Demosaicing of True Color Images Using Adaptive Interpolation Algorithms

Nadia Tarik Saleh
Department of Computer Science
College of Computer and Mathematics Sciences
University of Mosul

Received 28 / 08 / 2008 **Accepted** 16 / 02 / 2009

المستخلص

يتم إستخلاص الألوان في أجهزة التصوير الملون الإلكترونية، كما في الكاميرا الرقمية، عن طريق متحسس للإضاءة والذي يقوم بأخذ عينة حسب نمط معين من ألوان عناصر الصورة النقطية، وهي الأحمر، الأخضر، والأزرق. تسمى عملية استرجاع الألوان المفقودة و إعادة توليدها من جديد ، بإسترجاع الألوان من الحالة الفسيفسائية أو ما يطلق عليه الـ Demosaicing، وهو أحد أنواع الإستكمال.

يتناول البحث الحالي تطبيق ومقارنة عدد من خوارزميات الإستكمال التكييفية المعروفة على مجموعة من الصور ذات الألون الحقيقية، كما تمت مقارنة هذه الطرائق مع الطرائق التقليدية. بالإضافة إلى استحداث طريقة جديدة معتمدة في عملها على النوعين.

الصور المستخدمة في العمل من نوع Bitmap. وقد استخدم مقياس نسبة الإشارة إلى الخطأ وهو Signal to noise ratio لقياس مدى دقة الصورة الناتجة. تم إدراج جميع النتائج التي تم الحصول عليها بالإضافة إلى عينات من الصور المستخدمة. أما لغة البرمجة المستخدمة فهي لغة Visual C++ 6.0.

ABSTRACT

In an electronic color-imaging device, such as a digital camera, using a single CCD (Charge Coupled Device) sensor, the color information is usually acquired in sub-sampled patterns of Red (R), Green (G) and Blue (B) pixels. The methodology to recover or interpolate the missing color components is known as color demosaicing, a type of image interpolation.

This paper introduces and compares some of the most popularly demosaicing algorithms of adaptive type applied on true color images. Another comparison is done between these methods and the non-adaptive traditional methods. Also, a new method is invented, which depends on both types.

The used images are of Bitmap type. Signal to noise ratio measure is used as an objective quality metric. The results of comparison are presented with some sampled images. Programming has been applied using Visual C++ 6.0 programming language.

1. Introduction

Due to the cost and packaging consideration, in digital imaging devices, such as the commercially available Digital Still Cameras (DSC), the image color is captured in a sub-sampled pattern. Typically each pixel in the captured raw image contains only one of the three primary color components, R (Red), G (Green), or B (Blue). This sub-sampled color image is generated using certain pattern of a Color Filter Array (CFA).

This CFA is realized by coating the surface of the electronic sensor array or a single CCD array using some optical material that acts as a band-pass filter. This coating allows the photons corresponding to only one color component (frequency range) to be transmitted to the sensor and the other two color components are blocked. A typical and widely used CFA pattern is called Bayer Pattern. As shown in Figure 1, a Bayer pattern image of dimension 8*8 [1][2][3].

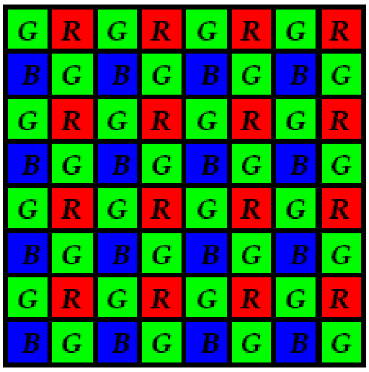


Figure 1: Bayer Pattern.

Many other implementations of a color-sampling grid have been incorporated in commercial cameras, most using the principle that the luminance channel (green) needs to be sampled at a higher rate than the chrominance channels (red and blue). The choice for green as representative of the luminance is due to the fact that the luminance response curve of the eye peaks at around the frequency of green light[3].

Each cell in Bayer pattern represents a pixel with only one color component as indicated by either R or G or B.

A full-color image needs the information of all the three colors in each pixel location. As a result, each pixel needs to be represented by 24-bit colors, assuming 8 bits for each of R, G, and B, as in true color images. So that, it is essential to interpolate the missing two colors in each pixel location using the information of the neighboring pixels.

The methodology to recover or interpolate these missing colors is known as Color Interpolation or Color Demosaicing [1][2][4].

The aim of this paper is to apply commonly adaptive and non-adaptive demosaicing algorithms on different Bayer images to recover the original true color images.

A combination of two methods of adaptive and non-adaptive type is performed to get a new method, which gives better results.

The applied methods are compared together in order to observe the differences.

2. Principles of Capturing Colors in Digital Camera

The sensors, which are used in most cameras, are either Charge Coupled Device (CCD) or CMOS (Complimentary Metal Oxide Semiconductor) sensors. The CCD camera comprises a very large number of very small photo diodes, called photo-sites. The electric charges, which are accumulated at each cell in the image are transported and are recorded after appropriate analog to digital conversion.

In CMOS sensors, on the other hand, a number of transistors are used for amplification of the signal at each pixel location. The resultant signal at each pixel location is read individually [1][5][6].

There are several ways in which a digital camera can capture colors. One approach uses red, green, and blue filters and spins them in front of each single sensor sequentially one after another and records three separate images in three colors at a very fast rate. Thus the camera captures all the three colors components at each pixel location. While using this strategy an automatic assumption is that during the process of spinning the three filters, the colors in the image must not change, i.e., they must remain stationary. This may not be a very practical solution.

A practical solution is based on the concept of color interpolation or demosaicing, which is a more economical way to record the three primary colors of an image. In this method, only one type of filter would be permanently placed over each individual photo-site. Usually the sensor placements are carried out in accordance to a specified pattern. The most popular pattern used is Bayer pattern. It is possible to make very accurate guesses about the missing color component in each pixel location by a color interpolation or demosaicing algorithm [1][6].

3. Color Interpolation Algorithms

Color demosaicing algorithms can be broadly classified into two categories: non-adaptive algorithms and adaptive algorithms, as described below [1][2]:

- **Non-adaptive Algorithms:**

  In non-adaptive color interpolation algorithms, a fixed pattern of computation is applied in every pixel location in the sub-sampled color image to recover the missing two color components. Usually, this type is easy to implement with low cost in terms of computational requirements.

- **Adaptive Algorithms:**

  In adaptive color interpolation algorithms, intelligent processing is applied in every pixel location based on the characteristics of the image in order to recover the missing color components. This type of algorithms yield better results in terms of quality as compared with the non-adaptive algorithms. However, effective algorithms in this category are usually more computationally intensive

  In this paper four algorithms of non-adaptive type with the other four of adaptive types have been applied at Bayer pattern of images of different specifications. Also, a new method is invented, which is a combination of both adaptive and non-adaptive methods.

3.1 Non-adaptive Interpolation Algorithms

Nearest neighbor, bilinear, median, and smooth hue transitions are the non-adaptive methods used in this paper.

3.1.1 Nearest Neighbor Algorithm

Probably, nearest neighbor is the most basic form of image interpolation. In this method, each missing color is approximated by nearest pixel representing that color in the input image. The nearest neighbor can be any one of the upper, lower, left, and right pixels.

For example, the pixel p at location (m,n) has four horizontal and vertical neighbors that are given by (m+1,n), (m-1,n), (m, n+1), (m, n-1). The four diagonal neighbors of p have coordinates (m+1,n+1), (m+1,n-1), (m-1,n+1), (m-1,n-1). The missing components of these neighbors can take the component value of p.

The advantage of this approach is that the computational requirement is very small and suitable for applications where speed is very crucial. However, the significant color errors make it unacceptable for a still imaging system, such as high-resolution digital cameras.

With the most basic nearest neighbor interpolation, just copy the exact same pixel values over to the filler pixel closest to the pixel [1][2][3][7].

3.1.2 Bilinear Interpolation

A slightly more sophisticated way of accomplishing gray-level assignments, using the four nearest neighbors of a point. In this algorithm, a missing color component is interpolated by linear average of the adjacent pixels representing the missing color.

Consider pixel location (m,n), to estimate $G_{m,n}$, given $G_{m-1,n}$, $G_{m,n-1}$, $G_{m,n+1}$, $G_{m+1,n}$, the four horizontal and vertical neighbors. The estimate for $G_{m,n}$ is given by:

$$G_{m,n} = (G_{m-1,n}+ G_{m,n-1}+ G_{m,n+1}+ G_{m+1,n})/4.$$

To determine $R_{m,n}$, given $R_{m-1,n-1}$, $R_{m-1,n+1}$, $R_{m+1,n-1}$, $R_{m+1,n+1}$, the four diagonal neighbors. The estimate for $R_{m,n}$ is given by:

$$R_{m,n} = (R_{m-1,n-1}+ R_{m-1,n+1}+ R_{m+1,n-1}+ R_{m+1,n+1})/4.$$

Blue component is estimated accordingly.

This algorithm is better than nearest neighbor that takes into account the gradual transition of pixel color values. Some times the results suffer from pixel artifacts. This may be acceptable in a moderate quality video application because the artifact may not be immediately visible by the human eye.

It is possible to use more neighbors for interpolation. Using more neighbors implies fitting the points with a more complex surface, which generally gives smoother results [1][2][3][7].

3.1.3 Median Interpolation

Median interpolation allocates the missing color component with the “median” value of the color components in the adjacent pixels, as opposed to the linear average used in bilinear interpolation.

The median value is computed for the pixel p at location (m,n) from the four horizontal and vertical neighbors, for the case of missing Green component. And from the four diagonal neighbors, for Red and Blue missing components [1][2].

3.1.4 Smooth Hue Transition Interpolation

The key problem in both bilinear and median interpolation is that the hue values of adjacent pixels changed suddenly because of the nature of these algorithms. On the other hand, the Bayer pattern can be considered as a combination of a luminance channel (green pixels, because green contains mostly luminous information) and two chrominance channels (red and blue pixels). The smooth hue transition interpolation algorithm treats these channels differently. The missing Green component in every Red and Blue pixel in the Bayer pattern can first be interpolated using bilinear interpolation. The idea of chrominance channel interpolation is to impose a smooth transition in hue value from pixel to pixel. In order to do so, it defines “blue hue” as B/G, and “red hue” as R/G. To interpolate the missing Blue component $B_{m,n}$ in pixel location (m,n) in the Bayer pattern, the following three cases may arise:

1. The pixel at location (m,n) is Green and the adjacent left and right pixels are Blue color. The missing Blue component in location (m,n) can be estimated as:
   $B_{m,n} = G_{m,n}/2 * (B_{m,n-1}/G_{m,n-1} + B_{m,n+1}/G_{m,n+1})$.
2. The pixel at (m,n) is Green and the adjacent top and bottom pixels are Blue. The missing Blue component can be estimated as:
   $B_{m,n} = G_{m,n}/2 * (B_{m-1,n}/G_{m-1,n} + B_{m+1,n}/G_{m+1,n})$.
3. The pixel at (m,n) is Red and four diagonally neighboring corner pixels are Blue. The missing Blue component can be estimated as:
   $B_{m,n} = G_{m,n}/4 * (B_{m-1,n-1}/G_{m-1,n-1} + B_{m-1,n+1}/G_{m-1,n+1} + B_{m+1,n-1}/G_{m+1,n-1} + B_{m+1,n+1}/G_{m+1,n+1})$.

The missing Red component can be interpolated in each location in a similar fashion [1][2][3].

3.2 Adaptive Interpolation Algorithms

Pattern matching-based interpolation, block matching, edge sensing interpolation, and linear interpolation with second order correction, are the adaptive methods used in this paper. Besides, an enhanced linear interpolation method is proposed here.

3.2.1 Pattern Matching-Based Interpolation Algorithm

In the Bayer pattern, a Blue or Red pixel has four neighboring Green pixels. A simple pattern matching technique for reconstructing the missing color components based on the pixel contexts. It defines a green pattern for the pixel at location (m,n) containing a non-Green color component as a four-dimensional integer-valued vector:

$g_{m,n} = (G_{m-1,n}, G_{m+1,n}, G_{m,n-1}, G_{m.n+1})$.

The similarity (or difference) between two green patterns g1 and g2 is defined as the vector norm, in other word, the Euclidean distance between two vectors.

$\| g_1 - g_2 \| = \sum_{i=0}^{3} | g_{1i} - g_{2i} |$.

When the difference between two green patterns is small, it is likely that the two pixel’s locations, where the two green patterns are defined, will have similar Red and Blue color components.

A weighted average proportional to degree of similarity of the green patterns is used to calculate the missing color component. For example if the pixel at location (m,n) is Red, the missing Blue component

$B_{m,n}$ is estimated by comparing the green pattern $g_{m,n}$ with the four neighboring green patterns $g_{m-1,n-1}$, $g_{m+1,n-1}$, $g_{m-1,n+1}$, and $g_{m+1,n+1}$. If the difference between gm,n and each of the four neighboring green patterns is uniformly small (below a certain threshold), then a simple average is used to estimated the missing Blue color component:

$$B_{m,n} = \frac{B_{m-1,n-1} + B_{m-1,n+1} + B_{m+1,n-1} + B_{m+1,n+1}}{4}.$$

Otherwise, only the top two best-matched green pattern information items are used. For example, if $||g_{m,n} - g_{m-1,n-1}||$ and $||g_{m,n} - g_{m+1,n-1}||$ are the two smallest differences, then the missing Blue color is estimated as:

$$B_{m,n} = (B_{m-1,n-1}*\Delta_1+B_{m+1,n-1}*\Delta_2)/(\Delta_1+\Delta_2).$$

Where $\Delta_1 = ||g_{m,n} - g_{m+1,n-1}||$ and $\Delta_2 = ||g_{m,n} - g_{m-1,n-1}||$.

The missing Red components can be interpolated in as similar fashion. While the missing Green component in every Red or Blue pixel is interpolated using bilinear interpolation [1][2][3].

3.2.2 Block Matching Algorithm

A block matching algorithm based on a concept of Color Block. The color block of a non-Green pixel is defined as a set $x = (x_1,x_2,x_3,x_4)$ formed by the four neighboring Green pixels, say, x_1, x_2 ,x_3, and x_4. A new Color Gravity is defined as the mean $\bar{x} = (x_1+x_2+x_3+x_4)/4$.

The similarity between two color blocks is defined as the absolute difference of their color gravities. The block matching algorithm is developed based on the selection of a neighboring color block whose color gravity is closest to the color gravity of the color block under consideration.

For any non-Green pixel in the Bayer pattern image, as shown in Figure 1, there are four neighboring Green pixels G_N (the North neighbor), G_S (South), G_E (East), and G_W (the West neighbor), which form the color block $g = (G_N, G_S, G_E, G_W)$. Its color gravity is $\bar{g} = (G_N + G_S + G_E + G_W)/4$. The missing Green value is simply computed by the median of G_N, G_S, G_E, and G_W. If the pixel at (m, n) is Blue, it will have four diagonally Red pixels R_{NE} (the Northeast neighbor), R_{SE} (the Southeast neighbor), R_{SW} (the Southwest neighbor), and R_{NW} (the Northwest neighbor), whose color blocks are g_{NE}, g_{SE}, g_{SW}, and g_{NW} and the corresponding color gravities are $\overline{g_{NE}}$, $\overline{g_{SE}}$, $\overline{g_{SW}}$ and $\overline{g_{NW}}$ respectively. The missing Red component is assumed to be one of these four diagonal Red pixels based on best match of their color gravities. The best match (or minimal difference Δ_{min}) is the minimum of $\Delta_1=|\bar{g} - \overline{g_{NE}}|$, $\Delta_2=|\bar{g} - \overline{g_{SE}}|$, $\Delta_3=|\bar{g} - \overline{g_{SW}}|$ and $\Delta_4=|\bar{g} - \overline{g_{NE}}|$. The missing Blue component in a Red pixel location could be estimated similarly due to the symmetry of Red and Blue sampling position in a Bayer pattern image. For the Green pixel location, only two color blocks (either up-bottom or left-right positions) are considered for the missing Red or Blue color. The algorithm can be described as [1][2]:

```
Begin
   for (each pixel in the Bayer pattern image) do
   {  if (the pixel at location (m, n) is not Green) then
      {     G_{m,n} = median (G_N, G_S, G_E, G_W);
            Δ1=|ḡ - ḡ_NE |; Δ2=| ḡ- ḡ_SE |; Δ3=| ḡ - ḡ_SW |; Δ4=|ḡ - ḡ_NE |;
            Δmin = min (Δ1, Δ2, Δ3, Δ4);
            if (the pixel at location (m, n) is Red) then
            {
            if (Δmin = Δ1 ) then B_{m,n} = B_NE;
            if (Δmin = Δ2 ) then B_{m,n} = B_SE;
            if (Δmin = Δ3 ) then B_{m,n} = B_SW;
            if (Δmin = Δ4 ) then B_{m,n} = B_NW;
            }
            if (the pixel at location (m, n) is Blue) then
            {
            if (Δmin = Δ1 ) then R_{m,n} = R_NE;
            if (Δmin = Δ2 ) then R_{m,n} = R_SE;
            if (Δmin = Δ3 ) then R_{m,n} = R_SW;
            if (Δmin = Δ4 ) then R_{m,n} = R_NW;
            }
      }
      if (the pixel at location (m, n) is Green) then
      {
      Δu = |G - ḡ_u |; Δb = |G - ḡ_b |; Δl = |G - ḡ_l |; Δr = |G - ḡ_r |;
      if (Δu < Δb) then    B_{m,n} = B_u     else B_{m,n} = B_b;
      if (Δt < Δr)  then   R_{m,n} = R_t     else R_{m,n} = R_r;
      }
   }
End.
```

Where $_u$, $_b$, $_l$, and $_r$ are the abbreviations for upper, bottom, left, and right respectively.

3.2.3 Edge Sensing Interpolation Algorithm

In this algorithm, different predictors are used for the missing Green values depending on the luminance gradients. For each pixel containing only the Red or the Blue component, two gradients (one in the horizontal direction, the other in the vertical direction) are defined as:

$\delta H = |G_{m,n-1} - G_{m,n+1}|$, $\delta V = |G_{m-1,n} - G_{m+1,n}|$

Based on these gradients and a certain threshold T, the interpolation algorithm can be described as follows:

Begin
if ($\delta H<T$ and $\delta V>T$ (that is, smoother in horizontal direction)) then
$G_{m,n} = (G_{m,n-l} + G_{m,n+1})/2$;
else
if ($\delta H>T$ and $\delta V<T$ (that is, smoother in vertical direction)) then
$G_{m,n} = (G_{m-l,n} + G_{m+l,n})/2$;
else
$G_{m,n} = (G_{m,n-l} + G_{m,n+1} + G_{m-l,n} + G_{m+l,n})/4$.
End.

To estimate Blue and Red components two diagonal gradients are defined; one for left diagonal and the other for right diagonal. Also, the same algorithm described above is implemented, the only difference is that the diagonal values are used rather than the horizontal and vertical values [1][2].

3.2.4 Linear Interpolation with Second-Order Corrections

This algorithm was developed with the goal of enhanced visual quality of the interpolated image when applied on images with sharp edges. Missing color components are estimated by the following steps:

1- Estimation of Green Component: estimating the missing Green component $G_{m,n}$ at a Blue pixel $B_{m,n}$ at location (m,n), as an example. Interpolation at a Red pixel location is done in the similar fashion. Horizontal and vertical gradients are defined in this pixel location as follows:

$$\delta H = |G_{m,n-1} - G_{m,n+1}| + |(B_{m,n} - B_{m,n-2}) - (B_{m,n+2} - B_{m,n})|$$
$$\delta V = |G_{m-1,n} - G_{m+1,n}| + |(B_{m,n} - B_{m-2,n}) - (B_{m+2,n} - B_{m,n})|$$

In the expression of δH, the first term $|G_{m,n-l} - G_{m,n+l}|$ is the first-order difference of the neighboring green pixels, considered to be the gradient and the second term $|(B_{m,n} - B_{m,n-2}) - (B_{m,n+2} - B_{m,n})|$ is the second-order derivative of the neighboring blue pixels. Using these two gradients, the missing green component $G_{m,n}$ at location (m,n) is estimated as follows:

if ($\delta H < \delta V$) then
$G_{m,n} = (G_{m,n-1} + G_{m,n+1})/2 + (2*B_{m,n} - B_{m,n-2} - B_{m,n+2})/4$;
else
if ($\delta H > \delta V$) then

$$G_{m,n} = (G_{m-1,n} + G_{m+1,n})/2 + (2*B_{m,n} - B_{m-2,n} - B_{m+2,n})/4;$$

else

$$G_{m,n} = (G_{m,n-1} + G_{m,n+1} + G_{m-1,n} + G_{m+1,n})/4 + (4*B_{m,n} - B_{m,n-2} - B_{m,n+2} - B_{m-2,n} - B_{m+2,n})/8.$$

The second-order correction term based on the neighboring blue or red components.

2- Estimation of Red (or Blue) component: The missing Red (Blue) color components are estimated in every pixel location after estimation of the missing Green components is complete. Depending on the position, we have three cases:

a) Estimate the missing Red (Blue) component at a Green pixel $G_{m,n}$, where nearest neighbors of Red (Blue) pixels are in the same column, as:
$R_{m,n} = (R_{m-1,n} + R_{m+1,n})/2 + (2*G_{m,n} - G_{m-1,n} - G_{m+1,n})/4.$

b) Estimate the missing Red (Blue) component at a Green pixel $G_{m,n}$, where nearest neighbors of Red (Blue) pixels are in the same row, as:
$R_{m,n} = (R_{m,n-1} + R_{m,n+1})/2 + (2*G_{m,n} - G_{m,n-1} - G_{m,n+1})/4.$

c) Estimate Red (Blue) component at a Blue (Red) pixel. In this case two diagonal gradients must be defined as:

$$\delta D1 = |R_{m-1,n-1} - R_{m+1,n+1}| + |(G_{m,n} - G_{m-1,n-1}) + (G_{m,n} - G_{m+1,n+1})|$$

and

$$\delta D2 = |R_{m-1,n+1} - R_{m+1,n-1}| + |(G_{m,n} - G_{m-1,n+1}) + (G_{m,n} - G_{m+1,n-1})|$$

Using these diagonal gradients, the algorithm for estimating the missing color components is described as:

if ($\delta D1 < \delta D2$) then

$$R_{m,n} = (R_{m-1,n-1} + R_{m+1,n+1})/2 + (2*G_{m,n} - G_{m-1,n-1} - G_{m+1,n+1})/2;$$

else

if ($\delta D1 > \delta D2$) then

$$R_{m,n} = (R_{m-1,n+1} + R_{m+1,n-1})/2 + (2*G_{m,n} - G_{m-1,n+1} - G_{m+1,n-1})/2;$$

else

$$R_{m,n} = (R_{m-1,n-1} + R_{m+1,n+1} + R_{m-1,n+1} + R_{m+1,n-1})/4 + (4*G_{m,n} - G_{m-1,n-1} - G_{m+1,n+1} - G_{m-1,n+1} - G_{m+1,n-1})/4.$$

This algorithm provides much better visual quality of the reconstructed image containing a lot of sharp edges. In other case, the second-order calculations of the gradients give some unacceptable results [1][2][3].

3.2.5 Modified Linear Interpolation with Second-Order Corrections

This new invented method is based on linear interpolation method discussed above in addition to bilinear algorithm. The idea is to compare the second-order correction term with a certain threshold value, if this term is greater than the threshold value then the correction term is eliminated and only a linear average is computed. Otherwise, the correction term is added. For example, the missing Green component $G_{m,n}$ at location (m,n) is estimated as follows:

```
if (δH < δV) then
{       Correction term = (2*B_{m,n} – B_{m,n-2} – B_{m,n+2})/4;
        if (Correction term < Threshold) then
                G_{m,n} = (G_{m,n-1} + G_{m,n+1})/2 + Correction term;
        else    G_{m,n} = (G_{m,n-1} + G_{m,n+1})/2;
}
else {
if (δH > δV) then
{       Correction term =  (2*B_{m,n} – B_{m-2,n} – B_{m+2,n})/4;
        if (Correction term < Threshold)  then
                G_{m,n} = (G_{m-1,n} + G_{m+1,n})/2 + Correction term;
        else    G_{m,n} = (G_{m-1,n} + G_{m+1,n})/2;
}
else
{       Correction term = (4*B_{m,n} – B_{m,n-2} – B_{m,n+2} – B_{m-2,n} – B_{m+2,n})/8;
        if (Correction term < Threshold) then
                G_{m,n} = (G_{m,n-1} +G_{m,n+1} +G_{m-1,n} +G_{m+1,n})/4 + Correction term;
}
        }
```

4. Program Interface

The program is created using Visual C++ programming language, as MFC AppWizard (exe) of type Single Document Interface (SDI) [8][9].

Besides its basic function i.e., image demosaicing, the project has other properties such as image scrolling, file serialization, and providing (Signal to Noise Ratio) SNR results.

The base class that is used for viewing is derived from CScrollView rather than CView class, because the latter doesn't provide the utilities for scrolling, which is very important in viewing high-resolution images [8][9], see appendix A.

File Serialization is the method used by MFC programs to read and write application data to files, with the aid of CArchive class, which is used to model a generic storage object. In most cases, a CArchive object is attached to a disk file[8][9], see appendix A. Figure 2 shows the main program interface.

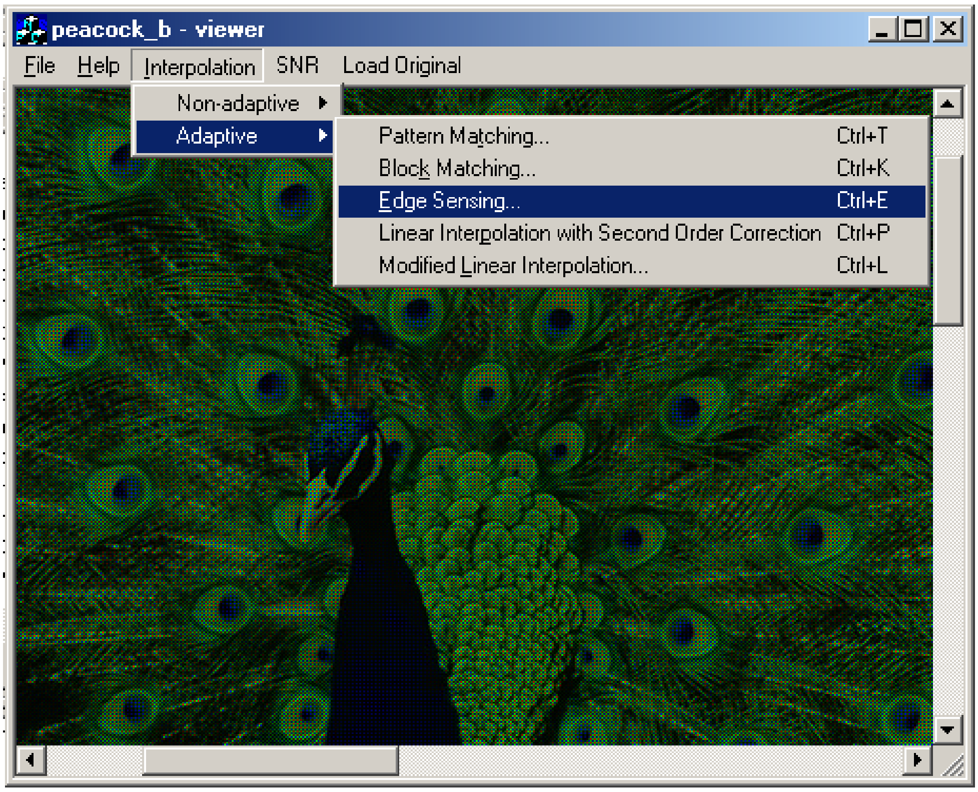


Figure 2: Main Program Menu.

5. Creating Bayer Pattern

To show the effectiveness of the demosaicing algorithms, the Bayer pattern is synthetically generate from a full-color RGB image by simply dropping two color components in each pixel coordinate (if the original Bayer pattern image cannot be grabbed from a digital camera directly) [1][2][4].

6. Experimental Results

More than twenty images have been chosen to work with depending on their high frequency nature and other interesting image characteristics to demonstrate performance of different color interpolation algorithms.

In order to compare between the color intensity of the original image the resultant image, signal to noise ratio measure is used. The following formula clarifies the relation between two images [10]:

$$SNR = -10 \times \log_{10} \left(\sum (X_i - W_i)^2 / \sum X_i^2\right).$$

Where X refers to the original image and W refers to the image produced from the selected demosaicing process. The formula is applied three times for Red, Green, and Blue intensities of each pixel in the true color bitmap. Table 1 shows the comparison results of interpolation algorithms depending on Green component, as Green is sampled in Bayer image at a higher rate than Red and Blue. Table 2 and 3 give the SNR results.

Figure 3(a) is a full color image of a parrot. Bayer pattern image is generated from this image, as shown in Figure 3(b). The Figures 3(c), (d), (e), (f), (g), (h), (i), (j), and (k) show the resultant images produced by applying the methods: nearest neighbor, bilinear, median, smooth hue transitions, pattern matching, block matching, edge sensing, linear interpolation with second order correction, and the new enhanced linear interpolation, respectively, on Bayer pattern image.

Table 1: Methods Comparison.

Method	SNR Average for Green Color Component	Quality Percentage
Nearest Neighbor	21.2005	57.2987
Bilinear	25.5773	69.1279
Median	24.0728	65.0618
Smooth Hue Transition	26.0348	70.3643
Pattern Matching	26.0348	70.3643
Block Matching	26.0348	70.3643
Edge Sensing	25.8151	69.7706
Linear with Second Order	21.3119	57.5997
Enhanced Linear Intr.	26.7709	72.3539

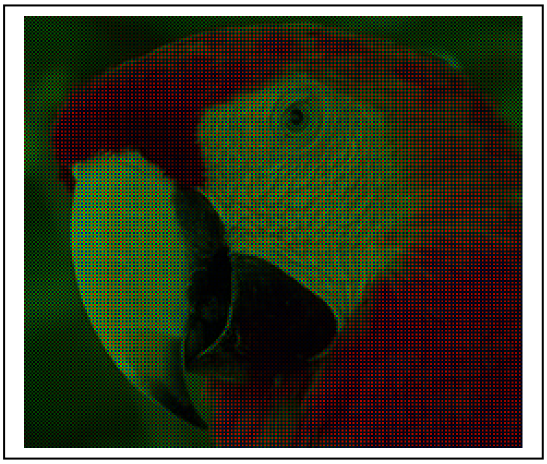
Table 2: SNR Values.

Method	Color Component	Image			
		Butterfly1 (768 * 512)	Parrot (768 * 512)	Pussy (800 * 600)	Peacock (768 * 512)
Nearest Neighbor	Red	17.2642	21.5438	24.2019	8.24024
	Green	15.4311	26.5922	27.7852	13.9387
	Blue	12.5823	19.7319	22.6687	8.1647
Bilinear	Red	20.3529	27.5987	33.1196	10.4388
	Green	19.2571	33.6041	29.4472	17.1379
	Blue	14.4144	21.4710	25.3966	10.1288
Median	Red	17.7912	23.6269	24.5813	8.0758
	Green	17.9295	31.1944	30.4355	15.4444
	Blue	12.0589	18.2850	22.1538	7.7778
Smooth Hue Transition	Red	19.6328	14.2431	26.4471	14.0538
	Green	19.2571	33.6041	33.1196	17.1379
	Blue	17.3365	19.6136	28.8633	13.3238
Pattern Matching	Red	21.6453	29.3152	27.5663	11.6035
	Green	19.2571	33.6041	33.1196	17.1379
	Blue	15.5485	22.6358	25.3364	11.2409
Block Matching	Red	20.6620	25.7953	26.2346	10.9742
	Green	19.2571	33.6041	33.1196	17.1379
	Blue	14.5794	21.6791	24.1764	10.6060
Edge Sensing	Red	21.4997	29.9995	29.7173	11.3670
	Green	19.1911	33.5922	32.9667	16.9402
	Blue	15.5158	24.3467	27.0233	11.0573
Linear with Second Order	Red	15.3675	12.6327	12.5763	8.7051
	Green	15.2444	19.6021	32.7494	18.2609
	Blue	9.27391	12.0230	25.3256	7.6137
Enhanced Linear Intr.	Red	20.5277	25.3514	25.3639	10.5497
	Green	19.9202	36.2225	34.5294	18.1374
	Blue	15.3721	22.9022	25.5705	10.9676

Table 3: SNR Values.

Method	Color Component	Image			
		Eye (384 * 288)	Calyx (640 * 427)	Butterfly5 (768 * 512)	Field (1024 * 768)
Nearest Neighbor	Red	20.3526	22.4850	14.8668	14.8262
	Green	24.4613	24.0084	18.3724	19.0149
	Blue	19.5368	17.7169	13.1508	16.5368
Bilinear	Red	22.6645	28.2561	17.6342	17.5685
	Green	28.2766	29.5607	23.4976	23.8376
	Blue	22.1791	20.1288	15.5018	18.2977
Median	Red	20.4266	24.2902	14.7751	14.8609
	Green	26.7556	27.3613	21.2890	22.1733
	Blue	19.3490	16.9487	12.8151	15.8505
Smooth Hue Transition	Red	21.4924	11.5052	21.3284	19.7337
	Green	28.2766	29.5479	23.4976	23.8376
	Blue	20.1592	21.9343	18.9117	21.1158
Pattern Matching	Red	20.4796	30.2983	19.4524	18.4214
	Green	28.2766	29.5479	23.4976	23.8376
	Blue	21.3188	21.7146	17.0740	19.0850
Block Matching	Red	19.0501	27.3461	18.2049	17.6655
	Green	28.2766	29.5479	23.4976	23.8376
	Blue	21.2567	19.6617	16.0073	18.3378
Edge Sensing	Red	23.5193	30.5532	19.0462	18.8273
	Green	27.2448	29.5247	23.3417	23.7196
	Blue	24.0469	21.7802	16.8060	19.5277
Linear with Second Order	Red	9.8191	8.9287	11.2109	11.6121
	Green	23.766	15.9485	20.6218	24.3021
	Blue	18.246	4.2126	9.6298	15.9991
Enhanced Linear Intr.	Red	17.5466	18.2836	18.2226	17.0685
	Green	28.5913	27.3947	24.4639	24.9084
	Blue	20.3406	21.9026	16.8629	19.1218

(a)



(b)

(c)

(d)

(e)

(f)

(g)

(h)

(i)

(j)

(k)

Figure 3: (a) Original 24-bit full color image, (b) Bayer Image (c) Nearest Neighbor, (d) Bilinear, (e) Median, (f) Smooth Hue Transitions, (g) Pattern Matching, (h) Block Matching, (i) Edge Sensing, (j) Linear interpolation with second order correction, (k)The new enhanced linear interpolation.

7. Conclusion

The comparison between the different applied demosaicing algorithms presents the following conclusions:

- Applying nearest neighbor method gives some significant color errors, which make it unacceptable for all types of images.
- Bilinear algorithm gives better results than other non-adaptive methods and in some cases adaptive methods, as bilinear takes into account the gradual transition of pixel color values.
- Median method provides a slightly better result as compared with nearest neighbor.
- Applying smooth hue transition on images having the characteristic of hue values changing suddenly of adjacent pixels, gives results much better than other non-adaptive or adaptive methods.
- Best results are achieved when using pattern matching, block matching, and the new enhanced linear algorithms. This is because of the accurate approach to find the best color component match. Also, they rely on bilinear method, as the latter gives good results.
- Edge sensing gives better results and high quality images depending on the threshold value and when applying on sharp edges images.
- Linear interpolation with second order correction method gives good image quality for sharp edges image. However, it has some unacceptable results and disparate pixels can be obviously noticed.
- The new enhanced linear interpolation method, proposed in this paper, provides better image quality as compared with the both types. Also, it gives the best image quality than the original linear interpolation method for all the used images.

References

1) T. Acharya and A. K. Ray, “Image Processing Principles and Applications”, John Wiley & Sons, INC., Publication, 2005.

2) P. Tsai, T. Acharya, and A. K. Ray, “Adaptive Fuzzy Color Interpolation”, Journal of Electronic Imaging, Vol. 11 No. 3, 293-305, July 2002.

3) Rajeev Ramanath, Wesley E. Snyder, and Griff L. Bilbro, “Demosiacking Methods for Bayer Color Array”, Journal of Electronic Imaging, July 26, 2003.

4) Gary L. Embler, “Correlation-based color mosaic interpolation using connectionist approach”, Society of Photo-Optical Instrumentation Engineers (SPIE) Vol. 4669, 2002.

5) John C. Russ, “The Image Processing Handbook”, Third Edition, CRC Press, 1998.

6) Rafael C. Gonzalez and Richard E. Woods, “Digital Image Processing”, Second Edition, Prentice-Hall, Inc., 2002.

7) L. Guan, S. Kung, and J. Larsen, “Multimedia Image and Video Processing”, CRC Press, 2001.

8) M. Williams, “Teach Yourself Visual C++ 5 in 24 Hours”, Macmillan and Sams Publishing, 1999.

9) Davis C. and Jon B., “Teach Yourself Visual C++ 6 in 21 Days”, Macmillan Publishing, 1999.

10) Pratt, W.K., "Digital Image Processing", John Wiley & Sons, 1978.

Appendix (A)

```
void CViewerView::OnInitialUpdate()
{
	CScrollView::OnInitialUpdate();
	CSize sizeTotal;
	sizeTotal.cx = x_res;	sizeTotal.cy = y_res;
	SetScrollSizes(MM_TEXT, sizeTotal);
}
void CViewerDoc::Serialize(CArchive& ar)
{
	if (ar.IsStoring())
	{
ar.Write(ImageSave,54+infoHeader.biHeight*infoHeader.biWidth*3 );
	ar.Close();
	}
	else
	{
		CFile* pFile = ar.GetFile();
		pFile->Read(&fileHeader, sizeof(fileHeader));
		pFile->Read(&infoHeader, sizeof(infoHeader) );
		pFile->SeekToBegin();
	if( infoHeader.biSize != sizeof(infoHeader) )
		AfxMessageBox(_T("Error in BMP file!"));
pFile->Seek(fileHeader.bfOffBits, CFile::begin);
m_pPixels = new BYTE [fileHeader.bfSize - fileHeader.bfOffBits];
pFile->Read( m_pPixels,fileHeader.bfSize - fileHeader.bfOffBits);
	/*covert m_pPixels from 1D array to 2D array*/
for(int y=0;y<infoHeader.biHeight;y++)
	for(int x=0;x<infoHeader.biWidth;x++)
	{
pixels[y][x].red=m_pPixels[(((infoHeader.biHeight-1)-y)*
			(infoHeader.biWidth*3)+(x*3))+2];
pixels[y][x].green=m_pPixels[(((infoHeader.biHeight-1)-y)*
			(infoHeader.biWidth*3)+(x*3))+1];
pixels[y][x].blue=m_pPixels[(((infoHeader.biHeight-1)-y)*
			(infoHeader.biWidth*3)+(x*3))];
	}
}
}
```