An Improved Demosaicing Algorithm*

Alexey Lukin, Denis Kubasov
Faculty of Applied Mathematics and Computer Science,
State University of Moscow, Russia
{ lukin, dkubasov } @graphics.cs.msu.su

In this paper, we present a new algorithm for the demosaicing of digital images, i.e. for the interpolation of bayer patterns. A review of existing demosaicing algorithms is made, and a new high-quality algorithm is proposed featuring significant improvements to interpolation with respect to visual quality and PSNR. The algorithm introduces several improvements upon a well known Kimmel method: more complex interpolation of the green component, adaptive control of the number of iterations, and projection on initial data.

Keywords: bayer pattern, mosaic, demosaicing, demosaicking, interpolation, Kimmel method, NEDI.

1. INTRODUCTION

Most modern digital photo and video cameras use a mosaic arrangement of photo-sensitive elements. This enables using only one matrix of photo-sensors instead of 3 matrices (one for each basic color component). In such a matrix, the elements sensitive to different basic colors are interleaved. They form a mosaic which is called the bayer pattern (fig. 1).

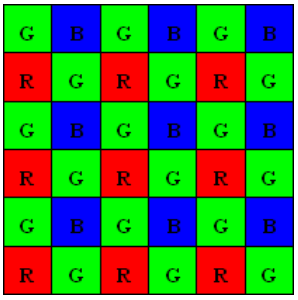


Fig 1. Bayer pattern

Thus, each element of the matrix stores the information on only one of 3 color components, whereas the output "full-color" digital image should contain all 3 basic components (R,G, B) for each pixel.

The problem of demosaicing involves the interpolation of color data to produce the "full-colored" image from the bayer pattern. The demosaicing algorithm interpolates each of color planes at the positions where the corresponding values are missing (fig. 2).

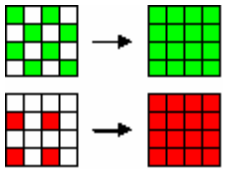


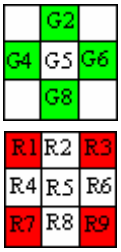
Fig 2. Interpolation of color planes

2. REVIEW OF EXISTING METHODS

2.1 Linear methods

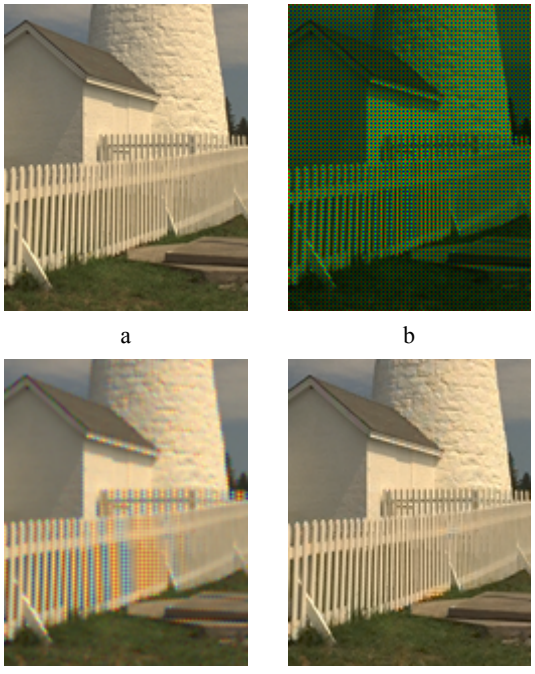
2.1.1 Independent interpolation of color planes

The simplest method of demosaicing just interpolates each color plane independently using some kind of interpolation algorithm (e.g. bilinear or bicubic interpolation), fig. 3. This is the fastest method, but it has the lowest quality.



1) Green:
   G5=(G2+G4+G6+G8)/4
2) Red:
   R2=(R1+R3)/2
   R4=(R1+ R7)/2
   R5=(R1+R3+R7+R9)/4
3) Blue:
   Similarly to red

Fig 3. Bilinear interpolation of a pixel in position 5



a b c d

Fig 4. Original image (a), bayer pattern (b), bilinear interpolation (c), and the proposed method (d)

* The paper is to appear in Graphicon'2004 conference proceedings.

One of the common artifacts – a color moiré (fig. 4-c) – is present in all demosaicing methods. It results from different space positions of different color sensors. Many demosaicing methods employ the fact that there are twice as many green pixels as red or blue pixels, in order to restore the high-frequency information of the image better. After that, the restored green component is used to interpolate red and blue components.

2.1.2 Color ratios interpolation

Interpolation of red and blue colors using the green color is based on some assumptions about correlation of color planes. One of possible assumptions states that ratios of basic color components (e.g. red/green or blue/green) remain equal within objects of the image [2]. Then, once we have interpolated the green components, we can interpolate ratios of red (or blue) to green (fig. 5) instead of interpolating red (or blue) colors on their own. This produces better results, compared to independent color plane interpolation, because the green component of a bayer pattern has higher sampling frequency and hence can be restored more accurately (even using a linear method). For interpolation of color ratios linear or other methods can be used.

The weak point of this algorithm is the image areas where the green color component is low. In these areas, the color ratios grow large and become sensitive to the noise. This problem can be addressed by interpolation of logarithms of the ratios (i.e. color differences) instead of ratios themselves.

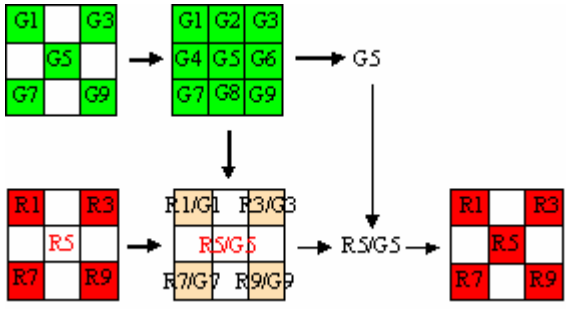


Fig 5. Color ratios interpolation

2.2 Adaptive methods

Adaptive methods switch different types of interpolation filters on a pixel-to-pixel basis depending on some heuristic or mathematical models of the local image area around the current pixel.

2.2.1 Edge-adaptive methods

The resulting quality of color ratios (or differences) interpolation heavily depends on quality of initial interpolation of the green component. It is desirable to improve the quality of interpolation of green color by replacing linear interpolation with edge-directional interpolation.

The simplest edge-directional interpolation algorithm calculates vertical and horizontal gradients using neighbors of the interpolated pixel [1] and assumes that the direction of the edge near this pixel is corresponding to the direction of a smaller gradient. After this, the interpolated pixel value is calculated as average of two pixel values in the direction of the edge (fig. 6).

1) Horizontal gradient: H=|G2-G4|
2) Vertical gradient: V=|G1-G5|
3) If H>V
   G3=(G1+G5)/2
   Else If V>H
   G3=(G2+G4)/2
   Else
   G3=(G1+G2+G4+G5)/4

Fig 6. Edge-directional interpolation. G1, G2, G4, G5 – given green pixels, G3 – interpolated green pixel.

This method can be improved by calculating gradients in a wider window around the current pixel, and by using other color components for estimation of the gradient [2].

After the green color component has been interpolated, the red and green components can be interpolated using color ratios algorithm.

2.2.2 Kimmel algorithm

A Kimmel demosaicing algorithm [3] includes 3 stages:

1. Interpolation of a green color.
2. Interpolation of red and blue colors using the interpolated green color.
3. Correction stage.

Let's describe each stage in details.

2.2.2.1 Interpolation of green color

The missing green pixel is calculated as a linear combination of 4 nearest neighbors of this pixel (the values of these neighbors are known). The weights E_i in the linear combination are calculated from probability that pixel G_i belongs to the same image object as pixel G_5 (fig. 7).

$$G_5 = \frac{E_2 G_2 + E_4 G_4 + E_6 G_6 + E_8 G_8}{E_2 + E_4 + E_6 + E_8}$$

E_i – weight function

Fig 7. Green color interpolation in Kimmel algorithm

The weights E_i are calculated as follows. Firstly, the concept of directional derivatives for 4 directions (vertical, horizontal, and 2 diagonals) from each point is introduced. Let's calculate the derivatives in the point P_5 (we use "P" notation instead of "R", "G", and "B", to show that calculation of derivatives doesn't depend on a particular color component given in the point P_5). The derivatives are calculated as:

$$D_x(P_5) = \frac{P_4 - P_6}{2},$$

$$D_y(P_5) = \frac{P_2 - P_8}{2},$$

$$D_{xd}(P_5) = \frac{P_3 - P_7}{2\sqrt{2}},$$

$$D_{yd}(P_5) = \frac{P_1 - P_9}{2\sqrt{2}},$$

where P_i is the intensity value in the corresponding point of the mosaic. It should be noted that whichever component is given in a point P_5, the derivatives are always calculated between intensities of the same color.

If P_5 stores the green value, then the derivatives can be calculated more accurately as:

$$D_{xd}(P_5) = \max\left\{\left|\frac{P_3 - P_5}{\sqrt{2}}\right|, \left|\frac{P_7 - P_5}{\sqrt{2}}\right|\right\},$$

$$D_{yd}(P_5) = \max\left\{\left|\frac{P_1 - P_5}{\sqrt{2}}\right|, \left|\frac{P_9 - P_5}{\sqrt{2}}\right|\right\}$$

Then the weighting function can be calculated as:

$$E_i = \frac{1}{\sqrt{1 + D^2(P_5) + D^2(P_i)}},$$

where $D(P_i)$ are the directional derivatives in $P_5 - P_i$ direction. For example: $E_3 = (1 + D_{xd}(P_5)^2 + D_{xd}(P_3)^2)^{-1/2}$.

2.2.2.2 Interpolation of red and blue colors using the green color

For interpolation of red and blue colors the previously described color ratios interpolation algorithm is used. The ratios are interpolated similarly to green pixels on the previous stage using the weights E_i defined earlier (fig. 8).

$$R_5 = G_5 \frac{E_1 \frac{R_1}{G_1} + E_3 \frac{R_3}{G_3} + E_7 \frac{R_7}{G_7} + E_9 \frac{R_9}{G_9}}{E_1 + E_3 + E_7 + E_9}$$

E_i – weight function

Fig 8. Red color interpolation in Kimmel algorithm

The formula for interpolation of blue color is quite similar.

2.2.2.3 Correction stage

Correction is a critical stage to the overall algorithm, because it suppresses most artifacts, such as color moiré. The main idea of correction is as follows. During the interpolation of red (or blue) color we assumed that the ratio of red (or blue) to green is constant within each image object. This means that the ratio of green to red (or blue) should also be constant in this area. Therefore, after the interpolation of red and blue colors we can correct green pixels so that this requirement is met. But such correction will alter the original ratios of red (or blue) to green, so they have to be corrected again. Authors of [3] suggest repeating these steps 3 times, adjusting the green and red/blue color planes alternatively.

So, the correction stage works as follows:

Repeat 3 times:

- Correct green pixels according to the green/red and green/blue ratios:

$$G_5^B = B_5 \frac{E_2 \frac{G_2}{B_2} + E_4 \frac{G_4}{B_4} + E_6 \frac{G_6}{B_6} + E_8 \frac{G_8}{B_8}}{E_2 + E_4 + E_6 + E_8},$$

$$G_5^R = R_5 \frac{E_2 \frac{G_2}{R_2} + E_4 \frac{G_4}{R_4} + E_6 \frac{G_6}{R_6} + E_8 \frac{G_8}{R_8}}{E_2 + E_4 + E_6 + E_8},$$

$$G_5 = \frac{G_5^R + G_5^B}{2}$$

- Correct red and blue pixels according to the red/green and blue/green ratios:

$$B_5 = G_5 \frac{\sum E_i \frac{B_i}{G_i}}{\sum E_i}, i \neq 5, \quad R_5 = G_5 \frac{\sum E_i \frac{R_i}{G_i}}{\sum E_i}, i \neq 5$$

- End of loop.

2.3 Mathematical methods

2.3.1 Optimal recovery

For high-quality interpolation of green color (which significantly influences the overall quality) some advanced image interpolation methods can be used, such as NEDI [7] or optimal recovery [5], which is an extension of NEDI.

Authors of [5] suggest replacing the edge-directional interpolation of green color in Kimmel algorithm by optimal recovery interpolation, and interpolation of color ratios – by interpolation of color differences.

The idea of optimal recovery is to extend the well known NEDI method by imposing some additional constraints on resulting values of pixels. This produces better results than NEDI does, because additional constraints allow suppressing the artifacts.

2.3.2 Alternating Projections

The method of alternating projections [6] belongs to a class of so called POCS (Projections on Convex Surfaces) methods. The main idea is iterative improvement of some initial approximation of the resulting image. The improvement is performed using adjustment of image to alternatively meet 2 classes of constraints ("projection of the image on constraint sets").

The first set of constraints requires that pixels of the interpolated image do not violate the original (given) mosaic data. In other words, the color components in the positions where they were specified by the mosaic shouldn't change.

The second set of constraints results from the assumption that the local high-frequency information (details) in all the color planes is similar. In the simplest case, we can require that details of all 3 color components in the surrounding of each pixel are strictly equal (this can be satisfied by copying of their high-frequency wavelet coefficients). In the method [6] it is assumed that high-frequency wavelet coefficients of red and blue colors should not differ more than by some predefined threshold from the coefficients of green color. If the color planes in the locality of the current pixel are close, then the threshold can be set low. If the significant difference between color planes is possible (or is admitted), then the threshold can be set higher. For grayscale images the threshold can be set to 0.

So, the outline of the algorithm is as follows:

- The initial interpolation of color planes using one of the existing methods (e.g. edge-adaptive interpolation of color differences).
- Repeat several times (or until converged):
  - Take the wavelet transform of all the color planes.
  - Change the high-frequency wavelet coefficients of red and blue colors so that they differ from coefficients of the green color not more than by a specified threshold.
  - Take the inverse wavelet transform.
  - Substitute the color components given in the original bayer pattern.
- End of loop.

2.4 Algorithm quality evaluation

A widely used metric for evaluation of bayer pattern interpolation quality is PSNR – Peak Signal to Noise Ratio. For PSNR calculation the original full-color image is artificially mosaiced (by throwing out 2 of 3 color components for each pixel) and then – demosaiced and compared with the original image using PSNR measure. There are different metrics for evaluation of difference between two images. For example, the color differences can be evaluated using perceptually uniform color spaces (e.g. ΔE_{00} [9]) or models of a human vision system. In this work, we will use a standard PSNR measure and ΔE_{00} measure, alongside with a subjective visual evaluation of image quality.

The most common artifacts of the interpolated images include: color moiré, zipper effect, loss of sharpness, and jagged edges [11].

3. THE PROPOSED ALGORITHM

Our algorithm includes the following stages:

1. A high-quality interpolation of green color using gradient interpolation and NEDI algorithms.
2. Modified Kimmel method with adaptively varied number of iterations.
3. Projection on source image data.
4. Iteration of the whole algorithm using the interpolated image.

Let's review each of these stages in details.

3.1 Green color interpolation

In our method of green color interpolation, we determine the direction and weights of interpolation only using green component, without reference to red or blue components. This choice is justified by the fact that red and blue components have lower sampling rate in a bayer pattern, and lower resolution can cause aliasing, which can result in a wrong selection of direction of the interpolation.

3.1.1 NEDI

The NEDI method for edge-directional interpolation has been proposed in [7] and advanced in [8] and [5]. NEDI features high-quality interpolation of edges without jagged edges artifact. However NEDI has several disadvantages as well: high computational complexity, watercolor artifacts in areas of fine texture, singularity of the algorithm in smooth image areas [10]. That is why we propose to hybridize NEDI with simpler edge-directional interpolation methods so that the artifacts of both methods are reduced.

In our method, we use a modification of NEDI proposed in [8] (smoothing of the lightness map, spatial aperture extension). Also, for improvement of stability of the algorithm in smooth image areas, we use the dithering of the lightness map with a white noise of small amplitude. This improves the stability of a matrix inversion operation involved in a calculation of interpolation weights in NEDI.

3.1.2 Gradient interpolation

A simpler interpolation method used in our algorithm is a gradient interpolation. In a basic method of gradient interpolation, we select between horizontal and vertical interpolation directions. The selection is based on a comparison of vertical and horizontal gradients of the green color component. The direction of lower absolute gradient is selected for interpolation (fig. 9).

1) Calculation of horizontal and vertical gradients H and V
2) If H≥V
   G3=(G2+G8)/2
   Else
   G3=(G4+G6)/2

Fig 9. Simple gradient interpolation of a green color

Each of the gradients is averaged within some area around the current pixel. On a first step, the horizontal gradient H is averaged within a small area:

$$D[i, j] = |G[i, j] - G[i, j+2]|$$

$$H = D[i, j-1] + D[i-1, j-2] + D[i+1, j-2] + D[i-1, j] + D[i+1, j] + D[i, j-3] + D[i, j+1] + D[i-2, j-1] + D[i+2, j-1] + (D[i, j-5] + D[i, j+3] + D[i-2, j-3] + D[i+2, j-3] + D[i-2, j+1] + D[i+2, j+1] + D[i-1, j-4] + D[i+1, j-4] + D[i-1, j+2] + D[i+1, j+2]) / 2$$

Here D[ ] is the absolute difference between two horizontally adjacent pixels.

Then the horizontal and vertical gradients are compared. If their values are relatively different, the direction of the interpolation is selected according to the lower gradient. On the other hand, if their values are relatively close, we decide that the direction of the interpolation is not well determined, and in this case the gradients are evaluated in a wider area (window):

$$\text{If } |H - V| < \beta \cdot (H + V) \text{ then Use bigger window}$$

Totally, 3 different window sizes are used. The small window, according to the formula above, is approximately 4x4 pixels. The medium window is 10x10 pixels. The larger window is approximately 22x22 pixels. After selection of the interpolation direction, the interpolation is performed using averaging of 2 adjacent pixels in the direction of the interpolation (fig. 9).

In a more complex modification of our method, the weights of the interpolation are calculated according to the formula on a fig. 10.

1) $E_2 = E_8 = 1/(\varepsilon + V^8)$
2) $E_4 = E_6 = 1/(\varepsilon + H^8)$
3) G3 is calculated as on fig. 7.

Fig 10. Gradient interpolation of a green color

A high power of gradients in the denominator minimizes the mixing of vertical and horizontal interpolation directions, yet allows such mixing in a case of very close values of the gradients. This reduces the color moiré artifact, often resulting from such mixing.

The next modification improves the quality of interpolation in smooth image areas. In such areas, the image noise influences the gradient values significantly and this can lead to watercolor artifact. At the same time, the simple bilinear interpolation does not produce this artifact. So, lowering of the power in the denominator of the mixing formula enforces more mixing of vertical and horizontal interpolation directions and makes the interpolation closer to bilinear interpolation. In our algorithm, this power in the denominator is lowered from 8 to 4 or 2, depending on the smoothness of the area.

The result of the gradient interpolation is the interpolated green component, and also the additional array storing the coordinates of pixels that required the maximal gradient window size to determine the interpolation direction. This array will be used further to determine the "problem" image areas, where the direction of the interpolation has been determined unstably.

3.1.3 Hybridization of gradient interpolation and NEDI

In our method of interpolation of a green component, we combine results of NEDI and gradient interpolation in order to reduce the artifacts of both approaches and to improve the speed of calculation (compared to NEDI alone), since NEDI will be now applied only to a relatively small part of image pixels. To achieve this, we use a special classification algorithm, which is applied to the resulting image after gradient interpolation of green color. This classification algorithm determines the mixing proportion between results of NEDI and gradient interpolation for each pixel. The purpose of the classification algorithm is separation of pixels onto 2 classes. The first class should include all the edges in the image. The second class should include smooth image areas, areas of high-frequency content (e.g. some fine repeating patterns), and areas of fine texture (e.g. leaves, grass, etc.) The NEDI algorithm will be used for interpolation of pixels from class 1. The gradient interpolation will be applied for the pixels of class 2, since NEDI will produce artifacts there.

Our classifier uses the following criteria to form the final decision:

1. *The presence of details near the current pixel.* This criterion is evaluated using a simple linear filter approximating the second derivative ((0, 1, 0), (1, -4, 1), (0, 1, 0)). This criterion allows classifying the smooth areas in the image as pixels of class 2.
2. *Ratio of high-frequency energy to low-frequency energy near the current pixel.* This criterion is evaluated using spectral amplitude coefficients of a windowed 2D FFT (the window size is set to 8x8 pixels). The energies of high-frequency and low-frequency coefficients are calculated. The low frequencies here are defined from 0 to 0.25 cycles per pixel. The zero frequency is not considered since it corresponds to the average block intensity and shouldn't influence the edge detection). This criterion allows classifying areas with predominantly high-frequency content (fine repeating patterns) to class 2 and classifying edges to class 1 (since edges contain predominantly low-frequency content).
3. *The complexity of spatial structure of the image near the current pixel.* This criterion is evaluated as follows. Firstly, the 8x8 block around the current pixel is translated into the locally optimal binary palette (using few iterations of the K-means clustering algorithm applied to pixel colors). Then, in the resulting 2-color image, the number of coherent areas is calculated (the 8-coherent areas are considered here, i.e. each pixel has 8 neighbors). This criterion allows to classify areas of simple structure (edges usually have only 2 coherent areas: 2 different sides of the edge) to the class 1, and classify areas of fine texture (grass, leaves) to the class 1, because latter usually have more than 2 coherent areas.

In the result of the classification, each pixel is assigned with a "probability" of belonging to the class 1 or 2. This probability determines the proportion of mixing of NEDI and gradient interpolation results (fig. 11-a).

Since the classification is performed before the NEDI algorithm, we can perform NEDI interpolation only for pixels where the corresponding probability is not zero (for further speed-up of the algorithm we can raise this threshold from 0 to 0.3 without noticeable loss of quality).

3.2 The Kimmel algorithm modification

After interpolation of a green color, we perform the interpolation of red and blue colors using the Kimmel algorithm with several modifications. The main modification is the use of adaptively variable number of iterations.

3.2.1 Variable number of iterations

The standard implementation of Kimmel algorithm suggests using 3 iterations for correction of red and blue colors. We have conducted experiments in order to find how the number of iterations influences the visual quality and PSNR in different image areas. The experiments have shown that with increase of the number of iterations the suppression of color moiré improves (since all the 3 color components are better synchronized), but at the same time color saturation decreases in the places of color transitions [11]. 3 iterations are usually acceptable for most images and produce best average PSNR. However the results can be improved by adaptively varying the number of iterations.

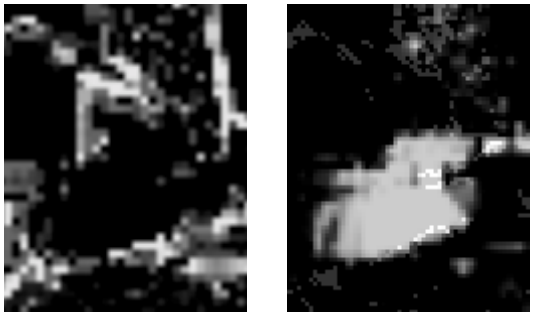
The main task of the Kimmel algorithm is suppression of a color moiré. The color moiré is caused by the aliasing of red and blue color components and appears in areas where low sampling rate of red/blue colors is not enough for capturing high spatial frequencies. The idea of our method is to find such areas and increase the number of Kimmel iterations in them. The search of such areas is performed using 2 criteria.

The first criterion is the uncertainty of detection of interpolation direction (stored in a special array as described in a paragraph 3.1.2).

The second criterion is the image area classifier similar to that described in paragraph 3.1.3. The difference from the paragraph 3.1.3 is that the second criterion of the classifier is used reversely (since we have to classify high-frequency regions as class 2 regions), and the third criterion of the classifier is not used. In our implementation of the algorithm, these 2 classifiers are calculated simultaneously since they work on the same input data and have many common steps.

As a result of this classifier, we obtain a probability for each pixel to be belonging to the class 2. We call this probability array the “problem map”, since it shows image areas, where color moiré is likely to occur (fig. 11-b).

This “problem map” controls the number of iterations in the Kimmel algorithm: in “problem-free” areas the number of iterations can be low: 1…2. But in “problem areas” the number of iterations can raise to 10…12.



a b

Fig 11. Pixel classification for switching of the interpolation type (a), «problem map» (b). The source image is on a fig. 4-a.

3.2.2 Extension of the gradient calculation window

For calculation of interpolation weights in the Kimmel algorithm several directional gradients are evaluated. Each of gradients is calculated as difference of 2 pixel values: Grad = |G[i, j] – G[k, m]|

In our algorithm, we have modified the calculation of gradients by averaging differences in a wider spatial window:

$$\text{Grad} = 8\cdot|G[i, j] - G[k, m]| + |G[i+1, j] - G[k+1, m]| + |G[i-1, j] - G[k-1, m]| + |G[i, j+1] - G[k, m+1]| + |G[i, j-1] - G[k, m-1]|$$

This has slightly improved the resulting PSNR quality.

3.2.3 Half-recursive color update

In the original Kimmel algorithm, the calculation of the corrected colors is based on one color array, while the corrected colors are written into another color array. Then, at the next iteration, the arrays are interchanged. In our algorithm, we write the corrected pixel values into the same array, and they are used for interpolation of next neighboring pixels. On odd iterations of the correction, the whole array is walked from left to right, from top to bottom, and on even iterations – in the reverse order. This small modification also improves the resulting PSNR.

3.3 Projection on a source color data

The Kimmel algorithm adjusts all 3 color components of the image to reduce the color moiré. However the algorithm changes the color values in all pixel positions, including those color values originally given in the bayer pattern. It is evident that substitution of the original color components from the bayer pattern into the corresponding positions of the demosaiced image will improve PSNR. It was found that the visual quality also improves after this operation. Speaking in terms of alternating projections algorithm [6], this operation is called "projection on the source data".

In our algorithm, we use more complex model of projection on the source data. The idea of the method is adjusting all the color components in each pixel according to the local color model of the image.

The local color model is constructed as follows. We try to approximate all the pixel colors from some local window around the current pixel using a straight line in a 3-dimensional color space. The colors of the window pixels are a cloud (in 3-dimensional color space), and it is required to draw a straight line that optimally approximates all these colors. In our algorithm, we obtain this line approximately by connecting 2 points of the cloud that are most distant from each other. We call vector between these 2 points a "color direction vector". It shows the dominant direction of a color variation around the current pixel in the image. For example, if the pixel lies close to the boundary between black and white colors (including intermediate grayscale colors), then the color direction vector has equal R, G, and B components.

In our algorithm, we use a window of 3x3 pixels for search of a color direction vector. After the vector is found, we calculate the average distance of window pixels from the approximating line (i.e. from the color direction vector). This distance can be considered as a measure of consistency of such 1-dimensional color approximation model.

After the color direction vector is obtained, the substitution of the source color data is performed according to the following rule: for each pixel the substituted color component determines the amount of pixel color shift, and the *direction* of the shift is determined by the color direction vector. For example, if the color direction vector is (100, 100, -100), and the value of the pixel is (r=20, g=40, b=30), and the substituted source color component is g=42, then the resulting shift of the pixel color will be (2, 2, -2), and the resulting value of the pixel will be (22, 42, 28).

This simple principle requires some protection against situations when color direction vector is approximately orthogonal to the substituted color component, which can result in very high changes of other color components. To implement such protection in out algorithm, we have imposed some limits on maximal changes of the color components. The example for the case of a green color substitution is demonstrated below:

```
Green = OriginalGreen;
if (CDV[Green]>eps)
{
  Amount = 1 – eps/CDV[Green] + GrayscaleMeasure;
  Red  += (OriginalGreen – Green) * CDV[Red] /
              CDV[Green] * Amount;
  Bound(Red, –Limit, +Limit);
  Blue += (OriginalGreen – Green) * CDV[Blue] /
              CDV[Green] * Amount;
  Bound(Blue, –Limit, +Limit);
}
```

Here *Red*, *Green*, and *Blue* are the values of color components, *OriginalGreen* is the source color component substituted from the original bayer pattern, {CDV[Red], CDV[Green], CDV[Blue]} is the color direction vector, *Limit* is the limit on changes of color components (it can be calculated from the consistence of the local color model).

The *Amount* variable softens the effect of substituted color component on other color components of the current pixel. Its value is calculated from 2 criteria:

1. The effect should be reduced when CDV vector is approximately orthogonal to the green color component (this criterion is implemented by eps/CDV[Green] term).
2. The effect should be increased when the direction of CDV vector is close to (1, 1, 1). This increases the coherence of color components in grayscale images which increases the quality of the result. At the same time it reduces the coherence of color components is areas of transitions between colors of different hue, which also increases the quality and reduces the "zipper effect".

3.4 Iteration of the whole algorithm

For further improvement of the image quality we use the additional iteration of the whole algorithm. On this iteration the directional gradients (used for interpolation of a green component) are calculated using the whole-resolution image, restored on the pre-

vious iteration. This significantly increases the quality of green color interpolation, esp. in areas of high-frequency information. On a first iteration, the direction of interpolation could have been chosen erroneously due to aliasing. But after the projection on a source color data, the lost high frequencies are partly restored, and this helps to determine correct interpolation directions on a second iteration of the algorithm.

4. RESULTS

We have conducted experiments to compare the proposed method with several state-of-the-art algorithms. For comparison we have used a set of test images widely spread in publications on image processing and demosaicing.

The image quality has been measured using PSNR measure in RGB space, and PSNR measure in CIEDE2000 space of ΔE_{00} color differences [9], taken between the original full-color image and the demosaiced image (as discussed in section 2.4). The averaged results across our test set are given in the table 1.

Method	Average PSNR-RGB	Average PSNR-CIEDE2000
Bilinear	27.50	35.56
Kimmel	33.50	42.57
Aqua-2	34.63	42.99
Alternating Projections	35.24	42.76
Proposed method	**37.10**	**44.36**

Table 1. Average PSNR values across test images

PSNR values on each image separately are graphed on fig. 12.

The fig. 13 shows the example result of our algorithm applied to the "Lighthouse" image. Other examples can be found on our web-page [11].

PSNR measurements and visual assessment suggest that the proposed algorithm outperforms all known to us demosaicing algorithms. The computational complexity of our method is around 3 times higher than the complexity of Kimmel algorithm.

5. REFERNECES

[1] R.H. Hibbard, "Apparatus and method for adaptively interpolating a full color image utilizing luminance gradients", *U.S. Patent 5,382,976, January 1995*.

[2] C.A. Laroche and M.A. Prescott, "Apparatus and method for adaptively interpolating a full color image utilizing chrominance gradients", *U.S. Patent 5,373,322, December 1994*.

[3] R. Kimmel, "Demosaicing: image reconstruction from CCD samples", *Proc. Trans. Image Processing, vol. 8, pp. 1221–1228, 1999*.

[4] D.D. Muresan, "Review of Optimal Recovery", http://dsplab.ece.cornell.edu/papers/technical_reports/review_or.pdf

[5] D.D. Muresan, T.W. Parks, "Optimal Recovery Demosaicing", *IASTED Signal and Image Processing Conference (Hawaii 2002)*.

[6] B.K. Gunturk, et al. "Color Plane Interpolation using Alternating Projections", *IEEE Trans. Image Processing, vol. 11, no. 9, pp. 997-1013, September 2002.*

[7] X. Li, M.T. Orchard, "New Edge-Directed Interpolation". *IEEE Trans. On Image Processing, Vol. 10, No. 10, October 2001*.

[8] J.A. Leitao, M. Zhao and G. de Haan, "Content-adaptive video up-scaling for high-definition displays". *IVCP 2003 Proc. Vol. 5022, January 2003*.

[9] M. R. Luo, G. Cui and B. Rigg, "The Development of the CIE 2000 Colour Difference Formula: CIEDE2000", *Colour & Imaging Institute, University of Derby, UK*.

[10] A. Lukin, "Image Resampling Algorithms" *(demo webpage),* http://audio.rightmark.org/lukin/graphics/resampling.htm

[11] A. Lukin, D. Kubasov, "Bayer pattern interpolation" *(webpage),* http://audio.rightmark.org/lukin/graphics/demosaicing.rus.htm

About the authors

Alexey Lukin is a PhD student of a State University of Moscow, faculty of Applied Mathematics and Computer Science

E-mail: lukin@graphics.cs.msu.su

Denis Kubasov is a graduate student of a State University of Moscow, faculty of Applied Mathematics and Computer Science

E-mail: dkubasov@graphics.cs.msu.su

We would like to thank our university supervisors Dr. Y.M. Bayakovskiy, head of our Graphics & Media Lab, and Dr. A.S. Krylov, head of our joint research projects with SAIT (Samsung Advanced Institute of Technology). We also acknowledge Dr. D.S. Vatolin's introduction to the field of demosaicing.

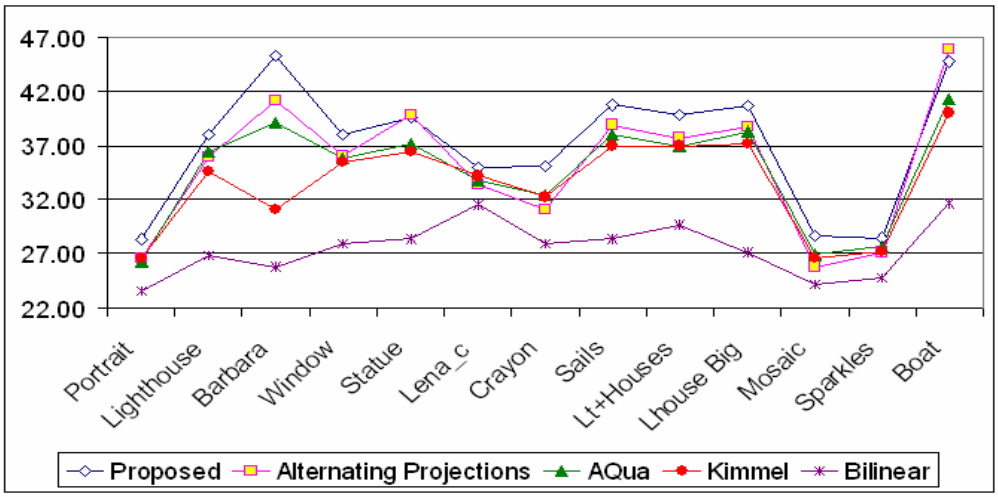


Fig 12. PSNR-RGB values for each test image

Original image Bilinear interpolation Proposed method

Fig 13. Example of our algorithm working on a “Lighthouse” image