

FAST TEXT/GRAPHICS RESOLUTION IMPROVEMENT USING WAVELET BASED DENOISING AND CHAIN-CODE TABLE LOOKUP

Onur G. Guleryuz and Anoop Bhattacharjya

Epson Palo Alto Laboratory
3145 Porter Drive, Suite 104
Palo Alto, CA 94304
oguleryuz@erd.epson.com, anoop@erd.epson.com

ABSTRACT

We propose a fast text/graphics resolution improvement algorithm with boundary parameterization and wavelet based denoising. Given input images containing labeled text/graphics objects, local boundary segments on each object are traced, parameterized, smoothed, and subsequently rerendered at the desired resolution. All of the critical operations are delegated to lookup tables, resulting in an algorithm that is fast, computationally inexpensive, with low memory requirements. A very flexible framework is proposed which can be utilized in a variety of text/graphics applications requiring resolution improvement.

1. INTRODUCTION

In many applications digital material consisting of text and graphics that have been created at a particular resolution have to be printed/displayed at higher resolutions offered by printers and other high resolution devices. We will refer to a technique that achieves the necessary resolution conversion as a *resolution improvement algorithm* (RIA). Briefly, the aim of resolution improvement algorithms is to take digital images containing text and graphics rendered at a particular resolution and somehow produce the same image at a higher resolution with better quality. Figure 1 illustrates an example of RIA operation where the low resolution text and graphics showing obvious aliasing artifacts are converted to a higher resolution where quality is improved and artifacts are eliminated.

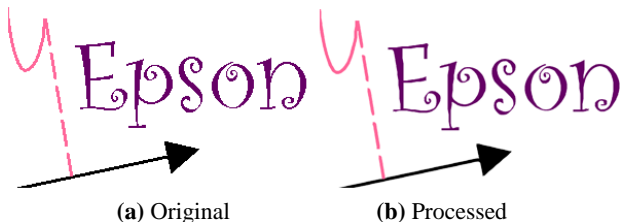


Fig. 1. Resolution Improvement.

There are various techniques for performing resolution improvement. Perhaps the most favorable scenario involves the use of scalable text and graphics, where text/graphics are generated in terms of various parameters (see for e.g. [1] and references therein). These parameters can be used to do very high quality rendering for a variety of resolutions. Hence, resolution improvement is simply achieved by passing the parameters to the high resolution device, which uses an appropriate rendering algorithm. However, in many cases (such as in some laser printing applications), the rendering of scalable text/graphics is typically done up

to dot per inch (dpi) resolution and further conversion to sub-dpi resolution may be necessary. Most importantly, in important text/graphics applications involving scanned documents, bitmap data, hand generated data, data on computationally simple devices such as some client/server applications and personal digital assistants, etc., one does not have access to scalable text/graphics and one must achieve resolution improvements by starting with the low resolution data itself (as opposed to starting with given parameters that enable scaling) and obtain higher resolution versions with quality improvements by using signal processing algorithms.

For text/graphics, classical interpolation techniques utilizing splines and other filters [2, 3], even when accompanied by subsequent sharpening, are in general unsatisfactory for high quality results. More elaborate techniques that account for discontinuities (see for e.g., [4] and references therein) are applicable to domains that are much broader than text/graphics, and typically produce less than optimal results on text/graphics images. These general techniques also do not result in fast algorithms necessary to tackle some of the aforementioned applications. Specifically for text/graphics images, one can employ approaches where patterns in the input are recognized and replaced with resolution improved patterns in the output via sophisticated pattern matching techniques utilizing large lookup tables designed by human operators. However, such techniques generally require large amounts of memory and significant amounts of computation for pattern matching. They are also encumbered by an important inflexibility since they require expert human operators that can design high quality output patterns.

In this paper we propose a fast text/graphics resolution improvement algorithm with boundary parameterization and wavelet based denoising. Given input images containing labeled text/graphics objects, the boundary of each object is traced, parameterized, denoised, and subsequently rerendered. All of the critical operations are delegated to lookup tables, resulting in an algorithm that is very fast and computationally inexpensive with low memory requirements. A very flexible framework is proposed which can be utilized in a variety of applications requiring resolution improvement. The proposed algorithm can be used to do resolution conversion to devices having higher resolutions and/or to devices having the same grid/dpi resolution but more color planes, sub-dpi resolution, etc. Application specific constraints are easily incorporated into the algorithm. Section 2 introduces the proposed algorithm. The three main parts of the proposed algorithm are discussed in Sections 2.1, 2.2, and 2.3. Section 3 considers simulation examples followed by Section 4 of generalizations and concluding

remarks.

2. MAIN ALGORITHM

Figure 2 presents a conceptual overview of the proposed algorithm. As shown, the objective of the algorithm is to convert the low resolution character depicted by the input image into a high resolution one depicted by the output image. This conversion is done by invoking the algorithm for each text and graphics pixel at the boundary of each text/graphics object (we assume that the input document is such that text/graphics pixels are labeled).

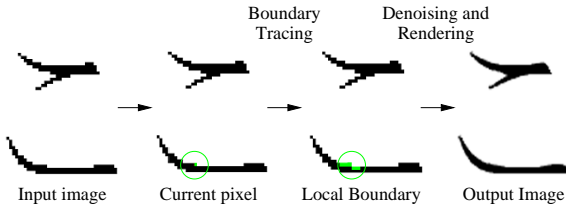


Fig. 2. Overview of resolution improvement.

The proposed RIA operates by smoothing the “jagged” boundaries of low resolution text/graphics and it can be summarized as follows:

1. For each text/graphics pixel that is on the boundary of the respective text/graphics object, determine the pixels that make up the local boundary segment (Section 2.1).
2. Parameterize and denoise/smooth the boundary segment using overcomplete wavelets (Section 2.2).
3. Establish rendering for the pixel with the aid of the smoothed boundary segment, resulting in resolution improved text/graphics objects (Section 2.3).

In short, the algorithm views the jagged boundary as a noisy version of an underlying smooth boundary. The traced local boundary segment is therefore parameterized in terms of pixel coordinates, these coordinates are denoised/smoothed using an overcomplete wavelet transform and finally, a smoothed boundary segment is constructed using the denoised coordinate data. The algorithm is thus composed of three stages given by the boundary tracing, boundary smoothing and final rendering.

All of the computationally complex operations are conveniently delegated to lookup tables (LUTs) for fast and computationally simple operation. The traced boundary is used to construct a chain-code which addresses the LUTs containing the results for subsequent stages. Furthermore, memory requirements for the algorithm and the sizes of the LUTs can be precisely controlled by adjusting the size of the pixel neighborhood in the construction of the boundary segment.

2.1. Boundary Tracing

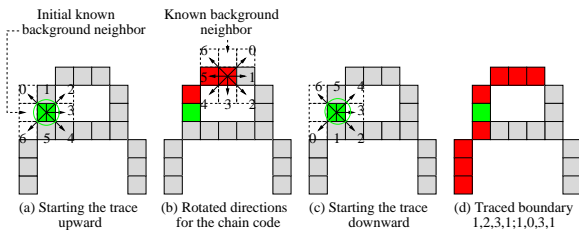


Fig. 3. Chain-code for the boundary segment.

Figure 3 illustrates an example trace of a boundary segment. The trace is carried out for each text/graphics pixel that lies on

the boundary of a text/graphics object. A text/graphics pixel belongs to the boundary of a text/graphics object if at least one of its 4-connected (or 8-connected, if desired) pixel neighbors does not belong to a text/graphics object. The current pixel serves as the initial point from which the trace starts in two directions, each producing N points for a $2N + 1$ point boundary segment¹. As shown in Figure 3 (a), the initial pixel has one or more non-text/graphics neighbors which we term “background” neighbors. Each one of these background neighbors is possibly associated with a different boundary segment (multiple boundary segments for a pixel are handled independently, in turn). For example, in Figure 3 (a) using 4-connectivity, there are two such neighbors: The left background neighbor is associated with the boundary segment making up the leftmost portion of the letter (as shown traced in Figure 3 (d)) and the right background neighbor is associated with the boundary segment making up the closed boundary segment making up the center of the letter (not shown). Given this initial pixel and say the left background neighbor, there are 7 directions which are searched *in turn* to decide the next pixel on the segment. The first text/graphics pixel found (as the directions are searched in turn) is determined as the next pixel on the boundary segment and the algorithm jumps to this pixel to continue the trace until N pixels are found. Observe that the *order* of the search is with respect to the background neighbor and the background neighbor is propagated from pixel to pixel during the trace: For example, after direction 1 is taken in the initial step onto the next pixel, the next background neighbor will remain to the left of the next pixel. Thus, the order of the boundary search will remain the same, i.e., the upper-left diagonal location will be searched first at the next iteration. However, depending on the traced boundary segment, this order may easily change as shown in Figure 3 (b), where the upper-right diagonal will be the initial point for the search.

Associating the order of the search with the background neighbor allows the algorithm to utilize locally relative directions which become important when the number of directions to search is constrained to be a number P ($P \leq 7$) for speed and memory requirements. As illustrated in Figure 3 (b), the search order “rotates” during the trace and the shown boundary can actually be traced by using only 4 distinct but relative directions² (i.e., $P \leq 3$ is enough for this example). Once the initial portion of the trace is concluded (upward with an upper-left diagonal starting point in Figure 3 (a)), the remaining portion of the trace is accomplished by starting with and propagating a symmetric order for the search (downward with a lower-left diagonal starting point in Figure 3 (c)). The final traced boundary consists of $2N + 1$ points ($N = 4$ in Figure 3), and can be indexed by $2N$ point chain-codes plus the one of 4 orientations for the boundary segment. As we will see, the boundary resolution improvement techniques we propose are insensitive to 4-fold rotations by 90° and a $2N$ chain-code (Figure 3 (d), with the semicolon separating the chain-codes for each direction) is sufficient to index the boundary segment.

2.2. Boundary Denoising

This step of the proposed algorithm is carried out off-line and only the results are stored in a LUT. The LUT is accessed using the

¹If N points cannot be found in either direction then the algorithm extrapolates additional points as needed in order to provide direct and fast access to LUTs using a fixed length chain-code.

²It is clear that there will be instances where a constrained number $P < 7$ of directions will not be sufficient to determine N points during either portion of the trace. In such cases, the algorithm will again provide an extrapolation.

index constructed from the chain-code of the traced boundary segment at run-time.

The chain-code for a given boundary segment is used to establish the order of the pixels in the segment by starting from one end of the chain and moving to the other. It will become clear below that in addition to 4-fold rotations by 90° , the boundary denoising algorithms are also not sensitive to the starting point in the chain. Prior to coordinate decomposition, the boundary segment is first upsampled by an amount U ($U \geq 2$) to form an upsampled boundary segment. The upsampling amount is chosen to correspond to the desired integer resolution conversion factor. However, due to a technical issue discussed below, even when the grid resolution is to remain unchanged, i.e., say when the algorithm will be used to take advantage of detailed color planes, grayscale, etc., of the display device, $U \geq 2$ is still enforced and a down-conversion is introduced later.

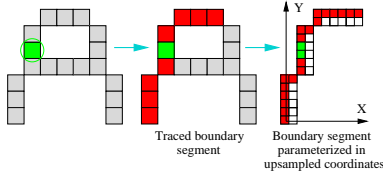


Fig. 4. Boundary segment parameterization.

As shown in Figure 4 ($U = 2$), the upsampled boundary segment is parameterized by establishing an orthogonal (X, Y) coordinate system. Each pixel in the segment is represented in terms of its coordinates, i.e., a M point segment results in M (X, Y) coordinates. The formed (X, Y) coordinates are considered as two independent sequences and are denoised using an overcomplete wavelet transform to form a new sequence (\tilde{X}, \tilde{Y}) of coordinates. The overcomplete wavelet transform consists of a wavelet transform and its overcomplete shifts [5]. Hence, for each sequence, denoising is established by transforming the sequence with each wavelet transform, thresholding the absolute values of the wavelet coefficients with a threshold T (hard thresholding), inverse transforming to obtain an intermediate sequence, and finally averaging the intermediate sequences corresponding to each wavelet transform to obtain the overall denoised result [5, 6]. Note that the independent coordinate parameterization and hard thresholding of wavelet coefficients ensures that the denoising stage is not sensitive to the starting point and earlier mentioned 4-fold rotations.

The reason for considering the upsampled version of the boundary segment (even for cases where the grid resolution remains the same) can be seen as follows: Suppose the original boundary segment corresponds to a 45° line. If we consider the X and Y coordinates of this line without upsampling, we end up with two sequences which are linear. Since both of these linear sequences are first order polynomials it is clear that they are smooth on their own right. For example, a wavelet transform evaluated over a signal that is a first order polynomial will produce zero valued wavelet coefficients (except for the trivial case of a Haar wavelet), indicating the high degree of smoothness of the signal. Hence, while the 45° line will suffer from significant aliasing artifacts and require RIA correction, there will be little or no correction affected by the proposed denoising technique with $U = 1$. On the other hand, when we incorporate $U \geq 2$ upsampling, the resulting boundary segment is no longer linear or smooth and will be handled correctly by the proposed algorithm.

The difference of the denoised coordinates and the original coordinates is determined as the differentials $(dx, dy) = (\tilde{X}, \tilde{Y}) -$

(X, Y) . The differentials are clipped to an appropriate level (1 for $U = 2$) with the assumption that the noisy boundary is within a half pixel of the original smooth boundary. The rendering stage only requires the differentials that correspond to the current pixel for which the local boundary segment is traced. With $U \geq 2$, the current pixel can be considered to be upsampled by U resulting in U^2 child pixels. Of course only a portion of the child pixels will belong to the upsampled boundary segment. Hence, only the differentials for the child pixels on the upsampled boundary that have the current pixel as a parent are required in the rendering stage.

2.3. Final Rendering

We will discuss the rendering stage primarily for the case where $U = 2$. Further generalizations can be found in Section 4. After boundary denoising one can think of the child pixels that reside on the boundary as having been “displaced” with the calculated differentials (dx, dy) . Since the algorithm will be repeated for all parent pixels on text/graphics boundaries, we are only interested in the displacement of the child pixels whose parent is the current pixel. Pixels that are not on text/graphics objects (background pixels) and non-boundary text/graphics pixels are handled at the final stage of rendering.

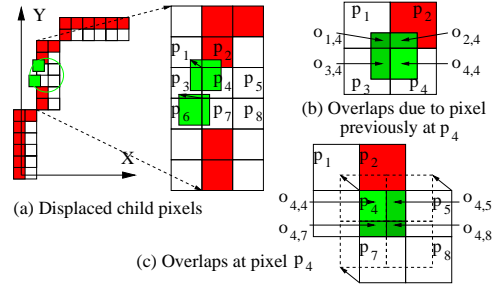


Fig. 5. Displaced child pixels and overlaps.

For the current pixel, the rendering stage amounts to rendering the child pixels to their new locations and rendering the correct information to the previous locations of the child pixels. Figure 5 (a) shows the displacement of the child pixels that are on the boundary as determined by the calculated differentials (dx, dy) . Superimposed on the figure is a grid of pixels labeled p_1, \dots, p_8 . As illustrated, the child pixel previously at p_4 has been displaced to overlap portions of pixels p_1, p_2, p_3 , and p_4 . The overlapping shown in the figure is in terms of simple box occupancy filters and as is discussed in Section 4, more sophisticated filters such as Gaussians can easily be incorporated into the algorithm.

As shown in Figure 5 (b), the rendering algorithm calculates the amount of overlap in pixels p_1 through p_4 caused by the displacement of the child pixel previously at p_4 . This results in an overlap factor $o_{k,4}$ at pixel k due to the displacement of pixel p_4 ($k = 1, \dots, 4$ in the figure). Similarly, as shown in Figure 5 (c), the algorithm also calculates the amount of overlap at pixel p_4 ($o_{4,l}, l = 4, 5, 7, 8$) assuming that the relevant pixels neighboring p_4 (p_5, p_7 , and p_8 in the figure, as determined via the differentials) move the same way p_4 does via a locally rigid motion assumption.

Let $c_i(p_4)$ denote the input color of the child pixel at p_4 with the understanding that the input color of a child pixel is the same as the color of its parent pixel in the input image. In the rendering stage we keep track of the output image color $c_o(\dots)$ as well as the percentage occupancy $a(\dots)$ of each child pixel ($a(\dots) = 0 \Rightarrow 0\%$ and $a(\dots) = 1 \Rightarrow 100\%$). We start the rendering stage by coloring all pixels in the output image with the color 0 and zeroing

out the percentage occupancies. The rendering *due* to the child pixel that was at p_4 modifies the subpixel p_1 variables by

$$c_o(p_1) = c_o(p_1) + o_{1,4} \times c_i(p_4) \quad (1)$$

$$a(p_1) = a(p_1) + o_{1,4} \quad (2)$$

and similarly for p_2 and p_3 , assuming $o_{k,l}$ are normalized to 0 – 1, corresponding to 0 – 100% overlap. The variables at p_4 are updated as

$$c_o(p_4) = c_o(p_4) + \sum_{l \in \{4,5,7,8\}} o_{4,l} \times c_i(p_l) \quad (3)$$

$$a(p_4) = a(p_4) + \sum_{l \in \{4,5,7,8\}} o_{4,l} \quad (4)$$

which corresponds to a locally rigid motion. This ensures that no holes are left at the locations of moving pixels. Observe that in Figure 5 (c), due to the assumed rigid motion, $o_{4,5} = o_{3,4}$, $o_{4,7} = o_{2,4}$, and $o_{4,8} = o_{1,4}$.

After all boundary child pixels are rendered in this fashion, the output image consists of child pixels with occupancy values that are greater than 100% and that are less than or equal to 100%. The final color at child pixels with occupancy greater than 100% is determined by normalizing by the corresponding occupancy, i.e., the color is divided by 2 if the occupancy is 2, etc. The final color at child pixels with occupancy less than or equal to 100%, including the background and non-boundary text/graphics pixels, is determined by

$$c_o(p) = c_o(p) + (1 - a(p)) \times c_i(p) \quad (5)$$

where p denotes the location of such a child pixel. This ensures that after resolution improvement text/graphics pixels blend smoothly into the background.

We have illustrated the rendering process using upsampled boundary segments ($U = 2$) and box occupancy filters. Depending on the desired output resolution factor the algorithm computes the necessary occupancy values, converts these to lower resolutions if necessary (as in the case where grid resolution is to stay the same and the algorithm will be taking advantage of the color planes, grayscales, etc., in the output device), and stores the results in the LUT off-line. In this fashion, given the chain-code for the boundary segment, the algorithm avoids all unnecessary calculations and directly renders the output image at the desired resolution.

3. SIMULATION EXAMPLES

Figure 6 illustrates the performance of the proposed algorithm on 12pt monochrome text. In order to illustrate the performance of the algorithm under very tight resource allocation, as would be the case in very simple hardware/software implementations, we have chosen the parameters in a very conservative fashion. For this example $P = 4$, i.e., only the first 4 directions are allowed in the boundary trace, $N = 3$, i.e., local boundary segments are limited to 7 pixels, and $T = 4$. Wavelets used in denoising are given by the Daubechies 7-9 biorthogonal bank. The conservative choice for P and N minimizes buffering requirements (necessary in the tracing and rendering stages) and allows for very fast operation. While the algorithm provides excellent performance even under these settings, we note that the performance does improve with increased P and especially N .

4. GENERALIZATIONS AND CONCLUSION

We have proposed a resolution improvement algorithm that meets the important memory, computational speed and computational

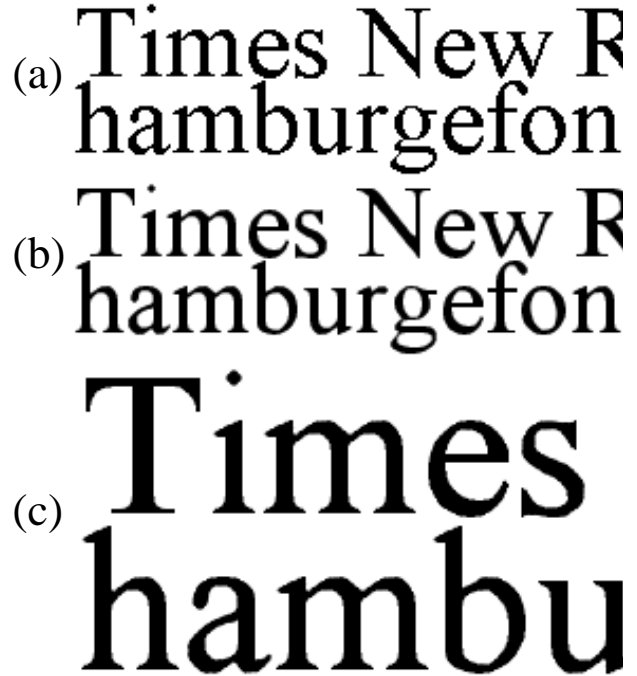


Fig. 6. Resolution Improvement: (a) 12pt monochrome text at 300 dpi, (b) resolution improved to 300 dpi plus grayscale, (c) resolution improved to 600 dpi plus grayscale.

complexity requirements we have outlined. The rendering process is discussed for the specific case of $U = 2$. For higher values the algorithm is scaled in a straightforward fashion, however, attention must be given to ensure that holes are not created after displacing the boundary pixels. The algorithm can likewise be extended to handle non-integer resolution conversions by converting to a higher integer resolution followed by conversion to the desired resolution. Of course, the simple filters for occupancy and conversions to lower resolutions (if necessary) can be done with sophisticated filters and LUTs modified appropriately. One of the key benefits of the algorithm is its *direct* access to LUTs without requiring intermediate computationally significant search steps. The LUTs can also be designed to ensure application specific constraints such as serif and corner preservation for text using selective denoising based on input boundary segment classification, etc.

5. REFERENCES

- [1] C. Hu and R. D.Hersch, "Parameterizable Fonts Based on Shape Components," *IEEE Computer Graphics and Applications*, May/June, 2001, pp. 70-85.
- [2] H. H. Hou and H. C. Andrews, "Cubic splines for image interpolation and digital filtering," *IEEE Trans. Acoust. Speech and Signal Proc.*, vol. ASSP-26, no. 6, pp. 508-517, 1978.
- [3] A. K. Jain, *Fundamentals of Digital Image Processing*. Englewood Cliffs, NJ: Prentice Hall, 1989.
- [4] R. R. Schultz and R. L. Stevenson, "A Bayesian approach to image expansion for improved definition," *IEEE Trans. Image Processing*, vol. 3, pp. 233-242, May 1994.
- [5] R. R. Coifman and D. L. Donoho, "Translation invariant denoising," in *Wavelets and Statistics*, Springer Lecture Notes in Statistics 103, pp. 125-150, New York:Springer-Verlag.
- [6] P. Moulin and J. Liu, "Analysis of Multiresolution Image Denoising Schemes Using Generalized - Gaussian and Complexity Priors," *IEEE Trans. Info. Theory*, Vol. 45, No. 3, pp. 909-919, Apr. 1999.