Efficient Conversion of Digital Documents to Multilayer Raster Formats

Léon Bottou, Patrick Haffner and Yann LeCun

AT&T Labs - Research

{leonb,haffner,yann}@research.att.com

Abstract

How can we turn the description of a digital (i.e. electronically produced) document into something efficient for multilayer raster formats [1, 6, 4]? It is first shown that a foreground/background segmentation without overlapping foreground components can be more efficient for viewing or printing. Then, a new algorithm that prevents overlaps between foreground components while optimizing both the document quality and compression ratio is derived from the Minimum Description Length (MDL) criterion. This algorithm makes the DjVu compression format significantly more efficient on electronically produced documents. Comparisons with other formats are provided.

1 Introduction

Most document description languages such as PostScript, PDF and MSWord are generally slow to render, may produce very large files and are often platform dependent. The distribution of electronic documents through the web is better achieved with efficient "rasterized" formats such as DjVu [1]. Such formats use different layers for text or graphic elements

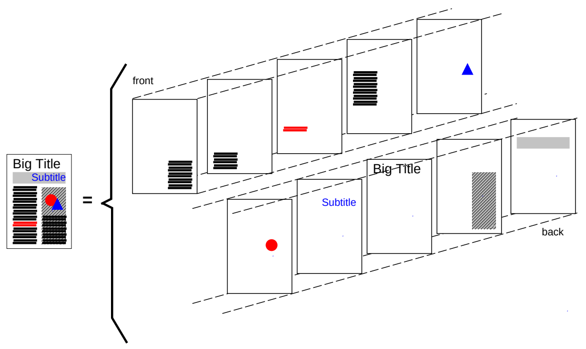

Figure 1: Decomposition of a digital document as a list of drawing operations.

(i.e. foreground) and pictures or background. Obtaining a correct foreground/background segmentation has a critical impact on the performance of layered document raster formats such as DjVu or ITU T.44 [6]. The current DjVu segmenter [3] analyzes a raw high resolution color document image and produces a bitonal mask that relegates pixels to the background or the foreground with good accuracy.

In many practical cases, one would like to compress a document produced using computerized methods that do not rely on a pixel based representation of the document. A text processing software, for instance, represents a document using high level objects such as text, fonts, colors, embedded images, etc. This *structured page information* obviously provides considerable help for the segmenter and should yield very high quality compressed images.

Structured page information for electronic documents come in a large variety of file formats such as the MSWord `doc` files, PDF files, or PostScript files. Printing such files converts the structured information into a list of drawing operations (Figure 1) such as as “fill a rectangle”, “draw a line”, “draw an image” or “draw a piece of text”. This can be interpreted as a list of predefined foreground components with a drawing order as they may overlap. However, overlapping components may cause problems for layered document raster formats.

First, most of these formats (DjVu2 [1], ITU T.44 [6]) assume that the foreground is represented by one or several pixel maps, which are encoded as low-resolution color images (as the pixels are raster-ordered, this greatly simplifies printing operations). To simply paste the overlapping components on the foreground pixel map would create sharp changes in color that cannot be encoded at low resolution.

Even when, as this is done with the most recent versions of DjVu or Digipaper[4], the foreground can be represented as a list of components, overlapping components may not be desirable. The quality of the user experience with DjVu depends critically on the speed of browsing, zooming and panning through a document. Our experience in implementing a browser plug-in showed that non-overlapping foreground components enabled faster sub-sampling and rendering algorithms.

When the foreground is obtained through the segmentation of a scanned document, it is easy to ensure that these components are not overlapping. However, we found that the situation was much more complex with electronic documents. Several *naive* strategies are possible, e.g. to place all the text into the foreground and all the rest into the background. But the optimal approach would be, from this list of foreground components, to select only those (or part of those) which result in the best compression. This paper first presents the *naive* approaches and gives examples that defeat them. The second section presents the *perimeter ratio criterion* and the corresponding segmentation algorithm. The third section discusses implementation issues. Experimental results are presented in the last section.

2 Naive approaches

In this section, three *naive* algorithms are considered to select which components go in the foreground.

All text in foreground A naive segmentation consists in placing all the text into the foreground and all the other details into the background. This approach often fails because printing software must often navigate around printer limitations, sometimes using low-level primitives to draw characters, sometimes using text primitives to render images. Even if we assume that all text can be properly identified, this approach handles complicated documents such as geographical maps poorly.

All monochromatic components in foreground Most drawing operations simply assign a solid color to a set of specified pixels called the *component shape*. Another naive segmentation would place all these monochromatic components into the foreground layer, even though that should not necessarily be the case. Let us consider again the example of a geographical map. Some large solid color components represent the ground, the vegetation type or the sea. Rivers, roads, names and other symbols are overlaid on them. Coding these large components in the foreground would be costly because the corresponding foreground mask components must describe their visible part only and therefore need to be carved precisely to avoid the overlaid roads, rivers, names and symbols. These large components on the other hand would be very easily encoded in the background layer as large color patches and accurately delimited by the boundaries of the overlaid objects.

First drawn component in background Neither is it sufficient to assign the first drawn components to the background layer and all the remaining components to the foreground layer. Roads might be rendered by first drawing a fat black segment and then drawing a thinner red segment over the center of the black segment. The resulting image shows a red road surrounded by two black edges resulting from the occlusion of the fat black segment by the thinner red segment. Logically, they belong to the black segment and are drawn before the red segment. However, they would be better encoded as foreground objects drawn over a red background.

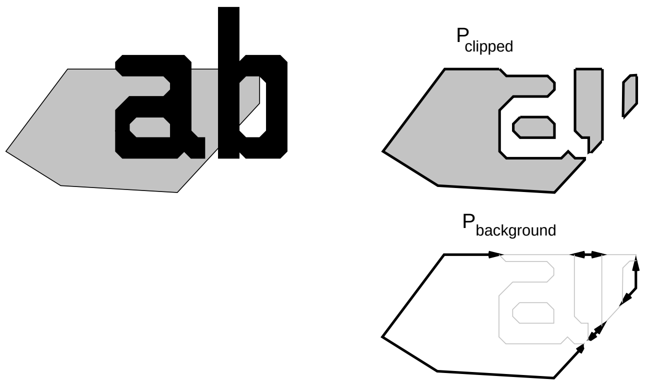

Figure 2: A polygonal component is partially occluded by two letters. The perimeter of its visible part is named P_{clipped}. The length of the perimeter segments that do not result from the occlusion is named $P_{\text{background}}$.

3 Component Classification

This section describes the segmentation algorithm for deciding whether each monochromatic component belongs to the foreground or the background. This segmentation criterion is based on the MDL principle[5]: each decision is made to minimize the overall coding cost. This coding cost is the sum of the number of bits necessary to encode the image (*the encoding bit cost*) and the number of bits necessary to encode the discrepancy between the encoded image and the original image (*the discrepancy bit cost*).

Comparing the true bit costs for each decision would be very expensive as it would require coding both the foreground and the background layers and measuring the resulting file size and quality *for each possible decision*. The proposed criterion relies instead on bit cost estimates derived from simple measurements on the page components. This is summarized in the following steps and illustrated in Figure 2.

A component classified as foreground must be encoded with high spatial resolution. The foreground encoding bit cost is roughly proportional to the perimeter P_{clipped} of the visible part of the component (i.e. after removing the component shape pixels that are occluded by other page components). This foreground encoding scheme is assumed to be lossless (in DjVu, loss in the bitonal coding is hardly noticeable): there is no foreground discrepancy bit cost.

The background encoding scheme is optimized for continuous tone images and typically requires more bits to encode sharp transitions such as the component edges. However, the encoding cost for some of these sharp edges has already been paid for in the foreground: because DjVu background encoding scheme is able to reduce the bit rate allocated to masked parts of the background [2], there is no need to waste bits for encoding edges that arise from occlusions by foreground components and are already accurately defined by the boundary of overlapping components.

This background encoding scheme is lossy and most of the quality loss is concentrated along those edges of the component shape that touch other background components.

In summary, in the case of the background, both the encoding and the discrepancy bit costs are roughly proportional to the length $P_{\text{background}}$ of the perimeter segments that do not result from occlusions by foreground components. Furthermore, the proportionality coefficient depends on the color differences along the object boundary.

The proposed classification algorithm proceeds in a greedy way. First we prepare two empty bitmaps $\mathcal{F}$ and $\mathcal{B}$ representing the pixels currently classified as foreground and background. Then we perform the following operations on every monochromatic component starting from the topmost component and proceeding towards the bottommost component.

i) Determine the part of the component shape that is occluded by background components drawn above the current component. This is achieved by computing the intersection of the component shape and the current background $\mathcal{B}$. Remove these occluded pixels from the component shape.

ii) Determine the part of the component shape that is occluded by foreground components drawn above the current component. This is achieved by computing the intersection of the component shape and the current foreground $\mathcal{F}$. Remove these occluded pixels from the component shape.

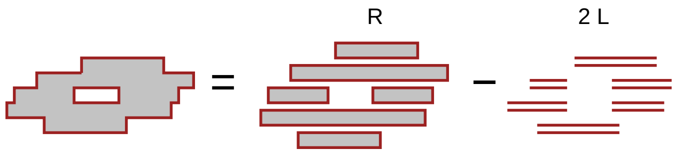

Figure 3: The perimeter of a run-length encoded bitmap is easily computed by adding the perimeters of each horizontal run and subtracting twice the length of the contact segments between runs located on adjacent rows.

iii) The component shape now contains only the visible pixels of the component. Compute its perimeter P_{clipped}. Compute the length $P_{\text{background}}$ of the perimeter segments that do not result from occlusions by foreground components. Estimate the color difference δ along the perimeter segments that do not result from occlusions by foreground components.

iv) Compute the ratio $\delta P_{\text{background}}/P_{\text{clipped}}$. If this ratio is smaller than a predefined threshold T, the component is deemed a background component and the clipped component shape is added to bitmap $\mathcal{B}$. Otherwise the component is deemed a foreground component and the clipped component shape is added to bitmap $\mathcal{F}$.

4 Implementation Issues

The proposed algorithm makes a large number of boolean operations between bitmaps (i.e. $\mathcal{B}$, $\mathcal{F}$ and the component shapes). Our implementation represents these bitmaps using run-length encoding and performs boolean operations in time proportional to the number of runs on the relevant scan lines.

The proposed algorithm also requires the quantities P_{clipped} and $P_{\text{background}}$. These quantities can be computed as a side effect of processing the component occlusions in steps (i) and (ii).

- Quantity P_{clipped} is simply the perimeter of the clipped component shape computed

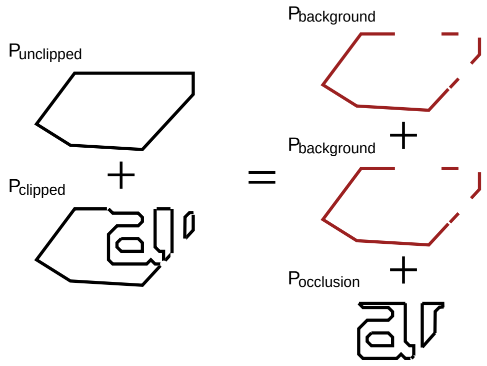

Figure 4: The sum of the clipped and unclipped perimeter is equal to twice the background contour plus the perimeter of the occluded part of the unclipped object. This equality provides a convenient way to compute the length of the background contour.

at step (ii) of the algorithm. The perimeter of a run-length encoded bitmap is also computed in linear time by making a single pass on the bitmap runs and simultaneously computing the sum R of the run perimeters and the sum L of the lengths of the contact segments between runs located on adjacent scan lines. As shown in Figure 3, the bitmap perimeter P is equal to $R - 2L$.

- Quantity $P_{\text{background}}$ is easily computed using the relation illustrated in Figure 4. It is sufficient to compute the perimeter $P_{\text{unclipped}}$ of the component shape after step (i) and the perimeter $P_{\text{occlusion}}$ of the bitmap representing the component shape pixels occluded by foreground objects. This bitmap is computed during step (ii) of the algorithm. The desired quantity $P_{\text{background}} = \frac{P_{\text{unclipped}} + P_{\text{clipped}} - P_{\text{occlusion}}}{2}$

5 Results

We implemented the proposed algorithm as a printer driver for the well known Ghostscript PostScript/PDF interpreter (`http://www.ghostscript.com`). This program produces foreground and background images for each page. These images are then encoded using the regular DjVu encoder (`http://www.djvuzone.org`).

Document	Type	Pages	PS/PDF	**PS2DjVu**
mask.ps.gz	LaTeX	10	400K	**78K (23s)**
paper2web.pdf	Book	327	4230K	**3424K (1235s)**
sgi.pdf	Flyer	4	484K	**106K (27)**
stanford.pdf	Map	1	412K	**170K (30s)**

Table 1: File sizes and compression times for four different documents. Results for the proposed segmenter at 300 dpi are given in the last column. These results can be compared with the initial `ps.gz` or `pdf` file sizes.

Figure 5: Test documents from `http://www.research.att.com/h̃affner/pdf2djvu`

Table 1 summarizes the results obtained on four very different documents at 300 dpi, shown in Figure 5. Files sizes compare favorably with the initial files, either compressed into PostScript or PDF. Visual quality matches the quality of the original document. From a user experience viewpoint, DjVu enables a much faster zooming and panning. DjVu, PostScript or PDF documents can be examined at `http://www.research.att.com/h̃affner/pdf2djvu`.

The same documents have also been encoded by first rendering each page as a 300 dpi image and then running the regular DjVu segmenter and encoder. This procedure produces files with comparable sizes but with much lower visual quality and after much longer encoding times (3 to 8 times depending on the document).

6 Conclusion

We propose a segmentation algorithm for electronically produced document images. This algorithm takes advantage of the structured information generated by typical document creation software. Such document images are typically decomposed as a list of drawing operations. Each component of this list is affected to either the foreground or the background layer according to a MDL criterion based on perimeter ratios. This MDL-principled approach only requires one threshold and considerably reduces the need for tuning.

Empirical results show that this algorithm provides an extremely robust and accurate segmentation. It does not rely on the textual or non textual nature of each image component. It is able to handle documents with complex layouts such as geographical maps. Using this segmenter with the DjVu system yields very high quality images whose size is smaller than the size of the initial compressed PostScript of PDF document.

References

[1] L. Bottou, P. Haffner, P. G. Howard, P. Simard, Y. Bengio, and Y. LeCun. High quality document image compression with DjVu. *Journal of Electronic Imaging*, 7(3):410–425, 1998.

[2] L. Bottou and S. Pigeon. Lossy compression of partially masked still images. In *Proceedings of IEEE Data Compression Conference*, Snowbird, UT, March-April 1998.

[3] P. Haffner, L. Bottou, P. G. Howard, and Y. LeCun. DjVu : Analyzing and compressing scanned documents for internet distribution. In *Proceedings of the ICDAR'99.*, pages 625–628, 1999.

[4] D. Huttenlocher, P. Felzenszwalb, and W. Rucklidge. Digipaper: a versatile color document image representation. In *Proceedings of the ICIP'99*, volume 1, pages 219–223, 1999.

[5] W. Niblack J. Sheinvald, B. Dom and D. Steele. Unsupervised image segmentation using the minimum description length principle. In *Proceedings of ICPR'92*, 1992.

[6] MRC. Mixed rater content (MRC) mode. ITU Recommendation T.44, 1997.