FAST DIRECTIONAL IMAGE INTERPOLATION WITH DIFFERENCE PROJECTION

Zhiwei Xiong[1*], *Yonghua Zhang*[2], *Xiaoyan Sun*[2], *Feng Wu*[2]

[1]Department of Electronic Science and Technology, University of Science and Technology of China, Hefei, China
[2]Internet Media Group, Microsoft Research Asia, Beijing, China

ABSTRACT

This paper presents a new directional image interpolator, aiming to increase image resolution with high perceptual quality and low computational complexity. In our method, missing pixels in a magnified image are generated through linear interpolation on certain fixed supports to facilitate fast implementation, while local directional features are imposed on the adaptive interpolation weights which are determined by the gradients diffused from the low resolution image. Afterwards, a novel difference projection strategy is proposed to enforce the continuity of the magnified image by reusing the directional interpolator. Experimental results show that our method outperforms conventional bicubic and some existing adaptive interpolators, in terms of both the perceptual and quantitative quality.

Index Terms— Difference projection, directional interpolator, gradient diffusion, image interpolation

1. INTRODUCTION

There are many occasions that we want to increase the resolution of images to have better observation experience. Thus image interpolation finds its wide application in real world [1]. Basically, there are two criterions to evaluate the performance of an image interpolator: perceptual quality and computational complexity. Conventional bilinear and bicubic operators have advantages in simplicity and fast implementation, but they often introduce annoying "jaggy" artifacts around the edges because local directional features in images are not taken into consideration. Consequently, various adaptive image interpolators [2]-[4] have been designed to better preserve the edges by utilizing more accurate models. Still, there are some of the following disadvantages to overcome. First, the iterative or large-scale property of adaptive coefficient estimation often results in high computational complexity. Second, the edge orientations utilized are sometimes restricted to several predefined choices, which affect the accuracy of the imposed model. Third, many interpolators are limited to a 2× magnification, and it is inconvenient to achieve other magnification ratios.

Besides the complicated interpolators mentioned above, there are also some simple yet effective methods. For example, the 2× interpolator proposed in [5] generates a missing pixel only from its four nearest available neighbors, which are divided into two symmetric pairs centered at the missing pixel. The weight of each pair is inversely proportional to the difference between the two pixels in the pair. This interpolator can alleviate the "jaggy" artifacts along diagonal edges. However, edges in other orientations have not been discussed. Also, the continuities between original and interpolated pixels are not well reconstructed, which is an inherent deficiency of the first-order interpolation in our view.

In this paper, we develop a new directional adaptive and computational efficient interpolator. First, gradients from the low resolution image are diffused to the desired high resolution to determine the edge orientations at missing pixels in the magnified image. Second, linear interpolation is performed with position-fixed supports (i.e. involved available pixels) and gradient-adaptive weights. Third, the continuities between original and interpolated pixels are enforced by *difference projection*, which simply reuses the proposed directional interpolator. In our scheme, arbitrary edge orientations can be detected and utilized, non-iterative, linear algorithm requires low computational complexity and the continuity of the reconstructed image is improved.

The rest of this paper is organized as follows. Section 2 describes our proposed directional interpolator. The difference projection process is detailed in Section 3. Section 4 presents the experimental results and briefly analyzes the complexity of our algorithm. Section 5 concludes the paper.

2. PROPOSED DIRECTIONAL INTERPOLATOR

The proposed interpolator is to magnify a low resolution image **X** of size H×W into a high resolution image **Y** of size nH×nW. Without loss of generality, we assume n=3 for the illustration of our algorithm, yet it can be easily extended to other integer magnification ratios.

As depicted in Fig. 1, pixels in **Y** are divided into three categories. The black dots are the copies of original pixels from **X**; the gray dots and white dots are the missing pixels to be interpolated. Imagining four black dots form a square block (marked in dashed line), we call the gray dots *in-block* pixels and the white dots *on-block* pixels. In our scheme, the in-block interpolation is first performed in the square to generate the gray dots, and then the on-block interpolation is carried out in the hexagons (marked in dot-

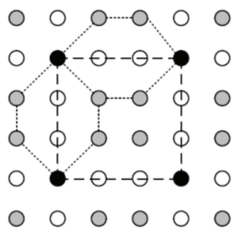

Fig. 1. High resolution image layout

[1] *This work was done during Z. Xiong's internship at Microsoft Research Asia.

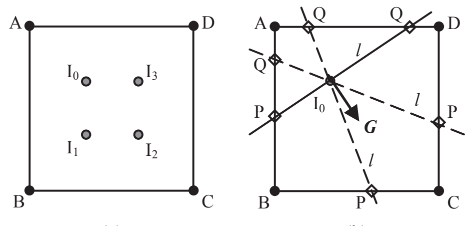

Fig. 2. In-block interpolation layout

TABLE I. IN-BLOCK INTERPOLATION WEIGHTS

k	(-∞, -1] ∪ [2, +∞)	[-1, 1/2]	[1/2, 2]
α_A	$(4k-2)/9k$	$(4-2k)/9$	1/3
α_B	$(2k+2)/9k$	$(2+2k)/9$	1/3
α_C	$(k-2)/9k$	$(1-2k)/9$	0
α_D	$(2k+2)/9k$	$(2+2k)/9$	1/3

TABLE II. IN-BLOCK INTERPOLATION SUPPORTS MIRRORING

I_0	A	B	C	D	k
I_1	B	A	D	C	$-k$
I_2	C	D	A	B	k
I_3	D	C	B	A	$-k$

ted line) to generate the white dots. We found this two-pass strategy can adequately exploit the correlations between neighboring pixels. Generally, a missing pixel U is interpolated as

$$U = (\alpha_1, \ldots, \alpha_N)(V_1, \ldots, V_N)^T \tag{1}$$

where $V_1, \ldots, V_N$ are the available neighbors of U in certain fixed positions, and $\alpha_1, \ldots, \alpha_N$ are directional adaptive weights determined by the estimated gradients at U.

2.1. Gradient Diffusion

Gradient diffusion is performed to estimate the gradients at missing pixels in the magnified image. Such gradients at individual pixels indicate the local directional features, which are thus involved in the later interpolation. To get the gradients on **Y**, we first calculate the gradients on **X** using the Sobel operator (denoted by **M**)

$$\mathbf{G}_h^{\mathbf{X}} = \mathbf{M} * \mathbf{X}, \quad \mathbf{G}_v^{\mathbf{X}} = \mathbf{M}^T * \mathbf{X}, \quad \mathbf{M} = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \tag{2}$$

$\mathbf{G}_h^{\mathbf{X}}$ and $\mathbf{G}_v^{\mathbf{X}}$ are the low resolution gradients in horizontal and vertical directions and * means convolution. The high resolution gradients $\mathbf{G}_h^{\mathbf{Y}}$ and $\mathbf{G}_v^{\mathbf{Y}}$ are then diffused from $\mathbf{G}_h^{\mathbf{X}}$ and $\mathbf{G}_v^{\mathbf{X}}$ using the bilinear interpolator (denoted by L)

$$\mathbf{G}_h^{\mathbf{Y}} = L(\mathbf{G}_h^{\mathbf{X}}) \uparrow \beta, \quad \mathbf{G}_v^{\mathbf{Y}} = L(\mathbf{G}_v^{\mathbf{X}}) \uparrow \beta, \quad \beta = 3 \tag{3}$$

2.2. In-block Interpolation

After the high resolution gradients are obtained, we first interpolate the four in-block pixels indexed as I_0, I_1, I_2 and I_3 in Fig. 2(a). Since they are symmetrical in geometry, we only take the interpolation of I_0 for illustration.

As shown in Fig. 2(b), we can draw a line l perpendicular to the gradient direction at I_0 (marked by arrow). Along l the local variation should be the minimum and thus the interpolation will be in this direction. With the gradients at I_0 we can easily calculate the slope of l, denoted as k

$$k = \mathbf{G}_v^{\mathbf{Y}}(I_0) / \mathbf{G}_h^{\mathbf{Y}}(I_0) \tag{4}$$

Then, l has two intersections P and Q with the square ABCD formed by the original pixels. As P and Q are situated on the square sides, they are generated by linear interpolation with the corresponding vertices; afterwards, they both generate I_0, again by linear interpolation. In short, we have

$$I_0 = (\alpha_A, \alpha_B, \alpha_C, \alpha_D)(A, B, C, D)^T \tag{5}$$

where the indices of pixels also refer to their intensity values. The interpolation weights are given in Table I and we omit the detailed deduction for brevity. Note that three sets of weights apply according to different k's, which are depicted in Fig. 2(b), respectively. Due to the geometric symmetry, the same weights can be used for I_1, I_2 and I_3 just by mirroring A, B, C, D and l. The permuted supports and rotated slope are presented in Table II.

2.3. On-block Interpolation

Now the in-block pixels are generated, we then interpolate the on-block pixels in the hexagons formed by both the original and the in-block pixels, as illustrated in Fig. 3. The procedure of the on-block interpolation is quite similar to that of the in-block interpolation, which can be finally formulated as

$$J_0 = (\alpha_A, \alpha_B, \alpha_C, \alpha_D, \alpha_E, \alpha_F)(A, B, C, D, E, F)^T \tag{6}$$

The interpolation details are listed in Table III and Table IV.

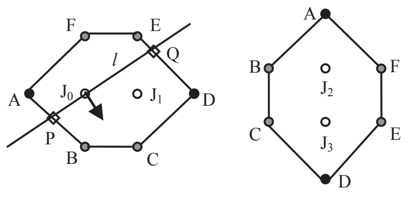

Fig. 3. On-block interpolation layout

TABLE III. ON-BLOCK INTERPOLATION WEIGHTS

k	(-∞, -1]	[-1, 0]	[0, 1]	[1, +∞)
α_A	$1/(1-2k)$	$2/(3-3k)$	$2/(3+3k)$	$1/(1+2k)$
α_B	$-(1+k)/(1-2k)$	0	$2k/(3+3k)$	$k/(1+2k)$
α_C	$1/(1-2k)$	$-2k/(3-3k)$	0	0
α_D	0	$(1+k)/(3-3k)$	$(1-k)/(3+3k)$	0
α_E	0	0	$2k/(3+3k)$	$1/(1+2k)$
α_F	$-k/(1-2k)$	$-2k/(3-3k)$	0	$(k-1)/(1+2k)$

TABLE IV. ON-BLOCK INTERPOLATION SUPPORTS MIRRORING

J_0	A	B	C	D	E	F	k
J_1	D	C	B	A	F	E	$-k$
J_2	A	B	C	D	E	F	$1/k$
J_3	D	C	B	A	F	E	$-1/k$

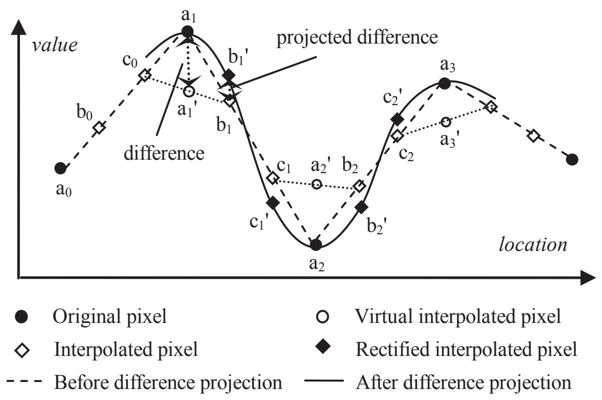

Fig. 4. Difference projection in 1-D case

Note that the gradient diffusion in our scheme is quite straightforward, only simple Sobel and bilinear operators are used. Moreover, once we get the gradients at the missing pixels, two passes of linear interpolation will reconstruct a high resolution image. The novelty of our method, however, lies in that we preserve the property of position-fixed interpolation supports as in conventional operators (e.g., bilinear and bicubic) while imposing the directional adaptivity onto the interpolation weights, which are dependent on a single parameter (the slope k) and extremely easy to compute (Table I – IV are prefixed). Different from existing methods that first select the interpolation supports and then calculate the interpolation weights, our designment greatly facilitates fast implementation when parallel computation is practical, such as in GPU and some hardware acceleration platforms.

3. DIFFERENCE PROJECTION

For image interpolators, first-order algorithm (including linear interpolation) is most computational efficient, especially when directional features need to be taken into consideration. However, since few pixels are involved, first-order interpolation bears a disadvantage that the continuities between original and interpolated pixels are not well reconstructed. In this paper, we propose a novel difference projection strategy to solve this problem.

We start from the 1-D case for simplicity. Still, the magnification ratio is assumed to be 3. As shown in Fig. 4, a_i (i = 0, 1, …) represent the original pixels, b_i and c_i are the linearly interpolated pixels from a_i

$$b_i = \mathbf{C}(a_i, a_{i+1})^T, \quad c_i = \mathbf{C}(a_{i+1}, a_i)^T, \quad \mathbf{C} = (2/3, 1/3) \tag{7}$$

It can be observed from the dash line that the first-order derivatives at a_i are not continuous. Mathematically, we may directly design interpolators that imply continuous derivatives, but in this work we consider another approach. That is, for better continuity, pixels a_i' are interpolated from b_i and c_i. We call the gap between a_i and a_i' *difference* and denote it as

$$\Delta a_i = a_i - a_i', \quad a_i' = (c_{i-1} + b_i)/2 \tag{8}$$

Since the original pixels a_i are generally supposed to be reliable, we project this difference back to b_i and c_i with the same linear interpolator

$$\Delta b_i = \mathbf{C}(\Delta a_i, \Delta a_{i+1})^T, \quad \Delta c_i = \mathbf{C}(\Delta a_{i+1}, \Delta a_i)^T \tag{9}$$

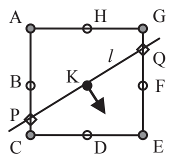

Fig. 5. Virtual interpolation layout

TABLE V. VIRTUAL INTERPOLATION WEIGHTS

k	(-∞, -1]	[-1, 0]	[0, 1]	[1, +∞)
α_A	-1/2k	-k/2	0	0
α_B	0	(1+k)/2	(1-k)/2	0
α_C	0	0	k/2	1/2k
α_D	(1+k)/2k	0	0	(k-1)/2k
α_E	-1/2k	-k/2	0	0
α_F	0	(1+k)/2	(1-k)/2	0
α_G	0	0	k/2	1/2k
α_H	(1+k)/2k	0	0	(k-1)/2k

Then the rectified pixels b_i' and c_i' can be easily deduced from (7)-(9) by adding back the projected differences

$$\begin{aligned} b_i' &= b_i + \Delta b_i = \mathbf{C}'(a_{i-1}, a_i, a_{i+1}, a_{i+2})^T, \\ c_i' &= c_i + \Delta c_i = \mathbf{C}'(a_{i+2}, a_{i+1}, a_i, a_{i-1})^T, \\ \mathbf{C}' &= (-1/9, 5/6, 1/3, -1/18) \end{aligned} \tag{10}$$

After difference projection, more original pixels contribute to an interpolated pixel and the overall continuity is improved, as shown by the solid line in Fig. 4. Actually, reusing a first-order interpolator on the difference equals to a higher-order algorithm. For example, the corresponding weights of cubic spline interpolation are $(-2/27, 7/9, 1/3, -1/27)$ in (10), which is very similar to our result.

In the 2-D scenario, the above difference projection can be directly applied by reusing our proposed linear directional interpolator, which can achieve similar continuity as higher-order algorithms. Note again that, we may directly design higher-order interpolators, but since directional features need to be involved, the designment can be quite complicated. Therefore, the consideration of computational efficiency suggests our approach.

The 2-D different projection works as follows. Suppose $\tilde{\mathbf{X}}$ is a high resolution image with original pixel copies from $\mathbf{X}$ and zeros at missing pixels. F represents our directional interpolator described in Section 2. We first get an intermediate result $F(\tilde{\mathbf{X}})$. To calculate the difference, a *virtual interpolation* is then performed at each original pixel according to (11) (also referring to Fig. 5 and Table V)

$$K = (\alpha_A, \ldots, \alpha_H)(A, \ldots, H)^T \tag{11}$$

Denote F^+ the concatenation of F and the virtual interpolation, the obtained difference $F(\tilde{\mathbf{X}}) - F^+(\tilde{\mathbf{X}})$ is then projected to the interpolated pixels by employing F again. Finally, our interpolation algorithm can be formulated as

$$\mathbf{Y} = F(F(\tilde{\mathbf{X}}) - F^+(\tilde{\mathbf{X}})) + F^+(\tilde{\mathbf{X}}) \tag{12}$$

Note that the projected difference is added back to $F^+(\tilde{\mathbf{X}})$, as the original pixels need to remain unchanged.

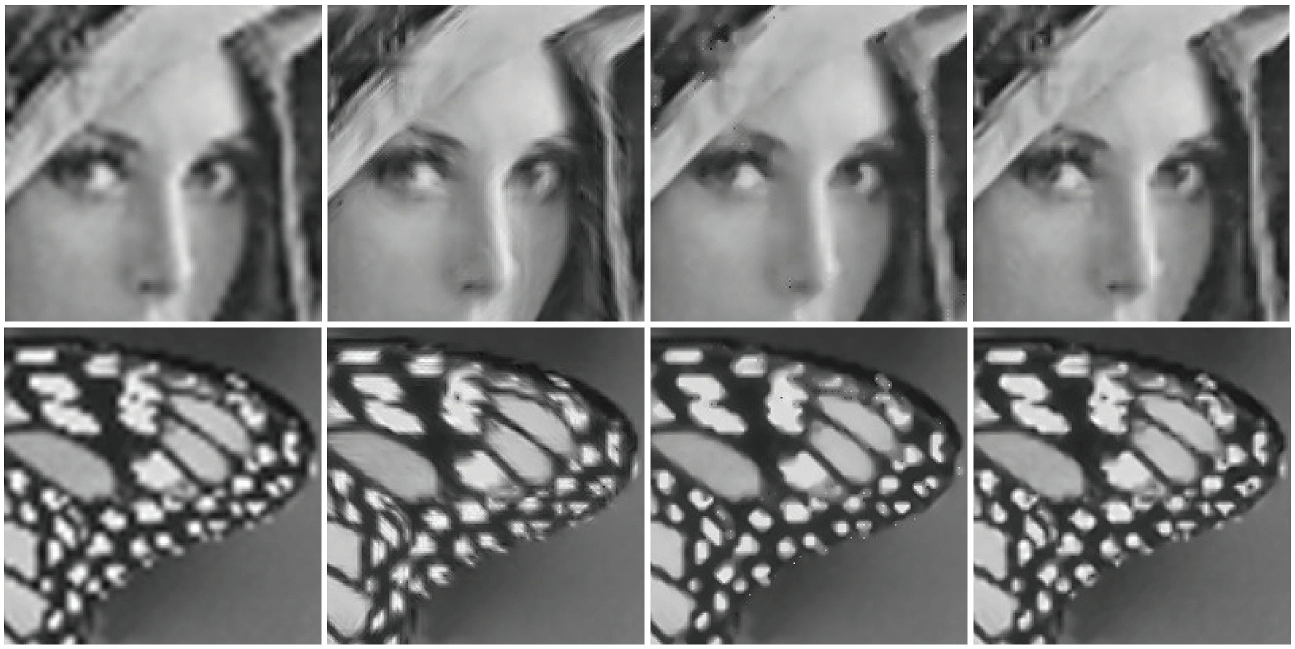

Fig. 6. Results of the *Lena* and *Monarch* image at 4× interpolation. From left to right: bicubic, NEDI [2], CAI [5] and proposed.

TABLE VI. PSNR(dB) Results of different Methods

Image	n×	Bicubic	NEDI	CAI	Proposed
Lena	3×	30.26	-	-	**30.33**
	4×	27.96	28.09	27.69	**28.20**
Monarch	3×	27.66	-	-	**27.82**
	4×	25.36	25.25	25.03	**25.41**
Mandrill	3×	20.17	-	-	**20.41**
	4×	19.43	19.58	19.57	**19.63**
Peppers	3×	27.93	-	-	**28.02**
	4×	26.82	26.89	26.86	**26.91**

4. EXPERIMENTAL RESULTS

We test the proposed interpolator on a variety of images, and compare its performance with three other interpolators: bicubic, new edge-directed (NEDI) [2] and content-adaptive (CAI) [5] (the last two are both 2× interpolators). The numerical results of several test images at 3× and 4 × magnifications are given in Table VI, where the low resolution images used are directly downsampled from the original ones. The 4× results of *Lena* and *Monarch* are shown in Fig. 6. It can be observed that bicubic introduces "jaggy" artifacts and gives the blurriest results. The other three adaptive methods give smoother edges. However, NEDI sometimes introduces unrealistic artifacts around the edges (e.g. the speckle of *Monarch*), whereas our interpolator gives more realistic results due to the simple yet effective designment. On the other hand, CAI presents many isolated original points which fuzzes the edges (e.g. the hat of *Lena*), and the proposed interpolator generates more continuous and clearer edges with difference projection. (Please see the electronic version for better visualization.)

Our method involves three procedures: gradient diffusion, in-block/on-block interpolation, and difference projection. The multiplication times required *per missing pixel* are 4.5 (0.5 for the Sobel operation on the low resolution image), 4 and 4.5 (0.5 for the virtual interpolation at original pixels), respectively. Besides, the adaptive interpolation weights cost another 3.5 multiplications (including 1 for calculating *k*). Contrastively, NEDI requires about 1300 multiplications per pixel, and CAI adopts exponential operations to calculate the weights. Therefore, the complexity of our interpolator is relatively low.

5. CONCLUSION

We present a fast directional interpolator in this paper, which can detect and utilize arbitrary edge orientations by gradient diffusion. Missing pixels in the magnified image are generated by position-fixed supports with gradient-adaptive weights through linear interpolation. Moreover, difference projection is introduced to enhance the continuity that the first-order interpolation lacks. Our algorithm has competitive efficiency compared with other adaptive methods, and it can be easily extended to any integer magnification ratios.

REFERENCES

[1] E. Meijering, "A chronology of interpolation: From ancient astronomy to modern signal and image processing," *Proc. IEEE*, vol. 90, no. 3, pp. 319–342, Mar. 2002.

[2] X. Li and M. T. Orchard, "New edge-directed interpolation," *IEEE Trans. Image Process*., vol. 10, no. 10, pp. 1521–1527, Oct. 2001.

[3] L. Zhang and X. Wu, "An edge-guided image interpolation algorithm via directional filtering and data fusion," *IEEE Trans. Image Process*., vol. 15, no. 8, pp. 2226–2238, Aug. 2006.

[4] Q. Wang and R. K. Ward, "A new orientation-adaptive interpolation method," *IEEE Trans. Image Process*., vol. 16, no. 4, pp. 889–900, Apr. 2007.

[5] T. Chan, O. C. Au, T. Chong and W. Chau, "A novel content-adaptive interpolation," in *ISCAS* 2005, pp. 6260–6263.