Improvements of Demosaicking and Compression for Single Sensor Digital Cameras

by

Colin Ray Doutre

B. Sc. (Electrical Engineering), Queen's University, 2005

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

Master of Applied Science

in

THE FACULTY OF GRADUATE STUDIES

(Electrical & Computer Engineering)

THE UNIVERSITY OF BRITISH COLUMBIA

February 2007

Abstract

Most consumer digital cameras capture colour images using a single light sensor and a colour filter array (CFA). This results in a mosaic image being captured, where at each pixel location either a red, green, or blue sample is obtained. The two missing colours at each location must be interpolated from the surrounding samples in a process known as demosaicking. In most current digital cameras, demosaicking is carried out in RGB colour space and later the image or video is converted to YCbCr 4:2:0 format and compressed with standard methods. Most previous research on demosaicking and compression in digital cameras has addressed the issues separately. That is, current demosaicking methods ignore the fact that the data will later be compressed, and compression techniques ignore the fact that the data was captured with a CFA.

In this thesis we propose two methods for optimizing the demosaicking and compression processes in digital cameras. First we propose a fast demosaicking method that directly produces an image in YCbCr 4:2:0 format (the colour format most commonly used in compression). This reduces the computational complexity relative to the conventional approach of performing demosaicking in the RGB space and later converting to YCbCr 4:2:0 format. Second, we propose two methods for compressing video captured with a CFA prior to demosaicking being performed. This allows us to take advantage of the smaller input data size, since demosaicking expands the number of samples by a factor of three. Our first CFA video compression method uses standard H.264, while our second method uses a modified motion compensation scheme that further increases compression efficiency by exploiting the properties of CFA data.

Table of Contents

List of Tables

List of Figures

1 Introduction

Consumer digital cameras have gained widespread popularity in recent years and have replaced analog film and tape cameras as the most common means for home users capturing still images and video. In capturing an image, many processing steps are carried out by the camera before it can be stored and viewed by the user. Two critical steps are demosaicking (colour filter array interpolation) and data compression.

Due to the properties of the human visual system, at least three colour samples are needed at each pixel location to represent a full colour image. Most digital cameras use red, green and blue samples. One way to design a digital camera is to have three separate sensors for capturing a red, green and blue sample at every pixel location. However, only high-grade cameras use this design as it is expensive to implement. Instead, most consumer digital cameras use a single light sensor together with a colour filter array (CFA). The CFA allows only one colour of light to reach the sensor at each pixel location. This results in a mosaic image, where at each pixel location either a red, green or blue sample is captured, with the CFA controlling the pattern of the colour samples. The two missing colour samples at each location must be interpolated from the surrounding samples. This process is known as colour filter array interpolation or demosaicking.

Another critical processing step in digital cameras is data compression. The large amount of data required for high-resolution images and video sequences makes efficient compression necessary in order to keep the camera's memory requirements reasonable. For still images, the JPEG (Joint Photographic Experts Group) standard [1] is by far the

most common compression method used. For video sequences, a number of different compression standards are available, the most popular being ITU-T H.263 [2], ISO/IEC MPEG-2 [3], and, most recently, H.264/AVC [4].

Virtually all of the work done in the area of single sensor digital cameras has addressed the issues of demosaicking and compression separately. Most demosaicking algorithms have been designed without regard as to how the image will be compressed afterwards, and most compression techniques ignore the fact that the data was originally captured with a colour filter array.

Advanced demosaicking algorithms put a lot of computational effort into reconstructing high frequency detail in the red and blue colour channels [5]-[15]. If the image is compressed afterwards, it will typically be converted to YCbCr 4:2:0 format. In this format, the chroma channels (Cb, Cr) are down-sampled by a factor of two in both the horizontal and vertical directions, resulting in a loss of the high frequency colour information. Therefore, it is wasteful to generate the high-frequency colour information in the demosaicking process.

The traditional approach to compressing image or video data captured with a CFA is to first perform demosaicking and then compress the resulting full-colour data. This approach is less than optimal, because demosaicking increases the size of the data without introducing new information. That is, the demosaicking process introduces redundancies into the data that the compression process must undo. Some work has been done on compressing raw CFA still image data, rather than first performing

demosaicking and then applying compression [16]-[21]. However, little work has been done on compressing video captured with a CFA.

In this thesis we develop schemes that jointly optimize the demosaicking and compression processes. Two approaches to doing this are considered:

1. Creating a demosaicking algorithm that directly produces an image in the format used for compression (YCbCr 4:2:0), thus reducing the complexity of the demosaicking process.

2. Compressing CFA video data prior to demosaicking rather than after, increasing compression efficiency by taking advantage of the smaller input data size.

The rest of this thesis is organized as follows. Chapter 2 provides background information on digital cameras, image and video compression, demosaicking methods, and existing techniques for CFA data compression. In Chapter 3, we present a new demosaicking algorithm that directly produces an image in YCbCr 4:2:0 format that can be immediately be compressed, without the needed for a separate colour space conversion step. Two methods for compressing CFA video prior to demosaicking are presented in Chapter 4. Finally, conclusions and directions for further research are provided in Chapter 5.

2 Background

In this chapter, we provide background information on digital camera design, the fundamentals of image and video compression, demosaicking in digital cameras, and the existing work on compression in single-sensor cameras. In Section 2.1 we provide basic information on the design of digital cameras, including the optical path, sensors and digital image processing steps. Section 2.2 provides an overview of the basics of image and video compression, including the YCbCr colour space, the discrete cosine transform, motion compensation, and the H.264 standard. In Section 2.3, an overview of demosaicking algorithms is provided, with emphasis placed on fast techniques that run directly on the digital camera itself. A detailed summary of existing research on direct compression of CFA images and video is provided in Section 2.4.

2.1 Digital Camera Design

2.1.1 Optical System and Sensors

A colour digital image can be represented using three colour samples. Often red, green and blue are used, which roughly correspond to the wavelengths that the three types of cones in the human eye are most sensitive to. By combining red, green and blue light in different ratios, almost any visible colour can be obtained.

To capture a digital image, the most straightforward design would be to use three separate sensors to capture red, green and blue light. The optical path for such a system is shown in Figure 2.1. A beam splitter would be used to project the light through three colour filters, and towards three sensors. This three sensor design is used in some high-

end cameras. However, since the sensor is one of the most expensive parts of a camera, typically accounting for 10-25% of the total cost [22], the three sensor design is not used in most consumer digital cameras.

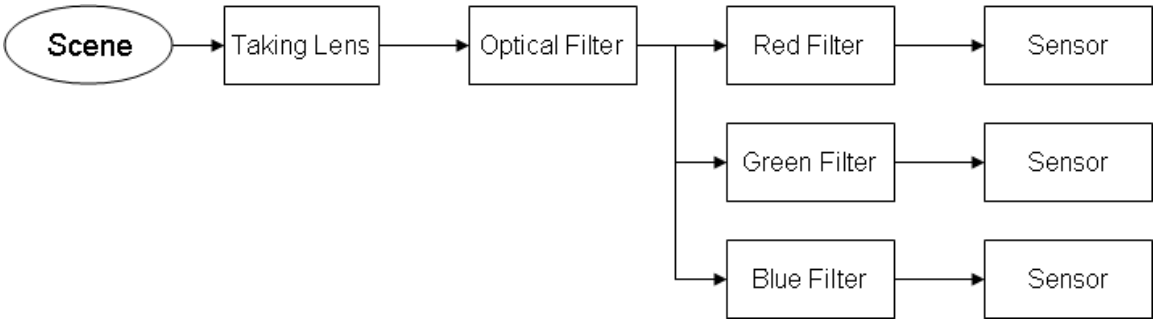


Figure 2.1: Typical optical path for a three sensor camera.

To reduce cost, most consumer digital cameras manufacturers use only a single light sensor. To capture colour information in such devices, a colour filter array (CFA) is placed before the sensor [23]. The optical path for such a camera is shown in Figure 2.2. The CFA only allows one colour of light to reach the sensor at each pixel location. Several CFA designs exist [24], the most popular being the Bayer pattern [25]. The Bayer pattern consists of a repeating 2x2 cell, each cell containing two green samples, one red sample and one blue sample, as shown in Figure 2.3. More green samples are captured than red or blue because the human visual system is more sensitive to the green portion of the light spectrum.

Figure 2.2: Typical optical path for a single sensor camera using a CFA.

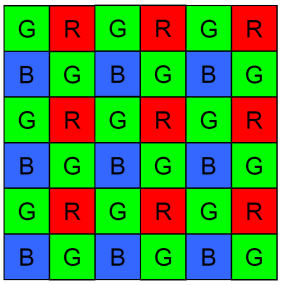


Figure 2.3: Bayer Pattern CFA

In the optical path for both the three sensor and the single sensor camera designs, optical filtering is done before the light passes through the colour filters. This is necessary to provide infra red (IR) rejection. Most colour dyes transmit light beyond 700 nm, which the human visual system does not perceive, but which the camera's light sensor may be sensitive to. This light must be filtered out to capture an image that corresponds to what a human observes. Typically, a hot mirror is used for IR rejection. A hot mirror transmits low wavelength light and reflects high wavelengths. An anti-aliasing filter may also be placed at the "optical filter" location in Figures 2.1 and 2.2. An anti-aliasing filter provides spatial blurring to filter out high frequencies that cannot be captured due to the spatial resolution of the sensor.

The sensor used in digital cameras is either a charge-coupled device (CCD) or a CMOS (Complementary Metal Oxide Semiconductor) device. CCD sensors were used in virtually all early cameras, but CMOS sensors are now becoming more common due to their lower cost, lower power consumption and their ability to be incorporated onto a single chip with other circuits. However, CCD sensors produce superior image quality and are used in high-end devices.

2.1.2 Digital Image Processing

The output from the image sensor goes through an analog to digital (A/D) converter to produce a digital image. After this, many digital image processing steps must be carried out in order to produce a final viewable image. A typical digital image processing pipeline for a single sensor camera is shown in Figure 2.4. It should be noted that this pipeline is an example only; different cameras perform different processing steps and the order of steps may be different in some cameras.

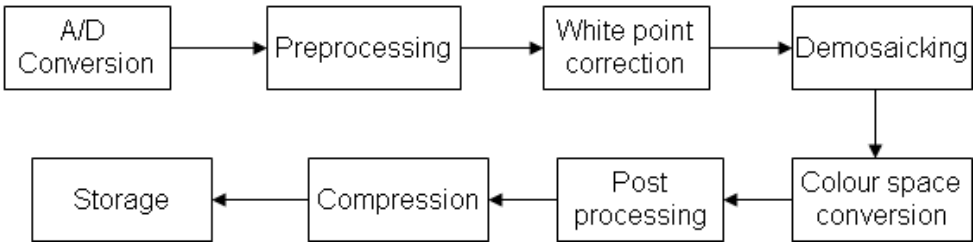


Figure 2.4: Example digital processing pipeline for a single sensor camera.

After the A/D converter has generated a digital image, some pre-processing may be applied. These steps vary significantly from camera to camera, but may include interpolating values at defective pixel locations, or converting the output from a non-linear sensor to a linear space.

The colour of light coming off an object is a function of both the incident light colour and the reflectance of the object. The human visual system corrects for the lighting conditions so that a white object is still perceived as being white regardless of the light source. To produce images that correspond to what a human perceives, a camera must also correct the colours in an image based on the lighting conditions. This processing is called white balancing or white-point correction. White balancing is done either by

having the user select from a number of pre-programmed settings, or using an auto white balance (AWB) algorithm [26].

When a CFA is used in a single sensor camera, the sensor captures a mosaic image, where there is either a red, green or blue sample at each pixel location. The two missing colours at each location much be estimated from the surrounding samples in a process known as demosaicking [5]. An overview of demosaicking algorithms is provided later in this chapter in Section 2.3.

The spectral characteristics of an image sensor are not likely to be matched to the spectral emission characteristics of the display device used. That is, if we took the RGB values recorded by the sensor in response to a scene and directly displayed those RGB values on a monitor, the colours produced by the monitor would not match the colours in the image scene. Thus, it is necessary to convert the RGB values produced by a sensor into a standard colour space, most often the sRGB (standard red, green, blue) space [27]. The sRGB space defines a mapping between a set of sRGB values and the CIE (Commission Internationale de l'Eclairage) chromaticity coordinates of the light produced by a display device. In digital cameras, the RGB values produced by the image processing chain discussed so far are typically converted into CIE XYZ (chromaticity) values through a linear transformation (that has been designed based on the sensors characteristics) [28]. Then a non-linear transformation is applied to convert from XYZ values to sRGB space [27].

Some cameras apply post-processing to the image before it is compressed. This may include steps such as sharpening and artifact removal to improve the subjective image quality.

The final step of the digital image processing chain is compression, where redundancies are removed from the image representation so it can be coded using fewer bits. The JPEG image compression standard [1] is by far the most commonly used image compression scheme in digital cameras. Section 2.2 of this thesis provides more details on image and video compression.

Some cameras provide a setting for the user to bypass all of the digital image processing steps above, and just store the raw data obtained by the sensor (perhaps with some minimal compression). This allows the digital processing to be performed later on a computer, with guidance from the user. By performing the digital processing on a computer, more advanced algorithms for white-balance, demosaicking, and post processing can be used, since a typical computer has much more computational power than a digital camera. This mode of operation is used by professional users, but rarely by typical consumers.

2.2 Image and Video Compression

In the following sections the basics of image and video compression are covered. Section 2.2.1 discusses the YCbCr colour space which is used in most still image and video compression standards. Still image compression techniques are covered in Section 2.2.2 and video compression is discussed in Section 2.2.3. Finally, a brief overview of H.264/AVC, the latest major video coding standard, is provided in Section 2.2.4.

2.2.1 YCbCr Colour Space

A digital image can be represented using three colour samples per pixel. Typically red (R), green (G) and blue (B) are used when capturing images with a digital camera. However, storing images in RGB space is inefficient, as there is a large amount of correlation between the channels. Instead, when images or video are to be compressed, they are usually converted into YCbCr colour space. In YCbCr space, an image is represented by one luma (Y) and two chroma (Cb, Cr) components. The luma channel contains "brightness" information; it is essentially a greyscale version of the image. The chroma values are colour offsets, which show how much a pixel deviates from greyscale in the blue (Cb) and red (Cr) directions.

The equations used for converting from RGB to YCbCr used in the JPEG JFIF format [29] are:

$$\begin{aligned} Y &= 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B \\ Cb &= -0.1687 \cdot R - 0.3313 \cdot G + 0.5 \cdot B \\ Cr &= 0.5 \cdot R - 0.4187 \cdot G - 0.0813 \cdot B \end{aligned} \qquad (2.1)$$

The reverse equations for converting from YCbCr to RGB are:

$$\begin{aligned} R &= Y + 1.402 \cdot Cr \\ G &= Y - 0.34414 \cdot Cb - 0.71414 \cdot Cr \\ B &= Y + 1.772 \cdot Cb \end{aligned} \qquad (2.2)$$

For most natural images, the RGB to YCbCr conversion strongly de-correlates the colour channels, so the Y, Cb, and Cr components can be coded independently without loss of efficiency. In YCbCr space, the energy of an image tends to be concentrated in

the Y channel. This leaves the Cb and Cr channels with less information, so they can be represented with fewer bits.

Another advantage of the YCbCr space comes from the properties of the human visual system. The human eye is more sensitive to brightness information than colour information. Consequently, the chroma signals can be down-sampled relative to the luma without significant loss of perceived quality. In fact, chroma down-sampling is almost always done when compressing image or video data.

Applying equation (2.1) at every pixel in an image results in an YCbCr 4:4:4 picture. In YCbCr 4:4:4 format, the chroma is sampled at the same rate as luma (Figure 2.5a). This format is rarely used, except in professional applications. In YCbCr 4:2:2 format, the chroma signals are down-sampled by a factor of two relative to the luma in the horizontal direction (Figure 2.5b). Some higher end digital video formats use YCbCr 4:2:2 sampling. However, the most common colour format used in compressed images and video is YCbCr 4:2:0 (Figure 2.5c). In YCbCr 4:2:0 format, the chroma signals are down-sampled by a factor of two in both the horizontal and vertical directions. It should be noted that there are different chroma positions used in YCbCr 4:2:0 format. Sometimes the chorma samples are considered to be half-way between the luma samples vertically, or in the center of a group of four luma samples. In the downsampling process the chroma channels should be low-pass filtered to limit aliasing.

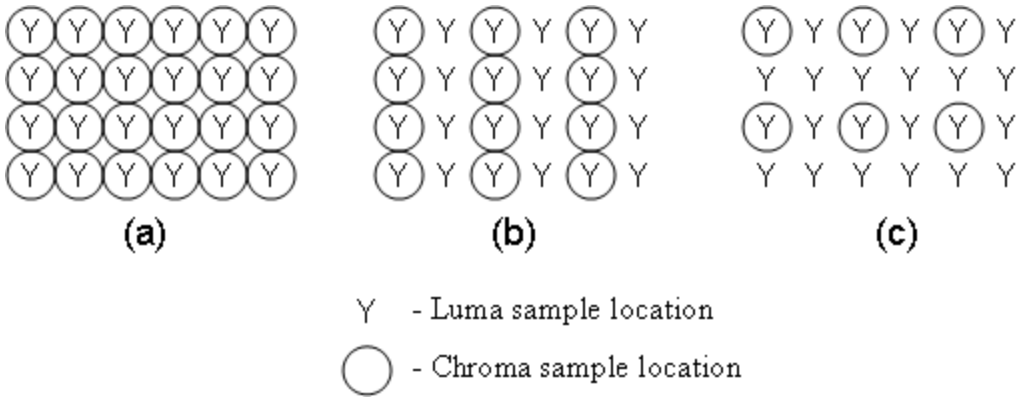


Figure 2.5: Illustration of YCbCr sampling formats. (a) 4:4:4 (b) 4:2:2 (c) 4:2:0.

There are minor variations on Equations 2.1 and 2.2 used for converting to YCbCr format. These typically involve scaling of the luma and chroma vales so the final samples will fall within a lower range. In digital images and video, the term YCbCr is typically usually used interchangeably with YUV to refer to the same colour space.

2.2.2 Image Compression

Still image compression involves removing spatial redundancies within an image. This is usually done with a transform, such as the discrete cosine transform (used in JPEG [1]) or the wavelet transform (used in JPEG 2000 [30]). The idea of transform coding is to apply a mathematical transformation to a group of pixels that will concentrate the energy of the signal into a smaller number of coefficients. After applying a transform, many coefficients are typically near zero, and hence can be discarded with minimal impact on the output image when the inverse transform is taken.

By far, the most popular image compression method used in digital cameras is baseline JPEG. JPEG is based on the 2D discrete cosine transform (DCT). The image

is divided into 8x8 blocks of pixels, and the DCT of each block is taken. The DCT coefficients are calculated with the following formula:

$$D(p,q)=\frac{1}{4}c(p)c(q)\sum_{x=1}^{7}\sum_{y=1}^{7}f(x,y)\cos\left(\frac{(2x+1)p\pi}{16}\right)\cos\left(\frac{(2y+1)q\pi}{16}\right)$$
$$c(x)=\begin{cases}1/\sqrt{2} & x=0\\ 1 & 1<x<8\end{cases} \qquad 0<p,q<7 \tag{2.3}$$

The DCT coefficients, *D(p,q)*, show how much energy the image block has at different spatial frequencies. Higher values of *p* and *q* correspond to higher frequencies in the *x* and *y* directions. The coefficients are quantized based on how the human visual system perceives different frequencies, with the higher frequency coefficients being more coarsely quantized. The quantized coefficients are scanned in zigzag order, in direction of increasing frequency. Entropy coding is applied to *(run, level)* pairs, where *level* is the value of a quantized coefficient, and *run* is the number of zero values preceding the coefficient in the zigzag scan.

2.2.3 Video Compression

Due to the large amount raw data needed to represent video, efficient data compression is critical in systems involving digital video. With a given amount of data storage capacity more efficient video compression schemes allow either the video quality to be increased or the play time to be extended. Efficient video compression is essential in applications such as personal video players (e.g., DVD), streaming video over the internet, digital camcorders, video conferencing, broadcasting, etc.

A digital video is a time sequence of 2D frames. Video compression attempts to remove redundancies both within a single frame and between frames.

Some frames in a video are coded independently of other frames using intra coding methods (I frames). These frames are compressed with still image techniques, typically similar to those used in JPEG.

The key technique that distinguishes video compression from image compression is motion compensation (MC). When MC is used, some frames are coded by predicting the frame from previously coded frames and storing the difference between the actual frame and the prediction. Such frames are called P frames. After forming a prediction for a block of pixels, the residual prediction error is calculated, which is the difference between the actual block and the prediction. Then a transform (e.g., DCT) is usually applied to the residual. The transform coefficients are then quantized, scanned and entropy coded.

MC involves dividing a frame into blocks and estimating and coding a displacement vector for each block. The displacement vector tells the decoder which block in a previously coded reference frame to use as a prediction. Using the displacement vector the decoder can form the same prediction for the block, and by adding the residual to the prediction the original block can be restored.

Figure 2.6 shows a simplified block diagram of a motion compensated, transform based hybrid video encoder. This basic structure is used in most major video coding standards including MPEG-1, MPEG-2, H.261, H.263 and H.264/AVC.

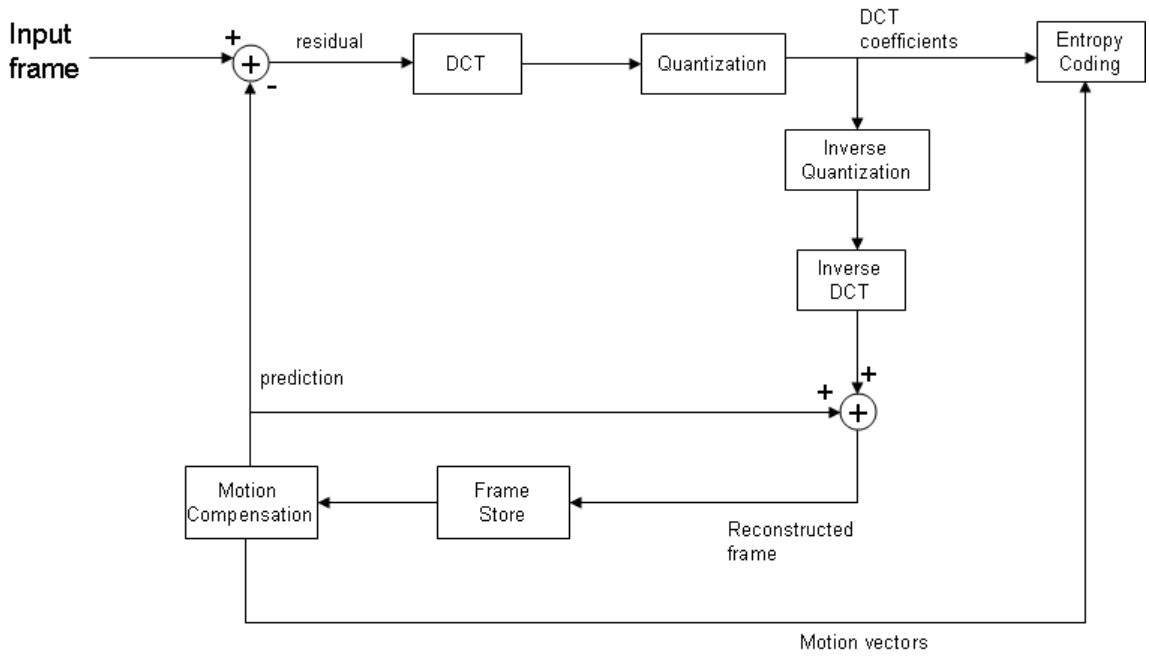


Figure 2.6: Block diagram of a typical motion compensated hybrid DCT based video encoder

2.2.4 H.264 Video Compression

H.264/AVC is the latest major video coding standard, and it is being used in applications such as next generation video players (Blu-ray, HD DVD). A number of new coding features were introduced in H.264, the most important of which are summarized below.

The transform used in H.264 is different from previous standards. Instead of using a DCT calculated with floating point arithmetic, as in JPEG and MPEG-2, H.264 uses an integer transform that approximates the DCT. The integer transform matrix only contains the values +1, -1, +2, and -2, so the transform can be calculated using only addition/subtraction and bit shift operations. In baseline H.264, the transform is calculated on blocks of 4x4 pixels, which is smaller than the 8x8 blocks used in most previous standards. The smaller block size helps reduce ringing artifacts.

When coding I frames, instead of taking the transform of blocks of pixels directly, each block is predicted in the spatial domain from previously coded blocks (usually blocks above or to the left of the one being coded). A number of different intra prediction modes are supported, for example each pixel in the block can be predicted from the pixel outside the block directly above it, to the left, or at a diagonal direction. A DC prediction mode is also supported, where the entire block is predicted based of the average value of the surrounding previously coded pixels. After the prediction has been made, the integer transform is applied to the prediction residual.

Motion Compensation (MC) is much more flexible in H.264 than in previous standards (e.g., MPEG-2). In H.264, variable block size motion compensation is supported. Blocks of size 16x16 pixels down to 4x4 pixels can be used. In addition, H.264 allows the encoder to select from multiple previously coded frames to find the one that will provide the best prediction for the frame being coded. Finally, motion vectors in H.264 can be defined in units of one quarter pixel. This significantly increases the accuracy of MC, resulting in higher compression efficiency. In order to support fractional pixel motion vectors, the reference frames must be interpolated. This is done with a 6 tap FIR (finite impulse response) filter to generate samples at half pixel positions, followed by bilinear interpolation to generate samples at quarter pixel positions.

2.3 Demosaicking Algorithms

Most consumer digital cameras capture colour information with a single light sensor and a colour filter array (CFA). The CFA only allows one light colour (red, green or

blue) to reach the sensor at each pixel location. The Bayer Pattern [25] is the most commonly used CFA design.

Demosaicking is the process of estimating the two missing colour samples at each pixel location. The simplest demosaicking methods interpolate each colour channel separately. One such technique is bilinear interpolation, where the average of the surrounding samples is used. When bilinear interpolation is used, each missing green sample is calculated as the average of the four surrounding green samples, and each missing red or blue sample is taken as the average of the two nearest neighbours or four nearest neighbours, depending on the position. Other standard interpolation methods, such as cubic spline interpolation, can be used to slightly improve the performance when processing each colour channel separately.

The problem with methods that interpolate the colour channels independently is that they usually fail at sharp edges in images, resulting in objectionable colour artifacts. Figure 2.7 illustrates this; it shows a full colour image and the result of using bilinear interpolation to reconstruct the image after it has been sub-sampled with the Bayer pattern. Note how the bilinear image has false colours in regions of high frequency image content and also appears significantly blurred.

Figure 2.7: Original RGB and the result of applying bilinear interpolation to the image when it is sampled with the Bayer pattern

To overcome the problems caused by simple methods that interpolate the colours channels separately, many adaptive demosaicking algorithms have been developed which exploit the correlation between the colour channels.

One class of adaptive demosaicking algorithms is edge-directed interpolation [6]-[8]. These algorithms attempt to preserve edges by calculating gradients from the CFA data and interpolating along the direction (typically either horizontal or vertical) that has the lower gradient.

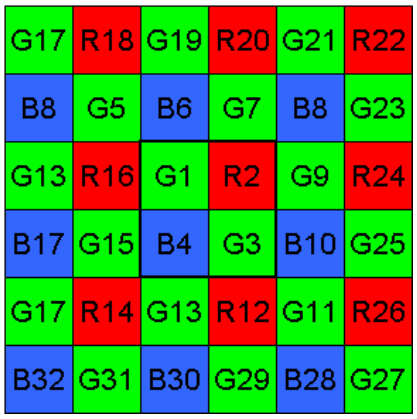


Figure 2.8: Location numbering of Bayer Pattern

In [6], Hamilton and Adams propose a method where Laplacian second order correction terms are used to enhance the estimates for missing pixels. Referring to Figure 2.8, consider the task of estimating the green sample at the location of sample R2 (we denote the missing green sample G2). In the method in [6] (which we will refer to as 'Laplacian demosaicking') gradients are calculated as:

$$\begin{aligned} DH &= |R16 + R24 - 2 \cdot R2| + |G1 - G9| \\ DV &= |R20 + R12 - 2 \cdot R2| + |G7 - G3| \end{aligned} \qquad (2.4)$$

Bilinear interpolation is carried out in the direction with the lower gradient, or both directions if the gradients are equal. The Laplacian of the red or blue channel is calculated in the same direction and added to the bilinear estimate. For example, G2 is calculated as follows:

$$
\begin{aligned}
&\text{if } (DH < DV) \\
&\quad G2 = \frac{G1 + G9}{2} + \frac{2 \cdot R2 - R16 - R24}{4} \\
&\text{else if } (DV < DH) \\
&\quad G2 = \frac{G7 + G3}{2} + \frac{2 \cdot R2 - R20 - R12}{4} \\
&\text{else} \\
&\quad G2 = \frac{G1 + G7 + G9 + G3}{4} + \frac{4 \cdot R2 - R16 - R20 - R24 - R12}{8}
\end{aligned}
\tag{2.5}
$$

In equation (2.5), the left term of every sum is the result of applying bilinear interpolation to the green channel in the lower gradient direction (horizontal, vertical, or both). The right hand term of each sum is the Laplacian of the red channel. The Laplacian is a high-pass filter. The RGB colour planes are typically very highly correlated, especially in high-frequency content, so adding the Laplacian of the red channel to the green channel enhances the bilinear estimate. The red and blue planes are filled in a similar manner (calculating gradients and using directional interpolation with Laplacian terms).

In [9], Laplacian correction terms are also used, but to reduce the number of computations required, the interpolation of the green channel always uses samples in both directions (the last case in equation (2.5)). This eliminates the need for calculating the gradients and performing comparisons, resulting in a very computationally simple method. After the green channel has been filled, the red and blue channels are estimated using bilinear interpolation on the difference between the red/blue channel and green channel. The signals R-G, and B-G are generally much smoother (less high frequency content) than the red and blue channels themselves and are thus more suitable for conventional linear interpolation.

Many demosaicking methods have been developed which are far more advanced than the methods described so far. Wavelet decomposition is used in [10], where the high-frequency sub-band of the green channel is used to iteratively update the high-frequency content in the red and blue channels. Techniques have also been proposed using neural networks [11], markov random fields [12] and a soft decision process for estimating edge directions [13]. These methods can achieve very good image quality but have high computational complexity. Thus, instead of performing demosaicking directly on the camera when the image is taken, the raw data would be stored on the camera and then transferred to a computer where demosaicking would take place. In this thesis, we focus on methods that run directly on the digital camera, so these advanced algorithms are not discussed in detail.

When video is captured with a CFA device, demosaicking is usually done on each frame independently. However, recent work has explored using motion estimation to improve performance when demosaicking is performed on a video [14],[15]. Through motion estimation, information from other frames can be used to improve the estimate for the missing pixels in each frame, at the expense of greatly increased complexity.

All the demosaicking algorithms described up to this point produce RGB output images. If the image is to be compressed afterwards (it almost always will be), it will be converted into YCbCr 4:2:0 format, which is more suitable for compression. Instead of performing demosaicking in the RGB space and then converting to YCbCr 4:2:0 space afterwards, the demosaicking algorithm can be designed to directly produce YCbCr 4:2:0 output.

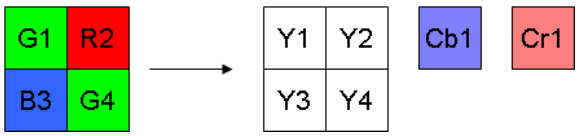


Figure 2.9: Conversion from Bayer cell to YCbCr 4:2:0

There is one previously published demosaicking algorithm that produces an YCbCr 4:2:0 output image [31]. Their algorithm, entitled "YUV through green interpolation with median filtering post-processing," (YUVGM), starts by creating a full green channel, using the method for interpolating the green channel in [6]. For each 2x2 cell in the Bayer pattern, four luma (Y) samples must be generated, together with one Cb and Cr sample (Figure 2.9). Let Y(R,G,B), Cb(R,G,B) and Cr(R,G,B) denote functions for the equations in (2.1) for converting from RGB space to YCbCr space. Then, the output samples are calculated by the following equations:

$$
\begin{aligned}
&\mathrm{Y1} = Y(\mathrm{R2}, \mathrm{G1}, \mathrm{B3}) \\
&\mathrm{Y2} = Y(\mathrm{R2}, \mathrm{G2}, \mathrm{B3}) \\
&\mathrm{Y3} = Y(\mathrm{R2}, \mathrm{G3}, \mathrm{B3}) \\
&\mathrm{Y4} = Y(\mathrm{R2}, \mathrm{G4}, \mathrm{B3}) \qquad (2.6) \\
&\mathrm{G}_{\mathrm{avg}} = (\mathrm{G1} + \mathrm{G2} + \mathrm{G3} + \mathrm{G4}) / 4 \\
&\mathrm{Cb1} = Cb(\mathrm{R2}, \mathrm{G}_{\mathrm{avg}}, \mathrm{B3}) \\
&\mathrm{Cr1} = Cr(\mathrm{R2}, \mathrm{G}_{\mathrm{avg}}, \mathrm{B3})
\end{aligned}
$$

After the above equations have been evaluated for every Bayer cell, median filtering is performed on the Cb and Cr channels to remove colour artefacts. Note that the YUVGM method is using zero-order hold interpolation on the red and blue channels, which produces severe false colours around edges. The median filtering post processing is an ad-hoc and computationally expensive method of reducing false colours.

2.4 Previous Work on CFA Image and Video Compression

The conventional approach used for compressing images and video generated with single-sensor cameras is to first perform demosaicking and then compress the resulting full-colour data with standard methods (Figure 2.10a). This approach is sub-optimal because demosaicking expands the size of data to be compressed by a factor of three and introduces further redundancy into the data that will be removed by the compression stage. Instead compression can be carried out on the CFA data, with demosaicking being performed after decompression (Figure 2.10b) [16]-[19].

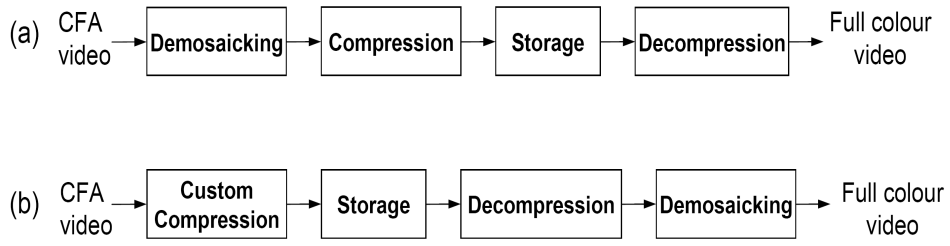


Figure 2.10: Flow diagram of CFA compression schemes: (a) Traditional demosaick-first approach. (b) Compression of CFA data prior to demosaicking.

There have been a few papers published on lossy compression of CFA image data prior to demosaicking. The goal of these is to provide better image quality at a given bit rate, allowing a camera to either store higher quality images or store more images with the same quality level.

In [16], Lee and Ortega propose a method starting with a modified YCbCr colour space conversion. The conversion is performed on each group of four Bayer pattern samples, and creates two Y samples and one Cb and Cr sample (Figure 2.11). The equations for the modified conversion are:

$$
\begin{aligned}
Yu &= 0.299 \cdot R + 0.587 \cdot Gu + 0.114 \cdot B \\
Yl &= 0.299 \cdot R + 0.587 \cdot Gl + 0.114 \cdot B \\
Cb &= -0.1687 \cdot R - 0.16565 \cdot Gu - 0.16565 \cdot Gl + 0.5 \cdot B \\
Cr &= 0.5 \cdot R - 0.20935 \cdot Gu - 0.20935 \cdot Gl - 0.0813 \cdot B
\end{aligned}
\qquad (2.7)
$$

When calculating each luma value with (2.7), the corresponding green sample is used together with the red and blue samples. For calculating the chroma values, the average of the two green samples is used along with the red and blue. After the conversion, the chroma samples are arranged into rectangular arrays and then compressed with standard JPEG. The Y samples, which are arranged in a quincunx lattice, are rotated 45 degrees to form a compact rhombus shape, as shown in Figure 2.12. Then the Y samples are compressed with a modified JPEG algorithm, where padding is done at the edges of the Y rhombus to create square 8x8 blocks that can be compressed with the JPEG DCT method.

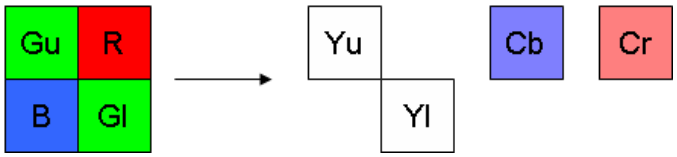


Figure 2.11: Modified YCbCr conversion performed on the Bayer unit cell in [17].

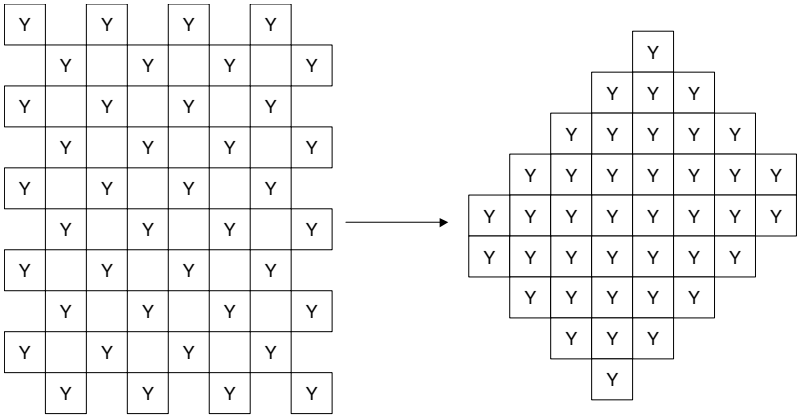


Figure 2.12: 45 degree rotation of luma samples used in [16].

Two CFA image compression methods are proposed by Koh et. al. in [17], which are called‘structure conversion’ and ‘structure separation’. Both methods start with the modified YCbCr conversion proposed in [16], followed by compressing the chroma channels with JPEG. The methods differ in how they process the luma samples. In the structure conversion method, the two luma samples generated from every Bayer cell are merged into a single column, as shown in Figure 2.13b. This creates a single luma array that has half the size of the CFA data. The process of merging the two samples distorts the image content somewhat. The structure separation method involves separating the even column and odd column luma samples into two arrays, each having size one quarter that of the CFA data (Figure 2.13c). In both methods, after rectangular arrays of luma samples have been formed they are compressed with standard JPEG.

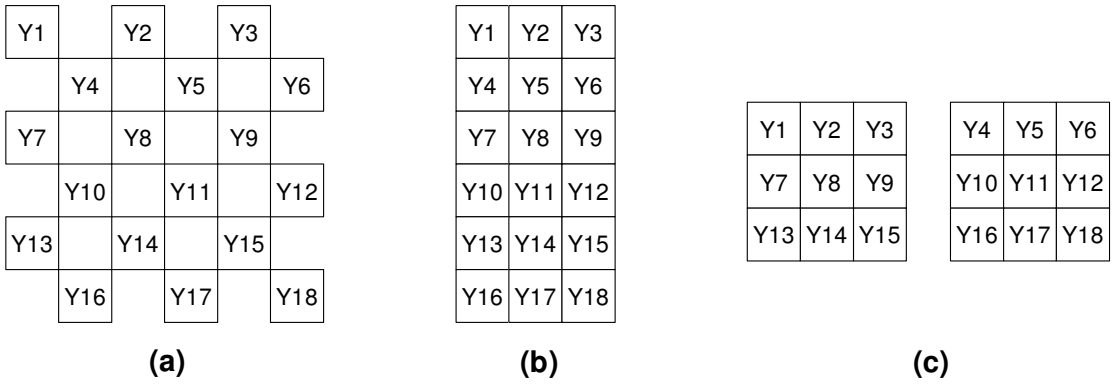


Figure 2.13: Methods for forming rectangular arrays of luma data in [17]. (a) Original luma arrangement (b) Structure conversion (c) Structure separation

Other techniques for compressing CFA images include sub-band decomposition [20] and vector quantization of groups of pixels [21]. However, these methods have worse compression efficiency than the JPEG based methods in [16] and [17]. A comparative analysis of the two workflows (compression-first vs. conventional demosaick-first) is presented in [18].

The only previously published work on lossy compression of CFA videos is presented in [19] by Gastaldi et. al. Their method is based on the structure conversion method in [17]. In [19], the structure conversion process is done in the RGB domain (without first applying an YCbCr colour space conversion), as shown in Figure 2.14. The arrays of green, red and blue are compressed with a custom MPEG-2 like coding scheme. A custom coding scheme was developed because no major video coding standard supports input data where one channel (green) has twice the vertical size and the same horizontal size as the other channels (red and blue). A the motion vectors from the green channel are used on the red and blue channels, with appropriate scaling and downsampling. An IBBPBBPBBPBB group of pictures structure is used, with JPEG compression used to compress the I-frames and residual for P and B frames.

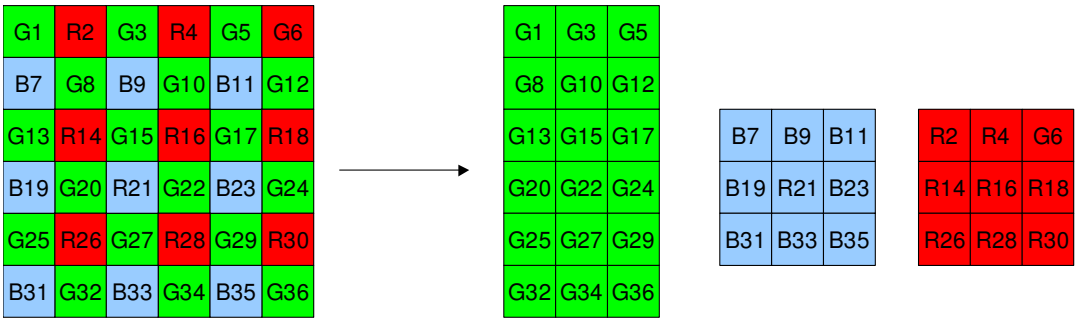


Figure 2.14: RGB based structure conversion method used in [21].

3 Demosaicking Directly to YCbCr 4:2:0

3.1 Introduction

Single sensor cameras capture colour information using a colour filter array (CFA). This results in a mosaic image being captured, where at each pixel location either a red, green or blue sample is captured. The two missing colours at each location are interpolated from the surrounding samples in a process known as demosaicking. As discussed in our overview of demosaicking methods in Section 2.3, virtually all demosaicking algorithms produce an RGB output image. If the image is to be compressed afterwards, it will typically be converted to YCbCr 4:2:0 format.

The green channel is the dominant component in determining luma, and the red and blue samples contribute most to the Cr and Cb channels, respectively. Advanced demosaicking methods put a lot of computational effort into reconstructing fine detail in the red and blue colour planes. This is a difficult task because red and blue are more sparsely sampled in the Bayer pattern. It is also somewhat unnecessary as most of the high frequency detail in those channels is lost in the conversion from RGB to YCbCr 4:2:0 format.

In this chapter, we present a demosaicking method that directly produces an YCbCr 4:2:0 output image. This reduces computational complexity by avoiding the need for performing demosaicking in the RGB domain and then converting to YCbCr 4:2:0 format afterwards.

The rest of this chapter is organized as follows. Our demosaicking method is described in Section 3.2. Simulation results comparing our proposed method against other fast demosaicking algorithms are presented in Section 3.3. A complexity analysis of our method is given in Section 3.4, where the complexity of our proposed method is compared against a very fast RGB based method. Finally, conclusions are given in Section 3.5.

3.2 Proposed Demosaicking Method

The Bayer pattern consists of cells of size 2x2 pixels, each cell containing two green samples, one red sample and one blue sample. To produce YCbCr 4:2:0 output, four luma samples and one Cb and Cr samples must be generated for each cell. Figure 3.1 shows a 2x2 cell (locations 1-4) and the surrounding Bayer pattern samples that will be used to calculate the luma and chroma samples in the cell. In the following section we describe how our algorithm generates chroma samples at location 1 (denoted as Cb1, Cr1), and luma samples at locations 1-4 (denoted Y1, Y2, Y3, Y4). The steps described are repeated on every 2x2 cell to generate the entire YCbCr 4:2:0 image. The location numbering given in Figure 3.1 will be used throughout the rest of the chapter.

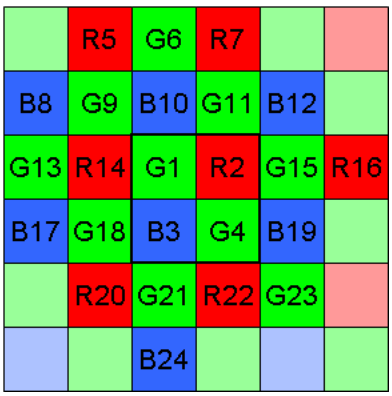


Figure 3.1: Neighbourhood of pixels used for generating the luma and chroma samples in the cell containing positions 1-4.

Our demosaicking algorithm consists of four steps:

1. Generating a full green channel

2. Generating low-pass filtered, down-sampled red-green and blue-green colour difference values (R-G, B-G)

3. Calculating low-pass filtered, down-sampled chroma channels

4. Filling the luma channel.

Each step is discussed in turn in the following sections.

3.2.1 Generating a Full Green Channel

Since a full luma channel is needed, and green is the dominant component in determining luma, our method begins by filling the green channel. A complete green channel allows us to estimate a high-quality luma channel.

Many existing demosaicking methods start by generating a complete green channel. We base our method for calculating the missing green samples on the method by Hamilton and Adams [6], which is a popular low-complexity demosaicking algorithm.

The idea behind the method for filling the green channel is to calculate horizontal and vertical gradients at the current location, and interpolate along the direction that contains the lower gradient. This results in interpolation being performed along edges rather than across edges. If the gradients are similar in magnitude, interpolation is done using samples in both directions. There are two cases that need to be considered when calculating the missing green samples; generating a green at a red location (R2) or blue location (B3).

At red location (R2), the gradients are calculated as:

$$\begin{aligned} DH &= |R14 + R16 - 2 \cdot R2| + |G1 - G15| \\ DV &= |R7 + R22 - 2 \cdot R2| + |G11 - G4| \end{aligned} \quad (3.1)$$

The missing green sample is calculated as follows:

$$\begin{aligned} &\text{if } (DH + T < DV) \\ &\quad G2 = \frac{G1 + G15}{2} + \frac{2 \cdot R2 - R14 - R16}{4} \\ &\text{else if } (DV + T < DH) \\ &\quad G2 = \frac{G11 + G4}{2} + \frac{2 \cdot R2 - R7 - R22}{4} \\ &\text{else} \\ &\quad G2 = \frac{G1 + G11 + G15 + G4}{4} + \frac{4 \cdot R2 - R7 - R14 - R16 - R22}{8} \end{aligned} \quad (3.2)$$

The second term in each sum in (3.2) is the second order gradient (Laplacian) of the red channel. The red, green and blue channels are typically very highly correlated,

especially in high frequency content [10], so adding the Laplacian term improves the bilinear interpolation [32].

A threshold '*T*' is used to ensure that the gradients are sufficiently different for interpolation to happen in only one direction. If the difference between the gradients is less then *T* (the last case in equation (3.2)), the interpolation uses samples in both directions. A threshold is not used in the method by Hamilton and Adams [6]. If the threshold is set to zero, the method for filling the green channel would be identical to their method. Experimentally, we found a threshold of 35 to provide good results for a wide range of images.

At the blue location (B3), the gradients and missing green sample are calculated analogously to the red location using the following equations:

$$\begin{aligned} DH &= |B17 + B19 - 2\cdot B3| + |G18 - G4| \\ DV &= |B10 + B24 - 2\cdot B3| + |G1 - G21| \end{aligned} \tag{3.3}$$

$$\begin{aligned} &\text{if } (DH + T < DV) \\ &\quad G3 = \frac{G18 + G4}{2} + \frac{2\cdot B3 - B17 - B19}{4} \\ &\text{else if } (DV + T < DH) \\ &\quad G3 = \frac{G1 + G21}{2} + \frac{2\cdot B3 - B10 - B24}{4} \\ &\text{else} \\ &\quad G3 = \frac{G1 + G4 + G18 + G21}{4} + \frac{4\cdot B3 - B17 - B19 - B10 - B24}{8} \end{aligned} \tag{3.4}$$

3.2.2 Calculating Low-pass R-G and B-G Samples

Since the conversion from RGB to YCbCr is a linear process, we can equivalently perform loss-pass filtering in the RGB domain rather than the YCbCr domain. We apply

low-pass filtering to generate R-G and B-G values located at the location of G1 in Figure 3.1. Instead of performing interpolation on the red and blue channels themselves, we perform interpolation on the difference between green and red or blue. The R-G and B-G images are generally much smoother than the R and B channels, so they are more suitable for interpolation [9]. From the low-pass R-G and B-G values, we can generate low-pass chroma samples, as will be explained in the next section.

The following 2D filter is used on the R-G channel:

$$h_{KR} = \begin{bmatrix} 1/8 & 0 & 1/8 \\ 0 & 0 & 0 \\ 1/4 & 0 & 1/4 \\ 0 & 0 & 0 \\ 1/8 & 0 & 1/8 \end{bmatrix} \qquad (3.5)$$

This filter provides low-pass filtering in both the horizontal and vertical directions, while only using positions that have red samples available. The filter in equation (3.5) is separable, and equivalent to using the following two filters in the horizontal and vertical directions:

$$\begin{aligned} h_{hor} &= \begin{bmatrix} 1/2 & 0 & 1/2 \end{bmatrix} \\ h_{vert} &= \begin{bmatrix} 1/4 & 0 & 1/2 & 0 & 1/4 \end{bmatrix} \end{aligned} \qquad (3.6)$$

The vertical filter in (3.6) provides stronger low-pass filtering than the horizontal filter, as it has two zeros at π radians/sample rather than just one. However, due to the required positioning of the output, a filter with an even number of taps is needed in the

horizontal direction. Using a four tap horizontal filter would require significantly more operations to implement.

The resulting equation for calculating the low-pass R-G value, denoted KR is:

$$\begin{aligned} \mathrm{KR1}_{lp} \equiv (R-G) * h_{KR} = & \frac{\mathrm{R14}-\mathrm{G14}+\mathrm{R2}-\mathrm{G2}}{4} \\ & + \frac{\mathrm{R5}-\mathrm{G5}+\mathrm{R7}-\mathrm{G7}+\mathrm{R20}-\mathrm{G20}+\mathrm{R22}-\mathrm{G22}}{8} \end{aligned} \tag{3.7}$$

For calculating a low-pass B-G sample at location 1, an equivalent filter to that of equation (3.5) is used, only this time it is rotated 90 degrees due to the different positioning of the blue samples in the Bayer pattern:

$$h_{KB} = \begin{bmatrix} 1/8 & 0 & 1/4 & 0 & 1/8 \\ 0 & 0 & 0 & 0 & 0 \\ 1/8 & 0 & 1/4 & 0 & 1/8 \end{bmatrix} \tag{3.8}$$

The low-pass KB sample at location 1 is calculated equivalently to the low-pass KR sample using:

$$\begin{aligned} \mathrm{KB1}_{lp} \equiv (B-G) * h_{KB} = & \frac{\mathrm{B10}-\mathrm{G10}+\mathrm{B3}-\mathrm{G3}}{4} \\ & + \frac{\mathrm{B8}-\mathrm{G8}+\mathrm{B12}-\mathrm{G12}+\mathrm{B17}-\mathrm{G17}+\mathrm{B19}-\mathrm{G19}}{8} \end{aligned} \tag{3.9}$$

3.2.3 Calculating Down-sampled Chroma Channels

The conversion from RGB to YCbCr space is linear, so instead of performing linear low-pass filtering on the Cb and Cr channels, the filtering can be equivalently performed on the RGB samples:

$$\begin{aligned} \mathrm{Cb}_{lp} &= -0.1687 \cdot \mathrm{R} * h - 0.3313 \cdot \mathrm{G} * h + 0.5 \cdot \mathrm{B} * h \\ \mathrm{Cr}_{lp} &= 0.5 \cdot \mathrm{R} * h - 0.4187 \cdot \mathrm{G} * h - 0.0813 \cdot \mathrm{B} * h \end{aligned} \qquad (3.10)$$

where h is the low-pass filter used to limit aliasing after downsampling, Cb_{lp} and Cr_{lp} are low-pass chroma channels, and $*$ denotes 2D convolution.

The equations in (3.10) can easily be rewritten in terms of the colour differences R-G and B-G using the linear property of convolution:

$$\begin{aligned} \mathrm{Cb}_{lp} &= -0.1687 \cdot (\mathrm{R} - \mathrm{G}) * h + 0.5 \cdot (\mathrm{B} - \mathrm{G}) * h \\ \mathrm{Cr}_{lp} &= 0.5 \cdot (\mathrm{R} - \mathrm{G}) * h - 0.0813 \cdot (\mathrm{B} - \mathrm{G}) * h \end{aligned} \qquad (3.11)$$

By allowing different low-pass filters to be applied to the R-G and B-G signals in (3.11), the chroma values at location 1 can be calculated using the KR and KB samples generated with equations (3.7) and (3.9):

$$\begin{aligned} \mathrm{Cb1} &= -0.1687 \cdot \mathrm{KR1}_{lp} + 0.5 \cdot \mathrm{KB1}_{lp} \\ \mathrm{Cr1} &= 0.5 \cdot \mathrm{KR1}_{lp} - 0.0813 \cdot \mathrm{KB1}_{lp} \end{aligned} \qquad (3.12)$$

In our method, equation (3.12) is used to calculate the final chroma samples using the low pass filtered, down-sampled, KR and KB values.

3.2.4 Calculating the Full Luma Channel

Once the down-sampled chroma values have been calculated, the only task left is to generate the full luma channel. Since different samples are available at each location, we considered the task of generating luma samples at locations 1 through 4 separately.

At the location of G1, we already have G, KR and KB samples available. By rearranging the equation for luma in (2.1) in terms of R-G and B-G, the Y1 sample can be calculated with:

$$\mathrm{Y1} = 0.299 \cdot \mathrm{KR1}_{lp} + \mathrm{G1} + 0.114 \cdot \mathrm{KB1}_{lp} \quad (3.13)$$

Note that we are using low-pass KR and KB values, when ideally unfiltered values should be used. However, since the green sample has not been filtered and green is the dominant component in calculating luma, the value calculated with equation (3.13) is still a good estimate.

At the location of R2, we have red and green samples available. An assumption often made in demosaicking methods is that chroma varies smoothly in natural images, so bilinear interpolation provides a good estimate for missing chroma samples. Using this assumption, an estimate for the blue chroma at R2 is found as:

$$\mathrm{Cb2} = \frac{\mathrm{Cb1} + \mathrm{Cb15}}{2} \quad (3.14)$$

The Cb2 sample in (3.14) does not need to be calculated and stored, but the equation will be used for deriving an expression for Y2. We would like to calculate the luma value at location 2 using R2, G2 and Cb2. This can be done by substituting the equation for B in equation (2.2) into the Y definition in equation (2.1).

$$\mathrm{Y2} = 0.299 \cdot \mathrm{R2} + 0.587 \cdot \mathrm{G2} + 0.114 \cdot \left(\mathrm{Y2} + 1.772 \cdot \mathrm{Cb2}\right) \quad (3.15)$$

Further substituting the estimate for Cb2 given by (3.14) and solving for Y2 yields:

$$Y2 = 0.3375 \cdot R2 + 0.6625 \cdot G2 + 0.114 \cdot (Cb1 + Cb15) \quad (3.16)$$

At location 3, green and blue samples are available. Using an analogous method as described for calculating Y2, only now with bilinear interpolation performed on Cr samples, Y3 is calculated as:

$$Y3 = 0.299 \cdot (Cr1 + Cr21) + 0.8374 \cdot G2 + 0.1626 \cdot B3 \quad (3.17)$$

At location 4, only a green sample is available. So here we use bilinear interpolation on both the Cb and Cr channels to calculate Y4. Substituting the R and B equations from (2.2) into the definition of luma gives:

$$Y4 = 0.299 \cdot (Y4 + 1.402 \cdot Cr4) + 0.587 \cdot G + 0.114 \cdot (Y4 + 1.772 \cdot Cr4) \quad (3.18)$$

Using bilinear interpolation on the four surrounding samples to estimate Cb4 and Cr4, and solving for Y4 yields:

$$Y4 = 0.1785 \cdot (Cr1 + Cr15 + Cr21 + Cr23) + G4 + 0.086 \cdot (Cb1 + Cb15 + Cb21 + Cb23) \quad (3.19)$$

3.2.5 Summary of Complete Algorithm

Our complete demosaicking algorithm for producing YCbCr 4:2:0 output consists of the following steps. Each step must be carried out on every 2x2 cell before proceeding to the next step.

1. Fill the missing green samples with equations (3.1), (3.2) (3.3) and (3.4).

2. Using (3.7) and (3.9) find low-pass R-G and B-G values.

3. Using (3.12) calculate the final Cb and Cr samples.

4. Fill the luma channel with equations (3.13), (3.16), (3.17), (3.19).

3.3 Experimental Results

The 24 RGB images from the Kodac set where used in our experiments. These images have been extensively used in demosaicking research. Thumbnails of the images are provided in Figure 3.2. CFA images were obtained by sampling the RGB images with the Bayer pattern.

Figure 3.2: Test Images used in evaluating demosaicking algorithm performance. Numbered 1-24, from top left to bottom right.

We compared our proposed method against the YUV method in [31], and some fast demosaicking methods that operate in RGB space. The RGB methods tested were bilinear interpolation, and the methods in [6] and [9]. The quality of the different methods is evaluated by comparing the demosaicked image to the original full colour image, as illustrated in Figure 3.3.

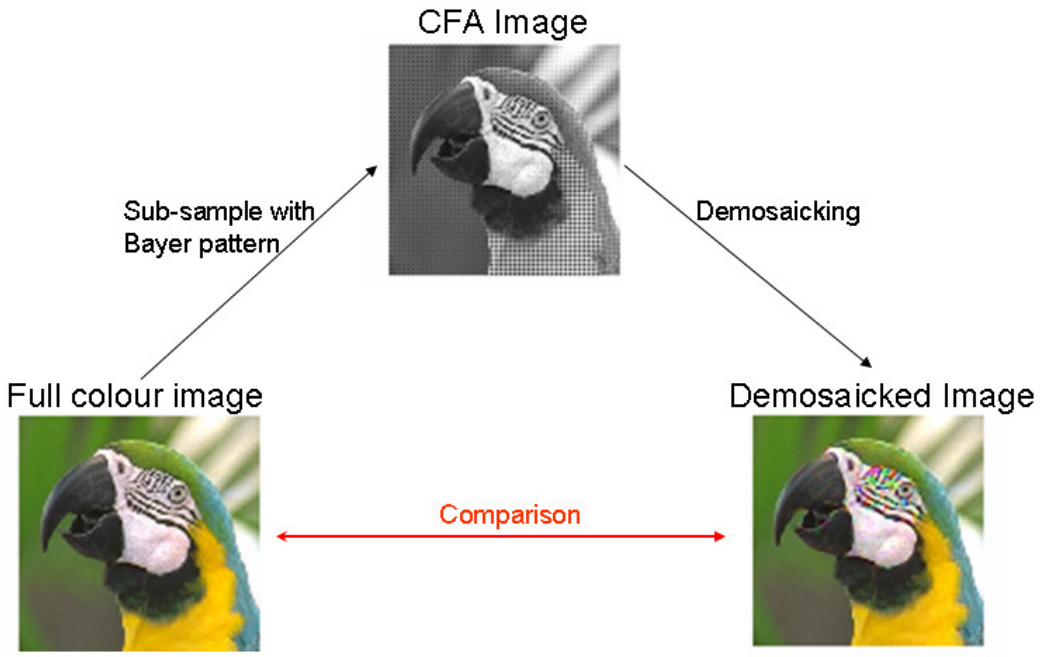


Figure 3.3: Procedure used for measuring demosaicking algorithm performance.

Here objective quality is measured in the YCbCr 4:2:0 domain using the peak signal-to-noise ratio (PSNR). The PSNR (in dB) of a demosaicked image channel, $I_{dem}(i,j)$ with 8 bit precision is calculated as:

$$PSNR = 10\log\left(\frac{255^2}{\frac{1}{MN}\sum_{i=1}^{M}\sum_{j=1}^{N}\left(I_{dem}(i,j)-I_{ref}(i,j)\right)^2}\right) \qquad (3.20)$$

where i denotes the row, j the column, M is the height of the channel, N is the width of the channel, and $I_{ref}(i,j)$ is the reference channel against which quality is measured.

The reference images were obtained by converting the full colour RGB images to YCbCr space with (2.1), filtering the Cb and Cr channels with a 9-tap FIR low-pass filter and downsampling. The 9-tap filter closely approximates an ideal low-pass filter with

cut-off 0.5π radians/sample, so the reference images contain very little aliasing in the down-sampled chroma channels.

For the demosaicking methods that operate in RGB space, the following low-complexity filter was used for filtering the chroma channels in the downsampling process:

$$h = \begin{bmatrix} 1/4 & 1/2 & 1/4 \end{bmatrix} \qquad (3.21)$$

This filter provides a good tradeoff between complexity and limiting aliasing.

Tables 3.1, 3.2 and 3.3 show the PSNR (in dB) obtained with each demosaicking method in the Y, Cb and Cr channels, respectively. In almost all cases, the proposed method gives higher PSNR than the other methods. On average the proposed method gives about 1 dB higher PSNR in each channel than the RGB based methods in [6] and [9]. For all images, the proposed method gives far better performance (over 5 dB higher PSNR in luma) than the only other YUV 4:2:0 based demosaicking method presented in [31].

Image	Bilinear	Method in [6]	Method in [9]	YUV Method in [31]	Proposed
1	29.58	35.99	35.97	30.90	37.43
2	36.31	41.68	41.28	37.04	42.21
3	37.45	43.44	43.40	38.17	44.32
4	36.82	41.71	42.33	37.65	42.52
5	29.60	37.49	37.09	30.85	38.36
6	31.04	37.47	37.38	32.53	38.87
7	36.59	43.60	42.42	36.88	43.92
8	27.08	34.76	33.11	28.55	35.85
9	35.67	42.75	41.55	36.57	43.81
10	35.57	42.63	42.36	36.78	43.58
11	32.33	38.57	38.44	33.58	39.65
12	36.82	43.32	42.47	38.12	43.93
13	26.90	32.18	33.51	28.18	33.57
14	32.23	38.50	38.16	33.07	39.09
15	35.98	40.32	41.43	36.64	41.26
16	34.62	40.94	40.42	36.04	42.24
17	35.17	41.04	41.50	36.31	42.11
18	31.06	36.51	37.48	32.15	37.71
19	31.49	39.75	37.38	32.94	40.94
20	34.78	41.23	41.02	35.83	42.40
21	31.69	37.80	38.00	32.90	39.18
22	33.67	39.24	39.33	34.88	40.35
23	38.21	44.58	43.75	38.69	44.86
24	29.90	35.03	36.36	31.06	35.88
Average	33.36	39.60	39.42	34.43	40.58

Table 3.1: Y-PSNR comparison of different demosaicking methods (dB)

Image	Bilinear	Method in [6]	Method in [9]	YUV Method in [31]	Proposed
1	35.03	40.03	39.37	37.85	42.68
2	41.73	45.01	44.25	42.46	45.48
3	42.76	45.46	45.36	41.85	45.94
4	43.45	45.69	46.07	43.31	47.56
5	37.22	40.87	40.73	36.61	41.59
6	36.17	41.30	40.63	38.29	43.35
7	42.88	45.60	44.67	39.87	45.40
8	32.05	38.44	35.73	33.49	40.35
9	41.19	45.22	44.06	40.84	45.38
10	41.53	45.61	45.21	41.64	46.17
11	37.86	42.53	41.84	40.11	44.36
12	41.97	46.35	45.15	43.01	46.60
13	32.78	36.18	37.08	35.50	38.90
14	37.75	40.60	40.03	36.09	40.35
15	42.01	43.61	44.93	41.58	45.81
16	39.10	44.32	43.15	41.03	45.68
17	41.78	44.44	44.68	41.40	45.27
18	37.09	40.29	40.76	37.83	41.75
19	36.78	43.00	40.37	37.16	43.85
20	40.64	43.91	43.45	40.73	44.38
21	37.12	41.34	41.12	38.69	43.26
22	39.12	42.17	41.80	39.15	42.57
23	44.66	46.79	45.87	41.62	45.46
24	35.23	38.04	38.95	36.15	39.53
Average	39.08	42.78	42.30	39.43	43.82

Table 3.2: Cb-PSNR comparison of different demosaicking methods (dB)

Image	Bilinear	Method in [6]	Method in [9]	YUV Method in [31]	Proposed
1	35.97	41.36	39.33	37.83	42.36
2	39.88	42.04	40.62	38.55	41.53
3	44.34	46.36	44.24	42.16	46.01
4	41.18	42.36	41.40	39.13	41.90
5	36.74	42.02	39.06	37.24	42.18
6	38.93	42.77	40.77	39.24	43.76
7	42.51	46.23	43.65	40.51	45.89
8	30.25	39.29	35.19	34.03	40.03
9	40.88	46.34	43.66	42.05	46.65
10	42.03	46.30	44.23	42.24	46.45
11	38.02	42.62	40.58	39.29	42.82
12	42.70	46.10	44.27	42.51	46.03
13	34.78	38.57	38.19	37.01	40.28
14	37.92	40.29	38.41	36.37	39.57
15	39.77	41.03	40.69	38.46	41.05
16	43.35	45.92	43.92	42.13	46.68
17	41.48	46.09	44.55	43.10	46.87
18	37.58	41.40	40.24	38.57	42.05
19	36.36	44.24	39.80	38.05	44.48
20	40.53	45.86	43.03	42.60	46.41
21	38.86	43.15	41.22	39.92	44.19
22	38.27	42.23	41.00	39.01	42.36
23	43.91	45.97	43.56	40.98	45.28
24	37.12	40.32	39.72	37.58	40.71
Average	39.31	43.29	41.31	39.52	43.56

Table 3.3: Cr-PSNR comparison of different demosaicking methods (dB)

Since PSNR is not always an accurate measure of perceived image quality we also provide images for subjective quality comparisons. Figure 3.4 and Figure 3.5 show cropped portions of images 6 and 8, respectively, and the result of applying each demosaicking method to the image. For the RGB based demosaicking methods, the images have been converted to YCbCr 4:2:0 format using the filter in (3.21) for the chroma downsampling process. Close visual inspection of the images show that the

proposed method produces fewer color artifacts and results in less blurring of edges than the other methods. Also note how that despite providing competitive PSNR, the method in [9] produces unpleasing zipper artefacts along some edges (Figure 3.4d, Figure 3.5d).

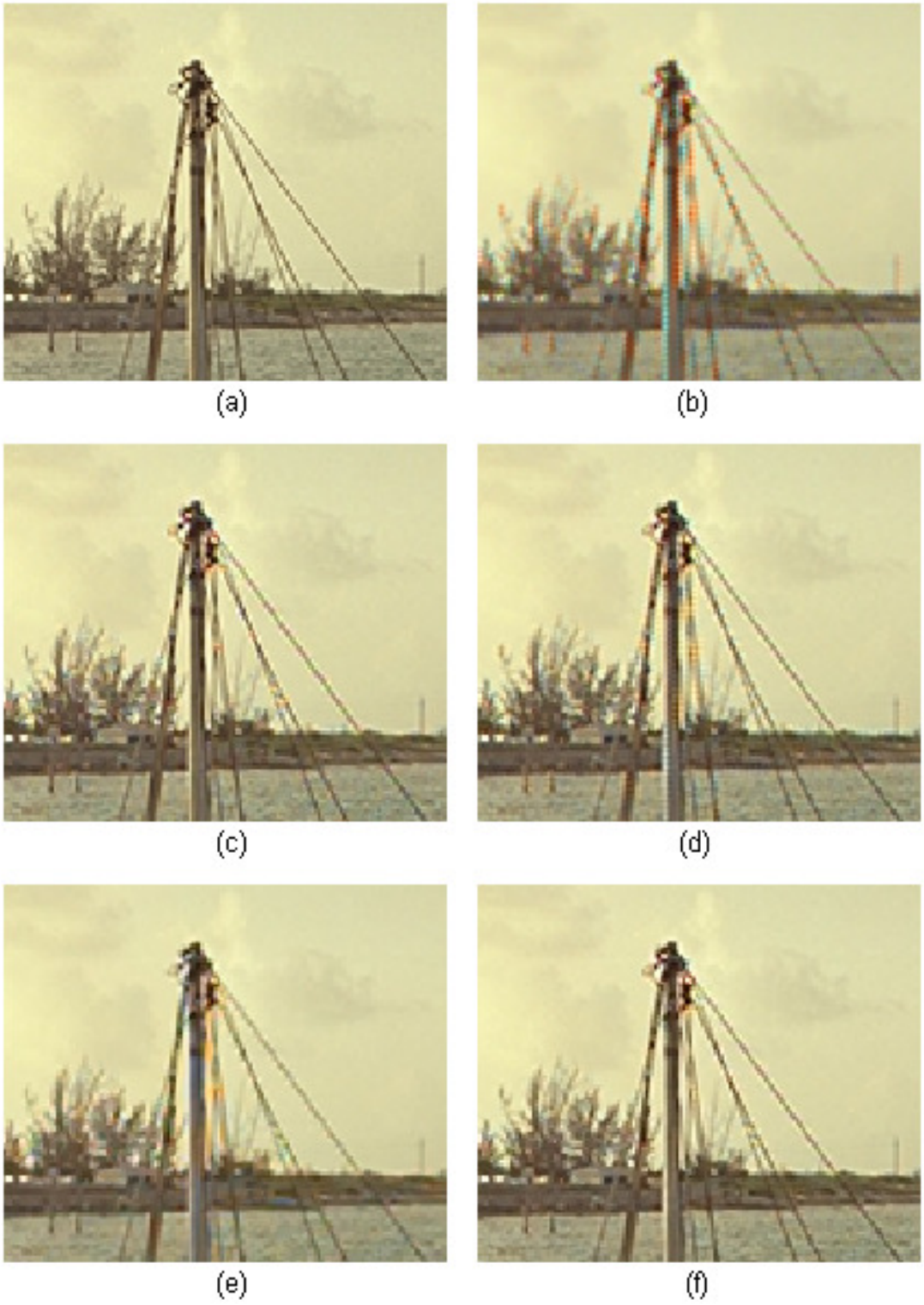


Figure 3.4: Comparison of demosaicking methods on a cropped portion of image 6. (a) original image (b) bilinear (c) method in [6] (d) method in [9] (e) YUV method in [31] (f) proposed method

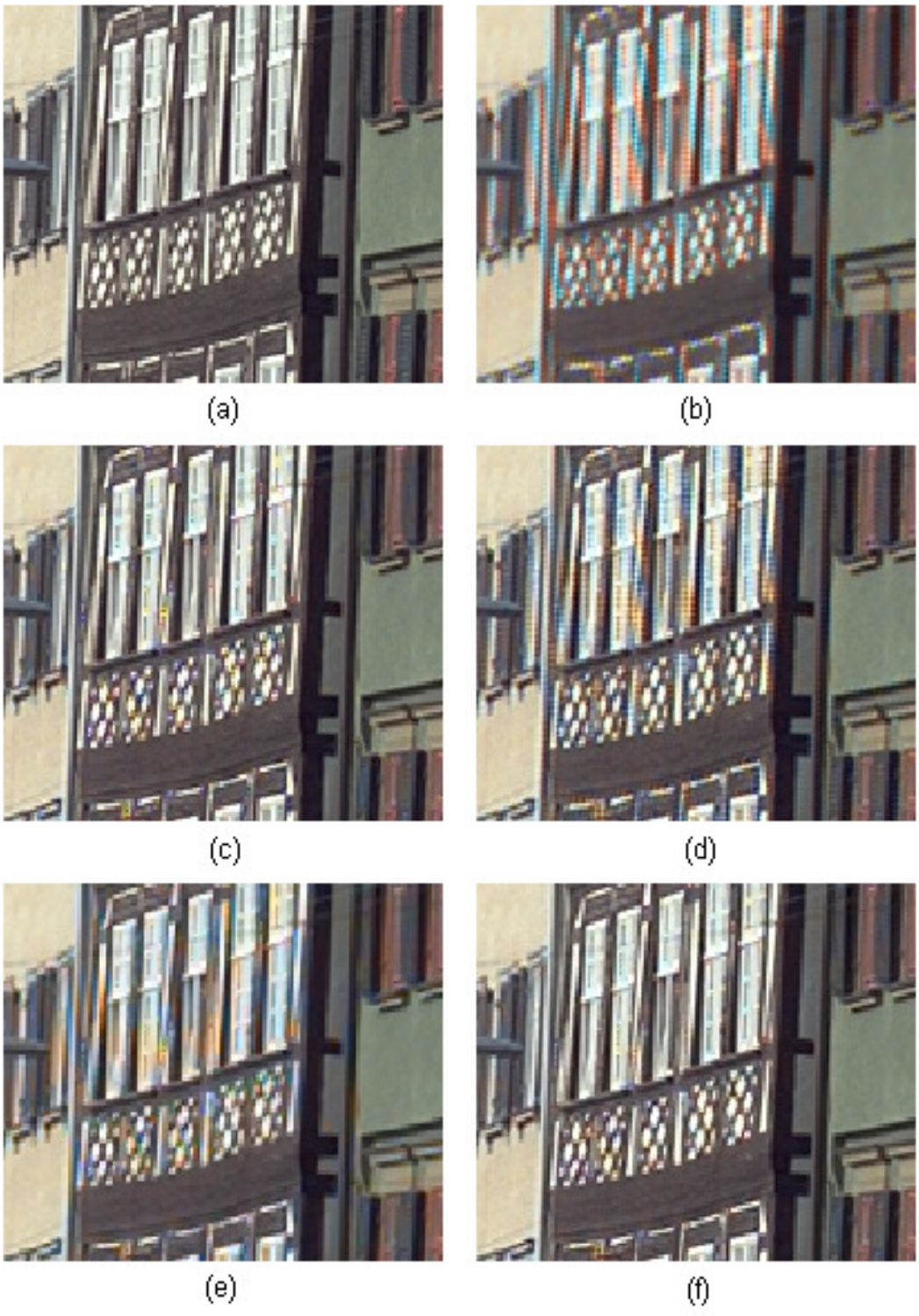


Figure 3.5: Comparison of demosaicking methods on a cropped portion of image 8. (a) original image (b) bilinear (c) method in [6] (d) method in [9] (e) YUV method in [31] (f) proposed method

3.4 Complexity Analysis

A key advantage of the proposed method is the computational complexity saved by directly producing YCbCr 4:2:0 output rather than performing demosaicking in RGB space and then converting to YCbCr 4:2:0. Table 3.4 shows a summary of the number of operations per pixel in the CFA image needed for the proposed demosaicking method. Note there are some fractional values in Table 3.4 because many equations are not evaluated out at every pixel location. The number of operations performed when evaluating equations (3.2) and (3.4) is variable depending on the result of the comparisons; only the worst case complexity is shown in Table 3.4.

For comparison, Table 3.5 presents a complexity analysis of the method in [9], which is one of the lowest complexity RGB based demosaicking algorithms reported in the literature. This table shows the number of operations required for demosaicking with the method in [9] and then converting to YCbCr 4:2:0 format. In this analysis, the simple filter in equation (3.21) is used for limiting aliasing in the Cb and Cr channels and an efficient downsampling scheme is used (where the filtering operations are performed at the lower sampling rate).

Step	Addition	Shift	Multiplication	Absolute Value	Comparison
Green interpolation (worst case)	9	2.5	0	2	1
Low-pass KR, KB	5.5	1	0	0	0
Generating chroma	0.5	0	1	0	0
Calculating Luma	4	0	2.75	0	0
Total (worst case)	19	3.5	3.5	2	1

Table 3.4: Number of operations per pixel required for the proposed method.

Step	Addition	Shift	Multiplication	Absolute Value	Comparison
Green Interpolation	4	1.5	0	0	0
Red Interpolation	4	0.75	0	0	0
Blue Interpolation	4	0.75	0	0	0
RGB to YCbCr Conversion	6	0	9	0	0
Filtering and downsampling rows, Cr	1	1	0	0	0
Filtering and downsampling columns, Cr	1	1	0	0	0
Filtering and downsampling rows, Cb	0.5	0.5	0	0	0
Filtering and downsampling columns , Cb	0.5	0.5	0	0	0
Total	21	6	9	0	0

Table 3.5: Number of operations per pixel required for the demosaicking method in [9] plus conversion to YCbCr 4:2:0 format.

Comparison of Tables 3.4 and 3.5 shows that the proposed demosaicking method has lower complexity than the method in [9]. In the worst case, our method requires slightly fewer additions and shift operations. More importantly, the proposed method uses far fewer multiplication operations, which are expensive to implement in the low cost DSP (digital single processor) chips typically used in digital cameras. The multiplications are required for the RGB to YCbCr conversion, so our proposed method uses fewer

multiplications than performing any RGB based demosaicking method and subsequently converting to YCbCr space.

The demosaicking method in [6] has considerably higher complexity than the method in [9], so our method has much lower complexity than that of [6]. We are not aware of any demosaicking methods with complexity lower or equal to [9] that provide comparable image quality.

3.5 Conclusions

In this chapter, we have presented a fast demosaicking algorithm that directly produces YCbCr 4:2:0 output. Our method saves considerable computational complexity by avoiding the need for performing demosaicking in the RGB colour space and then converting from RGB to YCbCr 4:2:0 format.

The proposed method generates a full green channel, and low-pass filtered, down-sampled red and blue samples. The green channel contains the fine detail needed to generate a high quality luma channel, while the low-pass R-G and B-G values allow us to directly compute low-pass, down-sampled chroma channels.

The proposed method achieves much higher PSNR than the only other demosaicking method that produces luma and chroma output. It also achieves better quality than fast RGB based demosaicking methods, with lower complexity than performing demosaicking in RGB space and then converting to YCbCr 4:2:0 format.

Our demosaicking method prepares the image (or video) for compression. Thus, it would be used in the conventional camera workflow of first performing demosaicking

and then compressing the image/video with standard methods. In performing demosaicking to YCbCr 4:2:0 format, the number of samples is increased by a factor of 1.5, which is somewhat undesirable. To avoid this, in the next chapter we present methods for compressing a CFA video prior to demosaicking, taking advantage of the smaller data size.

4 H.264 Based Compression of Bayer Pattern Video

4.1 Introduction

The conventional approach to compressing CFA data (still image or video) is to first perform demosaicking and then compress the resulting full colour data. This approach is sub-optimal because the amount of data is expanded by a factor of three in the demosaicking stage, which increases the compression processing time and introduces redundancy that the compression stage must remove. To avoid these problems, compression can be carried out on the CFA data prior to demosaicking.

In this chapter two new methods are proposed for compressing CFA video data. One uses the H.264 video coding standard and one uses a modified version of the H.264. H.264 is the latest major video coding standard and it provides significant improvements in coding efficiency over previous standards such as MPEG-2. Basing our methods on H.264 allows us to exploit the latest powerful video coding tools. Our first proposed method compresses the CFA video with standard H.264 and achieves better quality (measured with mean-square error) than the demosaick-first approach at high bit-rates. Our second method further increases compression efficiency by introducing a modified motion compensation scheme into H.264, alleviating problems that arise due to aliasing in the CFA data. Both methods are suitable for devices such as digital camcorders where video is encoded with high quality.

The rest of the chapter is organized as follows. In Section 4.2, aliasing in CFA data and its negative effect on video coding are discussed, providing the motivation for our

modified motion compensation scheme. The proposed methods for compressing CFA video data are described in Section 4.3. Simulation results showing the performance our methods relative to the conventional demosaick-first approach are presented in Section 4.4, along with a comparison of the complexity the different approaches. Conclusions are presented in Section 4.5.

4.2 Impact of Aliasing on CFA Video Coding

Aliasing in video has been shown to negatively impact the coding of P frames [33]. If there is movement between frames the effect of aliasing will be different in each frame. This results in low correlation between frames, and hence large P-frame size. The negative effect of aliasing can be reduced by using sub-pixel accurate motion vectors together with adaptive interpolation filters [34].

CFA data can contain severe aliasing [35]. Single sensor cameras usually use an optical filter to limit aliasing [5]. When selecting how much filtering to use, there is a trade-off between limiting aliasing and capturing fine image detail. Furthermore, since the Bayer pattern contains more green samples than red or blue, different amounts of filtering are needed to limit aliasing in the different colours. If enough filtering were used so that there was little aliasing in the red and blue channels, then significant detail would be lost from the green channel which is undesirable, and defeats the purpose of sampling green at a higher rate than red or blue.

An assumption sometimes made in demosaicking research is that enough optical filtering is done so that if a full colour image were captured, it would contain negligible aliasing; however sampling with the Bayer CFA introduces aliasing [36]. Other work

makes the assumption that significant aliasing occurs in the red and blue channels, but not in the green [10],[37].

The effect of aliasing in CFA video is illustrated in Figure 4.1, which shows a frame of red data from the "Mobile and Calender" video, and a blowup of the highlighted region over four successive frames. Over these frames, the calendar is moving vertically. Notice how the moving portion, especially the number '3', looks considerably different in each frame due to aliasing.

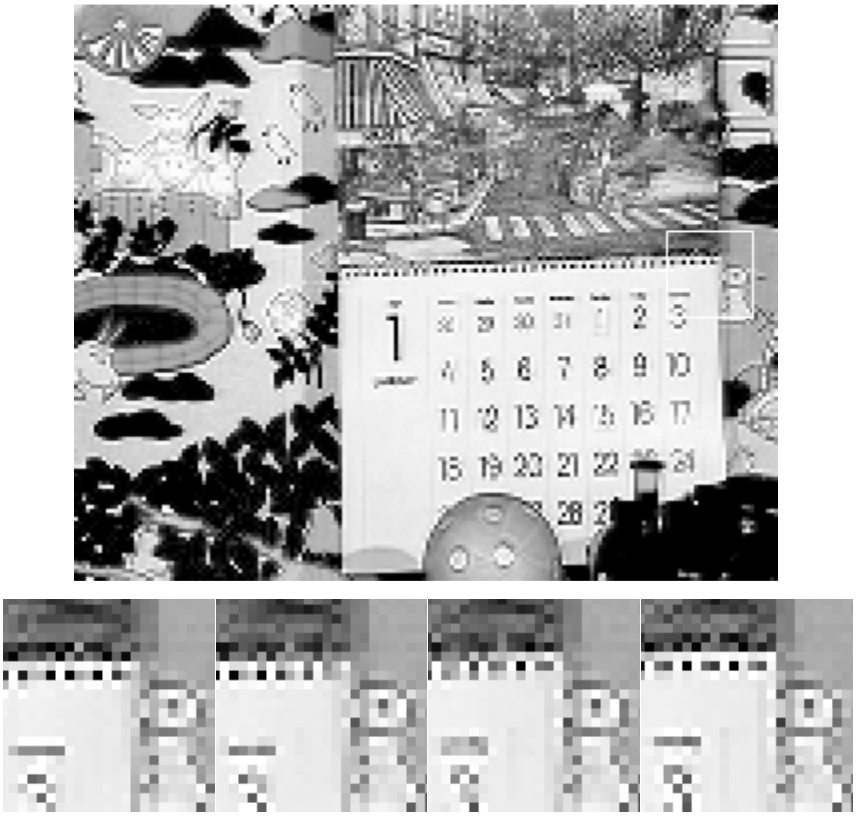


Figure 4.1: A frame of the red channel from the "Mobile and Calender" video (top), and a blowup of the highlighted region over four successive frames (bottom), illustrating the affect of aliasing and the low correlation between frames.

Advanced demosaicking algorithms attempt to reduce the effects of aliasing in each colour channel by using information from the other colour channels [9],[10],[37],[38].

An example of this is shown in Figure 4.2, which presents the four frames of red data in Figure 4.1 after demosaicking has been performed with the method presented in [38].

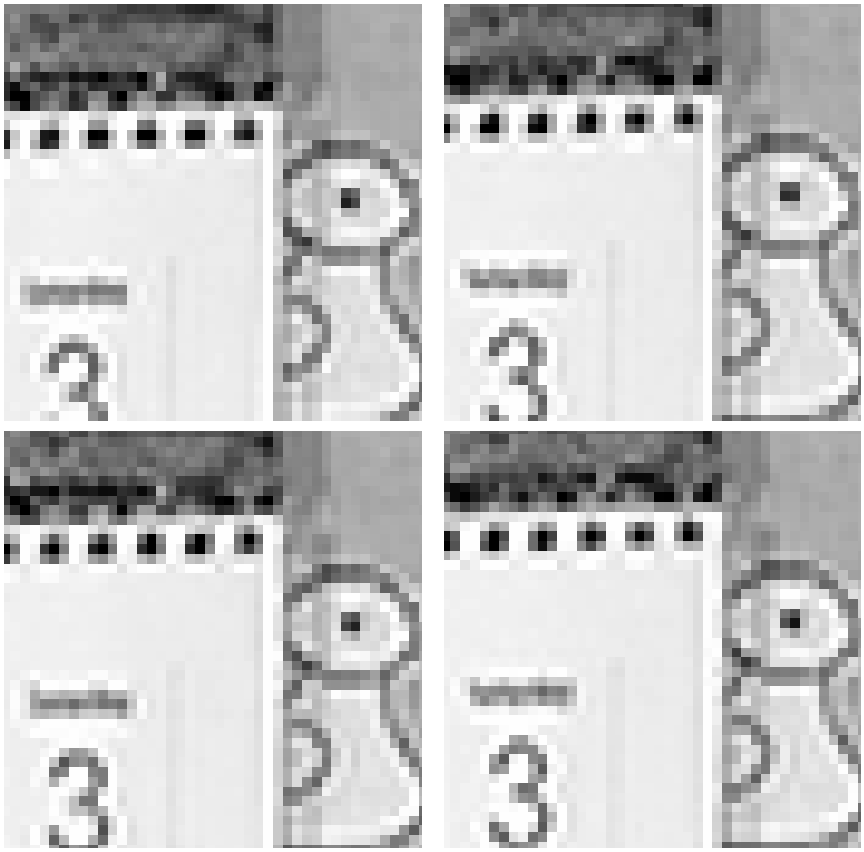


Figure 4.2: The four frames of red data in Figure 4.1 after demosaicking has been performed.

Demosaicking increases the amount of red data by a factor of four, so the frames in Figure 4.2 are bigger than in Figure 4.1. We observe that there is considerably more temporal correlation in the frames after demosaicking than there is in the original CFA data. Since each CFA frame is a subset of the corresponding demosaicked frame, the CFA frames can be effectively predicted from the demosaicked versions of the other frames.

4.3 Proposed Methods for CFA Video Compression

Both of our proposed methods involve dividing the CFA data into separate arrays of green, blue and red data, which are compressed in 4:2:2 sampling mode. In our first method, standard H.264 is used for compressing the arrays of red, green and blue. In our second method, a modified motion compensation (MC) scheme is also applied, where demosaicking is performed on the reference frames within the encoder and decoder to reduce the negative effects of aliasing on P-frames. The method of creating rectangular arrays of each colour and arranging the data for compression with H.264 is described in Section 4.3.1. The modified MC scheme used in our second method is described in Section 4.3.2.

4.3.1 Pixel Rearranging

Most video compression standards, including H.264, can only compress video of rectangular shape. So in order to compress the CFA data using H.264, the pixels must be rearranged into rectangular arrays. The red and blue data are sampled in a rectangular manner, so they can easily be separated into arrays one quarter the size of the Bayer data.

In [17] two options for creating rectangular arrays of quincunx sampled data are proposed (Figure 4.3). These methods are applied to luma samples after a color space conversion in [17], but they can also be applied to the green samples directly. If a frame of Bayer pattern data has dimensions *MxN* (height, width), the structure separation method involves separating the green data into two arrays of size *(M/2)x(N/2)*, one containing the even column samples and the other containing the odd column samples (Figure 4.3c). Implementing this method in a video codec would require extensive

modifications to allow four channels to be compressed together rather than the usual three. Also, downsampling the green data into two separate channels introduces further aliasing in each channel (in addition to the aliasing introduced due to Bayer sampling the green channel). In [17] a filter is applied to the green data before downsampling to limit the aliasing. However, applying filtering is undesirable since it removes high-frequency detail which cannot be recovered later.

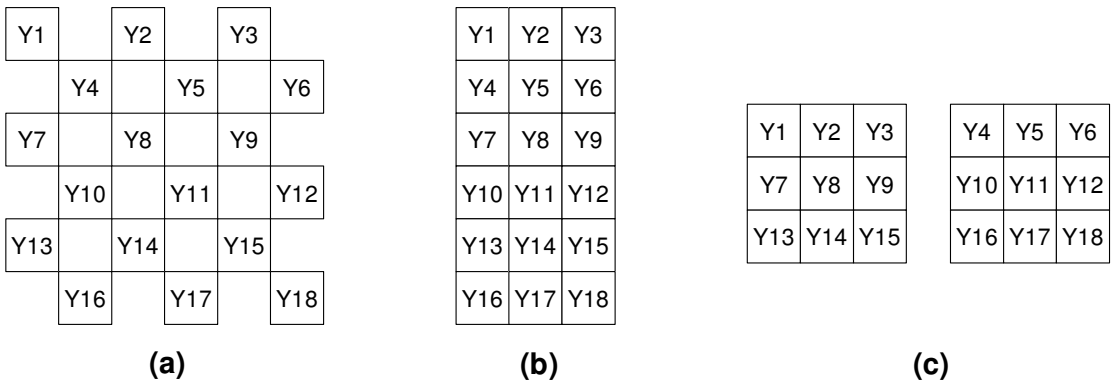


Figure 4.3: Methods for forming rectangular arrays of luma data in [17]. (a) Original luma arrangement (b) Structure conversion (c) Structure separation

The structure conversion method in [17] involves merging the two green samples from every group of four Bayer pattern samples into a single column, resulting in a green channel of size *Mx(N/2)*. This method does not suffer from the aliasing problems of the structure separation method, however the merging process does distort the data somewhat. In [19] the structure conversion method is used for compressing video, where they compress a green channel of size *Mx(N/2)* together with red and blue channels of size *(M/2)x(N/2)*. Since the relative dimensions of the three colour channels do not correspond to any sampling scheme supported in major video coding standards, a custom codec was used, where the motion vectors from the green channel are reused on the red and blue channels, with appropriate downsampling and scaling on the motion vectors.

We proposed a different structure conversion method, where the samples are merged into rows. Let *f(i,j)* be the value of the CFA data at spatial location *(i,j)* within the image, where *i* denotes the row and *j* the column. Let *R(i,j)*, *G(i,j)* and *B(i,j)* denote the arrays of red, green and blue data after conversion to separate arrays. The color arrays can be expressed in terms of the CFA data by the following equations:

$$\begin{aligned} G(i,j) &= \begin{cases} f(2i,j) & j \text{ even} \\ f(2i+1,j) & j \text{ odd} \end{cases} \\ B(i,j) &= f(2i+1,2j) \\ R(i,j) &= f(2i,2j+1) \end{aligned} \qquad (4.1)$$

The array *G(i,j)* has dimensions *(M/2)xN*, *B(i,j)* and *R(i,j)* have dimensions *(M/2)x(N/2)*, as illustrated in Figure 4.4. This approach allows the data to be compressed in 4:2:2 sampling mode, with green data in the luma channel and red and blue data in the chroma channels. Since 4:2:2 sampling support was added to H.264 with the Fidelity Range Extensions (FRExt) [39], the CFA data can be compressed with H.264 achieving the same effect as in [19] (reusing motion vectors from the green channel on the red and blue data) without the need for a custom codec. Our first proposed method simply consists of compressing the green, blue and red arrays given by (4.1) with standard H.264 using 4:2:2 sampling.

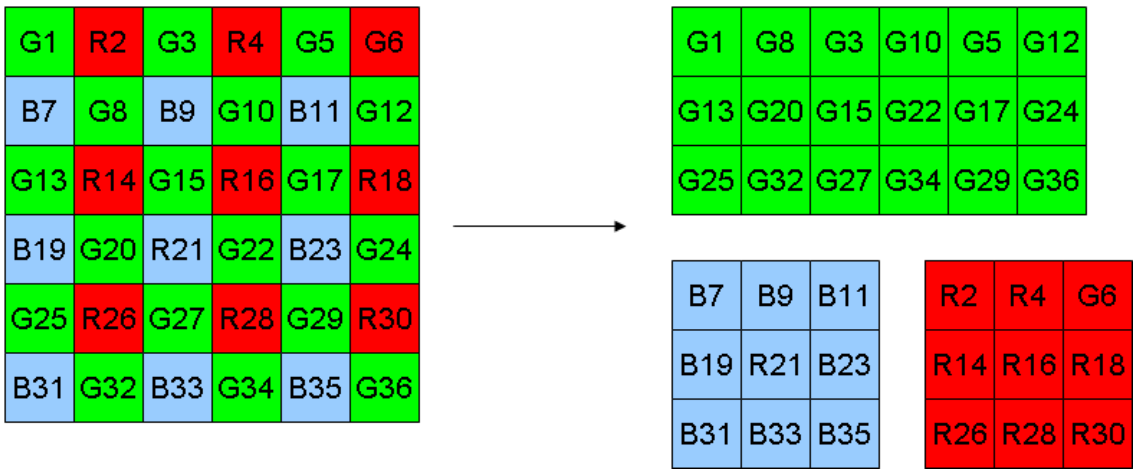


Figure 4.4: Conversion from mosaic data into separate R, G and B arrays used in our CFA video compression methods.

Our first proposed method has the advantage of using standard H.264. In the following section we describe a second proposed method which increases coding efficiency by using a modified motion compensation scheme, at the expense of increased complexity.

4.3.2 Modified Motion Compensation

As discussed in Section 4.2, aliasing in CFA data negatively effects P-frame coding. In our second proposed method we minimize this problem by performing demosaicking on the reference frames in the encoder and decoder for the purposes of motion compensation. Each P-frame of CFA data is predicted from the demosaicked reference frames, providing a better prediction for the frame being coded and hence lowering the bit-rate.

In H.264, I and P frames are used for predicting other frames of the video. After a frame has been encoded, it is decoded within the encoder, and the decoded version of the frame is used for prediction. In our method rather than directly using the decoded frame

for prediction, demosaicking is first performed on the decoded frame, and the demosaicked frame is used for prediction. This increases the size of the green data by a factor of two and the red and blue data by a factor of four. This is illustrated in Figure 4.5a, which shows a block of pixels from a decoded frame, and the block after demosaicking. The numbers in Figure 4.5a indicate the location of each of the original pixels in the demosaicked frame and the unlabeled pixels are generated in the demosaicking process. Any demosaicking method could be used on the reference frames. The choice of a particular method would depend on the application and the amount of complexity that can be tolerated. Different demosaicking methods are evaluated for this purpose in Section 4.4.2.

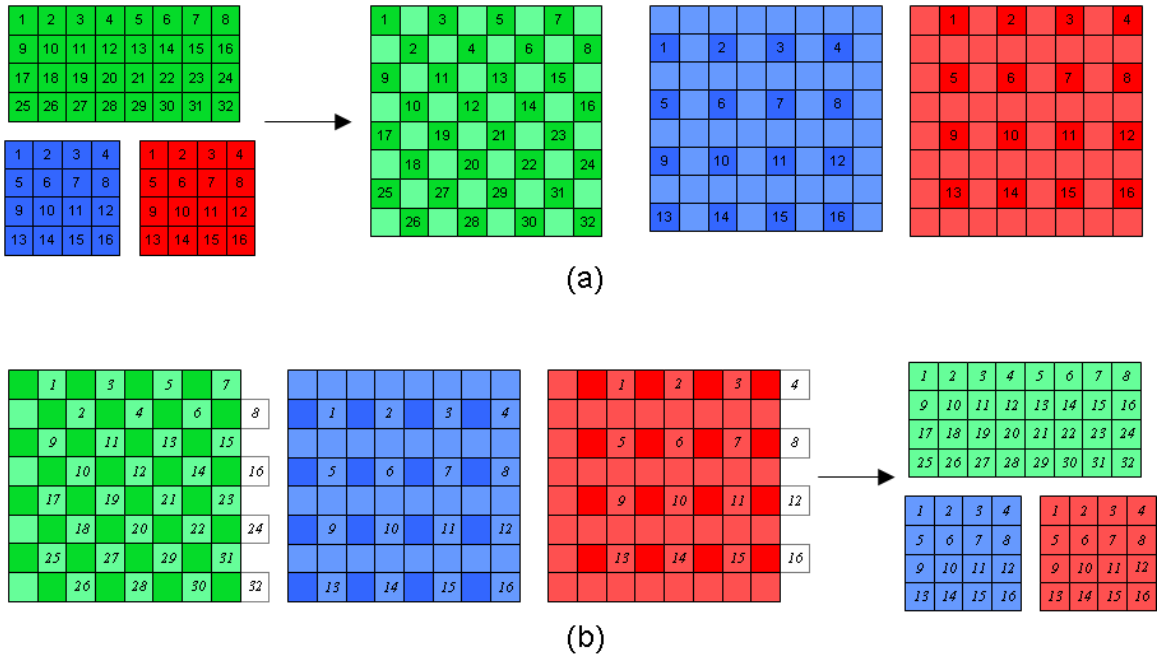


Figure 4.5: (a) Performing demosaicking on a block of pixels from a reference frame (b) Sampling the demosaicked reference frame to obtain a prediction for the block when the motion vector is (1,0) in full pel units

After demosaicking, all three colour channels are up-sampled by a factor of four in the horizontal and vertical directions to support quarter pel accurate motion vectors. This

is done using the method defined in the H.264 standard for upsampling the luma channel (where a 6-tap FIR filter generates the half-pel samples, and bilinear interpolation is used to generate quarter-pel samples). RGB data has properties more similar to luma than chroma, so the luma upsampling method is used to give better performance.

In the H.264 reference encoder, motion estimation (ME) is done on the luma channel. When ME is performed on RGB data, the green channel is usually used for ME [14][15], since the green data is more high correlated with luma than red or blue. So in our second proposed method, ME is done on the green channel. Consider a green pixel at location (i_G, j_G) in a CFA frame. After demosaicking, this pixel will be located at position $(2i_G, j_G)$ in the demosaicked frame if j_G is even and position $(2i_G+1, j_G)$ if j_G is odd (Figure 4.5a). Let Ω denote the set of coordinates of the green pixels within a block in the CFA data. A motion vector (m_i, m_j) is calculated for the block by minimizing the sum of absolute differences (SAD) given by:

$$SAD = \sum_{(i,j)\in\Omega} \begin{cases} G(i,j) - G_{dem}(2i + m_i, j + m_j) & j \text{ even} \\ G(i,j) - G_{dem}(2i + 1 + m_i, j + m_j) & j \text{ odd} \end{cases} \qquad (4.2)$$

where $G_{dem}(i,j)$ is the demosaicked reference frame being used for prediction. In equation (4.2), the demosaicked reference frame is being sampled with shape of the green data in the Bayer pattern, with the motion vector controlling the relative position of the sampling. After a full pel motion vector has been found with (4.2), the motion vector is refined to quarter pel accuracy, as is done in the H.264 reference encoder [4] (this basically involves minimizing the SAD given by (4.2), only now letting m_i and m_j take on values in increments of 0.25 pixels).

In our method, the motion vectors calculated from the green channel are also used on the red and blue channels. A red pixel at location (i_R,j_R) will be moved to $(2i_R, 2j_R+1)$ after demosaicking, and a blue pixel at (i_B,j_B) will be moved to $(2i_B+1, 2j_B)$. In order to obtain a prediction for a pixel in a CFA frame using motion compensation, the motion vector calculated is added to the corresponding position of the pixel in the demosaicked frame, and the demosaicked frame is sampled at that position. Let $B_{dem}(i,j)$, and $R_{dem}(i,j)$ be the values of the demosaicked frame being used for prediction, and the motion vector for a block of CFA data be (m_i, m_j). Then the predictions for the CFA pixels in the block will be:

$$
\begin{aligned}
G_{pred}(i_G, j_G) &= \begin{cases} G_{dem}(2i_G + m_i, j_G + m_j) & j_G \text{ even} \\ G_{dem}(2i_G + 1 + m_i, j_G + m_j) & j_G \text{ odd} \end{cases} \\
B_{pred}(i_B, j_B) &= B_{dem}(2i_B + m_i + 1, 2j_B + m_j) \\
R_{pred}(i_R, j_R) &= R_{dem}(2i_R + m_i, 2j_R + 1 + m_j)
\end{aligned}
\qquad (4.3)
$$

As an example, consider the task of predicting the 8x4 block of CFA pixels in Figure 4.5a in a future frame when the motion vector is (1,0) in full pel units. Figure 4.5b shows how the demosaicked frame is sampled to obtain a prediction for the block. The white squares represent pixels that are obtained by edge replication.

In summary, our MC scheme uses demosaicked versions of reference frames to predict the CFA frame being coded. This takes advantage of the increased temporal correlation of frames after demosaicking has been performed, without the need for compressing the larger demosaicked frames themselves.

4.4 Results

4.4.1 Testing Methodology

To evaluate the performance of our compression schemes two videos from the SVT High Definition Test Set [40], *CrowdRun* and *OldTownCross*, were used in our simulations. The *CrowdRun* sequence is a shot of hundreds of marathoners running through a park. It contains a high amount of motion due to the runners and also slight camera motion. The *OldTownCross* sequence is an aerial shot of a European city containing camera motion. Both videos contain large amounts of fine detail (high frequency image content). Screenshots of the test videos are shown in Figure 4.6.

Figure 4.6: Screenshots of the test videos, *CrowdRun* (top) and *OldTownCross* (bottom).

These videos were digitized at a resolution of 3840x2160, 16 bits per (RGB) colour plane, 50 frames/sec. We down-sampled and cropped the videos to give 720x480, 8 bit RGB data at 25 frames/sec. This is the quality of video typically captured by consumer digital camcorders, our target application. CFA videos were obtained by sampling the RGB videos with the Bayer pattern.

In all tests, 60 frames of each video were compressed, with I-frames inserted every 15 frames. CABAC and B-frames were disabled to lower the encoding complexity (these features are not likely to be used in digital camcorders). In the ME process, two reference frames were used and the search range was ±16 integer pixels.

4.4.2 Demosaicking Algorithm Choice

Here we evaluate the effectiveness of different demosaicking algorithms for use in our modified motion compensation scheme.

Three different demosaicking algorithms were tested; bilinear interpolation, an edge-sensing method by Hamilton and Adams using Laplacian second order correction terms [6] (referred to as Laplacian), and a state-of-the-art method based on estimating luma from the CFA data [38]. These algorithms vary in complexity and the quality of image they produce; bilinear interpolation has very low complexity but gives the poorest image quality. The luma-based method gives much better image quality (in terms of mean-square error) but has significantly greater computational cost. Laplacian demosaicking is intermediate in both image quality and complexity.

Rate-distortion curves obtained using different demosaicking algorithms for MC are shown in Figure 4.7. Here, the PSNR of the CFA data itself is measured (as opposed to measuring quality after demosaicking has been performed). Results are shown for our proposed MC scheme with each demosaicking algorithm, as well as compressing the CFA video with a standard H.264 encoder in 4:2:2 sampling mode (without our proposed MC scheme).

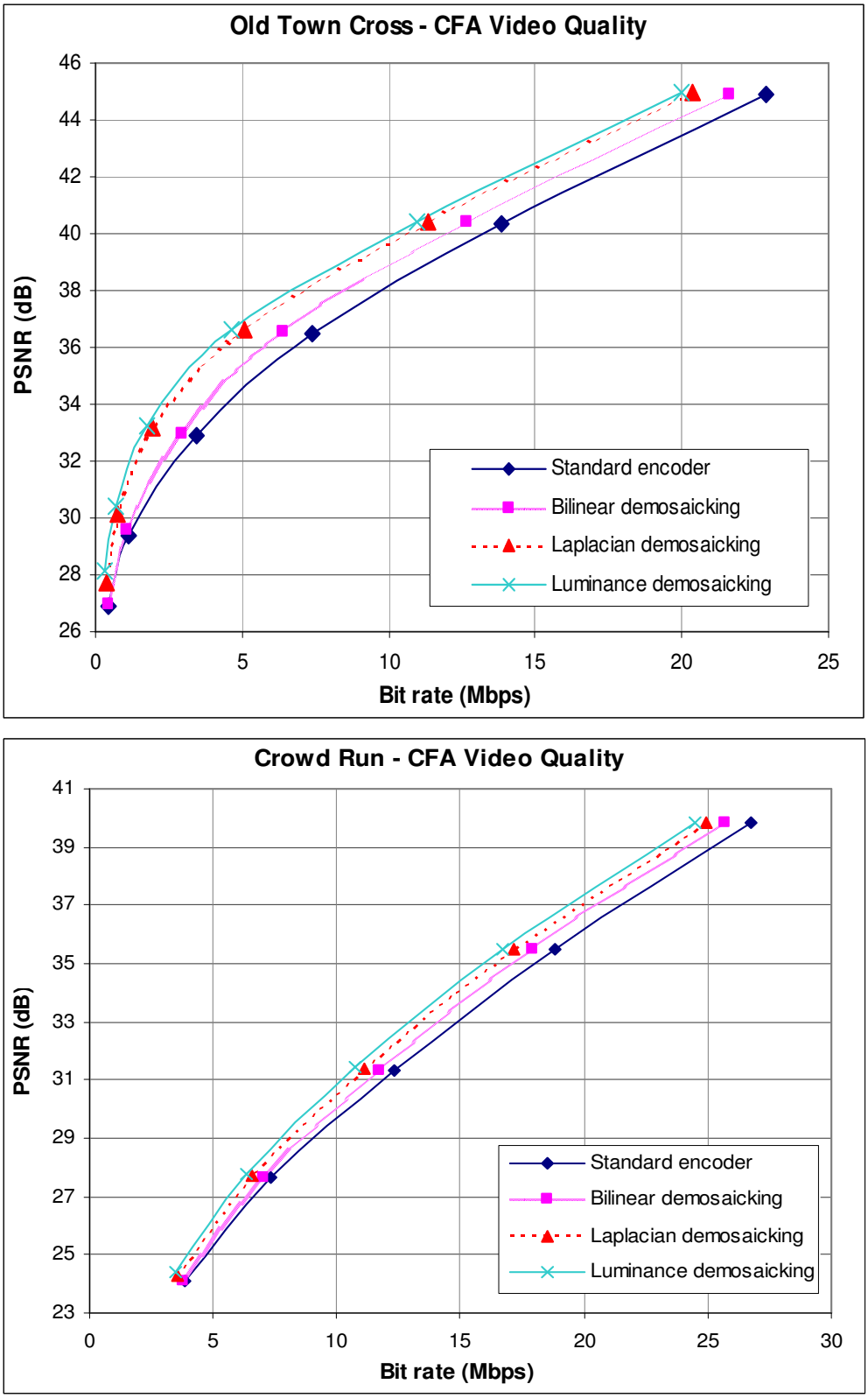


Figure 4.7: Plots of PSNR (of the CFA data) vs. bit rate for the two test videos obtained when different demosaicking algorithms are used for motion compensation.

The results for both videos show there is substantial benefit in using our MC scheme. The bit-rate reductions for the *CrowdRun* sequence are about 4%, 10%, and 12% using bilinear, Laplacian and luma-based demosaicking, respectively. Greater bit rate reduction is obtained on the *OldTownCross* sequence, up to 15% using bilinear, 43% using Laplacian and 48% using luma-based demosaicking. The *OldTownCross* sequence experiences greater bit-rate reduction when our MC scheme is used because that video contains camera motion. In general MC is very effective on videos with camera motion, because the picture experiences simple translational motion. When the motion is more complicated, such as the motion the marathoners in the *CrowdRun* sequence, MC is less effective, and thus enhancing MC provides less benefit.

These results show that the more advanced demosaicking schemes provide significant bit-rate reductions over simple demosaicking such as bilinear. Using Laplacian demosaicking for MC provides bit rates almost as low as the luma-based demosaicking method with considerably lower complexity, so Laplacian demosaicking is used in the rest of our tests.

4.4.3 Quality Comparison Against Demosaick-First Approach

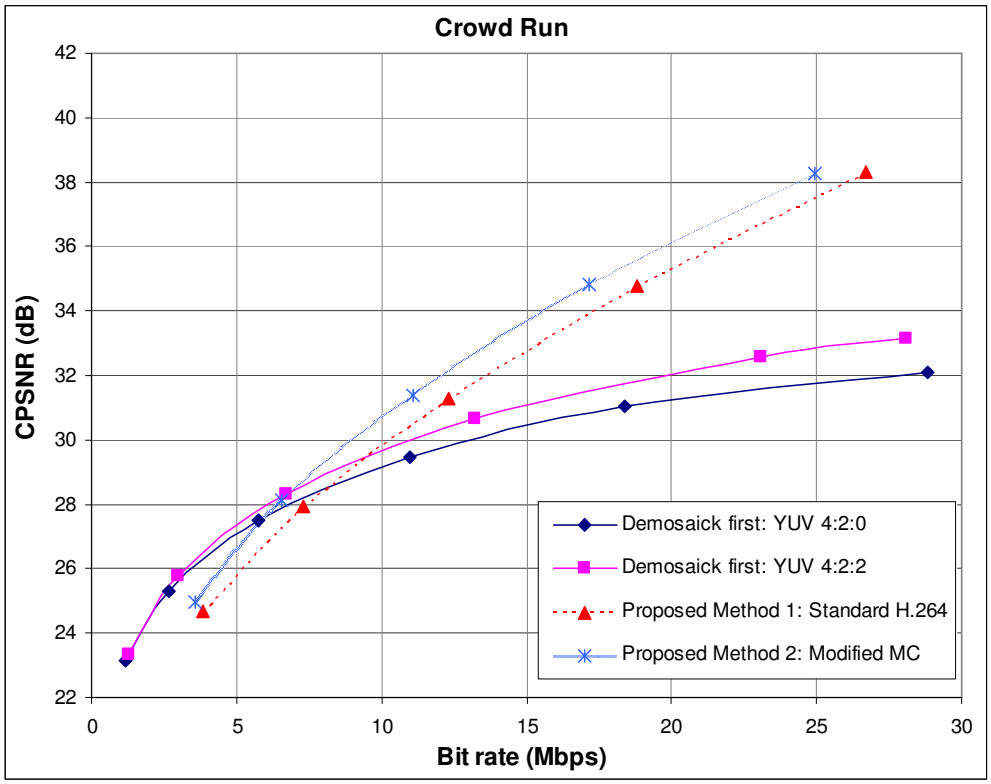
In this section we compare our two methods against the conventional demosaick-first approach. When comparing to the demosaick-first approach, the quality of the final RGB video, after both compression and demosaicking have been performed, needs to be measured. The quality measure used here is the Composite Peak Signal to Noise Ratio (CPSNR), which has been used in previous work on compressing CFA data [17],[19]. The CPSNR is calculated as the standard PSNR, but with the mean-square error

evaluated across the red, green and blue colour channels. The CPSNR for one frame of video with 8 bit data is given by:

$$CPSNR = 10\log\left(\frac{255^2}{\frac{1}{3MN}\sum_{i=0}^{M-1}\sum_{j=0}^{N-1}\sum_{k=1}^{3}\left(I_{comp}(i,j,k)-I_{ref}(i,j,k)\right)^2}\right) \qquad (4.4)$$

where k denotes the colour component (R, G, or B), $I_{comp}(i,j,k)$ is the compressed video after demosaicking, and $I_{ref}(i,j,k)$ is the reference video against which quality is measured. The reference is taken as the video obtained by demosaicking the CFA data without any compression. Thus, the reference video represents the best quality that can be obtained from the CFA data with a given demosaicking algorithm. The CPSNR for a video is taken to be the average CPSNR of the frames in the video.

The proposed methods were compared against the conventional demosaick-first approach. The demosaick-first results were obtained by demosaicking the CFA data (with the Laplacian method), converting from RGB to YUV and compressing with standard H.264. Results are presented for the demosaick-first approach using both YUV 4:2:0 and YUV 4:2:2 format during compression. Plots of CPSNR vs. bit-rate for the two test videos are shown in Figure 4.8.



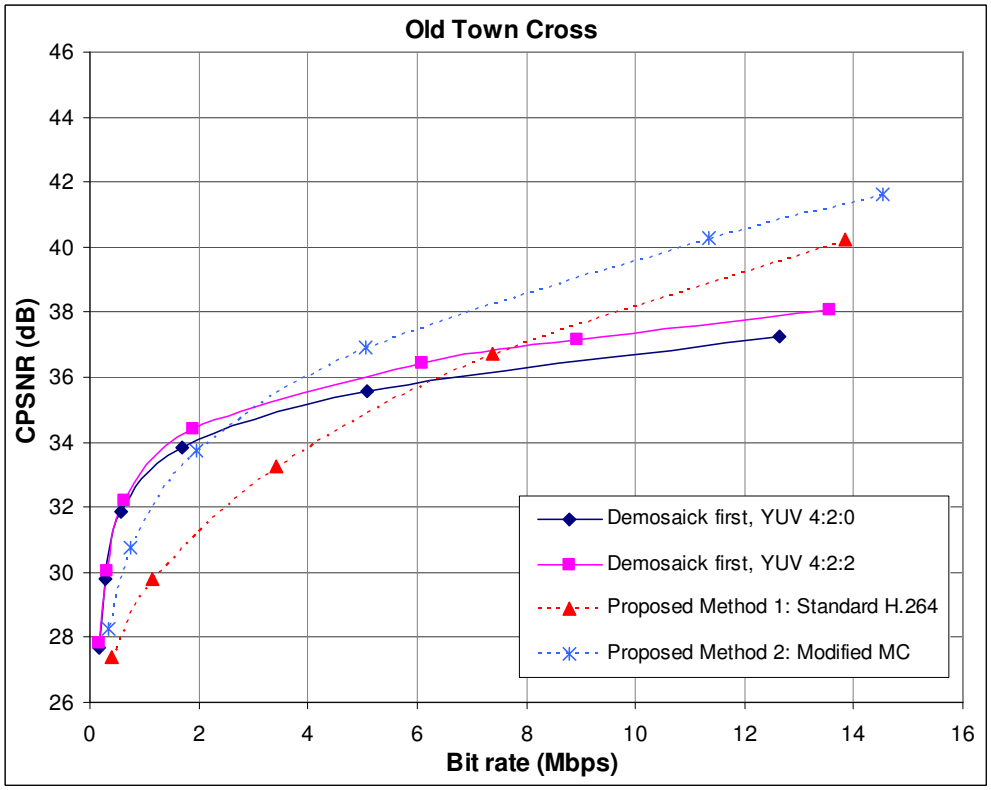


Figure 4.8: Plots of CPSNR (measured after compression and demosaicking) vs. bit rate.

Both proposed methods outperform the conventional demosaick-first approach at high bit-rates. This is primarily because the proposed methods compress data one third the raw size of the demosaick first approach. The proposed methods do not perform well at low quality levels. Highly compressing the CFA data removes detail necessary for demosaicking, and may introduce blocking artefacts which are interpreted as edges in the demosaicking process. Consequently demosaicking does not work well on highly compressed data. This problem obviously does not arise when demosaicking is performed prior to compression.

4.4.4 Complexity Comparison

In the conventional demosaick-first approach, a video of size *MxN* is compressed, usually in YUV 4:2:0 format, resulting in 1.5*MN* total samples. In either of our proposed methods, a video of size *(M/2)xN* is compressed in 4:2:2 format, which requires *MN* samples. Hence our proposed methods require two-thirds the number of samples to be compressed as the demosaick-first approach. Although the input video size is smaller in our proposed methods, the video will have more detail (high-frequency image content) as each frame has effectively been downsampled by a factor of two in the vertical direction.

The most computationally expensive operation of an H.264 video encoder is motion estimation (ME), which typically accounts for about 65% of the encoding time [41]. The complexity of ME varies greatly based on the algorithm used. If a full-search is used, the complexity of ME in either of our proposed methods will be about half that of the demosaick-first approach, since the luma channel has half the size. If a fast, content adaptive ME method is used, the ME complexity reduction obtained by using our

methods will be variable. Other significant functions that contribute to H.264 encoder complexity are intra prediction, interpolation, transform and quantization, loop filtering and entropy encoding [41]. With the exception of interpolation, the computational complexity of these functions varies depending on the video content. Generally video with more detail requires more computations. For these functions, our proposed methods will require fewer computations than the demosaick-first approach, but not by a factor of two-thirds since the smaller video will contain more detail.

The number of calculations for interpolation varies in our two methods. In Method 1 (standard H.264), luma interpolation requires half the number of computations of the demosaick-first approach, and chroma interpolation requires the same number of operations, due to the relative channel sizes. In Method 2 (Modified MC), the luma interpolation filter is applied to all of the red, green and blue channels. Therefore the same number of luma interpolation calculations are required as in the demosaick-first approach, because the combined size of the R, G, and B channels is the same as the size of the luma channel in the demosaick-first approach. However, no chroma interpolation calculations are required in Method 2. Therefore fewer interpolations operations are required in both proposed methods than in the demosaick-first approach.

In Method 1 (using standard H.264), demosaicking does not have to be performed at the encoder, which saves some computations relative to the demosaick-first approach. In Method 2, demosaicking is performed at the encoder, so the same number of demosaicking calculations are required as in the demosaick-first approach.

The most time consuming operations in H.264 video decoding are loop filtering, interpolation, inverse transform and quantization, and entropy decoding [42]. With the exception of interpolation, the amount of computations required for these operations is highly content and bit rate dependent. In general, videos with more detail require more computations. Hence the amount of computations required for the loop filter, inverse quantization and entropy decoding will be lower in the proposed method due to the smaller frame size, but not by a factor of two-thirds because the video will have more detail. Our proposed methods will require two-thirds the number of inverse transform calculations, because the amount of calculations required to perform the inverse transform is not content dependent. The relative number of interpolation calculations is the same at the encoder and decoder; so our proposed methods require fewer interpolation calculations than the demosaick-first approach at the decoder as well. In either of our proposed methods, the decoder will have to perform demosaicking, which it would not in the demosaick-first approach. The decoder complexity of our proposed methods relative to the demosaick-first approach will depend on the video coding options used and the demosaicking algorithm.

On average, H.264 decoding of a QCIF sized frame (176x144) requires 30-40 million RISC operations [43], which corresponds to about 1200-1600 operations per pixel. By comparison, the Laplacian demosaicking method in [6] requires about 60 operations per pixel (the exact number will be dependent on the hardware and software implementation details). Hence, if a computationally efficient demosaicking method is used, demosaicking will only take a small fraction, around 4-5%, of the decoding computations. This means that both proposed methods will have lower decoder

complexity than the demosaick-first approach, due to the smaller frame size of the encoded video.

4.4.5 Comparison Against Method in [19]

In order to compare our proposed methods against the method in [19], we use two standard test videos, foreman and carphone, for which results are presented in [19]. These videos are of QCIF resolution (176x144) and sampled at 30 frames/sec. Both videos were compressed at three quality levels and the resulting bit-rates are summarized in Table 4.1. The results for the method in [19] are taken directly from that paper. Both of our methods give far lower bit rates than the method in [19]. The bit rate reductions achieved range from 68% to 83% for our first method using standard H.264, and 76% to 90% for our second method using modified MC. Because are methods are based on H.264, which is a highly optimized, efficient video coding standard, they achieve much better compression than the custom built method in [19].

Video	CPSNR (dB)	Bit-rate (Kbps)		
		Method in [19]	Proposed Method 1: Standard Encoder	Proposed Method 2: Modified MC
Foreman	29.4	1360	246	182
	32.5	2080	478	350
	36.0	2780	882	664
Carphone	27.2	812	100	90
	30.0	1154	194	166
	34.5	1850	458	370

Table 4.1: Bit Rate Comparison of our proposed methods with the method in [19].

4.5 Conclusions

In this chapter, we have proposed two methods for compressing Bayer pattern CFA video prior to demosaicking. Our first method involves separating the CFA data into arrays of green, blue and red samples and compressing in 4:2:2 sampling with standard H.264. In our second method, a modified motion compensation scheme is also used, where demosaicking is performed on the references frames in the encoder and decoder in order to alleviate problems due to aliasing in the CFA data. Both proposed methods give better compression efficiency than the conventional demosaick-first approach at high bit-rates, and much better performance than the only previously proposed method for compressing CFA video prior to demosaicking.

5 Conclusions and Future Work

5.1 Conclusions

In this thesis, two means of jointly optimizing demosaicking and compression in single sensor cameras are investigated: 1) Creating a demosaicking algorithm that directly produces an image in the format used for compression (YCbCr 4:2:0), and 2) Compressing CFA video data prior to demosaicking, taking advantage of the smaller raw data size before demosaicking has been performed.

In Chapter 3, a new demosaicking method is proposed which directly generates an YCbCr 4:2:0 output image. This allows the image to be directly compressed after demosaicking, avoiding the need for a separate stage where the image is converted from RGB to YCbCr space. The proposed method provides better image quality than fast RGB based demosaicking methods and has lower computational complexity.

In Chapter 4, two methods for compressing colour filter array videos prior to demosaicking are proposed. Our first method uses standard H.264, and involves separating the CFA data into arrays of green, blue and red which are compressed in 4:2:2 sampling mode. Our second method also uses GBR 4:2:2 sampling, but with a modified motion compensation scheme where demosaicking is performed on reference frames. This alleviates problems due to aliasing in the CFA data. Both proposed methods given better compression efficiency than the standard demosaick-first approach at high bit-rates, and give far better performance than the only other method for directly compressing CFA video.

5.2 Future Work

Our demosaicking method that produces YCbCr 4:2:0 output works strictly on still images. If it were applied to a video sequence, the demosaicking would be done on each frame independently, only using information from the current frame. When performing demosaicking on a video, additional information is available from other frames. The demosaicking method could be altered to take into account temporal correlation within a sequence, which may improve performance.

In our work on compressing CFA video prior to demosaicking, we have relied on existing demosaicking methods for producing the final RGB video after demosaicking and compression. Instead, the demosaicking could use knowledge of how the data was compressed. Given knowledge of how a video has been degraded in the compression process, a demosaicking method could be designed to produce the highest quality image possible given the amount of compression that has been applied.

Bibliography

[1] ITU-T and ISO/IEC JTC1, "Digital Compression and Coding of Continuous-Tone Still Images," ISO/IEC 10918-1 – ITU Recommendation T.81 (JPEG), September 1992.

[2] ITU-T, "Video coding for low bitrate communication," ITU-T Recommendation H.263; version 1, November 1995; version 2, January 1998.

[3] MPEG-2: ISO/IEC JTC1/SC29/WG11 and ITU-T, "Revised text for ITU-T recommendation H.262—ISO/IEC 13 818-2: Information technology-generic coding of moving pictures and associated audio information: Video," ISO/IEC and ITU-T, Genf, Switzerland, 1995.

[4] ITU-T, "Draft ITU-T Recommendation and Final Draft International Standard of Joint Video Specification," ITU-T Rec. H.264/ISO/IEC 14496-10 AVC, Mar. 2003.

[5] B.K. Gunturk, J. Glotzbach, Y. Altunbasak, R.M. Mersereau, and R.W. Schafer, "Demosaicking: Color filter array interpolation," IEEE Signal Processing Mag., vol. 22, no. 1, pp. 44–54, 2005.

[6] Hamilton, J. F. and Adams, J. E., "Adaptive color plane interpolation in single sensor color electronic camera," U.S. Patent 5,629,734, May 1997.

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

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

[9] S. C. Pei and I. K. Tam, "Effective color interpolation in CCD color filter arrays using signal correlation," IEEE Trans. Circuits Syst. Video Technol., vol. 13, no. 6, pp. 503–513, Jun. 2003.

[10] B. K. Gunturk, Y. Altunbasak, and R. M. Mersereau, "Color plane interpolation using alternating projections," IEEE Transcations on Image Processing, vol. 11, no. 9, 2002.

[11] J. Go, K. Sohn, and C. Lee, "Interpolation using neural networks for digital still cameras," IEEE Transcations on Consumer Electronics., vol. 46, no. 3, pp. 610–616, Aug. 2000.

[12] J. Mukherjee, R. Parthasarathi, and S. Goyal, “Markov random field processing for color demosaicing,” Pattern Recognition Letters, vol. 22, no. 3-4, pp. 339–351, Mar. 2001.

[13] X. Wu and N. Zhang, “Primary-consistent soft-decision color demosaicking for digital cameras,” IEEE Trans. on Image Processing, vol. 13, pp. 1263–1274, 2004.

[14] X. Wu and L. Zhang, “Color Video Demosaicking via Motion Estimation and Data Fusing,” IEEE Transactions on Circuits and Systems for Video Technology, vol. 16, no. 2, pp. 231–240, Feb. 2006.

[15] X. Wu and N. Zhang, “Improvement of Color Video Demosaicking in Temporal Domain,” IEEE Transactions on Image Processing, vol. 15, no. 20, pp. 3138–3151, Oct. 2006.

[16] S.Y. Lee and A. Ortega, “A novel approach of image compression in digital cameras with a Bayer color filter array," IEEE Int. Conf. Image Processing 2001, vol. 3, pp. 482-485, Oct 2001.

[17] C.C. Koh, J. Mukherjee, S.K. Mitra. “New efficient methods of image compression in digital cameras with color filter array”. IEEE Transactions on Consumer Electronics 2003; 49(4):1448–56.

[18] N.X. Lian, L. Chang, V. Zagorodnov, Y.P. Tan, "Reversing Demosaicking and Compression in Color Filter Array Image Processing: Performance Analysis and Modeling," IEEE Transactions on Image Processing, vol.15, no.11, pp. 3261-3278, Nov. 2006.

[19] F. Gastaldi, C. C. Koh, M. Carli, A. Neri and S. K. Mitra, “Compression of videos captured via bayer patterned color filter arrays,” Proc. 13th European Signal Processing Conference (EUSIPCO-2005), Antalya, Turkey.

[20] T. Toi, and M. Ohta, “A subband coding technique for image compression in single CCD cameras with bayer color filter arrays,” *IEEE Transactions on Consumer Electronics*, vol .45, no. 1, pp. 176–80, 1999.

[21] A. Bruna, A. Buemi, F. Vella, A. Vitali, “A Low Cost Algorithm For CFA Data Compression”. Int Conf on Consumer Electronics, 2006 Digest of Technical Papers, pp. 385-386, Jan. 2006.

[22] J. Adams, K. Parsulski, and K. Spaulding, “Color processing in digital cameras,” IEEE Micro, pp. 20–29, Nov.–Dec. 1998.

[23] K.A. Parulski, “Color Filter Arrays and Processing Alternatives for One-Chip Cameras,” IEEE Trans. Electron Devices, Vol. ED-32, No. 8, Aug. 1985, pp. 1381-1389.

[24] R. Lukac, and K. N. Plataniotis, “Color Filter Arrays: Design and Performance Analysis,” IEEE Transactions on Consumer Electronics, vol. 51, no. 4, pp. 1260-1267, Nov. 2005.

[25] B.E. Bayer, “Color Imaging Array,” U.S. Patent 3,971,065.

[26] K. Barnard, V. Cardei, and B. Funt, “A comparison of computational color constancy algorithms - part i: Methodology and experiments with synthesized data," IEEE Trans on Image Proc, vol. 11, no. 9, pp. 972-983, 2002.

[27] IEC 61966-2-1 (1999-10), Multimedia systems and equipment - Colour measurement and management – Part 2-1: Colour management - Default RGB colour space - sRGB, International Electrotechnical Commission, www.srgb.com, 1999.

[28] R. Ramanath, W.E. Snyder, Y. Yoo, M.S. Drew, “Color image processing pipeline,” IEEE Signal Processing Mag., vol. 22, no. 1, pp. 34–43, 2005.

[29] E. Hamilton, “JPEG File Interchange Format, Version 1.02,” Sept., 1992. Available: http://www.w3.org/Graphics/JPEG/jfif.pdf.

[30] ISO/IEC JTC1, “Information Technology – JPEG 2000 image coding system – Part 1: Core coding system,” ISO/IEC 15444-1, 2000.

[31] Mukherjee, J., Lang, M. K., and Mitra, S. K. 2005. Demosaicing of images obtained from single-chip imaging sensors in YUV color space. Pattern Recognition Letters. 26, 7, pp. 985-997, May 2005.

[32] J.E. Adams, Jr., “Design of Practical Color Filter Array Interpolation Algorithms for Digital Cameras,” Proc. SPIE, Vol. 3028, SPIE, 1997, pp. 117-125.

[33] T. Wedi and H.G. Musmann, “Motion- and aliasing-compensated prediction for hybrid video coding,” IEEE Transactions on Circuits and Systems for Video Technology, vol. 13, pp. 577–587, July 2003.

[34] T. Wedi, “Adaptive Interpolation Filters and High-Resolution Displacements for Video Coding,” IEEE Trans. Circuits Syst. Video Technol., vol. 16, no. 4, pp. 484 - 491, Apr. 2006.

[35] D. Alleysson and S. Süsstrunk, “Aliasing in Digital Cameras”, SPIE EI Newsletter, Special Issue on Smart Image Acquisition and Processing, Vol. 14, Nr. 12, pp. 1,8, 2004.

[36] H. J. Trussell and R. E. Hartwig, “Mathematics for demosaicking,” IEEE Trans. Image Processing, vol. 11, pp. 485–492, Apr. 2002.

[37] J.W. Glotzbach, R.W. Schafer, and K. Illgner, “A method of color filter array interpolation with alias cancellation properties,” in Proc. IEEE Int. Conf. Image Processing, vol. 1, 2001, pp. 141–144.

[38] N. Lian, L. Chang and Y.P Tan, “Improved color filter array demosaicking by accurate luminance estimation," IEEE Int. Conf. Image Processing 2005, vol. 1, pp. 41-44, Sept 2005.

[39] Joint Video Team of ITU-T and ISO/IEC, “Draft Text of H.264/AVC Fidelity Range Extensions Amendment”, Doc. JVT-L047, Sept. 2004.

[40] L. Haglund, “The SVT High Definition Multi Format Test Set,” Available: ftp://vqeg.its.bldrdoc.gov/HDTV/SVT_MultiFormat/

[41] A. Hallapuro and M. Karczewicz, “Complexity Analysis of H.26L,” ITU-T SG16 Doc. VCEG-M50, 2001.

[42] V. Lappalainen, A. Hallapuro, T.D. Hamalainen, “Complexity of optimized H.26L video decoder Implementation,” IEEE Transactions on Circuits and Systems for Video Technology, pp.717–725, vol. 13 , no. 7 , July 2003.

[43] C. Xu, T.M. Le, and T.T. Tay, "H.264/AVC Codec: Instruction-Level Complexity Analysis", Proc. IASTED International Conference on Internet and Multimedia Systems, and Applications (IMSA 2005), Honolulu, Hawaii, Aug. 2005.

Appendix A – List of Acronyms

AWB	auto white balance
CCD	charge-coupled device
CFA	Colour filter array
CIE	Commission Internationale de l'Eclairage (International Commission on Illumination)
CMOS	Complementary Metal Oxide Semiconductor
CPSNR	Composite peak signal-to-noise ratio
DCT	Discrete cosine transform
FIR	Finite impulse response
JPEG	Joint Photographic Experts Group
MC	Motion compensation
ME	Motion estimation
MPEG	Moving Picture Experts Group
PSNR	Peak signal-to-noise ratio
sRGB	standard red, green, blue (colour space)