Greyscale Image Interpolation using Mathematical Morphology

Alessandro Ledda[1], Hiêp Q. Luong[1], Wilfried Philips[1], Valérie De Witte[2], and Etienne E. Kerre[2]

[1] Ghent University, TELIN-IPI, St. Pietersnieuwstraat 41, B-9000 Gent, Belgium
ledda@telin.UGent.be
http://telin.ugent.be/~ledda/
[2] Ghent University, Department of Applied Mathematics & Computer Science, Krijgslaan 281, S9, B-9000 Gent, Belgium

Abstract. When magnifying a bitmapped image, we want to increase the number of pixels it covers, allowing for finer details in the image, which are not visible in the original image. Simple interpolation techniques are not suitable because they introduce jagged edges, also called "jaggies".
Earlier we proposed the "mmINT" magnification method (for integer scaling factors), which avoids jaggies. It is based on mathematical morphology. The algorithm detects jaggies in magnified binary images (using pixel replication) and removes them, making the edges smoother. This is done by replacing the value of specific pixels.
In this paper, we extend the binary mmINT to greyscale images. The pixels are locally binarized so that the same morphological techniques can be applied as for mmINT. We take care of the more difficult replacement of pixel values, because several grey values can be part of a jaggy. We then discuss the visual results of the new greyscale method.

1 Introduction

A bitmap, or raster graphic, consists of *pixels*, aligned on a grid. The scene in the image is described in terms of those pixels' values. In order to magnify a bitmap image, the same scene must be represented using more pixels, i.e., image interpolation is needed.

The simplest interpolation method is *pixel replication* or *nearest neighbour interpolation*; pixel values in the enlarged image are copied from the pixels at the corresponding position in the original image, or – if that position does not correspond to a pixel centre – from the pixels nearest to that position. Fig. 1 shows an example for 4 times magnification. The result contains unwanted jagged edges, called *jaggies*. For non-integer scaling, other artefacts such as aliasing are even more undesirable.

Other linear (non-adaptive) techniques are the *bilinear* and *bicubic interpolation* [1]. Here, the new pixel values are computed as the (weighted) mean of the 4 and 16 closest neighbours, respectively. Other non-adaptive methods

(a) Original image (b) 4× magnified (c) Interpolated (mmINT)

Fig. 1. Pixel replication (b) creates "jaggies"

use higher order (piecewise) polynomials, B-splines, truncated or windowed sinc functions, etc. They create a greyscale image even for a binary input and most of them introduce additional artefacts, e.g., blurring and/or ringing.

Adaptive or non-linear interpolation methods incorporate prior knowledge about images to achieve better interpolation results. *Edge-based techniques* follow the principle that no interpolation across the edges in the image is allowed or that interpolation has to be performed along the edges. Examples of these techniques are EDI [2], NEDI [3] and AQua [4]. *Restoration methods* use regularization methods or smoothing to limit interpolation artefacts. Some restoration methods use PDE-based regularization [5], isophote smoothing [6], level curve mapping [7] and mathematical morphology [8]. Our proposed method mmINT(g) (see fig. 1(c)) also belongs to the class of restoration interpolation techniques.

Some other adaptive methods exploit the self-similarity property of an image, e.g., methods based on iterated function systems [9]. Example-based approaches are yet another class of adaptive interpolation methods. They map blocks of the low-resolution image into pre-defined interpolated patches [10, 11]. 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 earlier work [12], we presented a non-linear interpolation technique for *binary input images* based on mathematical morphology, *mmINT* (*Mathematical Morphological INTerpolation*). The technique performed quite well and had the advantage of producing a binary output image. The basic idea of the method is to detect jaggies from a pixel replicated image and then iteratively correct them by inserting "corner pixels" in or removing them from those jagged edges, while taking care not to distort real corners.

In this paper we present a greyscale extension of mmINT. The new method is called mmINTg. The basic idea is to locally binarize the greyscale input image, in order to detect the corners of the jagged edges, and then change their pixel values so the edges become smooth, but still remain sharp and not blurred. mmINTg is backward compatible with mmINT, in the sense that the interpolation result of mmINTg on a binary input image is binary too.

The next section gives an introduction to mathematical morphology and the morphological hit-miss transform used in our algorithm. Section 3 summarizes the principles of mmINT [12], while section 4 describes the changes made for the greyscale extension (mmINTg). Then we discuss the visual results of mmINTg and we will conclude that this method is good at interpolating line drawings.

2 Theoretical background

2.1 Morphological operators

Mathematical morphology [13–15], originally developed as a theory for binary images, is a framework for image processing based on set theory. Morphological image processing can simplify images, preserving objects' essential shape characteristics, while eliminating irrelevant objects or smoothing their border.

Binary mathematical morphology is based on two basic operators: *dilation* and *erosion*. The dilation and erosion of a set A by a structuring element (short: strel) B are defined as:[3]

$$\begin{aligned} \text{Dilation} &: A \oplus B = \bigcup_{\mathbf{b} \in B} T_{\mathbf{b}}(A) \\ \text{Erosion} &: A \ominus B = \bigcap_{\mathbf{b} \in B} T_{-\mathbf{b}}(A) \ , \end{aligned} \tag{1}$$

with $T_{\mathbf{b}}(A)$ the translation of image A over the vector $\mathbf{b}$. Computing $A \oplus B$ and $A \ominus B$ amounts to sliding the strel over the input image and computing for each position an output pixel based on the contents of the strel and the contents of the corresponding part of A.

In general, a dilation results in an enlarged version of the input object. The net effect of an erosion is to shrink or erode the input object.

The structuring element B can be of any size or shape and is chosen depending on the image and the application. The origin of the strel is also important, as it states how the strel is positioned relative to the examined pixel.

2.2 The hit-miss transform

The *hit-miss transform* "$\otimes$" is a morphological operator used extensively in our algorithm. It is defined in terms of two disjoint structuring elements: one for the erosion of object pixels (B), and one for the erosion of background pixels (C). Its definition is:

$$A \otimes (B, C) = (A \ominus B) \cap (A^c \ominus C) \ , \tag{2}$$

with A^c the complement set of A.

With the hit-miss transform it is possible to detect specific shapes in the image. We will use it to detect corners. To detect the upper-left corners of an object, we use the structuring elements (a) and (b) of fig. 2. An alternative for strel C for the corner detection is the use of the structuring element (c) in fig. 2.

[3] A is the set of the coordinates of the foreground pixels (value 1) in an image. In the remainder of this paper, we will often simply refer to A as "the (binary) image".

(a) Strel B (foreground) (b) Strel C (background) (c) Strel C' (b.g. alternative)

Fig. 2. Upper-left corner detection with the hit-miss transform. Specific strels are used. The black squares are pixels of the strel; the cross is the origin of the strel

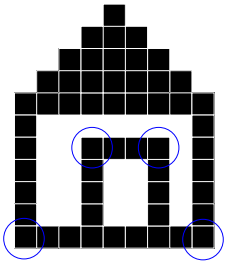


Fig. 3. The difference between "jagged corners" and "real corners" (encircled)

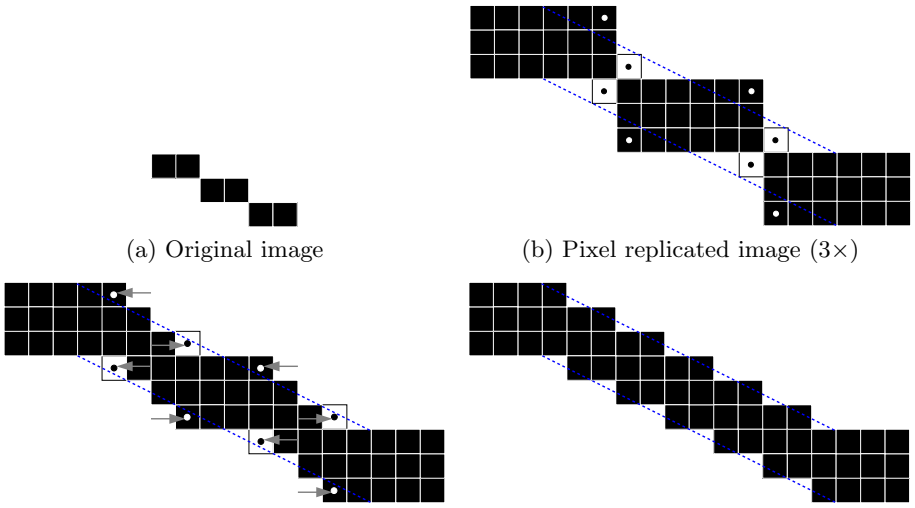
The result of the hit-miss transform is an image in which the foreground pixels indicate the position of the upper-left corners. Alternatively, the output can also be viewed as a set of corner coordinates. We further refer to this set as a *corner map*. In our method we apply the hit-miss transform several times, with rotated versions of the structuring elements in fig. 2 in order to produce four corner maps (upper-left, upper-right, lower-left and lower-right).

3 Binary interpolation scheme: mmINT

In this section we summarize the mmINT method presented in [12]. Its purpose is to remove the jagged edges from a pixel replicated image, by swapping specific pixels from background to the foreground and vice versa. We consider the most frequent colour 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 integer factor M. The resulting image contains strong staircase patterns because of the pixel replication (see for example fig. 1).
2. **Corner detection**: Using a combination of hit-miss transforms, the algorithm 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, the corners of the door and walls in fig. 3 are real corners. The corners detected on the roof are *jagged corners*, because the ideal roof is a diagonal line. The aim of corner validation is to distinguish false corners from real ones.



(a) Original image (b) Pixel replicated image (3×)

(c) Partial smoothing of jaggies (d) Interpolated image

Fig. 4. 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 show the pixels that will change value after interpolation

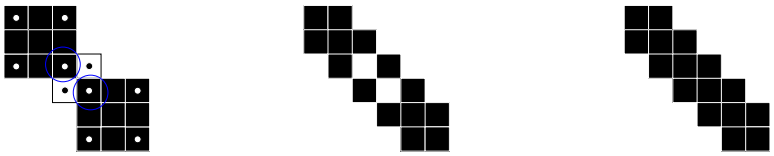
4. **Pixel swapping** (interpolation): We swap the colours of pixels classified as false corners, and the colours of some of their neighbours.

The above operations (except the first one) are repeated iteratively, each iteration operating on the output image from the preceding iteration.

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. 4). The result after pixel replication is shown in fig. 4(b) and is clearly jagged. 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. The iterative procedure aims to do just that.

As illustrated in fig. 4(b), the jaggies can be removed by altering the pixel values at the locations of object and background corners. The positions of these corners can be located with the morphological hit-miss transform as explained before, using the structuring elements from fig. 2. 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 (see fig. 5 ($M = 3$)), i.e., pixels with the same value that are only connected by 8-connectivity. Two object corners and two background corners are found at such 8-connectivity. If all those corner pixels change value, then holes are introduced



(a) Before pixel swapping (b) Without compensation (c) With more strict strels

Fig. 5. (a) Barely touching pixels (encircled) could give (b) artefacts after interpolation. (c) The use of more strict strels for the detection of object corners prevents this

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

Fig. 6. Complementary corners: the background corner (black dot) has different complementary corners (white dots) at specific relative coordinates. Magnification $M = 3$

(fig. 5(b)). To avoid these artefacts, the structuring element shown in fig. 2(b) is replaced by the one shown in fig. 2(c): for the detection of an upper-left corner, not only the pixel values to the left and above the current pixel are investigated, but also the neighbouring pixel at the upper-left. This new strel is only used for the detection of a foreground corner, because otherwise no interpolation will take place at all at lines with barely touching pixels.

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 an 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. 6(a) shows a pixel-replicated line (magnification $M = 3$) with a background corner and two complementary corners across the edge at a distance of M pixels. Complements along the edge (fig. 6(b)) are located at $M - 1$ pixels in one direction and $1 + (\theta - 1)(M - 1)$ in the other direction (with θ the iteration step). The existence of a complementary corner indicates the presence of a jagged edge, and thus a corner with at least one complementary corner will not be removed from the corner map.

In the pixel swapping step, we change the value (0 or 1) of the pixels (and surrounding neighbours) that are detected as corners of a jagged edge. Which pixels exactly need to change, is defined by the n times dilation of the corner maps with the structuring element B, with B shown in fig. 8(a) for the upper-left corner map. $n = \lfloor \frac{M}{2} \rfloor - 1$ for odd magnification M; for even M, $n = \lfloor \frac{M}{2} \rfloor - 1$ for

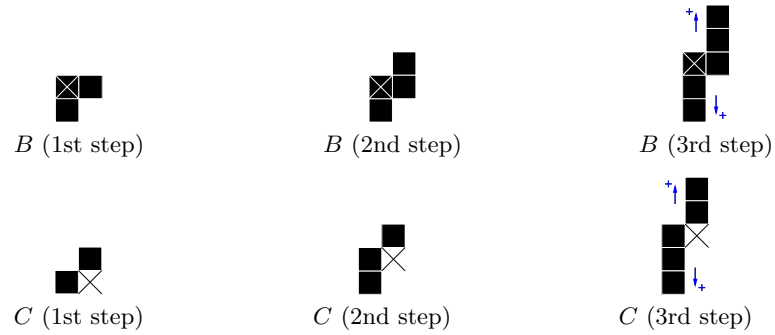


Fig. 7. The hit-miss structuring elements are different for every iteration step

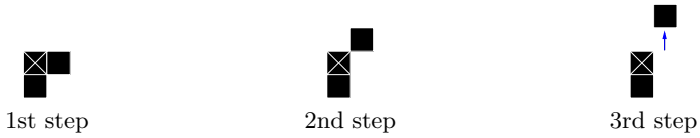


Fig. 8. The interpolation structuring element is different for every iteration step

the background and $n = \lfloor \frac{M}{2} \rfloor - 2$ for the objects in the odd iterations, and the situation is reversed in the even iterations. If $n < 0$, then no pixels are swapped.

At this point, lines with angles other than 0°, ±45° or 90° are not yet completely smooth. In our example, fig. 4(c) shows the result of the first iteration step of our method, while we want to obtain fig. 4(d). Therefore the algorithm is repeated until all appropriate changes have been made (on average 15 times), using different (and larger) structuring elements in successive iteration steps for the hit-miss transform (see fig. 7) and pixel swapping step (see fig. 8). Notice that the structuring elements are less symmetric for $\theta > 1$, so the number of corner maps doubles because we also have to look for corners using the mirrored strels, besides rotated versions [12].

The interpolation result of fig. 1(a) using mmINT is shown in fig. 1(c).

4 Extension to greyscale: mmINTg

In this paper we introduce the greyscale extension of mmINT. In the binary case, only two possibilities exist: a pixel is part of the fore- or the background. Also, when we look for jagged edges, the result is a detection of a corner or no corner.

The greyscale case is more complicated: the classification of a pixel to the foreground or the background is not straightforward, which makes the detection of corners using the binary hit-miss transform more difficult. We therefore adapt the corner detection step (see section 4.1). For this purpose the pixels are locally

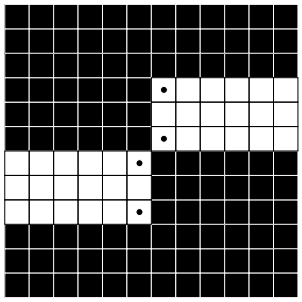


Fig. 9. If white is considered background, then no black corners will be detected, which results in a "real" edge

binarized, before applying the hit-miss operation. A majority ordering is used for the classification of a pixel as a foreground or a background corner.

In the pixel swapping step (see section 4.2), the values of the neighbouring pixels are taken into account to calculate the interpolated pixel value. This algorithm for greyscale interpolation is called *mmINTg*.

We will now discuss the new corner detection and pixel swapping algorithm.

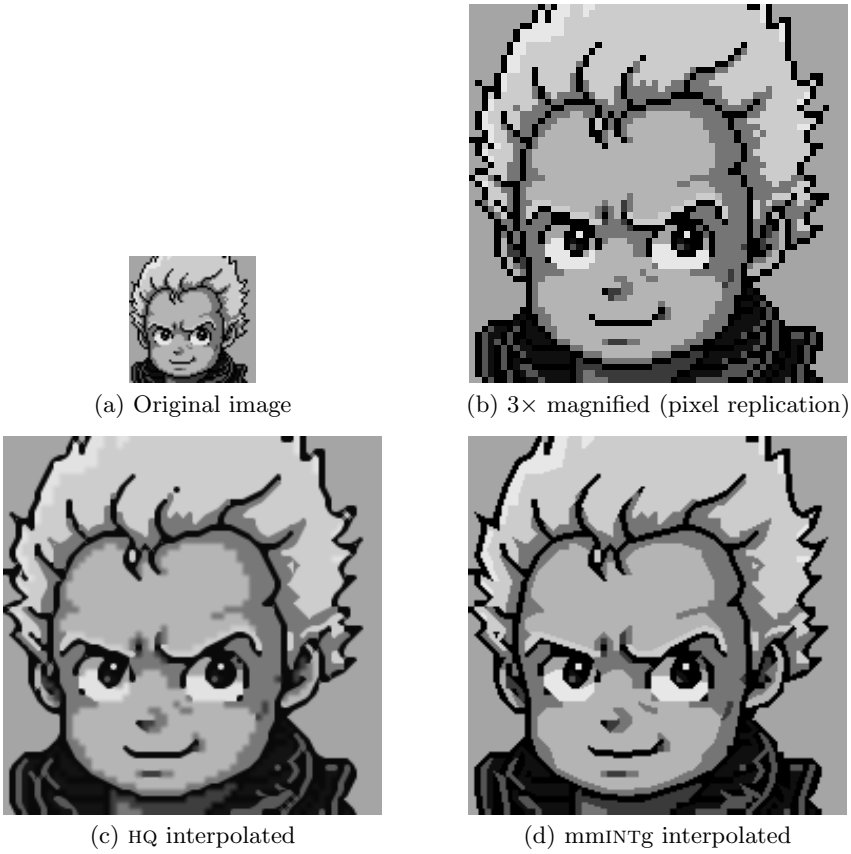
4.1 New corner detection method for step 2

Binarization. In order to apply the binary hit-miss transform to detect corners, we binarize the pixel values. A greyscale hit-miss transform would also be possible [16, 14], but then the result is still a greyscale image, while we wish for a binary result (i.e., corner or not). The binarization is done locally and only the pixels that are then covered by the structuring elements are taken into account (see fig. 2 for the strels used in the first iteration step). The threshold value $T(x, y)$ is defined by:

$$T(x,y) = \begin{cases} \frac{1}{2}(m+M) + \alpha, & \text{if } I(x,y) \geq \frac{1}{2}(m+M) \\ \frac{1}{2}(m+M) - \alpha, & \text{if } I(x,y) < \frac{1}{2}(m+M) \end{cases}, \tag{3}$$

with m and M respectively the minimum and maximum value of the set of pixels defined by the structuring elements, $I(x, y)$ is the grey value of the currently checked pixel, and α is a threshold for classifying more neighbouring pixels into a class different to the one of the current pixel. We have experimentally found that $\alpha = (M - m)/10$ is a good choice. The current pixel is always given the binary value 1, the value of the other considered pixels depends on their classification w.r.t. the current pixel.

Majority ordering. In section 3 we mentioned that different strels are used for foreground and background corner detection. Also, the corner validation step looks at complementary corners, which are part of the opposite class as the pixel



(a) Original image (b) 3× magnified (pixel replication)

(c) HQ interpolated (d) mmINTg interpolated

Fig. 10. The interpolation of a greyscale cartoon sprite

under investigation. This means that we need to classify the pixels as either possible object corner or background corner.

We utilize the *majority sorting scheme* (MSS) [17, 18] to order the grey values locally in function of their presence in a local window. If the grey value of the currently investigated pixel appears less than the grey values of the other pixels covered by the strels, then the pixel is considered foreground and strels (a) and (c) from fig. 2 are used for the upper-left corner detection.

The area in which we calculate an ordering map with the MSS is an 11×11 window in the original low-resolution image, since this size gives satisfying interpolation results. If the window is taken too small, then the ordering will be less accurate, because the probability to get the same counts for different grey values will become higher. If we take the window too large, then some pixels might not be interpolated. For example, fig. 9 shows a white line on a black background. We expect this line to be interpolated, but if in a larger window the white pixels are in the majority, then white is considered as background.

In this case, no interpolation will occur, since different structuring elements are used for object and background corner detection. There will only be detection of white corners, which will not pass the corner validation step, because there are no black corners detected in their neighbourhood.

The majority ordering only has to be performed in the first iteration step. From then on we know whether the pixel belongs to the fore- or background.

4.2 New interpolation method for step 4

In the binary case, the values of the jagged corner pixels and surroundings are replaced by the opposite colour, i.e., black becomes white and white becomes black. With greyscale images, we cannot simply swap the pixel values to the opposite grey value. The grey values of the surrounding pixels must be taken into account, so that the transition between grey values does not occur abruptly. The new pixel value is the average grey value of the pixels that are covered by the background hit-miss structuring element when positioned at the corner pixel (see fig. 2(b) for the strel used in the first iteration step). The resulting value is thus defined in function of the surrounding values of the other class. For a binary image, the effect is the same as with mmINT, since the average is taken of pixels of the same colour, a colour that is opposite to that of the current pixel. As a consequence, no blurring occurs.

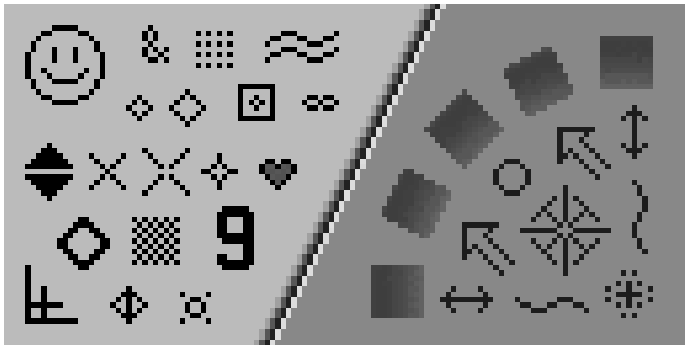
5 Results

mmINT is a technique that is very good at interpolating line art images, like logos, cartoons, maps, etc. Our proposed greyscale extension, mmINTg, also performs very well on this kind of images. When binary images are processed, the same results can be expected with mmINTg as with mmINT. When the MSS locally produces the opposite ordering as on the entire image (which is done for mmINT to define the background colour), this will lead to slightly different results. Situations like the ones in fig. 9 will be tackled by mmINTg.

The figures 10 and 11 show images with clear sharp edges. We compare our greyscale method with HQ, a technique that is very competitive with mmINT when interpolating binary images [10, 12]. In the first figure, we notice that mmINTg interpolates the lines better and the result is less blurred. The results shown in fig. 11 look very similar, except that the rotated squares look blurred with HQ. Also, HQ introduces too much unnecessary colours in the arrows at the right, i.e., when 8-connectivity lines are present.

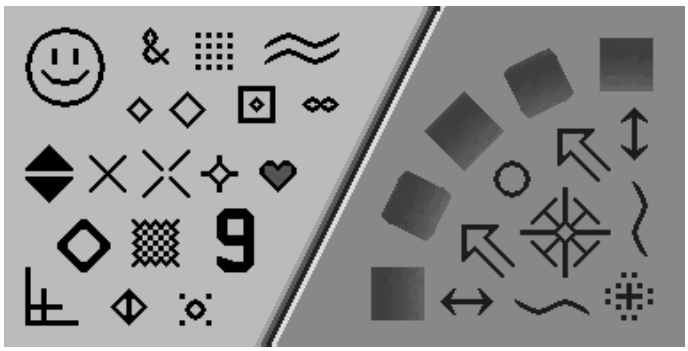
Fig. 12 shows a real life scene containing a lot of texture and grey value variation. mmINTg produces quite good results, but because the edges in the image are less sharply defined and more grey values are involved, the interpolation looks segmented, i.e., the grey value change after interpolation is too abrupt.

(a) Original image



(b) 3× magnified (pixel replication)

(c) HQ interpolated



(d) mmINTg interpolated

Fig. 11. The interpolation of a greyscale line graphic

(a) Pixel replication ($M = 4$) (b) mmINTg interpolated

Fig. 12. The interpolation with mmINTg of a real life scene (a cut-out of the Lena image)

6 Conclusions

We presented a greyscale solution for the binary morphological interpolation technique mmINT. A temporary local binarization of the grey values, combined with a local majority ordering, makes it possible to perform the morphological hit-miss transform on the image. The grey values of the pixels that need to change value, are defined in terms of neighbouring pixel values.

The visual quality of the new mmINTg is very good for cartoon sprites and line graphics. Its results are in most cases visually better than those of HQ.

References

1. Lehmann, T., Gönner, C., Spitzer, K.: Survey: Interpolations Methods In Medical Image Processing. IEEE Transactions on Medical Imaging **18** (1999) 1049–1075
2. Allebach, J., Wong, P.: Edge-directed interpolation. In: Proceedings of the IEEE International Conference on Image Processing ICIP '96. Volume 3., Lausanne, Switzerland (1996) 707–710
3. Li, X., Orchard, M.: New Edge-Directed Interpolation. IEEE Transactions on Image Processing **10** (2001) 1521–1527
4. Muresan, D., Parks, T.: Adaptively quadratic (AQua) image interpolation. IEEE Transactions on Image Processing **13** (2004) 690–698
5. Tschumperlé, D.: PDE's Based Regularization of Multivalued Images and Applications. PhD thesis, Université de Nice — Sophia Antipolis, Nice, France (2002)
6. Morse, B., Schwartzwald, D.: Isophote-Based Interpolation. In: Proceedings of the IEEE International Conference on Image Processing ICIP '98, Chicago, Illinois, USA (1998) 227–231

7. Luong, H., De Smet, P., Philips, W.: Image Interpolation using Constrained Adaptive Contrast Enhancement Techniques. In: Proceedings of the IEEE International Conference on Image Processing ICIP '05, Genova, Italy (2005) 998–1001
8. Albiol, A., Serra, J.: Morphological Image Enlargements. Journal of Visual Communication and Image Representation **8** (1997) 367–383
9. Honda, H., Haseyama, M., Kitajima, H.: Fractal Interpolation for Natural Images. In: Proceedings of the IEEE International Conference on Image Processing ICIP '99. Volume 3., Kobe, Japan (1999) 657–661
10. Stepin, M.: hq3x Magnification Filter. http://www.hiend3d.com/hq3x.html (2003)
11. Freeman, W., Jones, T., Pasztor, E.: Example-Based Super-Resolution. IEEE Computer Graphics and Applications **22** (2002) 56–65
12. Ledda, A., Luong, H., Philips, W., De Witte, V., Kerre, E.: Image Interpolation using Mathematical Morphology. In: Proceedings of 2nd IEEE International Conference on Document Image Analysis for Libraries (DIAL'06), Lyon, France (2006) 358–367
13. Serra, J.: Image Analysis and Mathematical Morphology. Volume 1. Academic Press, New York (1982)
14. Soille, P.: Morphological Image Analysis: Principles and Applications. 2nd edn. Springer-Verlag (2003)
15. Haralick, R., Shapiro, L.: 5. In: Computer and Robot Vision. Volume 1. Addison-Wesley (1992)
16. Ronse, C.: A Lattice-Theoretical Morphological View on Template Extraction in Images. Journal of Visual Communication and Image Representation **7** (1996) 273–295
17. Ledda, A., Philips, W.: Majority Ordering for Colour Mathematical Morphology. In: Proceedings of the XIIIth European Signal Processing Conference, Antalya, Turkey (2005)
18. Ledda, A., Philips, W.: Majority Ordering and the Morphological Pattern Spectrum. In: Proceedings of ACIVS. Volume 3708 of Lecture Notes in Computer Science., Antwerp, Belgium, Springer (2005) 356–363