A Consideration of Real-Time Imaging and Printing

Steven J. Simske, Margaret Sturgill, Jason S. Aronoff, Marie Vans, Hewlett-Packard Labs, 3404 E. Harmony Rd., MS 36, Fort Collins CO 80528, USA

Abstract

Modern variable data presses use substantial processing power. In many cases, a bank of processors is used to manage the RIP (raster image processing), and print jobs are performed using sophisticated parallel scheduling approaches. The high processing power of digital presses enables the possibility of performing valuable imaging tasks using the same processing units. Important imaging tasks include reading printed marks (such as barcodes), print validation and inspection.

In order to optimize the interleaving of real-time printing and imaging tasks, different imaging approaches must be considered. In this paper, we consider three different classes of imaging optimization in order to compare their relative effect on throughput and on amenability to processing on the press. These are (1) performing down-sampling before image segmentation versus performing native resolution image segmentation, (2) selecting different programming languages/compilers (e.g. Java versus C++ in our experiments) for the imaging, and (3) marshaling images into a single buffer versus allowing the system to manage the image,. Our results demonstrate that, in general, changes in structural approach to imaging, such as (1) provides, have the greatest positive impact on processing, while (2) has the least impact. The impact of approach (3) is more highly dependent on the architecture of the press, and so is perhaps the method that can be most positively affected by intelligent modeling.

Keywords: Down-sampling, image segmentation, memory locking, compiler

Introduction

Modern digital presses, capable of printing thousands of pages per hour – with each one different from the rest – use multiple processing units to prepare and process the page images [1][2][3]. Multiple units are needed to continue to "feed" pages at speed when there is significant page-to-page variability and resolutions are often 800-1200 dots/inch or more.

"Actionable printing" is the variable data digital printing domain wherein each printed item has its own customized content which can be later interrogated or "read", for example, as part of a mobile camera service. As its adoption continues to increase, it will be important to optimize the use of processing resources on the digital presses.

We have previously described how different factors in the "reading" environment can be combined into a mathematical model for deploying cameras, inspection devices, and forensic imagers in a geographically dispersed network of manufacturers, distributors, retailers and consumers [4]. In this paper, we consider three different aspects of real-time imaging and printing in an effort to model a system for image-intensive printing:

(1) Performing down-sampled segmentation vs. native resolution segmentation. This is a functional test for the improvement in performance possible with a memory-access intensive task. While we consider image segmentation here for ease of timing analysis, many other imaging – and some variable data pre-press – tasks will also require similar trade-offs (working on the original resolution image versus down-sampling and working on a reduced resolution image).

(2) Performing processing and image-accessing intensive analysis in an interpreted (e.g. Java) programming language vs. performing the same analysis in a compiled (e.g. C++) programming language. This test was performed simply to determine the relative impact of programming language selection on the system performance. This has larger system development importance, since for example (a) it may be simpler to program in the interpreted language since garbage clean up is not the responsibility of the programmer and (b) there may be legacy, related services, etc., codebase(s) written in one language that we wish to augment directly (i.e. in the same language).

(3) Marshaling images into a single (continuous) buffer in memory vs. allowing the system to manage the image – with re-location, discontinuous buffering, etc. possible. This test is meant to determine the efficiency of locking an image in memory for image-access intensive tasks, and is highly relevant to both variable data printing and image interpretation tasks.

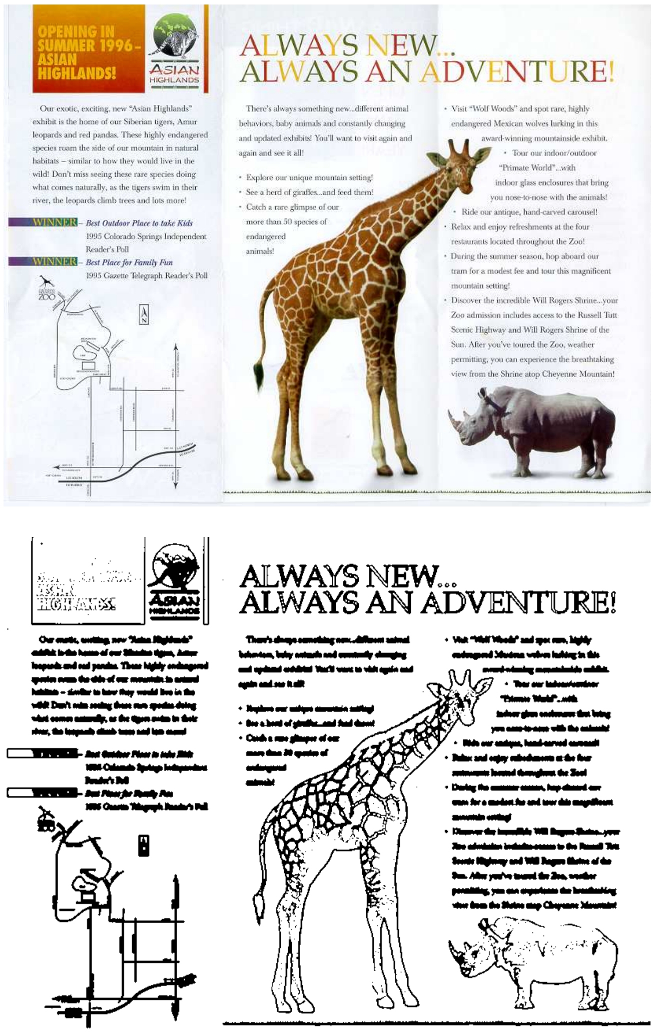
Methods and Materials

Downsampled-Segmentation vs. Native Resolution Segmentation

For this test, we considered two variations:

(1) Performing document image segmentation on a single document, originally scanned at 300, 400 or 600 pixels per inch (ppi), before and after down-sampling to 75 ppi, as described in [5]. Document image segmentation requires repeated, "random" access to parts of the image for a number of image processing tasks. The first, image thresholding, uses an image histogram followed by a binarization process. This requires accessing each image pixel at least twice, the bare minimum of which is a global thresholding approach such as the approach by Kittler et al. [6]. Methods which require local thresholds, multiple thresholds to accommodate background color differences, etc., will generally require accessing each pixel three or more times. Next, image segmentation is performed. Image segmentation is usually performed on the binarized image, and involves multiple procedures such as run length smearing, dilation/erosion, connected component formation, connected component histogram formation and analysis, connected component projection profile

computation and analysis, and clustering of related connected components [7]. Figure 1 illustrates the effect of thresholding on an image. The input image (top) is binarized to create the lower image using two passes through the entire image.


OPENING IN SUMMER 1996– ASIAN HIGHLANDS!

ASIAN HIGHLANDS

Our exotic, exciting, new "Asian Highlands" exhibit is the home of our Siberian tigers, Amur leopards and red pandas. These highly endangered species roam the side of our mountain in natural habitats – similar to how they would live in the wild! Don't miss seeing these rare species doing what comes naturally, as the tigers swim in their river, the leopards climb trees and lots more!

WINNER – *Best Outdoor Place to take Kids* 1995 Colorado Springs Independent Reader's Poll

WINNER – *Best Place for Family Fun* 1995 Gazette Telegraph Reader's Poll

ALWAYS NEW... ALWAYS AN ADVENTURE!

There's always something new...different animal behaviors, baby animals and constantly changing and updated exhibits! You'll want to visit again and again and see it all!

- Explore our unique mountain setting!
- See a herd of giraffes...and feed them!
- Catch a rare glimpse of our more than 50 species of endangered animals!
- Visit "Wolf Woods" and spot rare, highly endangered Mexican wolves lurking in this award-winning mountainside exhibit.
- Tour our indoor/outdoor "Primate World"...with indoor glass enclosures that bring you nose-to-nose with the animals!
- Ride our antique, hand-carved carousel!
- Relax and enjoy refreshments at the four restaurants located throughout the Zoo!
- During the summer season, hop aboard our tram for a modest fee and tour this magnificent mountain setting!
- Discover the incredible Will Rogers Shrine...your Zoo admission includes access to the Russell Tutt Scenic Highway and Will Rogers Shrine of the Sun. After you've toured the Zoo, weather permitting, you can experience the breathtaking view from the Shrine atop Cheyenne Mountain!

Figure 1*. Example of thresholded image. The full-color image (top) [8] is binarized in accordance with ref. [6] and then used for downstream tasks such as image segmentation.*

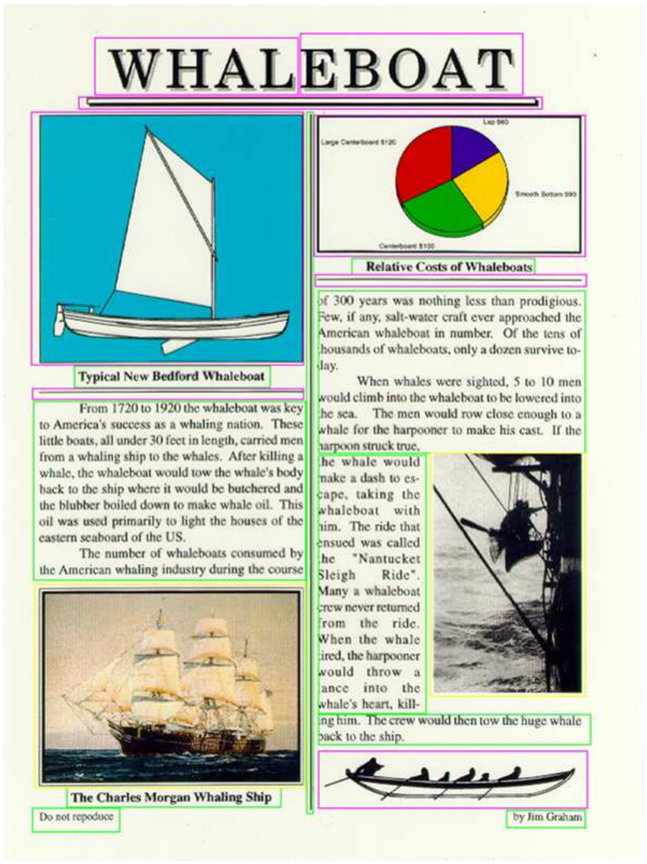
An example of document image segmentation is given in Figure 2. For this document image, image pixels were accessed, in the mean, more than 8 times throughout the processing (including the thresholding). The output of this software [9] is a set of labeled "regions" (clustered and "typed" connected components or connected component composites).

(2) The second test is to down-sample the image and then perform a two-stage segmentation. The first segmentation is performed using projection profiles [7] to define cuts through the document image and break up the image into as small of logical units as possible. This is readily accomplished for Manhattan layouts such as for the document page image in Figure 2. We performed our tests using a slide scanner, which can capture the images from several rows of film slides simultaneously.

We down-sampled all images to 75 ppi for both sets of tests in this section. The original images were scanned at 300 ppi or 600 ppi. Two different sets of 50 images were used for the timing data for (1) and (2).

C++ vs. Java

We considered an automated document image segmentation task as described in [10], for which we had originally performed image thresholding and segmentation on 1.2 x 10^6 document pages. To facilitate comparison we built a simplified (approximately 2 x 10^4 instead of 2 x 10^5 lines of code) segmentation engine simultaneously in both C++ and Java. The only difference—from the UML [11] to the object arguments, fields and methods—between the two engines was the employment of memory clean up in the C++ version. This resulted in slightly (~2%) more lines of code for the C++ version. Even the files (equivalent in number) were named identically.


WHALEBOAT

Lap $60
Large Centerboard $120
Smooth Bottom $90
Centerboard $100

Relative Costs of Whaleboats

Typical New Bedford Whaleboat

From 1720 to 1920 the whaleboat was key to America's success as a whaling nation. These little boats, all under 30 feet in length, carried men from a whaling ship to the whales. After killing a whale, the whaleboat would tow the whale's body back to the ship where it would be butchered and the blubber boiled down to make whale oil. This oil was used primarily to light the houses of the eastern seaboard of the US.

The number of whaleboats consumed by the American whaling industry during the course of 300 years was nothing less than prodigious. Few, if any, salt-water craft ever approached the American whaleboat in number. Of the tens of thousands of whaleboats, only a dozen survive today.

When whales were sighted, 5 to 10 men would climb into the whaleboat to be lowered into the sea. The men would row close enough to a whale for the harpooner to make his cast. If the harpoon struck true, the whale would make a dash to escape, taking the whaleboat with him. The ride that ensued was called the "Nantucket Sleigh Ride". Many a whaleboat crew never returned from the ride. When the whale tired, the harpooner would throw a lance into the whale's heart, killing him. The crew would then tow the huge whale back to the ship.

The Charles Morgan Whaling Ship

Do not repoduce

by Jim Graham

Figure 2*. Example of a document image segmentation. "Text" typed regions are outlined in green, "drawing" typed regions in purple (see oar boat in lower right for example if printed in grayscale) and the 2 photos/images in yellow.*

Figure 2 also suffices to illustrate the "work" performed by these two engines. Each of them produced fully typed {photo, text, drawing, and table} regions segmented from the document images.

We performed timing using system timers build into the invoking code. Timing was performed on a set of 500 files for both the C++ and Java-based engines.

Accessing Marshaled Image Data vs. Native Image Access

To marshal the image data into continuous memory, we used the LockBits functionality in .NET. LockBits, which locks the bitmap data in continuous memory, allows the marshaling out of the data. So, we performed timing on accessing reads and writes of image data with and without the use of LockBits:

(1) Load image in using the .NET Bitmap class.

(2) For no use of LockBits, reading is performed by sequentially reading in each pixel. Writing is performed by sequentially setting each pixel in the image to the Color.Red system color value.

(3) For the use of LockBits, we lock the bitmap, allocate the image buffer, and marshal out the bitmapData to a buffer. For reading, for each pixel in the image, we set the variables r,g,b to pixel values. For writing, for each pixel in image we set r, g, b bytes to {255,0,0) respectively (red). We then marshal out the new image to bitmap.

Results

The test results reported herein were performed on a variety of laptops and workstations. All systems had at least 1 GB of RAM and were run using Java or .NET on a Windows XP 32-bit or a Windows7 64-bit OS. Performance scores cannot be compared absolutely across the three types of experiments, nor is it necessary to do so. All reported comparisons were performed on exactly the same hardware with the same software, except where noted (that is, the C++ vs. Java comparison).

Downsampled-Segmentation vs. Native Resolution Segmentation

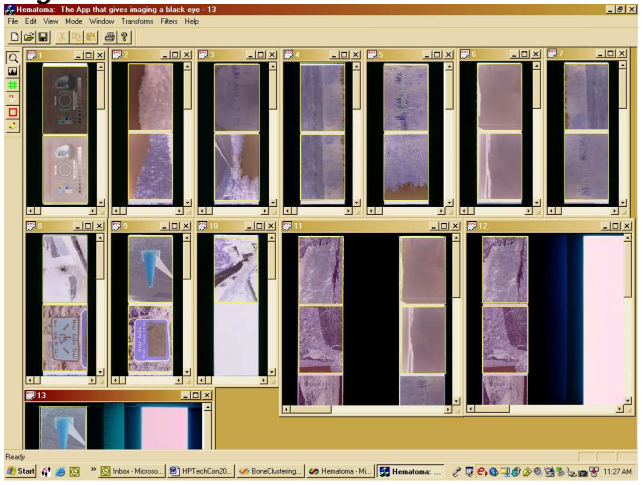


Figure 3. Sample of down-sampled and pre-segmented slide scans.

For our 50-file document image set, performing segmentation on the original 300 ppi images required a mean of 11.7 sec, while down-sampling to 75 ppi and segmenting at 75 ppi required a mean of 1.55 seconds (0.65 sec for down-sampling and 0.90 sec for segmentation).

For the slides (Figure 3), the original images at 300 ppi required less time than the general document image set, since the slides were organized in parallel rows. Segmentation required only a mean of 5.46 sec. Down-sampling required 0.65 sec and segmentation of the slide sets required only a mean of 16 msec/image.

C++ vs. Java

We performed benchmarking on 500 full page images already down-sampled to 75 ppi image, as described above. The C++ engine was consistently faster. On the mean, the Java engine required 34% more time to completion (1.73 sec compared to 1.29 sec).

Accessing Marshaled Image Data vs. Native Image Access

Two large images were timed. The results for the means of ten runs were computed. Image #1 is a 24-bit, 4264 x 5704 pixel image. The results comparing marshaling (using LockBits) and allowing native image access management (No LockBits) are presented in Table 1. Both reading and writing were approximately eight times faster when using LockBits (marshaling).

Table 1. Results for Image #1 (mean of 10 runs)

	Reading	Writing
No LockBits	38.00 sec	34.70 sec
LockBits	4.75 sec	3.97 sec
LockBits, pixel access only	4.71 sec	3.84 sec

Image #2 is a 24-bit, 2572 x 3696 pixel image. The results comparing marshaling (using LockBits) and allowing native image access management (No LockBits) are presented in Table 2. Writing was again eight times faster when using LockBits (marshaling), and reading was approximately 11 times faster.

Table 2. Results for Image #2 (mean of 10 runs)

	Reading	Writing
No LockBits	15.75 sec	15.91 sec
LockBits	1.43 sec	1.94 sec
LockBits, pixel access only	1.41 sec	1.91 sec

As both Table 1 and Table 2 indicate, the majority of the time for the LockBits approach is in accessing the pixels (~98% of the time used) and not in the actual marshaling of the data (~2% of the time used).

Discussion and Conclusions

Downsampled-Segmentation vs. Native Resolution Segmentation

For image processing tasks such as segmentation, it is not surprising that performing segmentation at lower resolution results in a significant speed-up in performance. It is worth noting, however, that the down-sampled image contains only 1/16 as many pixels, but did not perform segmentation 16 times as quickly (0.90 sec mean compared to 11.7 sec mean).

When the down-sampled images were pre-segmented, however, a significant speed-up (to just 0.016 sec mean processing time) was observed. The relative timings of these different processes are likely dependent on the hardware used: cache and RAM configuration and extent are likely important variables to consider, and are worth future work.

It is also worth noting that connected component boundaries created for 75 ppi document images will in general be less accurate than boundaries created for 300 ppi document images. However, the minor cropping differences are probably not a reason to preclude the clear performance provided by the down-sampling. In fact, we have been able to extricate useful information for scanning-related tasks with down-sampling to as little as 30 ppi, for which more than 95% of the processing time is the down-sampling itself.

C++ vs. Java

The second part of this paper compared and contrasted the implementation of an imaging application written simultaneously, line-for-line, in Java and C++. We considered UML, availability and ease of integration of existing APIs, testing and prototyping, UI, imaging and XML specification issues for comparing the two development platforms. We found that while Java was consistently an easier platform for which to find existing APIs, C++ offered an advantage in overall integration. Java is in many ways a more mature technology and for this certain advantages in image processing and ease of XML development were noted. C++, however, may provide a more consistent method of development that some developers may find more comfortable. We also find no advantage in the use of JNI vs. managed code wrappers incorporating existing native code (e.g. in C++). Overall, we did not find a broadly significant advantage to development on either platform, and the 34% difference in performance was small compared to that observed in the other experiments report herein. The choice of the platform, therefore, should be based on, not surprisingly, the developer's skill set & deployment platform issues.

That being said, there was a small but statistically significant processing penalty in using Java for intense imaging tasks. C++-based benchmarking completed in 25% less time. For an overall mathematical model, therefore, it is deemed generally advantageous to perform intensive imaging tasks on C++ or other C-based compilers/interpreters.

Accessing Marshaled Image Data vs. Native Image Access

The marshaling results were unequivocal. Using LockBits to marshal the images into continuous memory resulted in nearly an order of magnitude speed image reading and writing. This approach is recommended for any printing and imaging tasks.

Conclusion

Not surprisingly, some of the results presented indicate a dependency on hardware architecture—the amount of cache and RAM, for example. However, it is worth noting that none of the images tested required a significant portion of available RAM. The results presented herein suggest that down-sampling and performance of image processing tasks at as low a resolution – and, if possible, in pre-segmented regions – as possible is warranted.

References

[1] Digital Printing Press for Commercial Printers: Xerox, http://www.xerox.com/digital-printing/digital-printing-press/enus.html, last accessed 27 June 2011.

[2] NexPress Digital Production Color Platform, http://graphics.kodak.com/US/Product/Printers_Presses/Digital_Color/default.htm, last accessed 27 June 2011.

[3] HP Indigo Digital Presses, http://h10010.www1.hp.com/wwpc/us/en/sm/WF02a/18972-18972-236257.html, last accessed 27 June 2011.

[4] S. Simske, M. Sturgill, J. Aronoff and M. Vans, "Factors in a Security Printing & Imaging Based Anti-Counterfeiting Ecosystem," NIP26: 26th International Conference on Digital Printing Technologies and Digital Fabrication, pp. 368-371 (2010).

[5] Sturgill, M. and Simske, S.J. "A Ground-Truthing Engine for Proofsetting, Publishing, Re-Purposing and Quality Assurance," Hewlett-Packard Technical Report HPL-2003-234, available on-line at http://www.hpl.hp.com/techreports/2003/HPL-2003-234.pdf (2003).

[6] J. Kittler, J. Illingworth and J. Fõglein, "Threshold selection based on a simple image statistic," Comp. Vision Graph. Image Proc., Vol 30, pp. 125-147 (1985).

[7] F.M. Wahl, K.Y. Wong, and R.G. Casey, "Block segmentation and text extraction in mixed/image documents," Computer Vision Graphics and Image Processing, Vol. 2, pp.375-390 (1982).

[8] Cheyenne Mountain Zoo, http://www.cmzoo.org/

[9] S.J. Simske and S.C. Baggs, "Customized Capture: Techniques for Automating Scanner Workflows," ACM Symposium on Document Engineering 2004:171-177 (2004).

[10] S.J. Simske and M. Sturgill, "A Ground-Truthing Engine for Proofsetting, Publishing, Re-Purposing and Quality Assurance", ACM DocEng 2003, pp. 150-152 (2003).

[11] Free On-Line Dictionary of Computing (FOLDOC), Unified Modeling Language, http://foldoc.org/UML, last accessed 29 June 2011.

Author Biography

Steve Simske is an HP Fellow and the Director and Chief Technologist of the Document Lifecycle & Security Printing & Imaging portfolio in Hewlett-Packard Labs. Steve is currently on the IS&T Board. He is also an IS&T Fellow and a member of the World Economic Forum's Global Agenda Council on Illicit Trade. He holds more than 40 US patents and has more than 250 peer-reviewed publications. He holds advanced degrees in Biomedical, Electrical and Aerospace Engineering.