may take one second, but the program does not have to do this again, so each

subsequent frames will only take .15 seconds. Since a minute of video has 3600 frames, preprocessing the variables reduces computations to save time.

Instead of calculating the inverting filter using convex optimization, this program simply reads the values from a table. Since these coefficients rely only on the motion of the scene, they can be calculated using Matlab and stored offline. Reading from a table is much faster than recalculating coefficients using convex optimization equations each frame.

An additional way to decrease the processing time is to reduce unnecessary computations. Instead of zero padding, Matlab replicates the image on each side to get information for the sides of the image when convolving the image. This reduces the artifacts and provides a better inverted filter. Then, Matlab processes the replicated images in addition to the original image, which doubles or triples the processing time. The program in C/C++ only replicates the amount of image necessary to convolve the original image. For example, if the inverting filter has a length of 16, only 15 pixels are added to each side of the image, while Matlab would add the entire image on each side. Also, outputs from the scaled gradient magnitude equation and clipping high and low values are only performed on the output image. Calculations should be minimized since this program is used for thousands of images.

Conclusion

LCD screens are increasing in popularity among consumer devices. This LCD motion blur reduction approach using signal processing is a viable solution to remove the blur caused by the nature of LCD screens. Implementing this signal processing approach to reduce LCD motion blur using C/C++ leads to use on a real time system. Also, more resources are spent improving the algorithm or testing subjects instead of processing images. Images need to be processed another 10x faster to actualize it on a real time system. This can be achieved through a dedicated hardware such as an FPGA.

A C/C++ version of the deblurring algorithm frees up resources to improve the theory and algorithms for reducing motion blur. Instead of taking days to process images, those working on the algorithm can spend more time improving the theory by evaluating image results. Also, the decreased processing shows that the algorithm can be used on a real-time system, so more money will be devoted to the project.

Future Work

Since this deblurring algorithm does not completely remove the LCD motion blur problem, more steps will be taken to improve the algorithm or incorporate it with other existing technologies. To make the inversion more ideal, parameters will be adjusted based on motion vector error calculations and human subject results. Also, this deblurring algorithm can be used in tandem with interpolating frames on a 120Hz television.

Resulting video sequences from the deblurring algorithm must be tested on human subjects to quantize the quality of the algorithm. Since students working in the video processing laboratory work with these images frequently, they know how to find artifacts and errors in video sequences. This algorithm was created to deblur LCD screens for everyone, so video sequences will be tested using human subjects. They will look at a screen featuring both the regular video sequence and a sequence processed using the deblurring algorithm. Unknowing which sequence was processed beforehand, subjects will judge one scene compared to the other on a scale from -3 to +3. They will be shown multiple sequences with different rates of motion. Results from these tests will indicate the best parameters for the deblurring algorithm.

The scene's set motion is incorporated into the deblurring algorithm. Since each object in the scene may not have the exact frame motion used in the algorithm, there will be a slight error. Therefore, this motion vector error will be calculated for different rates of motion. This error will help adjust parameters to reduce the overall motion vector error. Also, these error results will be compared to error results from the Richardson-Lucy algorithm or no algorithm at all.

One way to reduce the overall motion blur from LCD screens would be to incorporate the deblurring algorithm on a 120 Hz LCD screen. This screen would use either frame interpolation or a flashing black frame. Since the algorithm can be used in a real-time system, the 120 Hz screen should provide the best results. This solution will be more economical when 120 Hz LCD screens become popular and cost less.

Since the LCD Motion Blur problem will never be solved absolutely, scientists can reduce the blur as much as possible by employing multiple solultions. Eventually, consumers will barely notice the motion blur on LCD screens since more and more research is spent reducing the blur and improving the overall device.

References

[1] M. Klompenhouwer and L. J. Velthoven, “Motion blur reduction for liquid crystal displays: Motion compensated inverse filtering,” presented at the SPIE-IS&T Electronic Imaging, 2004.

[2] B. W. Lee, K. Song, D. J. Park, Y. Yang, U. Min, S. Hong, C. Park, M. Hong, and K. Chung, “Mastering the moving image: Refreshing TFT-LCDs at 120 Hz,” presented at the SID Symp. Dig. Tech. Papers. SID, 2005.

[3] N. Fisekovic, T. Nauta, H. Cornelissen, and J. Bruinink, “Improved motion- picture quality of AM-LCDs using scanning backlight,” in Proc. IDW, 2001, pp. 1637–1640.

[4] S. Hong, B. Berkeley, and S. S. Kim, “Motion image enhancement of LCDs,” presented at the IEEE Int. Conf. Image Processing, 2005.

[5] N. Mishima and G. Itoh, “Novel frame interpolation method for holdtype displays,” in Proc. IEEE Int. Conf. Image Processing, 2004, vol. 3, pp. 1473–1476.

[6] S. Har-Noy, T. Nguyen, "LCD Motion Blur Reduction: A Signal Processing Approach," IEEE Transactions on Image Processing, vol. 17, no. 2, Feb 2008.

[7] L. B. Lucy, “An iterative technique for the rectification of observed distributions,” Astron. J., vol. 79, no. 6, Jun. 1974.