First published in the Proceedings of the 16th European Signal Processing Conference (EUSIPCO-2008) in 2008, published by EURASIP

BINARY IMAGE INTERPOLATION BASED ON MATHEMATICAL MORPHOLOGY

A. Ledda[1], *H.Q. Luong*[1], *W. Philips*[1], *V. De Witte*[2], *and E.E. Kerre*[2]

[1]Ghent University–TELIN–IPI–IBBT / [2]Ghent University–Dept. of Appl. Math. & Comp. Science
[1]St. Pietersnieuwstraat 41, B-9000 Gent, Belgium / [2]Krijgslaan 281, S9, B-9000 Gent, Belgium
phone: + (32) 9 264.89.00, fax: + (32) 9 264.42.95, email: ledda@telin.UGent.be
web: http://telin.UGent.be/

ABSTRACT

This paper presents an interpolation technique "mmINTone" for binary images, such as logos, diagrams, graphs and cartoons. The technique removes the jaggies from a pixel-replicated image, using concepts from mathematical morphology. As a result, the edges are smoother and jaggy-free.

In this paper we describe the algorithm, show some results and compare it to existing interpolation methods as well as to a previously developed technique of ours, also based on mathematical morphology. We can conclude that mmINTone is considerably faster and similar in quality.

1. INTRODUCTION

Image interpolation techniques are used to convert a bitmap to a higher resolution. The easiest approach is to copy the existing pixel values to their new available neighbouring pixels on the high resolution grid. This is called *pixel replication* or *nearest neighbour interpolation* [1]. The result is poor and suffers from unwanted jagged edges, called *jaggies*.

Linear techniques like *bilinear* and *bicubic interpolation* convert the binary image to an upscaled greyscale image, which can be binarized afterwards. Here, the (weighted) mean of respectively 4 and 16 closest neighbours is calculated for the missing pixel value. Other linear methods use higher order (piecewise) polynomials, B-splines, truncated or windowed sinc functions, etc. Most of them create a greyscale image with extra artefacts, like blurring and/or ringing.

Adaptive or non-linear interpolation methods incorporate prior knowledge about images. Several approaches exist, such as edge-based techniques [2], restoration methods [3, 4], self-similarity techniques [5], or example-based approaches [6]. Our proposed method mmINTone can be classified as a restoration method. Adaptive methods still suffer from artefacts: their results often look segmented, bear important visual degradation in fine textured areas or random pixels are created in smooth areas.

In this paper we present an efficient non-linear binary interpolation technique based on mathematical morphology, "mmINTone" (*Mathematical Morphological INTerpolation* in One step). In short, mmINTone removes the jaggies from a pixel-replicated binary image, by changing the values of the corner pixels of those jagged edges. mmINTone performs very well and is also much faster than its predecessor mmINT [7]. This method uses several iterations to remove jaggies from a pixel-replicated image. Other existing techniques that use morphological concepts for interpolation or morphing can be found in e.g. [3, 8, 9, 10].

This paper is organized as follows: section 2 introduces the morphological techniques used for mmINTone, while section 3 describes the algorithm. Section 4 compares mmINTone to other methods, both visually and computationally. We will see that this technique outperforms other methods. Finally, section 5 draws some conclusions.

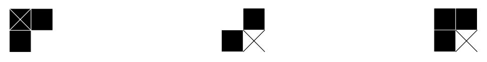

(a) Strel B (f.g.) (b) Strel C (b.g.) (c) Strel C' (b.g. alt.)

Figure 1: Upper-left corner detection with the hit-miss transform. Black squares: pixels of the strel; cross: origin of the strel. f.g.: foreground; b.g.: background.

2. CORNER DETECTION

Our interpolation technique is based on *mathematical morphology* [11]. A discussion of this theory can be found in many standard books about image processing.

Binary mathematical morphology is based on two basic operators: *dilation* and *erosion*. They are defined in terms of a *structuring element* (short: *strel*); this is a small window with an origin (the position of the current image pixel) which is moved over the image as in a linear filter and the pixels within the mask are used to compute a value for the output pixel.

In our algorithm, we use the morphological *hit-miss transform*. Two non-intersecting structuring elements are needed: one for the erosion of object pixels (B), and one for the erosion of background pixels (C). With this transform it is possible to detect specific shapes in the image.

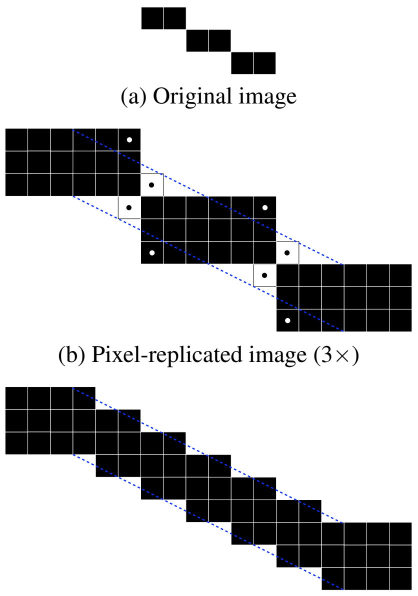
For example, the upper-left corner of an object can be detected by eroding the image by strel B in fig. 1(a), and by eroding the complement of the image by strel C (fig. 1(b)). The intersection of both erosions is a set that shows the locations of upper-left corners in the image. We call it a *corner map*. Similarly, other corners (upper-right, lower-left and lower-right) can be detected using rotated versions of the structuring elements B and C.

3. ALGORITHM DESCRIPTION

In this section, we explain the mmINTone method. Its purpose is to remove the jagged edges from a pixel-replicated image (only integer magnification factors) in one step, by swapping specific pixels from the foreground values to the background values and vice versa. We consider the most frequent value in the image to be the background.

Different steps can be distinguished in the algorithm:

1. **Pixel replication**: First, the image is pixel-replicated by an $M \times M$ square of the same values. The resulting im-



(a) Original image

(b) Pixel-replicated image (3×)

(c) Interpolated image

Figure 2: The jagged edges have to be removed, by replacing the values of specific pixels. The dotted lines show the orientation of the original line (a). The dots indicate jagged corners, obtained after step 3 in the algorithm.

age contains strong staircase patterns because of the pixel replication.

2. **Corner detection**: Using a combination of hit-miss transforms, mmINTone determines the positions of corners, both real and false (due to jaggies) in the image.
3. **Corner validation**: Some corners found in the preceding step are *real corners*, which have to be retained in the interpolated image. For example, consider a pixel-replicated drawing of a house containing walls, windows, a door and a roof. The corners of the walls, windows and door are real corners, as they represent the shape of the objects. The corners detected on the roof, on the other hand, are *jagged corners*, because the ideal roof is a slanted line. The aim of corner validation is to distinguish jagged corners from real ones.
4. **Pixel swapping**: We swap the values of pixels classified as jagged corners, and the values of some of their neighbours.

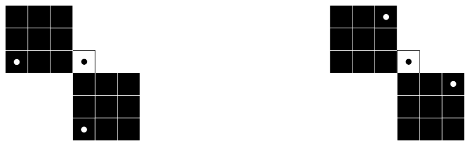
In order to illustrate the details of our method, we consider the case of enlarging an object consisting of a thin line of foreground pixels (see fig. 2(a)).

3.1 Pixel replication

The pixel-replicated result is clearly jagged (fig. 2(b)). The dotted lines show the ideal boundaries of the magnified line. The ideal solution would be to replace all background (white) pixels between the dotted lines with foreground (black) pixels and to replace all foreground pixels outside the dotted lines with background pixels. mmINTone aims to do just that.

3.2 Corner detection

As illustrated in fig. 2(b), the jaggies can be removed by altering the pixel values at the locations of object and background



(a) Complements *across* the edge (b) Complements *along* the edge

Figure 3: Complementary corners: the background corner (black dot) has different complementary corners (white dots) at specific relative coordinates. Magnification $M = 3$.

corners. The positions of these corners can be found with the morphological hit-miss transform as explained before, using the structuring elements in fig. 1 (and rotated variants). We not only look for corners of the objects, but also for corners in the background. This way, we will have 8 corner maps (4 *object* corner maps and 4 *background* corner maps).

If we use the same structuring elements for the detection of object corners and background corners, then artefacts will occur at lines with barely touching pixels, i.e., pixels with the same value that are 8-connected but not 4-connected. Two object corners and two background corners are found at such pixels. If all those corner pixels change value, then holes are introduced [7]. To overcome this connectivity problem, the structuring element shown in fig. 1(b) is replaced by the one shown in fig. 1(c) in the case we detect foreground corners: for the detection of an upper-left corner, not only the pixel values to the left and above the current pixel are then investigated, but also the neighbouring pixel at the upper-left.

3.3 Corner validation

In the corner validation step, we distinguish between real and jagged corners, by searching for one or more *complementary corner pixels* in a local neighbourhood, and this for every detected corner pixel, in each of the 8 corner maps. A complementary corner of a detected object corner c_o is a corner in the background (or vice versa) that lies at specific relative coordinates w.r.t. c_o.

Fig. 3(a) shows a pixel-replicated line (magnification $M = 3$) with a detected background corner (black dot) and two complementary corners *across* the edge at a distance of M pixels. They all are of the same type (here: lower-left corners). Complements *along* the edge are located at a Manhattan distance of M pixels, as can be seen in fig. 3(b) ($M-1$ pixels along the edge, 1 pixel in orthogonal direction). These complements have the opposite orientation (here: upper-right foreground corners and lower-left background corner).

The existence of a complementary corner suggests the presence of a jaggy, and therefore all corners without any complementary corner will be removed from the corner map. After this step in the algorithm, the corner maps will only contain jagged corners (the real corners are now removed), so the pixel swapping (the next step in the algorithm) will only occur on jagged edges.

3.4 Pixel swapping

The pixel swapping, step 4 in the algorithm, is the restoration part. The length of the *plateau* (a step in the staircase pattern, see fig. 4) of a jagged line (f.g. or b.g.) determines the number of pixels whose value will be changed at that location. In fig. 2 for example, the plateaus in the pixel-replicated image

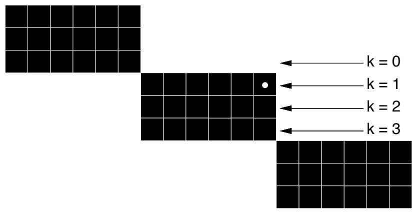


Figure 4: A plateau consists of several layers. The figure shows an $M = 3$ magnified image. This plateau is $2M$ pixels long.

are $2M$ pixels long, with M the magnification. Two object pixels should change to background and two background pixels should change to object value, which results in M smaller plateaus with length $\frac{2M}{M} = 2$. In general this becomes:

$$\text{length}(P) = nM \quad \Rightarrow \quad \text{length}(P_i) = n\ , \tag{1}$$

where P is the initial plateau (in the pixel-replicated image), n is a strictly positive integer value, and P_i are the plateaus that replace P after interpolation.

A plateau consists of several *layers* k (see fig. 4). Layer 1 is the layer of the detected corner pixel (the white dot in the figure). The next layer lies below the first layer. Note that a corner pixel can be part of two plateaus, a vertical one and a horizontal one. The layers of a vertical plateau are oriented columnwise.

Detecting the end of the plateau, and thus its length, is simple: scan the plateau, starting from the detected (and validated) jagged corner c_0 (the white dot in fig. 4), and check the following conditions:

- Has the pixel under investigation the same value as the corner pixel (the starting point c_0)?
- If so, has the pixel that lies "above" (layer 0 in fig. 4) the plateau the opposite value of the corner pixel?

If a pixel does not satisfy these two conditions, it lies next to the end of a plateau.

How the pixels are swapped, is defined by the magnification, the length of the plateau, and a distinction is made between odd and even magnifications.

Taking eq. (1) into consideration, we can derive for each layer k how many pixels Δ_k need to change value. For a magnification M by an even factor, the situation is a little more complicated than for a magnification by an odd factor. Object pixels are treated differently from background pixels, because it would otherwise lead to undesired interpolation results:

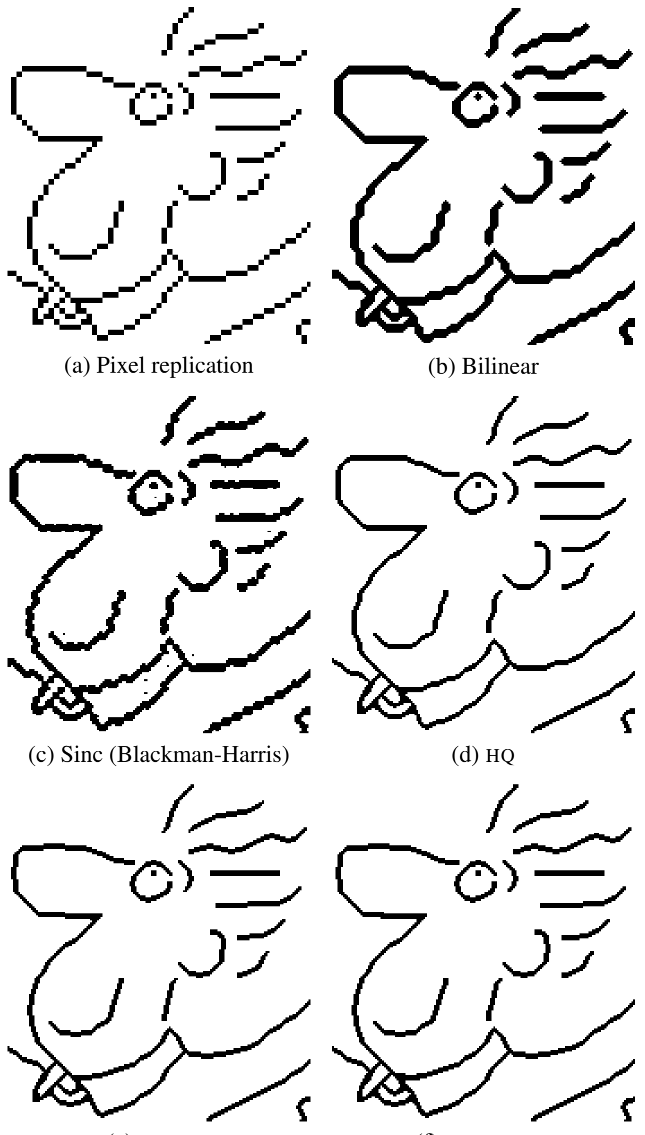
$$\text{F.g.} : \Delta_k = \frac{n}{2}(M - 2k + 1) - \frac{1}{2}(n \bmod 2)(M + 1 \bmod 2)\ ,$$
$$\text{B.g.} : \Delta_k = \frac{n}{2}(M - 2k + 1) + \frac{1}{2}(n \bmod 2)(M + 1 \bmod 2)\ .$$

n is a strictly positive integer value that defines the length of the plateau, nM. Index k is a strictly positive integer that depends on the magnification:

$$k \leq \left\lfloor \frac{M}{2} \right\rfloor .$$

The second term in the above equations is not present for an odd magnification, so the equation is the same for foreground and background. Note that if n is even, then these equations are equal to those for odd magnifications.

In the example illustrated in fig. 4, $k = 1$ is the only layer where the pixel values change. If $M = 3$ and $n = 2$, we can calculate that two pixels will change value, namely the detected corner pixel and its neighbour (with the same pixel value) on the same layer.



(a) Pixel replication (b) Bilinear

(c) Sinc (Blackman-Harris) (d) HQ

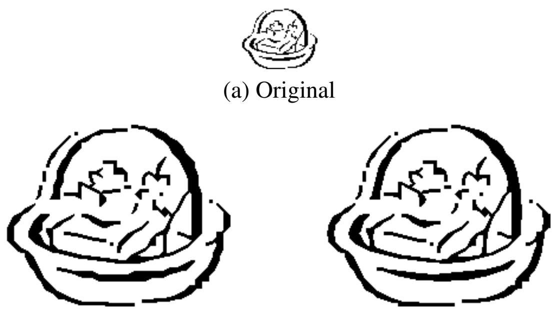
(e) mmINT (f) mmINTone

Figure 5: Results for different interpolation techniques (magnification $M = 3$).

4. RESULTS

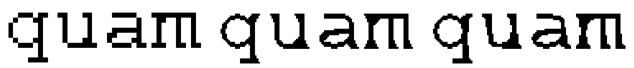
4.1 Visual comparison

mmINTone performs very well on binary images. Fig. 5(a) shows a 3× magnified image using pixel replication. Standard techniques, such as bilinear and sinc interpolation (their greyscale results were binarized using Otsu thresholding [12]), are not able to remove the jaggies when the image is magnified. While the technique HQ [6] is already very good,


(a) Original

(b) mmINT (c) mmINTone

Figure 6: mmINT vs mmINTone. Magnification $M = 3$.



(a) Pixel replication (b) mmINT (c) mmINTone

Figure 7: Interpolation of a text sample. Straight lines like those in the letter "u" are overinterpolated with mmINTone.

our techniques mmINT [7] and mmINTone perform even better. We notice smoother lines for the chin, nose and collar. None of the techniques can handle complex binary textures very well. In [7] we performed a psycho-visual experiment based on multi-dimensional scaling. mmINT scored best, except for text where HQ was better.

The visual difference between mmINTone and its predecessor mmINT is very small. This is because mmINTone follows a similar strategy as mmINT (i.e., removes jaggies from a pixel-replicated image), but using a faster non-iterative algorithm. The results are not identical. Fig. 6 shows the difference between interpolation with mmINT and interpolation with mmINTone. It seems that mmINTone does a slightly better job at smoothing the curved lines. In some situations, mmINTone is too ambitious: small protrusions can cause unwanted artefacts, because mmINTone only looks at the length of the plateaus. If the lines are very long, and the protrusions very small (compared to the lines), then the region of pixels that change values will expand more than with mmINT. This feature of mmINTone can be inconvenient for the interpolation of text (see fig. 7).

Table 1 shows the similarity values (in percent) for 7 interpolation techniques. We took 51 different binary images and scaled down 3 times (using downsampling, without filtering). The image similarities are only calculated in relevant regions, i.e., at edges. This region of interest (ROI) is defined by the morphological gradient (a dilation minus an erosion, using a 3×3 square strel) of the reference image. This edge image roughly defines the image pixels that might change due to interpolation. From the table, which shows the average similarity, we can conclude that the behaviour of the similarity is correlated with the visual quality. The average error on this measurement technique is about 0.015 %, i.e., only 15 out of 100 000 pixels that are different between the reference image and the interpolated image lie outside the ROI.

A PSNR calculation results in a similar trend, but such quality measurement technique is not well suited for binary images. Values lie between 18.13 dB and 19.96 dB.

Table 1: Average similarity calculation for 7 interpolation techniques (51 images).

Technique	**Similarity** (%)
mmINTone	80.78 ± 0.034
mmINT	80.37 ± 0.0043
HQ	79.29 ± 0.0028
Bilinear	76.28 ± 0.0022
Bicubic	76.08 ± 0.044
Sinc (B-H)	75.55 ± 0.020
Pixel replication	71.16 ± 0.00026

4.2 Speed comparison

We now test how much faster mmINTone is compared to mmINT, discussed in a previous paper [7]. While both methods are based on the hit-miss transform from mathematical morphology, there is a big difference between them: mmINT is an iterative procedure, where each iteration (on average 15 iterations are needed) removes more and more jaggies, using different (and larger) structuring elements for the hit-miss transform. mmINTone, on the other hand, is non-iterative.

Using mmINT can be time consuming when the input image is large and/or when a lot of lines are present in the image with "unfavourable" orientations. By unfavourable we mean lines with angles near 0° or 90°, because these are jagged lines with long plateaus that need many iterations before completely smoothened.

4.2.1 Artificial images

We take 16 artificial images (size 50×50) and interpolate them for different magnifications. Each image I_m ($m = \{1, 2, \ldots, 16\}$) contains lines under a specific angle. The angle of the lines in I_m is chosen in such way that mmINT would need m iterations for achieving optimal interpolation of I_m. The number of jagged corners in the images is kept constant (188). The timing experiments are performed on an AMD Athlon XP 2200+ (1.8 GHz, 1.5 GB RAM) running Linux, kernel v2.6.3. We used the `time` command in Linux for the calculation time of the process (sum of user and system time).

The processing time for mmINTone interpolation is independent of the image content, for a given magnification. This is logical, since all the necessary changes are made in one single step. On the other hand, the processing time depends on the magnification M. The averaged total calculation times for $4\times$, $11\times$ and $30\times$ magnification are respectively 0.098 s, 0.81 s and 5.1 s (see also fig. 8).

The processing time for mmINT interpolation increases polynomially with the number of iterations needed (this number depends on the image content). This increase is not only due to the number of iterations, but also due to the higher calculation cost for using larger structuring elements in the higher iteration steps. A plot is shown in fig. 8 for $M = 4$, $M = 11$ and $M = 30$. τ is the number of iterations needed. Notice the logarithmic scale of the ordinate. The dependency on τ is approximately cubic, i.e., $t \propto \tau^3$.

The processing time ratio mmINT/mmINTone ranges from 1.07 to 52.95 for $M = 4$, from 1.08 to 12.45 for $M = 11$, and from 1.37 to 31.39 for $M = 30$. The gain in speed when

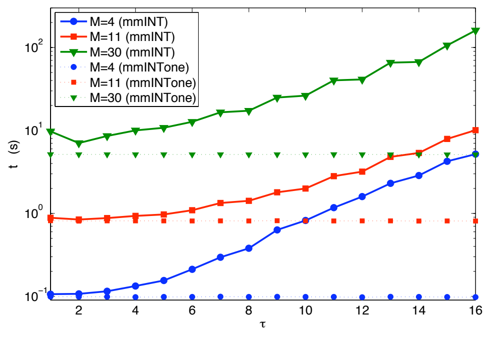


Figure 8: Calculation time needed for interpolating artificial images with different content, using mmINT for $M = \{4, 11, 30\}$. The calculation time ratio mmINT/mmINTone shows a similar behaviour. mmINTone is shown as reference.

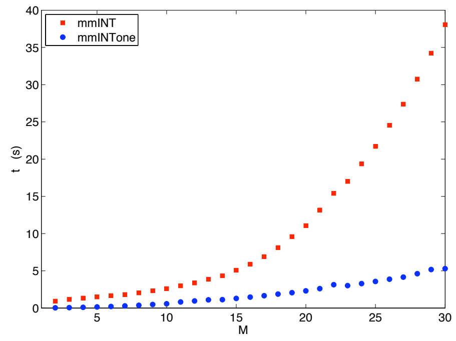


Figure 9: Calculation time needed for interpolating artificial images with different content. mmINT vs mmINTone.

using mmINTone instead of mmINT can thus be one or two orders of magnitude. The progress of the increases is similar to those shown in fig. 8.

Fig. 9 shows a plot of the average calculation time for the interpolation of the 16 artificial images with mmINT and mmINTone, from magnification 2 up to 30. By fitting a polynomial to the time measurements, we obtain a cubic relationship for mmINT ($t \propto M^3$) and a quadratic relationship for mmINTone ($t \propto M^2$).

With the hit-miss transform, $4\tau^2 + \tau + 4$ operations are needed for the detection of a corner pixel, which is small for mmINTone ($\tau = 1$), but can become large for mmINT.

4.2.2 *Realistic images*

We also take 33 realistic binary images, such as logos, cartoons, text and diagrams, with different sizes (ranging from 14×42 to 504×300 pixels) and different content (ranging from 4 to 38 needed iterations with mmINT). The timing experiments are performed on an AMD Athlon XP 4000+ (2.41 GHz, 2 GB RAM, 64 bit) running Linux, kernel v2.6.11.

The average processing time ratios mmINT/mmINTone increase with the desired magnification. For $M = 4$, $M = 11$ and $M = 30$, the ratio is 1.5, 2.6 and 12, respectively. The ratio strongly depends on the number of iterations needed with mmINT (maximum ratios are 4.3, 17 and 128, resp., the smallest ratio is 1.0). We can fit this ratio linearly with the number of iterations needed with mmINT.

5. CONCLUSIONS

We presented a binary interpolation technique, mmINTone, based on the morphological hit-miss transform. The visual quality of mmINTone is better than that of other methods and comparable to that of mmINT. mmINTone tends to better round curved lines than mmINT, but, on the other hand, it can generate small artefacts at protrusions in the image.

While mmINT needs several iterations for optimal interpolation, mmINTone only needs one iteration. Therefore, we now have a method with a processing time independent of the image complexity, and several factors lower than that of mmINT. In some cases, a speed gain of more than a factor 100 can be obtained.

REFERENCES

[1] T.M. Lehmann, C. Gönner, and K. Spitzer, "Survey: Interpolations Methods In Medical Image Processing," *IEEE Transactions on Medical Imaging*, vol. 18, no. 11, pp. 1049–1075, 1999.

[2] J. Allebach and P.W. Wong, "Edge-directed interpolation," in *Proceedings of the IEEE International Conference on Image Processing ICIP'96*, Lausanne, Switzerland, 1996, vol. 3, pp. 707–710.

[3] A. Albiol and J. Serra, "Morphological Image Enlargements," *Journal of Visual Communication and Image Representation*, vol. 8, no. 4, pp. 367–383, 1997.

[4] H.Q. Luong, P. De Smet, and W. Philips, "Image Interpolation using Constrained Adaptive Contrast Enhancement Techniques," in *Proceedings of the IEEE International Conference on Image Processing ICIP'05*, Genova, Italy, 2005, pp. 998–1001.

[5] H. Honda, M. Haseyama, and H. Kitajima, "Fractal Interpolation for Natural Images," in *Proceedings of the IEEE International Conference on Image Processing ICIP'99*, Kobe, Japan, 1999, vol. 3, pp. 657–661.

[6] M. Stepin, "hq3x Magnification Filter," `http://www.hiend3d.com/hq3x.html`, 2003.

[7] A. Ledda, H.Q. Luong, W. Philips, V. De Witte, and E.E. Kerre, "Image Interpolation using Mathematical Morphology," in *Proceedings of 2nd IEEE International Conference on Document Image Analysis for Libraries (DIAL'06)*, Lyon, France, 2006, pp. 358–367.

[8] V. Chatzis, "A Generalized Fuzzy Mathematical Morphology and its Application in Robust 2-D and 3-D Object Representation," *IEEE Transactions on Image Processing*, vol. 9, no. 10, pp. 1798–1810, 2000.

[9] M. Iwanowski, "Morphological Binary Interpolation with Convex Mask," in *Proceedings of ICCVG*, Zakopane, Poland, 2002, pp. 360–367.

[10] V. De Witte, S. Schulte, E.E. Kerre, A. Ledda, and W. Philips, "Morphological Image Interpolation to Magnify Images with Sharp Edges," in *Lecture Notes in Computer Science, ICIAR'06*, Póvoa De Varzim, Portugal, 2006, vol. LNCS 4141, pp. 381–393.

[11] P. Soille, *Morphological Image Analysis: Principles and Applications*, Springer-Verlag, 2nd edition, 2003.

[12] N. Otsu, "A Threshold Selection Method from Gray-Level Histograms," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 9, no. 1, pp. 62–66, 1979.