A Practical Bayer Demosaicing Algorithm for Gphoto

Theodore Kilgore
Mathematics and Statistics
Auburn University, Alabama 36849

Abstract ***The Gphoto project exists to provide free and open-source software support for USB and serial digital still cameras which use miscellaneous proprietary interface protocols and image data formats. The cameras supported range from very cheap to very expensive, but one thing that many of them need is a good and practical Bayer demosaicing algorithm. The algorithm should get good results with cheap cameras. The algorithm should not require a memory footprint which is unduly heavy, and the algorithm should not require excessive time to run, because gphoto should run even on fairly small computers. However, the priority is to support still cameras, not pccams or webcams, which makes the execution time to be a secondary issue, not a primary one. Recently, a practical implementation of the adaptive homogeneity-directed Bayer demosaicing algorithm has been written, which meets the above criteria and seems to give very nice results.***

Keywords: Visual imaging, Bayer interpolation, demosaicing, Gphoto project

1 Introduction

Behind the lens of a digital camera is a sensor chip. On the surface of the sensor chip, a large number of photosensitive microsensors are arranged in a rectangular grid. For an uncompressed bitmapped image, the rectangular array is w sensors wide and h sensors high, where w is the pixel width of the finished image and h is its pixel height. Thus, for example, a 640x480 image will be produced by an array containing 480 rows of microsensors, in which each row is 640 microsensors wide.

At each of these microsensors, the intensity of light striking it is recorded when a picture is snapped. The data may then be compressed. However, for the present purpose we will assume either that no such compression has taken place or that decompression has already taken place. With only few exceptions, a consumer-grade digital camera produces a color photo by having a grid of color filters placed between the microsensors and the lens. Often, the color filters are also microlenses which better capture the intensity of the light, as well as giving preference to a certain color. A standard practice is to use red (R), green (G), and blue (B). The microsensors with their filters on them are arranged in a pattern often called a Bayer array, named after its inventor, B. E. Bayer. There are four possible variants called tilings. A typical one looks like

row 0 RGRGRG ... (until w is reached)
row 1 GBGBGB ... (until w is reached)

and then the two-row pattern is repeated downwards, until there are h rows in all. This pattern is called RGGB, because it consists of repetitions of a basic 2x2 tile which looks like

```
R G
G B
```

Three other patterns, called BGGR and GRBG and GBRG, can achieve similar results. In all of these four Bayer tiling patterns there is twice as much green data as there is of either red or blue. The reason for this is based upon known visual qualities of the human eye. In any event, the raw, uncompressed data from many digital cameras consists of precisely $w * h$ bytes of data arranged in one of the standard Bayer tiling patterns.

Now, there is a fundamental problem. To construct an image requires data about each of the three colors at each pixel. In one of the basic, simple formats used both in BMP image data and in PPM (Portable PixMap), one byte of data is required for each of the colors R, G, and B, at each pixel location in the image. Intensity is measured at each microsensor, then, on a scale from 0 to 0xff for the color which is sampled there. Consequently, we have only $w*h$ pieces of data, and we need to have $3*w*h$ pieces of data (one byte for each color at the given pixel location) in our finished image. How to do that? The short answer is: Use educated guesswork. We now discuss some possible methods for doing that. In fact we have a problem which is known from the outset to have no exact solution, which to some extent accounts for the huge proliferation of methods for solving it.

2 Methods of doing Bayer interpolation.

The most primitive method for supplying a missing color at a pixel location is just to take one of the nearest actual readings of the same color. Suffice it to say that the crudity of the results thereby obtained is at least equal to the crudity of the method.

The most obvious approach is to use an averaging of nearest-neighbor data, called bilinear interpolation. This method requires special treatment at edges and corners of the image, obviously, because the number of nearest neighbors is not the same there. To describe the method with simplicity, therefore, let us assume we are well away from edges and corners. We might, for example, need to supply a missing datum for G in the center of

$$\begin{array}{ccc} & \mathrm{G}_1 & \\ \mathrm{G}_2 & \mathrm{C} & \mathrm{G}_3 \\ & \mathrm{G}_4 & \end{array}$$

in which C can signify that either R or B is the color actually sampled. Other possible arrangements can arise if we need to supply a missing datum for either R or B, giving minor but irritating variants on the required procedure. However, in the situation just described the way to do bilinear interpolation is simply to take the average of $\mathrm{G}_1, \mathrm{G}_2, \mathrm{G}_3, \mathrm{G}_4$. Obviously, this procedure can give very nice results if done in a region of an image where things are fairly homogeneous, but the procedure can also act quite badly, causing fuzziness, jaggedness, or other artifacting at edges in an image. One of the ugliest of these artifacts is called the “zipper” effect, or the “checkerboard” effect, in which the edge comes out as alternating pixels of different, contrasting colors or light intensity. An example of the zipper effect may be seen in the first of the images in Section 5.

Bilinear interpolation was in use in the Gphoto project for quite some time, the basic algorithm appearing in libgphoto2/libgphoto2/bayer.c, [5] written by Lutz Müller in 2001. In commercial camera software such as OEM camera driver software, bilinear interpolation is often replaced by similar but more complicated procedures, such as cubic spline interpolation instead of piecewise-linear averaging. In practice, though, the greater complexity does not seem always to do a very much better job.

Some of the more sophisticated methods for doing the interpolation involve such things as gradient searches for the likeliest weighted linear combination of readings at nearby pixels, or involve some basic schemes for edge detection and a method for deciding to use some of the nearby readings and to ignore others, in case there is suspicion of an edge. The method currently in use in the Gphoto project was introduced in [6], in 2007. That method involves some vertical and horizontal edge detection and also gives a special role to the green data, as there is twice as much green data as red or blue. Improvement over the previous bilinear interpolation method was obvious when this edge detection was introduced. Moreover, this improvement in artifact reduction and image clarity was done without great increase in processing complexity or memory footprint.

3 Adaptive Homogeneity-directed Demosaicing

Somewhere, while studying about the interpolation or demosaicing problem, someone stepped back and noticed that, in spite of the difficulties involved in re-creating the different colors at their missing locations all over an image, data about the entire image is present nevertheless. For, at every pixel location one piece of data has been gathered. However, for a long time it seems not to have been clear at all how this fact might actually be helpful toward a better reconstruction of the image. Indeed, it could be said without exaggerating very much, that the interpolation methods described above have lost sight of the fact that data is present for the whole image, while concentrating on reconstructing the missing data for the individual colors, viewed in isolation from one another.

One method which has put these observations to good use is given in [1] of Hirakawa and Parks, in 2005. In briefly describing what is done there, we will assume that the raw data from the camera has been spread out already to a size of $3*w*h$, so that three bytes are allocated for each pixel; the data already present has been put into its proper, respective locations, and the “missing” data is represented by filler bytes which are still 0. This then is the input of the procedure, which will fill in the 0 bytes with data and will give back the same data as an output but with the blank spaces now filled in. Then, the AHD algorithm proceeds as follows:

First, make two additional copies of the image. Let us call them image_v and image_h, because we are going to work in one of the copies only in the vertical direction and in the other one only in the horizontal direction. Quite apart from the hope for an ultimately better outcome, another good reason for doing this is that the estimation procedure is greatly simplified. For, each individual row of data from the image will in fact contain only two colors in an alternating pattern, not all three of the colors. The same is also true of each individual column.

Second, supply the missing green values in each of image_v and image_h. If the G data at a given location in a given row or column is missing and only R or B (depending on the row or column being considered) has been sampled, then the missing G value can be estimated by also using sampled R or B values (again, whichever of these two is present in the given row or column), along with the result of an averaging of the two nearest G values in the given row or column. As already explained, all computations are done only in columns in image_v and only in rows in image_h. The principle which is invoked in [1] for using the other color is, that the local differences in one color are on most occasions very similar to the local differences in the others, too. To describe in more detail what is done, let us assume that the missing G value is at position i in a row or column and for the sake of definiteness that the other color which occurs in the row or column is R, red. Then we have sampling data for G_{i-1} and for G_{i+i}. The initial estimate for G_i is the average of these. Then, Hirakawa and Parks add to this initial estimate a weighted sum of the sampled values $R_i, R_{i\pm 2}$ and $R_{i\pm 4}$ in order to accomplish the correction. Their intent is to choose the coefficients in the weighted sum in order to minimize the difference $G_i - R_i$, and their computation of the globally optimal weighting coefficients leads to some rather ugly decimal values.

After the missing G values have been estimated, the missing R and B values are estimated, too, in a similar fashion.

Finally, the missing values in the original image are estimated. The way this is done is to use a choice algorithm, which decides at each pixel location whether to choose the value from image_v, from image_h, or from an average of those two values. The description of the choice algorithm which is given in [1] is quite complicated and is obviously very resource-intensive to implement. The first step in the algorithm is to convert image_h and image_v, each of which has been completely filled out in accord with the previous steps, from RGB colorspace to CIELAB colorspace, and the choice algorithm is carried out there. The details have no place here, and so they are omitted. It should be obvious, though, that to do this step well is one of the keys to the whole procedure working well.

4 Adaptive Homogeneity-directed Demosaicing for Gphoto

Implementation of a new demosaicing algorithm for Gphoto has been done as part of the required work for a Diploma (Master's degree) in Computer Science by Eero Salminen, at the Helsinki University of Technology, working under Jorma Laaksonen. I provided a list of to-do items related to needs of the Gphoto project, and one of those items was a better Bayer demosaicing algorithm than what we were currently using, which can be found in libgphoto2/bayer.c [6]. As already mentioned, [6], which uses edge detection, is an improved version of the original bilinear algorithm [5]. In spite of the obviously better performance of the newer algorithm over the older one, it was hoped that the results could be improved still more. In this project, Eero Salminen used the AHD algorithm outlined in the previous section as a point of departure.

Before proceeding we should say something about the Gphoto project itself, because the design constraints for this implementation of the AHD demosaicing algorithm are based in large part upon the specific requirements of the Gphoto project. The Gphoto project is licensed under the LGPL license published by the Free Software Foundation. The project's mission is to support all USB-connected and serial-connected digital still cameras which do not use the standard Mass Storage protocol for communication by the computer. In libgphoto2-2.4.1, released on March 28, 2008, we support 967 different cameras. Furthermore, the intent is to provide support for every possible hardware and software host platform. The language of the project is C. The code is supposed to compile with any C compiler and is supposed to be able to run on, essentially, any hardware which has a USB host interface. Thus, the design priorities for our software are different from those for a project which supports only i386 processors, or only one operating system, or which provides code intended for only one compiler. In contrast to some other Free Software projects such as the Linux kernel, which has been designed upon the GCC compiler and relies upon many of its specific features, one of our explicit goals is maximum portability. The Gphoto project description is therefore the basis for the following design constraints upon the implementation of a Bayer demosaicing algorithm.

• First of all, the algorithm as a whole should give good and pleasing results but should not be inordinately slow, even if run on a somewhat slow host machine. However, we do not do webcams, pccams, or streaming video in Gphoto. Thus, even though speed is a consideration it is possible to make small sacrifices in speed for great gains in image quality, for reducing the memory footprint, or for producing code which is easier to read and easier to maintain.

• Second, our Bayer demosaicing algorithm should be able to run even on slow or low-powered devices. It should not have too great a memory footprint, in case that available RAM is barely sufficient to hold the the photo while processing it, in the first place. This is probably the most stringent requirement, besides the question of image quality.

• Third, if it is possible to simplify or to approximate what an "ideal" algorithm is doing, then that simplification or approximation should be implemented, assuming that there is little or no visible difference to the naked eye in the output. In other words, theoretical gains in image quality which cannot be obviously seen or which cannot in practice be realized with most of the supported cameras but which use a lot of resources, ought not to be included. Our tools can also get and save the raw image files from the camera, instead of making photos from them during the download. Thus, any user whose priority is perfection in image processing and for whom time and resource requirements are not a factor can still get the raw files and is free to use other software for processing them. For us, however, the perfect may be the enemy of the good.

• Fourth, extensive floating point calculations should simply be avoided whenever possible. Everything which can be done exactly or done well enough with integer arithmetic should be done with integer arithmetic. Not all platforms will run floating point calculations very fast or very well, and some do not even provide a math coprocessor.

• Fifth, the algorithm should be for general use and should be suitable for a large range of consumer cameras. At the opposite extreme from us would be, for example, a government agency, intending to put up a weather satellite or spy satellite with just one camera in it, and then the design objective would be to write the software which gets the absolute maximum performance out of that one unique specimen of a camera. We on the other hand cannot design our software around just one camera, or even one make and model of camera.

• Sixth, the knowledge of how to carry out the algorithm needs to be public and not, so far as we know, something which is covered by some patent. The AHD algorithm was published in the article [1]. We are aware of no impediments to the use of an algorithm which has appeared in a refereed research publication.

• Seventh, there should be no impediment to publishing the algorithm under the copyright license of the Gphoto project, which is the Lesser Gnu Public License, version 2 or any later version. The requirements of this copyright license are, I believe, well known. In any case the license is freely available for inspection, too.

The new version of the Gphoto demosaicing algorithm, which uses the AHD algorithm implemented by Eero Salminen and further adapted by T. Kilgore, is now added to the Gphoto source tree, as [4]. The implementation functions quite well, as evidenced by the photos in Section 5, where some samples of the same images are shown, processed with different algorithms. The new algorithm also meets the objectives given above.

As to the speed, the new AHD algorithm requires only about three times as long as the old bilinear algorithm with edge detection. This means, approximately 130 milliseconds is needed to do the interpolation of a 640x480 image, on a machine with a Sempron 2600+ and 1G of RAM. The speed seems to be too slow for a webcam but is barely noticeable to someone who is downloading the photos from a still camera and is thus quite acceptable for that application.

As to the memory footprint, the amount of memory required to do the AHD procedure is not significantly larger than the size of the image itself. The way this problem has been addressed is to use sliding windows in which to do the interpolation and choice procedures, with data written in, processed, put away, and then rotated out, instead of using two complete copies of the image, as was envisioned in the paper [1]. The sliding windows are of width equal to the image, but only need to contain six consecutive rows of image data.

As to simplifications, the algorithm of [1] uses a weighted sum of five nearby samples to do the basic interpolation in image_v and image_h, using as weighting coefficients float variables of four decimal places. We replaced these with using only three samples, with coefficients -1, 2, and -1, which are integers. Specifically, our formula for computing a missing value G_i is

$$G_i = (2 * R_i + 2 * G_{i+1} + 2 * G_{i-1} - R_{i-2} - R_{i+2})/4$$

with a similar method used to compute a missing R or B value. Presumably, the AHD algorithm of [1] does a better job, but this simple procedure seems to work quite well.

Another simplification occurs in the implementation of the choice algorithm, which at each pixel chooses whether to use the horizontally-interpolated data or the vertically-interpolated data in order to fill in the values for that pixel. The AHD algorithm of [1] maps the RGB data in the image over into the CIELAB color space in order to do the choice algorithm a little bit better. The cost in processing time and memory usage of doing this is obviously horrendous, and, in line with the design objectives described above, those steps have been done differently. For clarity, the explanation below pretends that we are deciding at a given pixel to choose the data from image_h or from image_v, as described in the previous section. We are in fact working in sliding windows, but the difference is not important for the description. Our simplified choice algorithm works as follows:

In image_h, a difference between the data at the given pixel and each of the ones to its left and to its right is computed, by taking the sum of the squares of the differences for each of R, G, and B. Then the max of these two differences is computed. In image_v, the procedure is analogous, but is performed vertically. Then, the radius of a neighborhood in which to work is computed as the smaller of the maximum vertical reading from image_v and the maximum horizontal reading deriving from image_h. The procedure just described is greatly simplified from what is done in [1], but seems to work well.

Then, in each of image_h and image_v, points are assigned to the given pixel location based on what happens in going to the next pixel in each of the four cardinal directions from the given location. The way this is done is, first of all each of these differences is computed in the same manner as above, but this time in image_v and again in image_h. in each of the four directions at the pixel location. Then, in each of image_v and image_h separately, if for a given direction the computed difference is less than the previously-calculated neighborhood radius, a point is tallied. If the pixel gets a higher score with the tally done in image_h, then the horizontal interpolation procedure seems to be superior at that particular pixel. Similarly, if the pixel gets a higher score with the tally done in image_v, then the vertically interpolated value would seem to be superior. However, this is anticipating a bit. The choice will depend not solely upon what happens at the point, but also upon a composite of what happens nearby.

The final act in the choice algorithm is to sum up at a given pixel location the points deriving from that location and deriving from the same computation steps done at each of its eight nearest neighbors in each of image_h and image_v. If the tally from image_h is greater, one now chooses the data calculated at the given pixel in image_h and uses that data at the corresponding location in the output image. If the tally from image_v is greater, then one uses that data instead. if the two tallies are equal, then at the given location one takes an average of the data from image_h and image_v.

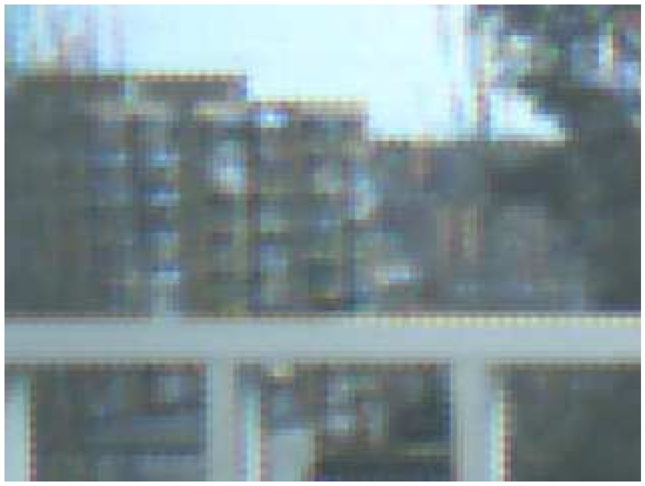
Several standalone image processing routines for raw files from various cameras were used to produce the images in the next section and also helped very much in the development of the AHD algorithm, making it possible to transmit images in raw format for identical testing and processing by people in distant locations. The processing routines were originally written for the purpose of experimentation, both with decompression routines for cameras which emit a compressed raw data format, and with problems relating to the improvement of image quality, of which the Bayer demosaicing problem is only one of many. For these purposes, programs which can process a raw image from the camera many times over, with small code changes for experimentation, are invaluable. This is especially true if the program can do the image processing on the fly, starting from the raw and possibly compressed data, processing it, writing an output file for each image, and displaying the result in a GUI window. In the development of this Bayer demosaicing algorithm, the fact that such raw-file processors already had been written was also very important for rapid progress, because then any raw files from supported cameras could be processed by anyone who had the programs. These raw-file processors may be found at [2]. They are licensed in part by the GPL and in part by the LGPL copyright licenses. By design, also, the code for each of these camera processors is modular, to the extent that one source file contains only the Bayer and gamma functions. It is easy, therefore, to replace this file in order to test a different version of any functions contained in it.

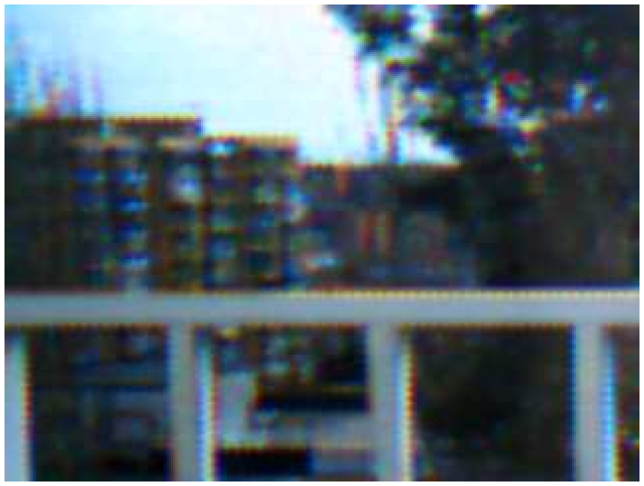
5 Images

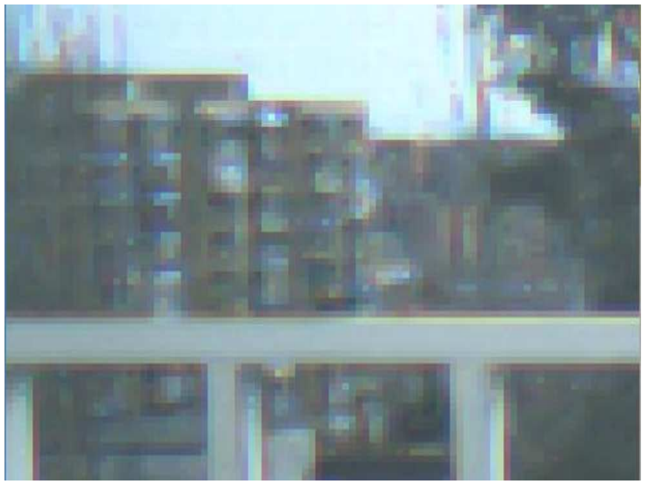
Five approximately identical details from the same raw image data follow. In image 1 we use the bilinear interpolation of [5], then in image 2 same bilinear algorithm, with color balance from [2] also applied. Then the third and fourth images use the edge detection scheme of [6], the second of the pair, again, is also processed with the color balance function of [2]. Finally, the last of the five uses the Gphoto AHD algorithm developed with Eero Salminen, as found both in [2] and in [4], along with the color balance from [2]. The pixel dimensions of the original image for these five were 320x240, and the camera uses a lossy differential Huffman encoded compression algorithm. We see a magnified small piece of the upper left quadrant. The photo was shot with a MiniShotz ms-350 by Ginés Carrascal de las Heras, in Madrid. Ginés also sent me images 1 and 3 balance back in 2007, during a test run of the edge detection function. In image 1, which uses bilinear interpolation only, the zipper effect is obvious and remains so even when the more recently developed correction technique is applied, too, in image 2. The second image which Ginés sent, image 3, uses the edge detection scheme which was introduced in [6]. In it, the zippering is gone. However, on close inspection the sharp edges are in fact all shadowed or are afflicted with color artifacting. The same is true in the later version, image 4, produced with the color balance function of [2]. The superiority of the adapted AHD algorithm in image 5 is obvious. Image 5, of course, also incorporates the routine for color balance which is in current use in show_sonix_raw, found at [2].

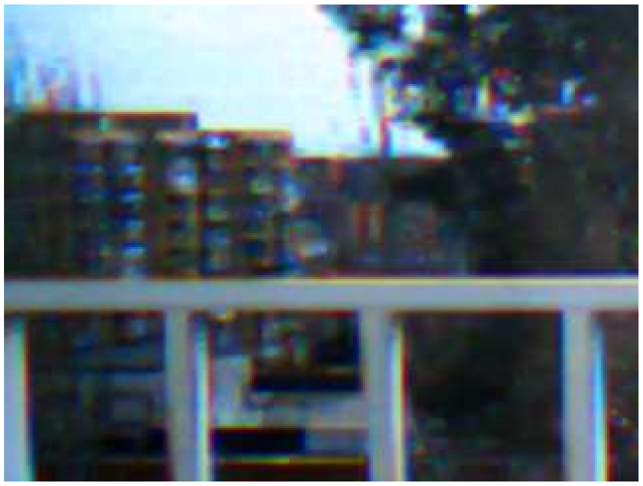
After this series follows in images 6 and 7 another comparison of the bilinear algorithm with edge detection and the adapted AHD algorithm, used upon a photo taken by Norayr Chilingaryan on a street in Zürich. Norayr's camera is a Genius Smart 300; the photo is 640x480 resolution, not compressed. Identical procedures were used for processing image 6 and image 7, except for the difference in the Bayer algorithm. Again, these photos were processed with show_sonix_raw as found at [2].

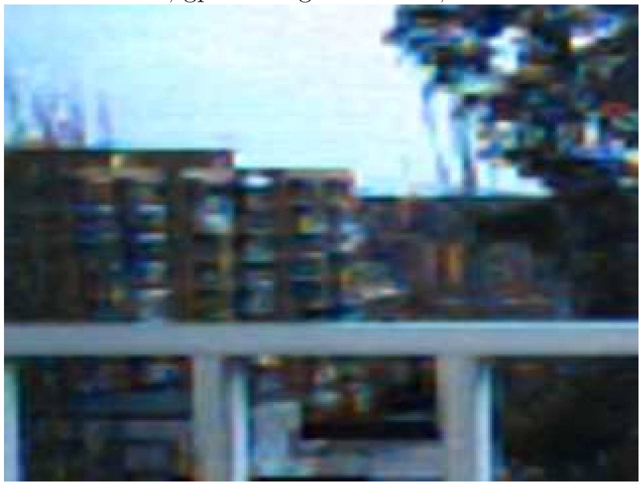
These and more sample images can also be found at [3].

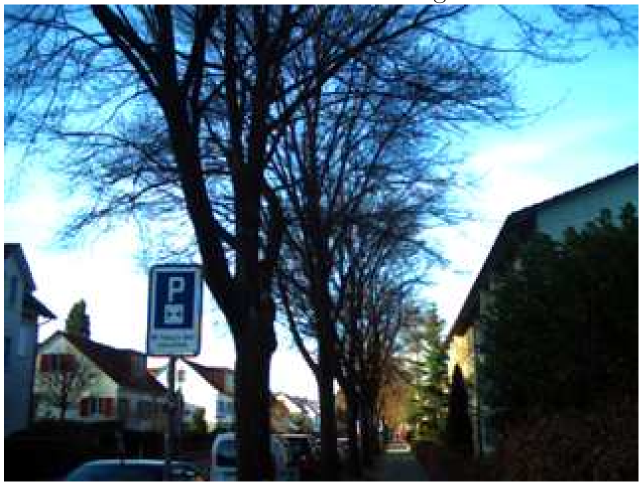

1. Bilinear interpolation, showing zipper effect

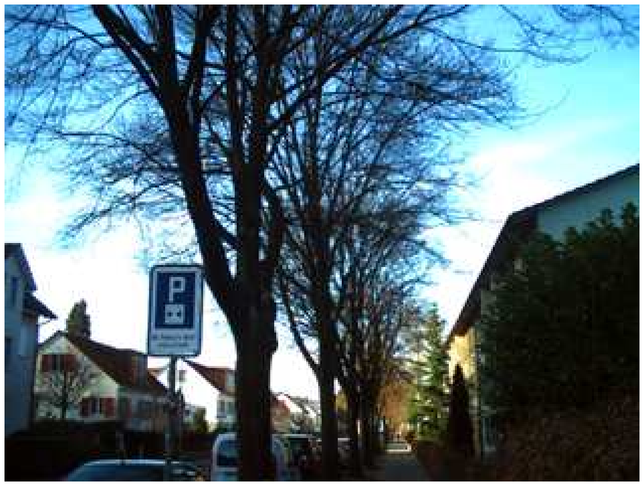

2. Bilinear with color balance


3. Bilinear interpolation with gphoto edge detection


4. Bilinear, gphoto edge detection, color balance


5. AHD demosaicing


6. Bilinear interpolation with gphoto edge detection


7. AHD demosaicing

References

[1] Keigo Hirakawa and Thomas W. Parks. “Adaptive Homogeneity-Directed Democsaicing Algorithm”; IEEE Transactions on Image Processing, 14. 3. 360-369, 2005

[2] Theodore Kilgore. /trunk/playground/raw_converters/*/, <www.gphoto.svn.sourceforge.net>, 2008

[3] Theodore Kilgore. <www.auburn.edu/∼kilgota/demosaic_samples>

[4] Theodore Kilgore and Eero Salminen. /trunk/libgphoto2/libgphoto2/ahd_bayer.c, <www.gphoto.svn.sourceforge.net>, April, 2008.

[5] Lutz Müller. /libgphoto2/libgphoto2/bayer.c, <www.gphoto.svn.sourceforge.net>, (2001 version)

[6] Lutz Müller and Theodore Kilgore. /trunk/libgphoto2/libgphoto2/bayer.c, <www.gphoto.svn.sourceforge.net>, 2007 version