ADAPTIVE DEMOSAICING WITH THE PRINCIPAL VECTOR METHOD

Ramakrishna Kakarala and Zachi Baharav

Abstract— **Demosaicing is the process of interpolating the missing colors in an image that is acquired from a digital image sensor equipped with a color filter array. This paper describes a spatially adaptive demosaicing algorithm that is based on the Jacobian matrix of the color map and neighborhood voting. The algorithm requires only additions, subtractions and shifts, and is therefore attractive from a computational point of view. Comparisons are provided to show that the algorithm improves on published algorithms in terms of complexity or image quality.**

I. Introduction

Most digital cameras use a single image sensor. In order to obtain a color image, the sensor is fitted with an array of color filters, with one filter for each pixel. The widely-used Bayer color filter array for red, green and blue (RGB) sensors uses the 2×2 pattern shown below, repeated across the sensor:

$$\begin{matrix} G & R \\ B & G \end{matrix}$$

Since each pixel obtains only one color, interpolation is required to obtain a full color RGB image. The interpolation process is known as *demosaicing*, since it attempts to invert the sampling process of the color filter mosaic.

Perhaps the simplest possible demosaicing method is bilinear interpolation, which may be realized by convolution and is not spatially adaptive. Bilinear interpolation produces blurry images and does not suppress aliasing. In particular, the aliasing that results from undersampling of colors is visible as colored fringes on high-frequency spatial patterns. The aim of adaptive interpolation schemes is to avoid blurriness and suppress aliasing by sensing edges and interpolating along them, rather than across them; see [6] for a discussion of this point.

Several adaptive demosaicing schemes have been published previously. The edge-weighted interpolation method with color cross ratio [2] (hereafter, CCR) produces high-quality images but requires considerable computation. CCR requires three image iterations to adjust ratios of color values, followed by an inverse diffusion stage. A much simpler algorithm [3] uses gradients in a single color plane as spatial predictors (hereafter, GRAD) to determine the direction of interpolation. Although simpler, GRAD does not suppress aliasing as well as CCR. Another simple algorithm [6] uses predictors from two color planes (hereafter, PRED) to determine the interpolation direction.

CORRESPONDING AUTHOR: R. Kakarala is with the Sensors Solutions Division, Agilent Technologies, 3175 Bowers Ave MS 87H, Santa Clara, CA 95054. Email: ramakrishna_kakarala@agilent.com

Zachi Barahav is with Agilent Laboratories, 3500 Deer Creek Rd, Palo Alto CA 94034. Email:zachi_baharav@agilent.com

In this paper, we propose an adaptive algorithm which produces images with quality comparable to CCR, but with a greatly reduced computational complexity which is comparable to GRAD or PRED. The new algorithm, which we refer to as the principal vector method (PVM), relies on the Jacobian matrix of the color mapping at every pixel and employs a voting scheme to determine the interpolation direction.

II. Preliminary concepts

A continuous color image may be viewed as a mapping from a two-dimensional (2-D) domain to a three-dimensional (3-D) space, the color space. The RGB color space is used in this paper, but the results apply to any color space. For RGB, the Jacobian is

$$J = \begin{bmatrix} \nabla R & \nabla G & \nabla B \end{bmatrix},$$

where ∇ denotes the gradient, e.g., $\nabla R = [\partial R/\partial x, \partial R/\partial y]^t$. The Jacobian captures the local first-order variation in the color images, and has been proposed as a measure suitable for color edge detection[1, pp 334-335] .

Since J is a matrix, it has a singular value decomposition (SVD), which may be written

$$J = USV^t = s_1 u_1 v_1^t + s_2 u_2 v_2^t.$$

Here $s_1 \geq s_2 \geq 0$ are the singular values, and u_k, v_k are respectively, the left and right k-th singular vectors. It is well known that the best rank one approximation to J (in terms of minimizing the sum of squared errors) is

$$J \approx s_1 u_1 v_1^t.$$

In particular, the 2×1 vector u_1 provides the direction of largest variation for the color image, and is the best fit in the rank-one sense to the three gradients $\{\nabla R, \nabla G, \nabla B\}$. We call u_1 the *principal vector* of the Jacobian.

The degree to which the principal vector fits the gradients may be determined from the singular values. Since

$$\|J - s_1 u_1 v_1^t\|_2 = s_2,$$

where the norm $\|\cdot\|_2$ is the sum of squared entries in the matrix, it follows that the degree of fit is measured by

$$\lambda = \frac{s_2}{s_1 + s_2}.$$

Because $s_1 \geq s_2$, we see that $0 \leq \lambda \leq \frac{1}{2}$. When $\lambda = 0$, the principal vector gives a perfect fit (the gradients are all parallel in this case), whereas if $\lambda = \frac{1}{2}$, the fit is poorest. An example of a situation when $\lambda = \frac{1}{2}$ occurs is when two of the three gradients point in orthogonal directions, and the third gradient is zero.

Contributed Paper

Manuscript received July 8, 2002

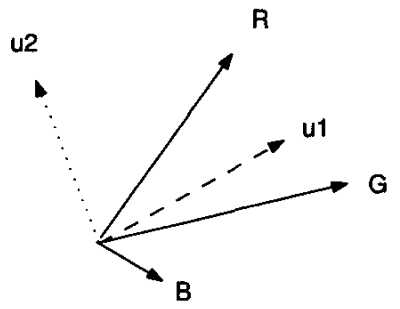


Fig. 1. A typical arrangement of red, green, and blue gradient vectors is shown, as well as the corresponding vectors u_1 and u_2.

III. Interpolation based on the principal vector

The previous section shows how, through the SVD, the information in the Jacobian J is usefully encapsulated by the principal vector u_1 and the measure of fit λ. An adaptive color interpolation scheme based on u_1 and λ is now described.

Adaptive interpolation works by interpolating a missing value from neighbors in the direction of least change. Consider the arrangement shown in Fig. 1. The R, G, and B gradient vectors point in the directions of greatest change for their respective colors. The principal vector u_1 is the best fit to the RGB gradients, and the vector u_2, which is always perpendicular to u_1, points in the direction of least overall change. Now suppose that I_1 is the result of interpolating along the direction u_1, and I_2 is the result of interpolating along u_2. Then the total interpolation, denoted I, should be a weighted combination of I_1 and I_2, where the weighting is based on the measure of fit λ as follows:

$$I = \lambda I_1 + (1-\lambda) I_2. \tag{1}$$

This weighting guarantees that the total interpolation varies continuously between $I = I_2$, when $\lambda = 0$ and therefore the principal vector perfectly fits the gradients, and $I = (I_1 + I_2)/2$, when $\lambda = \frac{1}{2}$ and the fit is poorest.

Since image data are sampled on a rectangular grid, we implement eq. (1) by using interpolated values in the horizontal and vertical directions instead of the continuously-varying directions of u_1 and u_2. Let I_h and I_v respectively denote the interpolated values in the horizontal and vertical directions. Intuitively, I_h and I_v are computed by forming simple linear combinations of pixel values along rows for I_h or, alternatively, along columns for I_v. Next, let $u_1 = [u_1(1), u_1(2)]^t$ and $u_2 = [u_2(1), u_2(2)]^t$, and note that both u_1 and u_2 are unit-length vectors. Noting that $u_1(1)^2$ measures the length of the horizontal component of the principal vector, and similarly $1 - u_1(1)^2 = u_1(2)^2$ measures the vertical component length, we define

$$I_1 = u_1(1)^2 I_h + u_1(2)^2 I_v,$$

and similarly

$$I_2 = u_2(1)^2 I_h + u_2(2)^2 I_v.$$

Combining these equations with eq.(1), we obtain a simpler formulation of (1) for sampled image data:

$$I = \alpha I_h + (1-\alpha) I_v, \text{ with } \alpha = \lambda u_1(1)^2 + (1-\lambda) u_2(1)^2. \tag{2}$$

Computing the value of I in eq. (2) requires three terms: I_h, I_v, and α. The first two terms, I_h and I_v, are straightforward to compute, since they involve only simple linear combinations of pixel values. The third term, α, requires the SVD of J and therefore involves much more computation. To avoid the computational burden of evaluating an SVD at every pixel, we now examine methods to estimate an approximate value for α.

IV. Voting and the majority rule

We make three heuristic assumptions in order to simplify computation of α, the weighting coefficient in (2).

1. Assume that the value of α can be quantized to three levels, $\alpha = 0$, 1, and $\frac{1}{2}$. This means the total interpolation is either purely horizontal, purely vertical, or an average of the two. Although this has the potential to produce a staircase approximation to straight lines, the degradation is hardly noticeable since we start with undersampled data in the first place.
2. Assume that the *direction* of the principal vector u_1 can be quantized to be one of only two possibilities: horizontal or vertical. We say that u_1 is horizontal if $|u_1(1)| \geq |u_1(2)|$, and vertical otherwise.
3. Assume that the decision whether the principal vector is horizontal or vertical can be made by applying the *majority rule*: if the majority of elements of the top row of the Jacobian (which are horizontal derivatives) exceed in absolute value the corresponding elements of the bottom row (which are vertical derivatives) then the principal vector is horizontal; otherwise it is vertical.

To illustrate the majority rule, suppose the Jacobian is

$$J = \begin{bmatrix} 1 & 3 & -2 \\ 3 & 1 & 1 \end{bmatrix}.$$

Then two of the elements of the top row exceed their counterparts in the bottom row in absolute value, and therefore the majority rule says that the principal vector is horizontal. Indeed, the principal vector for this matrix is $u_1 = [0.82\ 0.57]$, which has a larger first element and is therefore horizontal. The majority rule is not always correct. For example, if the red gradient is large and horizontally directed, but the green and blue gradients are small and vertically directed, the principal vector tends to align itself horizontally in opposition to the majority. This type of arrangement does not occur frequently in real images, since there is considerable correlation between colors. To test the validity of the majority rule, we performed an experiment with 1000 randomly-generated Jacobian matrices with independent, normally-distributed elements. The majority rule correctly determined the horizontal/vertical orientation of the principal vector 91% of the time, even though a Jacobian matrix with statistically independent elements (and therefore no correlation between colors) is the worst-case scenario.

While the majority rule gives us the direction of the principal vector, it does not by itself determine α. To do so, we examine the *number* of horizontal derivatives exceeding vertical derivatives. Let the notation $A \overset{?}{>} B$ denote a conditional whose value is 1 if $A > B$, and 0 otherwise. Then the number of horizontal derivatives exceeding vertical derivatives is

$$V_J = \sum_{k=1}^{3} \left(|J_{1k}| \overset{?}{>} |J_{2k}| \right).$$

If $V_J = 3$, then we would want $\alpha = 0$ so that the interpolation is vertical; if $V_J = 0$, then we would want $\alpha = 1$ to force horizontal interpolation. However, if $V_J = 1$ or $V_J = 2$, then there is not a unanimous indication, and it seems appropriate to set $\alpha = \frac{1}{2}$.

The selection mechanism for α described above is basically a voting scheme, where each column of the Jacobian gets one vote. However, there is clearly information in the neighborhood around a pixel that should influence the choice of α. For example, in a smoothly varying region, it makes sense to choose α in a consistent manner. The voting scheme may be extended to accomodate neighborhood votes as well as votes at each pixel, as follows. The pixel where interpolation is to occur gets three votes, as described above. Neighboring pixels get one vote each, which is equal to 1 if $V_J \geq 2$ at that pixel, and 0 otherwise. Specifically, let

$$\text{maj}\{J(x,y)\} = \begin{cases} 1, & \text{if } V_J > 1, \\ 0, & \text{otherwise} \end{cases}.$$

The total votes collected at each pixel is now defined to be the sum of the votes V_J from the Jacobian at that pixel, as well as the votes at neighboring pixels, denoted V_N:

$$\begin{aligned} V = V_J + V_N &= \sum_{k=1}^{3} \left(|J_{1k}(x,y)| > |J_{2k}(x,y)| \right), \\ &+ \sum_{(x,y)\in N} \text{maj}\{J(x,y)\}. \end{aligned}$$

The neighborhood N may be defined to be any subset of pixels preceeding the current pixel in a raster scan of the image. For example,

$$N_1 = \{(x-1,y),(x,y-1)\} \tag{3}$$

is the neighborhood of adjacent pixels to the left and also above the current pixel.

Once the votes are collected at the current pixel, the weight α is assigned. The votes are divided into three ranges: first, $V < T_1$, in which case $\alpha = 1$; second, $T_1 \leq V \leq T_2$, in which case $\alpha = 0.5$; and third, $T_2 < V$, in which case $\alpha = 0$. The choice of T_1 and T_2 depends on the size of the neighborhood N. For example, if N is chosen as in (3), then $T_1 = 2, T_2 = 3$. In this situation the maximum number of votes is five. With T_1 and T_2 so chosen, if less than two votes for vertical interpolation are received, then horizontal interpolation is chosen with $\alpha = 1$. If either four or five votes are obtained, then vertical interpolation is chosen with $\alpha = 0$. In the middle ground where two or three votes are obtained, then averaging of horizontal and vertical is applied with $\alpha = 0.5$. Other thresholds may be used if a different neighborhood N is chosen.

Note that if the neighborhood N contains vertically adjacent pixels, then an additional memory is required to store the value of $\text{maj}\{J(x,y)\}$ at each pixel on the previous lines. The memory requirement is only 1 bit per pixel for each line that is stored.

V. PVM ALGORITHM

The complete algorithm for demosaicing using interpolation along the principal vectors is now described.

1. The Jacobian is computed at every pixel. This requires estimates of red, green, and blue derivatives, both horizontally and vertically. Since only one color is present at each pixel, the derivatives must be computed from neighbors. Any discrete approximation to the first derivative may be used, from simple differences to regularized schemes for noise suppression. Simple, convolution-based derivative estimators [7] are used to obtain the results shown below in Section VI.
2. The missing green values are interpolated by applying the voting mechanism to determine α at each red or blue pixel, and subsequently employing eq.(2). The fully populated green plane after this interpolation is denoted $\tilde{G}$.
3. The missing red and blue values are interpolated in two stages. First, the difference images between the respective color and $\tilde{G}$ is formed. For example, for the red plane, let

$$D_{RG}(x,y) = R(x,y) - \tilde{G}(x,y),$$

if location (x,y) is a red pixel, and $D_{RG} = 0$ otherwise. A similar method is used to form D_{BG}, the difference image for the blue plane. The difference images may be interpolated using either bilinear interpolation, or alternatively, to provide a smoother result at low light levels, the "polyphase interpolation" method [4, pg 94-95]. Briefly, the polyphase method interpolates in two stages, the first being for values that have four diagonal neighbors with existing values, and the second being for values with two existing neighbors and two interpolated neighbors computed from the first stage. In the second stage the interpolated neighbors are weighted less than the existing neighbors; see [4] for details. This method achieves a smooth interpolation without propagating outlier values.
4. Using the now fully populated difference images, denoted $\tilde{D}_{RG}$ and $\tilde{D}_{BG}$, the interpolated red and blue values are obtained by addition with the original green values. For example, for the red plane, set

$$\tilde{R}(x,y) = \tilde{D}_{RG}(x,y) + G(x,y),$$

if (x,y) is not a red pixel, and $\tilde{R}(x,y) = R(x,y)$ otherwise. A similar process is followed to obtain the fully populated blue plane $\tilde{B}(x,y)$.
5. The three fully-populated color planes, $\tilde{R}$, $\tilde{G}$, and $\tilde{B}$ form the output of the interpolator.

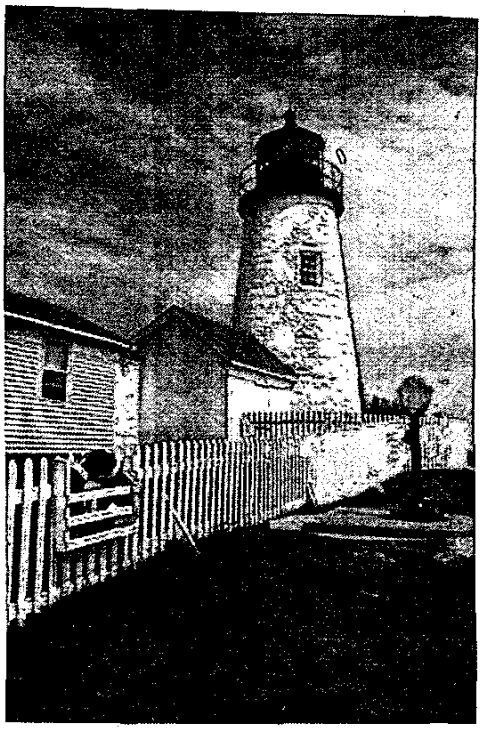


Fig. 2. Original image "lighthouse". The full-color image is available on the web [8].

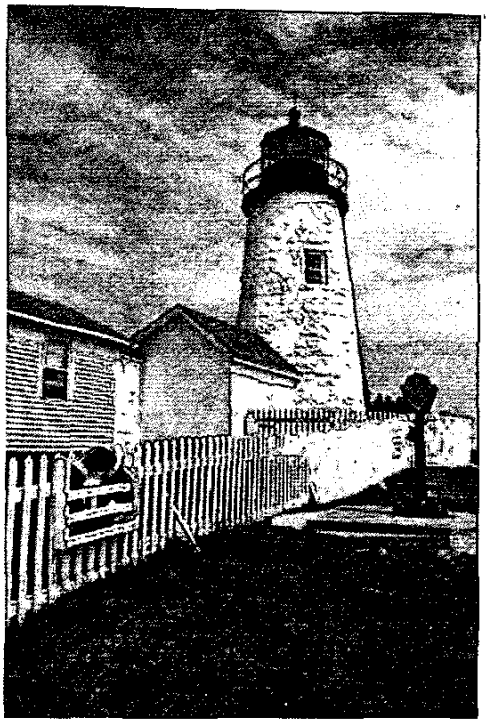


Fig. 3. Lighthouse image demosaiced with the method proposed in this report (PVM). See [8] for the full-color image.

Using standard computational methods, it is possible to perform all computations required above with only integer additions, comparisions, and bit shifts.

VI. Examples and discussion

Figure 2 shows the image "lighthouse", which is difficult to demosaic due to the high spatial frequency content. Also, the image has many neutral surfaces which makes it easy to observe color variations due to aliasing. Figure 3 shows the result using PVM with bilinear difference image interpolation. The result is comparable in quality to that obtained with CCR [2].

Figure 4 shows the result using the GRAD method. It it clear that PVM shows considerably less aliasing. Figure 5 shows that the PRED algorithm provides a result of comparable quality to PVM. Portions of the demosaiced images for both PVM and the PRED algorithms are shown in Fig. 6 to provide a more detailed comparison. It can be seen that PVM improves on PRED in two of the three regions shown.

The usefulness of neighborhood voting in PVM is now illustrated. Figure 7(a) shows a portion of the lighthouse image with N as specified in eq. (3). Figure 7(b) shows the same region, this time processed using $N = \emptyset$, or no neighborhood. It is clear that Fig. 7(a) improves on Fig. 7(b) in terms of reducing isolated interpolation errors, and hence using a neighborhood provides smoother interpolation.

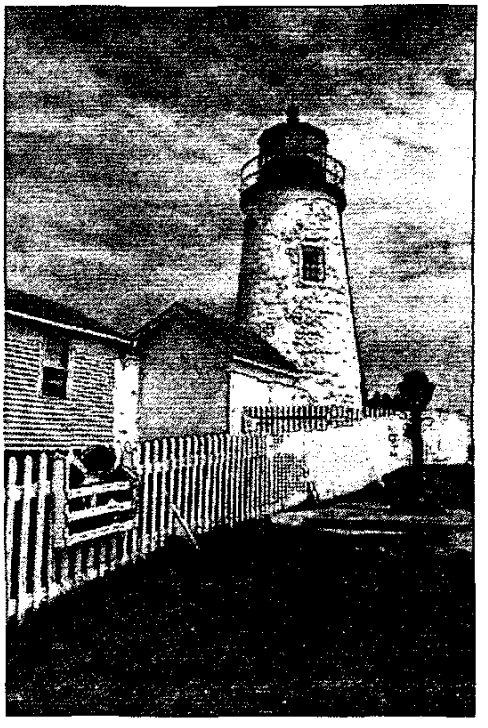


Fig. 4. Lighthouse image demosaiced with the GRAD method. See [8] for the full-color image.

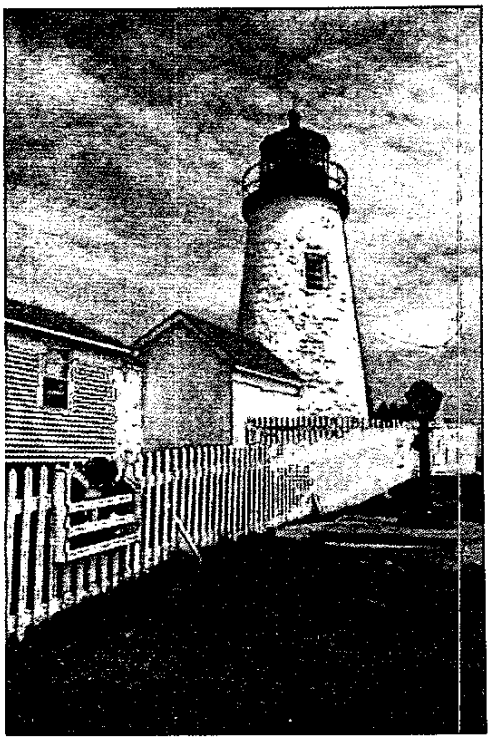


Fig. 5. Lighthouse image demosaiced with the PRED method. See [8] for the full-color image.

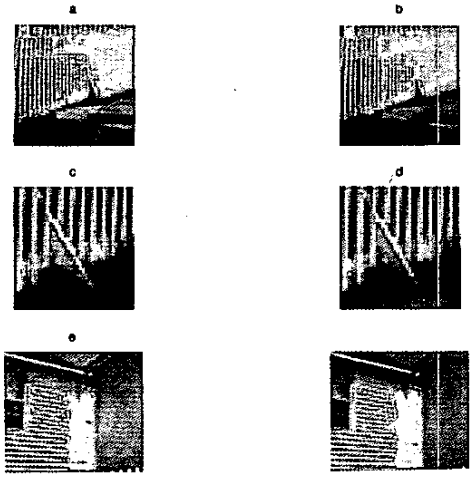


Fig. 6. Comparison between PVM and the PRED method. The left column (parts a, c, and e) show results obtained with PVM. The right column (parts b, d, and f) show corresponding results obtained with the PRED method. PVM shows somewhat less aliasing in parts a and c, compared to parts b and d, but more in part e compared to part f. Full-color images are available on the web [8].

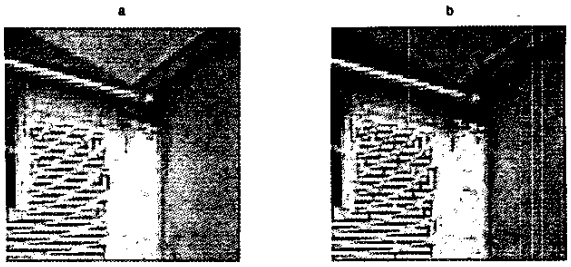


Fig. 7. Comparison showing advantage of using neighborhood voting. Part (a) shows result using neighborhood N_1, and in part (b), no neighborhood voting is used. See [8] for the full-color image.

VII. Summary

This paper describes a new adaptive demosaicing algorithm, which uses a voting scheme to determine the direction of interpolation at each pixel. Votes are counted from the neighborhood as well as from measurements taken at the pixel itself. The computational requirements are relatively low, and the resulting images are largely free of blurring and aliasing artifacts.

Acknowledgements

We would like to thank both Dieter Vook and Mark Alden for many helpful suggestions which improved the manuscript.

References

[1] B. Jahne, H. Haussecker, eds, *Computer vision and applications: a guide for students and practitioners*, San Diego: Academic Press, 2000.

[2] R. Kimmel, "Demosaicing: image reconstruction from color CCD samples", *IEEE Transactions on image processing*, Vol. 8, No. 9, September 1999, pp 1221-1228.

[3] T. Kuno, H. Sugiura, M. Asamura, Y. Hatano, and N. Matoba, "Aliasing reduction for color digital still camera with single-chip charge-coupled device," *Journal of Electronic Imaging*, Vol. 8, No. 4, October 1999, pp 457-466.

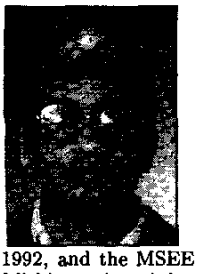
[4] G. R. Arce, J. L. Paredes, J. Mullan, "Nonlinear filtering for image analysis and enhancement," in *Handbook of Image & Video Processing*, A. Bovik, Editor, San Diego: Academic Press, 2000, pp 81-100.

[5] N. Herodotou & A. N. Venetsanopolous, "Color image interpolation for high resolution acquisition and display devices," *IEEE Transactions on consumer electronics*, Vol. 41, No. 4, November 1995, pp 1118-1126.

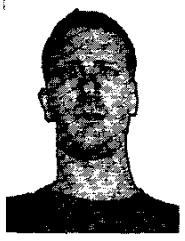
[6] J. Adams, K. Parulski, K. Spaulding, "Color processing in digital cameras," *IEEE Micro*, Vol. 18, No. 6, November-December 1998, pp 20-30.

[7] R. C. Gonzalez & R. E. Woods, *Digital image processing*, Reading, MA: Addison-Wesley, 1992.

[8] www.baharav.org



Ramakrishna Kakarala (SM '01) is with the Sensor Solutions Division (SSD) of Agilent Technologies, where he carries out research into algorithms for image processing, video compression, and optical navigation. Prior to joining SSD, he spent two years at Agilent Laboratories in Palo Alto, and six years as a Lecturer (promoted to Senior Lecturer) in Electrical Engineering at the University of Auckland in New Zealand. He received the PhD in Mathematics from the University of California, Irvine, in 1992, and the MSEE and BSCompE degrees from the University of Michigan, Ann Arbor in 1988 and 1986, respectively.



Zachi Baharav (SM '99) is with Agilent Technologies Labs, Palo Alto, CA. His areas of research include new applications and algorithms for image processing, and numerical Electromagnetics. He received the PhD in EE from the Technion, Israel, in 1998.