A GPU implementation of color digital halftoning using the Direct Binary Search algorithm

Conference Paper · May 2012

DOI: 10.1109/ISCAS.2012.6271629

4 authors, including:

Kartheek Chandu
Institute of Electrical and Electronics Engineers
10 PUBLICATIONS **50** CITATIONS
SEE PROFILE

Mikel Stanich
Ricoh, Boulder Colorado
25 PUBLICATIONS **77** CITATIONS
SEE PROFILE

Barry M. Trager
IBM
58 PUBLICATIONS **1,623** CITATIONS
SEE PROFILE

A GPU implementation of color digital halftoning using the Direct Binary Search algorithm

Kartheek Chandu*, Mikel Stanich*, Barry Trager†, Chai Wah Wu†
*Ricoh Production Print Solutions, LLC, 6300 Diagonal Hwy, Boulder, CO 80301, USA
†IBM T. J. Watson Research Center, P. O. Box 218, Yorktown Heights, NY 10598, USA
Contact email: chaiwahwu@ieee.org

***Abstract*—We illustrate how employing Graphics Processing Units (GPU) can speed-up intensive image processing operations. In particular, we demonstrate the use of the NVIDIA CUDA architecture to implement a color digital binary halftoning algorithm based on Direct Binary Search (DBS). Halftoning a color image is more computationally expensive than the single color case as there is a need to minimize dot interaction between different color planes as well. We propose processing all color planes in parallel. In addition we employ processing several non-overlapping neighborhoods in parallel, by utilizing the GPU's parallel architecture, to further improve the computational efficiency. This parallel approach allows us to use a large neighborhood and filter size, to achieve the highest halftone quality, while having minimal impact on performance.**

I. Introduction

Traditionally halftoning is done via a point-based operation such as dither mask halftoning [1]. These algorithms only require one comparison operation per pixel. More sophisticated neighborhood-based halftoning algorithms, such as error diffusion (ED) [2] [3] [4] [5] [6] and Direct Binary Search (DBS) [11] require knowledge of neighborhood pixels to process the current pixel. ED and DBS algorithms are orders of magnitude more demanding than dither mask halftoning in terms of number of operations, but generate markedly improved halftone image quality. Improvements include better sharpness, higher detail rendition and minimized texture artifacts. Color screening, since it involves multiple interacting masks, adds additional constraints to minimize texture and to avoid dot-on-dot printing. In general, an input CMYK image is provided to the screening module, which performs halftoning (e.g. dither mask halftoning) independently for each color plane. In this paper, we propose a simple but effective color halftoning algorithm with minimum dot-on-dot using DBS and its implementation on Graphics Processing Units (GPU).

Recently, GPU has emerged as a powerful low-cost parallel processing architecture for various applications requiring large amount of compute cycles [7]. In this paper, we studied how GPU can be used to implement high quality binary color halftoning algorithms for real-time operation in a high speed printer controller. In particular we focus on implementations using Nvidia's CUDA architecture.

II. Nvidia's CUDA GPU architecture

Nvidia's CUDA architecture provides a parallel computing architecture to capitalize on GPU technology. While GPUs in desktop computers' graphic cards can employ CUDA for application acceleration, specialized GPU hardware such as the Nvidia Tesla or Fermi boards provide specialized hardware containing a large numbers of GPUs. These boards are being assembled to provide solutions for High Performance Computing (HPC). China has employed these GPUs to create one of the fastest supercomputer in the world [9]. The CUDA Architecture consists of multiple components: 1) Parallel compute engines, 2) OS Kernel support, 3) Driver and, 4) Instruction set architecture for computing kernels and functions [8]. Multiple memory addressing modes with efficient memory bandwidth are supported which include texture memory, shared memory and constant memory. Mechanisms for scheduling work on processors are provided using streams and atomic operations.

III. Direct Binary Search halftoning

Direct Binary Search (DBS) can be thought of as an iterative/recursive optimization heuristic which is used to minimize a cost function ε, defined as the error (or difference) between the perceived halftone image and the perceived continuous tone image. This is represented as:

$$\varepsilon = |h(x,y) * *g(x,y) - h(x,y) * *f(x,y)|^2 dxdy,$$

where $**$ denotes 2-dimensional convolution, $h(x,y)$ (of size $P \times P$) represents the point spread function (PSF) of the human visual system, $f(x,y)$ is the continuous tone original image and $g(x,y)$ is the corresponding rendered halftone image, which are assumed to have values either 0 (white) or 1 (black). The halftone image itself incorporates a printer model

$$g(x,y) = \sum_m \sum_n g[m,n] p(x - mX, y - nX) \qquad (1)$$

This particular model represents a device with $X \times X$ addressability, a spot profile $p(x,y)$, and the addictive interaction between overlapping spots.

DBS is a computationally expensive algorithm that requires several passes or iterations through an image before converging to the final halftone. DBS starts by generating an initial halftone image, and local improvements to the halftone are produced by swapping and toggling of pixels. Swapping is the operation of exchanging the colors of nearby pixels and toggling is the operation of changing the polarity of individual pixels. The cost function ε can be represented as

$$\varepsilon = < \tilde{e}, \tilde{e} >,$$

978-1-4673-0219-7/12/$31.00 ©2012 IEEE

where $< \cdot, \cdot >$ denotes the inner product and $\tilde{e}(x,y) = h(x,y)**(g(x,y)-f(x,y))$ represents the perceptually filtered error. We will assume that the continuous tone image $f(x,y)$ can also be expressed in terms of its samples $f[m,n]$ similar to Eq. (1). Then the perceived error is given by

$$\tilde{e}(x,y) = \sum_{m,n} e[m,n]p\ (x - mX, y - nX),$$

where

$$e[m,n] = g[m,n] - f[m,n],$$

and $\tilde{p}(x,y) = h(x,y) * *p(x,y)$ is the perceived printer spot profile. Now, consider the effect of a trial change. The new error will be $\tilde{e}' = \tilde{e} + \Delta\tilde{e}$. Substituting this into the cost function ε equation and expanding the inner product, we obtain

$$\varepsilon' = \varepsilon + 2 < \Delta\tilde{e}, \tilde{e} > + < \Delta\tilde{e}, \Delta\tilde{e} >,$$

where we have used the fact all signals are real-valued. Either a toggle at pixel (m_0, n_0) or a swap between pixels (m_0, n_0) and (m_1, n_1) can be represented as

$$g'[m,n] = g[m,n] + \sum_i a_i \delta[m - m_i, n - n_i]$$

Then

$$\Delta\tilde{e}(x,y) = \sum_i a_i \tilde{p}(x - m_i X, y - n_i X)$$

and

$$\Delta\varepsilon = 2\sum_i c_{\tilde{p}\tilde{e}}[m_i, n_i] + \sum_{i,j} a_i a_j c_{\tilde{p}\tilde{p}}[m_i - m_j, n_i - n_j],$$

where

$$c_{\tilde{p}\tilde{e}}[m,n] =< \tilde{p}(x,y), \tilde{e}(x + mX, y + nX) >,$$

and

$$c_{\tilde{p}\tilde{p}}[m,n] =< \tilde{p}(x,y), \tilde{p}(x + mX, y + nX) > .$$

Assuming that $c_{\tilde{p}\tilde{p}}$ is symmetric, then

$$\Delta\varepsilon = 2(\sum_i c_{\tilde{p}\tilde{e}}[m_i, n_i] + \sum_{i<j} a_i a_j c_{\tilde{p}\tilde{p}}[m_i - m_j, n_i - n_j]) + \sum_i a_i^2 c_{\tilde{p}\tilde{p}}[0,0].$$

Each a_i represents the amount of change in the gray level toggle, and they are defined as:

$$a_i = g_{new}[m_i, n_i] - g_{old}[m_i, n_i].$$

In particular, if the pixel toggles between 0 and 1, then

$$a_i = 1 - 2g[m_i, n_i] = \pm 1.$$

If $a_i = \pm 1$, then the last term of $\Delta\varepsilon$ is $\sum_i a_i^2 c_{\tilde{p}\tilde{p}}[0,0] = nc_{\tilde{p}\tilde{p}}[0,0]$. Once every pixel location (m,n) is examined for a qualified swap or toggle, it updates the halftone image $g[m,n]$ and $c_{\tilde{p}\tilde{e}}[m,n]$. This operation is performed again on all pixels until the end criteria (e.g. $\Delta\varepsilon < \delta$ or no changes to the halftone are performed in the last iteration) is met. Any accepted operation requires updating of $c_{\tilde{p}\tilde{e}}$ at $g[m,n]$ using $c_{\tilde{p}\tilde{e}}[m,n]' = c_{\tilde{p}\tilde{e}}[m,n] + a_i c_{\tilde{p}\tilde{p}}[m - m_i, n - n_i]$.

Fig. 1. Continuous tone CMYK ($128 \times 128 \times 4$) image

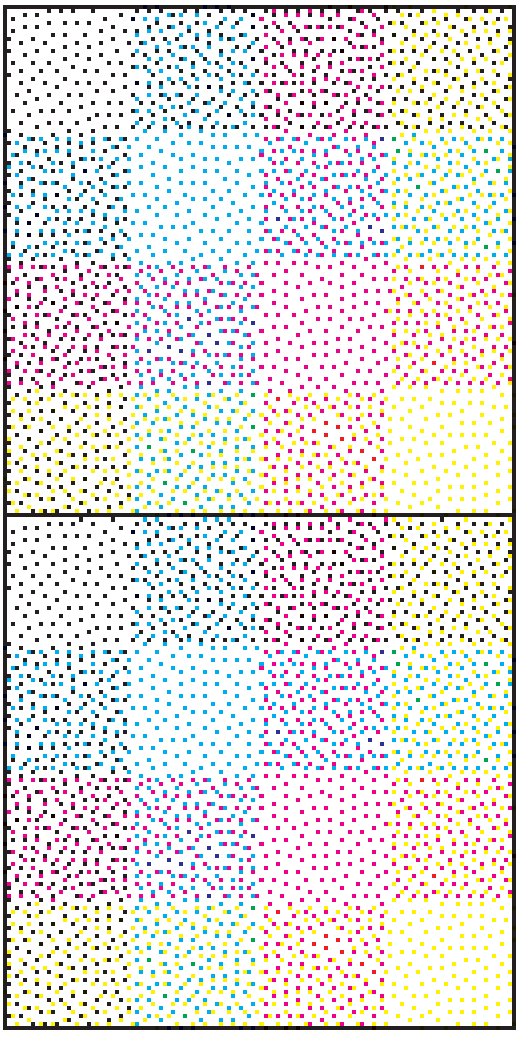

Fig. 2. Rendered halftone color image of Fig. 1, (top) Rendered halftone image using CPU, and (bottom) Rendered halftone image using GPU

IV. Parallelizing Direct Binary Search halftoning for color images

In the case of multiple color planes, all swaps and toggles are examined within a single neighborhood in each of color planes. Only the single best swap or toggle for each neighborhood, among all of the color planes, is performed to minimize dot-on-dot. This is referred to as a "qualified swap or toggle operation". The proposed algorithm is described below:

1) Read an input continuous tone image $f[m, n, d]$ of size $R \times C \times D$ (example shown in Figure 1).
2) Generate an initial halftone $g[m, n, d]$ of size $R \times C \times D$ with $g[m, n, d]$ = 0.
3) Compute the auto-correlation function $c_{\tilde{p}\tilde{p}}[m, n]$.
4) Compute the initial error $c_{\tilde{p}\tilde{e}}[m, n]_d$ for each color plane D.
5) **repeat for** each pixel (m, n) **do**
   - compute the change in $c_{\tilde{p}\tilde{e}}[m, n]_D$ due to toggling pixel $g[m_0, n_0, d]$, and swapping pixel $g[m_0, n_0, d]$ with $g[m_i, n_i, d]$ in each color plane D.
   - find the operation with maximum error decrease in $\Delta\varepsilon$ among all color planes D.
   - update $c_{\tilde{p}\tilde{e}}[m, n]_D$ and $g[m, n, d]$ with the above operation.
6) repeat step 5 until end criteria is met.

The serial DBS algorithm traverses all pixels of the image in a fixed order. At every pixel location (m_0, n_0) DBS looks for a qualified swap or toggle operation within its $(2P-1) \times (2P-1)$ neighborhood as described in step 5. This check is performed in each of the color planes. Only the single swap or toggle in all the color planes which reduces the error the most is performed. After any updating operation is completed, the algorithm selects the next pixel location (m_0, n_0) and again examines the new neighborhood for a qualified swap or toggle operation. The updating process only modifies the $(2P-1) \times (2P-1)$ neighborhood in $c_{p\tilde{e}_D}$ and $g[m, n, d]$ around the center pixel location (m_0, n_0) for a toggle. On the other hand a swap requires updating the $(2P-1) \times (2P-1)$ neighborhood around the center pixel location (m_0, n_0) and also the $(2P-1) \times (2P-1)$ neighborhood around the swapped pixel location (m_i, n_i). Therefore the effect of an update can extend to a $(4P-3) \times (4P-3)$ region, and pixel neighborhoods can be processed simultaneously only if the center pixel locations (m_0, n_0) are separated by at least $4P-3$.

Examining and updating these components is more computationally expensive when the size $P \times P$ of the filter $h(x, y)$ is large. High resolution halftone images require a larger HVS filter size. The neighborhood operations performed at the former pixel location are independent of the later pixel location neighborhood, only if the new pixel location neighborhood does not overlap the former neighborhood. The main goal of many of the previous articles regarding DBS is to decrease the execution time of the algorithm. One of the most common approaches is to decrease the neighborhood size or filter size. We proposed parallelizing these different non-overlapping neighborhood operations by utilizing the GPU's parallel architecture [12]. Our parallel approach allows us to use a large neighborhood and filter size, to achieve the highest halftone quality, while having minimal impact on performance.

The parallel GPU algorithm examines in parallel all swaps and toggles in the current neighborhood for each of the color planes D. In our approach, after the best swap or toggle among all the color planes is determined, we must update the cost function ε. This is performed in parallel by updating the regions of $c_{p\tilde{e}_D}$ and $g[m, n, d]$ corresponding to the pixels which have changed. A toggle only requires updating the $(2P-1) \times (2P-1)$ neighborhood of the current pixel. If a swap was performed, then the current neighborhood of the swapped pixel in the C_{pe} array is updated in parallel. In addition to processing entire neighborhoods and multiple color planes in parallel, it is also possible to process several pixel neighborhoods in parallel provided the update operations remain independent. As previously described, an update operation can extend to a $(4P-3) \times (4P-3)$ region. Before invoking the GPU kernel to process the next neighborhood, we examine several of the next pixels to be processed to determine whether their $(4P-3) \times (4P-3)$ regions are non overlapping. If they are determined to be non overlapping, the GPU kernel is instructed to process them in parallel.

For the next set of neighborhoods N_x, the algorithm has to wait until the updating of all components are completed. Note that there are three levels of parallelism employed here. First, the $\Delta\varepsilon$ computation and $c_{p\tilde{e}}$ updates are computed in parallel within each pixel's neighborhood. Second this is done simultaneously for all color planes. Third, pixels in several non-overlapping regions can be performed simultaneously.

Steps 1-4 of the algorithm outlined above are part of the initialization performed by the CPU. After initialization the various arrays are transferred into the memory on the GPU card. The three bullets in step 5 correspond to separate kernels to be performed on the GPU. The CPU decides how many non-overlapping pixel neighborhoods can be processed together. In addition the CPU issues a GPU kernel call to test all of the candidate swaps or toggles to select the best operation. Local memory of each thread block is used in this case. Then a separate kernel call compares the results of the various thread blocks for all of the color planes for each pixel neighborhood. This two stage process is required since local memory is only accessible by threads in the same block and the number of threads in a block is limited to 512. Once the best swap or toggle is determined for each pixel neighborhood, separate kernels are used to update the $(2P-1) \times (2P-1)$ neighborhood in the $c_{\tilde{p}\tilde{e}}[m, n]_D$ array around each swap or toggle. After these updates are complete, the next batch of pixel neighborhoods can be processed.

Due to the parallelism provided by the GPU, we are able to support larger search neighborhoods. We have chosen to make the search neighborhood as large as $(2P-1) \times (2P-1)$ even for larger HVS filter sizes.

The Nvidia GPU has four types of memory: local, global, constant, and texture. Employing the optimal memory for different operations provides the highest performance. Constant

DBS Timings (total time in sec)				
image size	color planes	Matlab time	GPU time	speed-up ratio
128x128	4	155	34	4.6
256x256	3	571	67	8.5
432x432	4	1678	146	11.5
768x512	3	4784	291	16.4
3048x2512	4	124781	6124	20.4

TABLE I
EXECUTION TIMES FOR GPU-ENABLED AND SERIAL DBS HALFTONING.

memory is cached and was used to store both the valid toggle levels and the set of pixel neighborhoods to be processed in parallel. Global memory stores information which can be read or written by any thread, but is not cached and has rather slow access. Local memory is only accessible to threads in a block, but has very fast access time. Thus we try to use local memory as much as possible to improve performance. We use local memory to store the results of $\Delta\varepsilon$ for each swap or toggle, and then search for the swap or toggle which reduces ε the most. While the $\Delta\varepsilon$ computation can be performed in parallel for our set of non-overlapping neighborhoods, searching for the best swap or toggle requires $log(n)$ iterations using a parallel scan of binary min operations, where n is the number of values being compared.

Parallelizing the neighborhoods as proposed significantly decreases the execution time thus supporting the extension to color halftoning.

V. PERFORMANCE RESULTS

The focus of this study is on the computational efficiency of the color halftoning algorithm when implemented on the GPU architecture. A GPU implementation of color halftoning on the input image shown in Fig. 1 ran in 34 secs, whereas the CPU implementation processed the same image in 155 secs i.e. the computation is 4.6 times faster using GPU. Results in Fig. 2 shows the output rendered halftone images implemented using CPU and GPU. Fig. 2 demonstrates that parallel neighborhood processing using GPU produces the same outcome and has minimum dot-on-dot interaction as the CPU implementation. We tested our algorithms on a C1060 Nvidia Tesla GPU accelerator board in a server with a Intel Xeon X5650 six-core processor with 12 GB physical memory. The C1060 provides 30 multiprocessors with a total of 240 processor cores and 4 GB of on-board memory.

The timings for the GPU based DBS implementation compared to a serial Matlab implementation are shown in Table I. The Matlab DBS code is optimized to use as much vectorized operations as possible. The results are presented for several image sizes (size is given in pixels). Larger images allow more neighborhoods to be processed in parallel yielding a larger speed-up factor.

REFERENCES

[1] R. Ulichney, Digital halftoning, MIT Press, 1987.
[2] R. W. Floyd and L. Steinberg, "An adaptive algorithm for spatial grey scale", Proceedings of the Society of Information Display 17 (2), pp. 7577,1976.
[3] C. P. Tresser and C. W. Wu, US Patent 6,006,011.
[4] H. R. Kang, Digital Color Halftoning, SPIE Press, 1999.
[5] J.F. Jarvis, C.N. Judice and W.H. Ninke, "A survey of techniques for the display of continuous tone pictures on bilevel displays," Computer Graphics and image Processing, vol. 5, pp. 13-40.
[6] T. Metaxas, "Parallel Digital Halftoning by Error Diffusion," Proc. of the FCRC2003 Paris C. Kanellakis Workshop, San Diego, CA, June, 2003.
[7] X. X. Wang and B. E. Shi, "GPU Implementation of Fast Gabor Filters," Proceedings of IEEE ISCAS 2010, pp. 373-376.
[8] NVIDIA CUDA Architecture, Introduction and Overview.
[9] www.top500.org
[10] C. W. Wu, G. Thompson and M. Stanich, "A unified framework for digital halftoning and dither mask construction: variations on a theme and implementation issues," Proc. IS&T's NIP19: International Conference on Digital Printing Technologies, pp. 793-796, 2003.
[11] J. P. Allebach, "DBS: retrospective and future directions," Proc. SPIE, vol. 4300, pp 358-376, 2001.
[12] B. Trager, C. W. Wu, M. J. Stanich, and K. Chandu, "GPU-Enabled Parallel Processing for Image halftoning application," IEEE International Symposium on Circuits and Systems (ISCAS), pp.1528-1531, Rio de Janerio, Brazil, May 2011.