CFA Interpolation Detection

Leszek Świrski

November 30, 2009

1 Introduction

1.1 Colour Image Sensors

To capture a still photograph, digital cameras have to employ a technology converting incoming photons into digital signals. The image sensor used for this is most often a charge-coupled device (CCD) or a complementary metal-oxide-semiconductor (CMOS) chip, however regardless of the technology, the effect is the same: light strikes an array of photosensors, where energy proportional to that of the light is converted into per-pixel voltages, and these voltages can be interpreted as digital pixel information.

The problem when faced with colour imaging using the majority of such image sensors is that the photosensors in the array can only detect the intensity of light, and not its wavelength; hence, the output image can only be greyscale. Three major methods exist to create colour images:

- **Colour filter arrays (CFA)**, which use an array of filters which filter the wavelengths of light that can reach each sensor.
- **Foveon X3** sensors, which use layered CMOS chips and rely on the wavelength-depended absorption of silicon
- **3CCD**, which uses three separate CCD sensors, and separates the incoming light into three beams of various wavelengths, each directed onto a separate CCD.

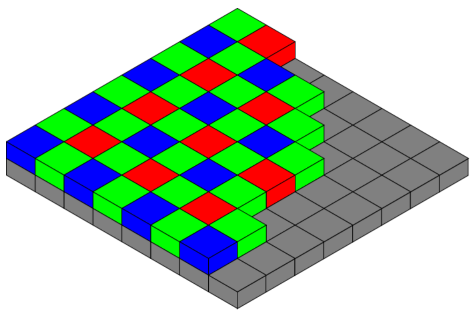


Figure 1: *A Bayer pattern CFA overlaying a photosensor array.*

Of these, the CFA is by far the most popular in still digital cameras, and is therefore the only colour separation technique considered in this essay. The most common CFA pattern is the Bayer pattern,

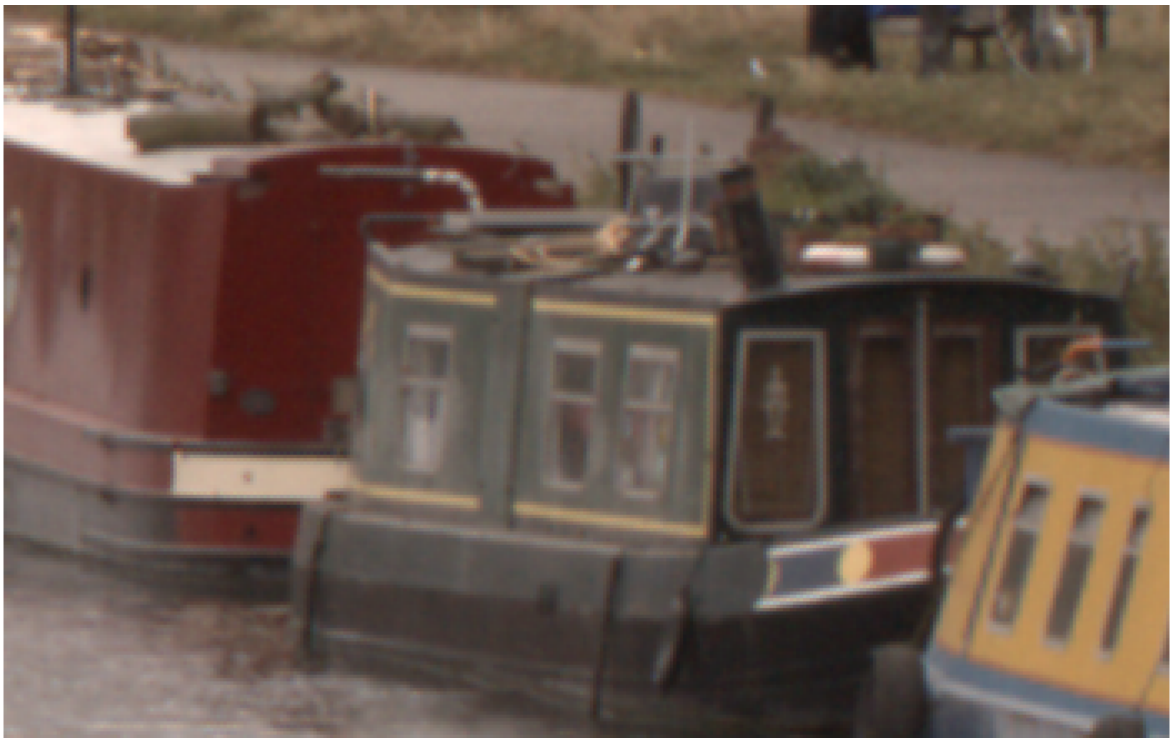


Figure 2: *An example image we want to receive by taking a photograph*

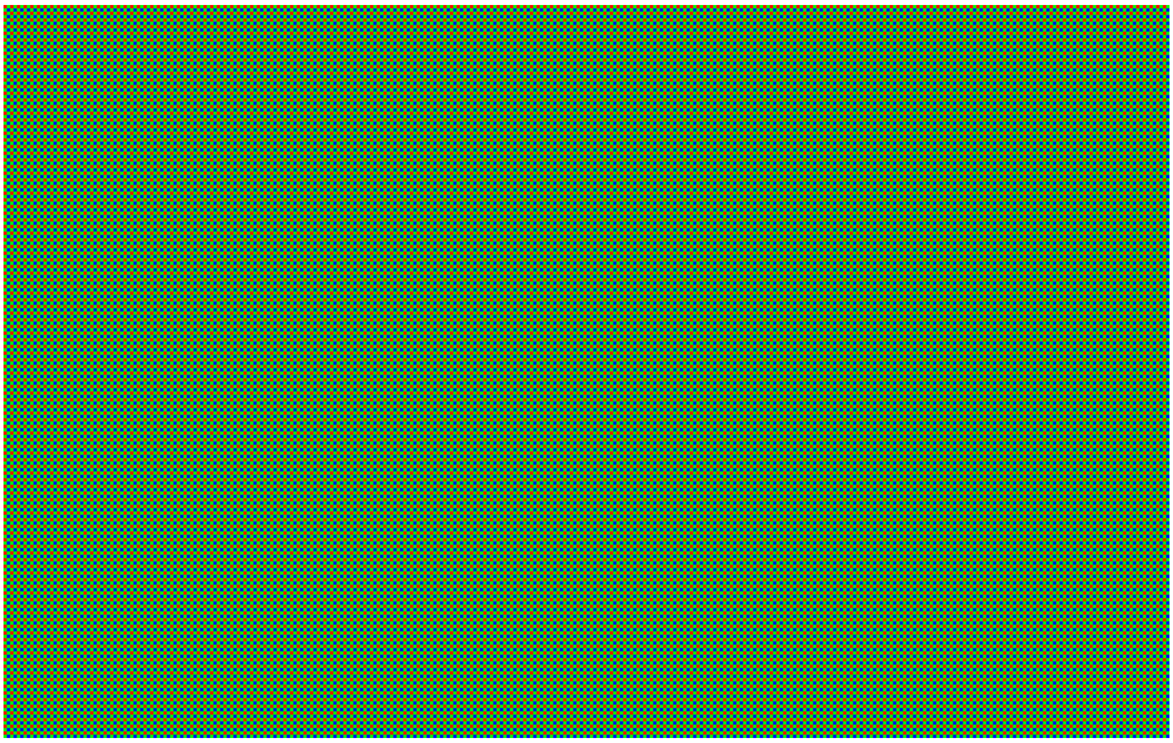


Figure 3: *The colour filter array that overlays the photosensor array*

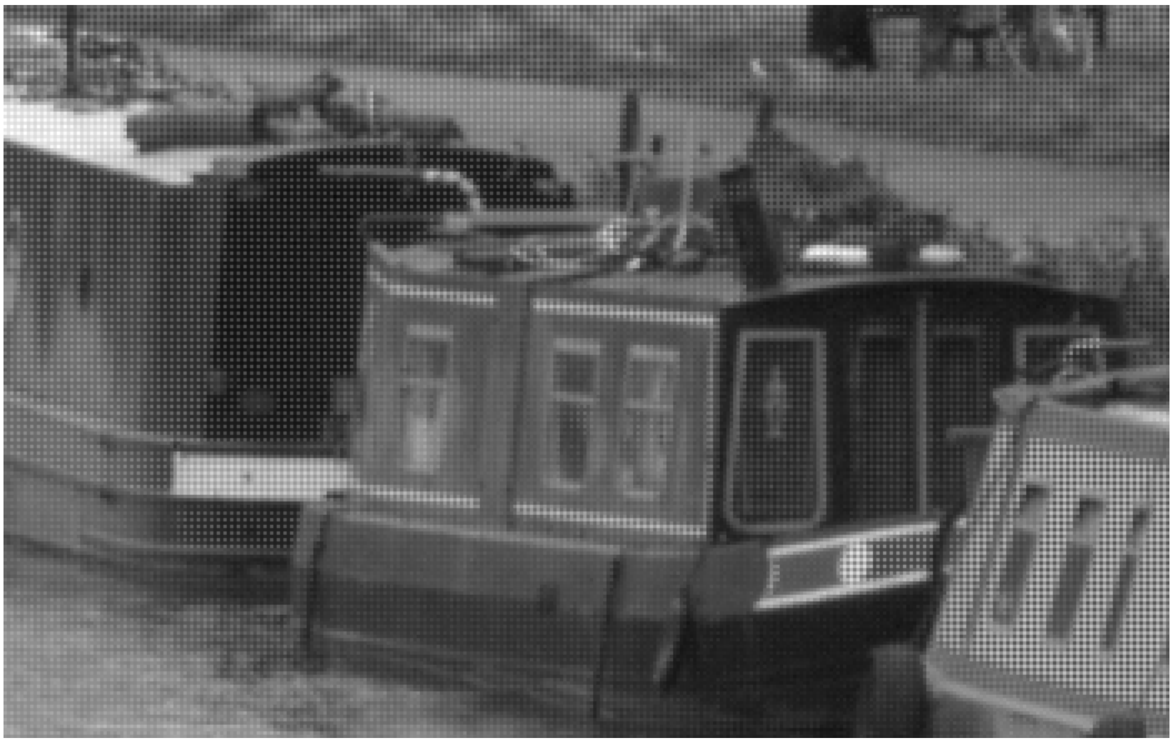


Figure 4: *The data received from the image sensor*

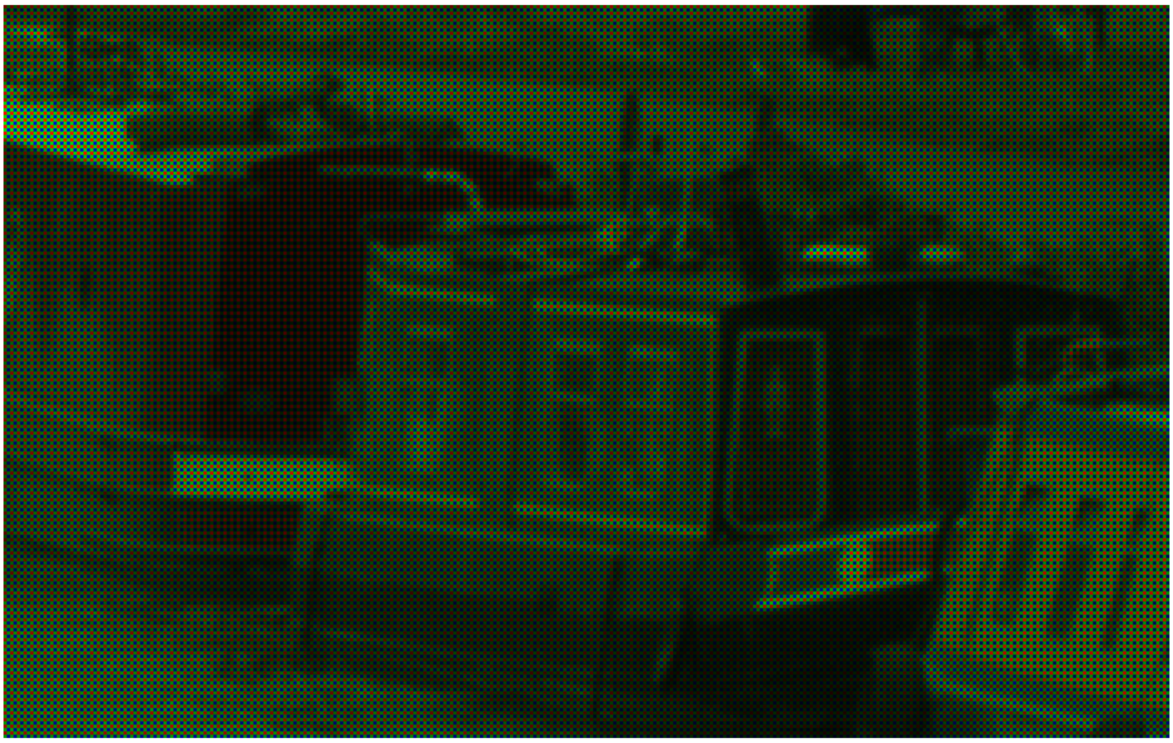


Figure 5: *A coloured version of the sensor data, coloured by the corresponding CFA filter colour*

which consists of repeating four pixel squares. Each square is coloured with two green pixels on the diagonal, and one each of red and blue pixels (figure 1). The ratio of two green pixels to one red and one blue was chosen because the human eye is much more sensitive to green light.

The problem with such a filter array is that we cannot interpret the sensor data as received. If we did, in the resulting image each pixel would only be one of red, green or blue, whereas for a full colour image we want each pixel to be a combination of these colours. This problem is demonstrated in the following figures: figure 2 show what we want the image taken to be, figure 3 shows the filter applied to the incoming light, and figures 4 and 5 demonstrated what the output of the image sensor actually is.

1.2 Interpolation

Since our target is a full-colour, full-size image, digital cameras have to interpolate the pixel values across each of the three colour channels. In this section, I describe several such interpolation algorithms. There are many different interpolation models used in commercial digital cameras, and an exhaustive list would be unfeasible—however most are similar in effect or principle to the following (as described in Popescu and Farid[3]). When describing the algorithms, let $S_{x,y}$ be the CFA image[1], and let $\hat{R}_{x,y}$, $\hat{G}_{x,y}$, $\hat{B}_{x,y}$ be the red, green and blue channels constructed from $S_{x,y}$, where

$$\hat{R}_{x,y} = \begin{cases} S_{x,y} & \text{if } (x \equiv 1 \mod 2) \wedge (y \equiv 1 \mod 2) \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

$$\hat{G}_{x,y} = \begin{cases} S_{x,y} & \text{if } (x \equiv 1 \mod 2) \oplus (y \equiv 1 \mod 2) \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

$$\hat{B}_{x,y} = \begin{cases} S_{x,y} & \text{if } (x \equiv 0 \mod 2) \wedge (y \equiv 0 \mod 2) \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

Then, let $R_{x,y}$, $G_{x,y}$, $B_{x,y}$ be the colour channels of the target, interpolated image. It is these three that we want to calculate. In all the below cases, my MATLAB implementation can be found in appendix A, and example outputs of the MATLAB implementations can be found in appendix B.

1.2.1 Nearest-Neighbour Interpolation

In nearest-neighbour interpolation, the empty pixel values are filled in with their nearest neighbour's value. An example of such an interpolation is:

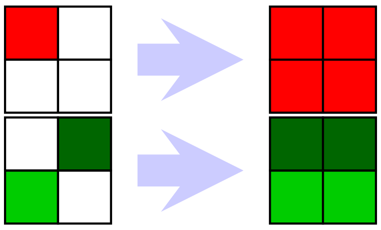


Figure 6: *Nearest-neighbour interpolation of the red and green channels*

[1]Note that x increases in the vertical direction, and y in the horizontal. This is due to these being cells in matrix notation, rather than image co-ordinates.

$$R_{2i,2j} = R_{2i+1,2j} = R_{2i,2j+1} = R_{2i+1,2j+1} = \hat{R}_{2i+1,2j+1} \tag{4}$$

$$G_{2i,2j} = G_{2i,2j+1} = \hat{G}_{2i,2j+1} \tag{5}$$

$$G_{2i+1,2j} = G_{2i+1,2j+1} = \hat{G}_{2i+1,2j} \tag{6}$$

$$B_{2i,2j} = B_{2i+1,2j} = B_{2i,2j+1} = B_{2i+1,2j+1} = \hat{B}_{2i,2j} \tag{7}$$

Figure 6 demonstrates the above algorithm visually.

Such an interpolation causes unpleasant blocky artefacts, and is not generally used unless a very high-speed implementation is necessary.

1.2.2 Bilinear Interpolation

Bilinear interpolation estimates the value of an empty pixel as the average of the values of its non-empty neighbours: more precisely, it performs a convolution of the matrix representing each channel with an bilinear interpolation matrix. Therefore, we have:

$$R = \hat{R} * \frac{1}{4}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \tag{8}$$

$$G = \hat{G} * \frac{1}{4}\begin{bmatrix} 0 & 1 & 0 \\ 1 & 4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \tag{9}$$

$$B = \hat{B} * \frac{1}{4}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \tag{10}$$

Figure 7 demonstrates the above algorithm visually.

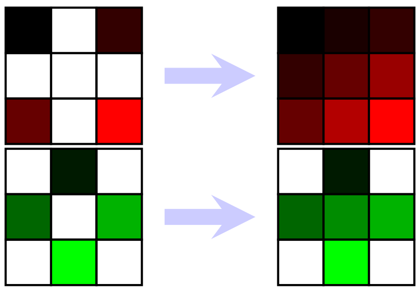


Figure 7: *Bilinear interpolation of the red and green channels*

Other forms of linear interpolation using convolutions also exist, including bicubic and biquadratic, however bilinear is the most common.

1.2.3 Smooth Hue Transition

In bilinear interpolation, there are visible chrominance aliases in areas with detail, where the interpolation of colour information has caused the detail to be slightly shifted between colour channels. To attempt to eliminate such chrominance distortions, one can make the assumption that in natural images colour hue varies smoothly. If we define two 'hues' as the ratios of each chrominance component to the luminance, and assuming we have interpolated the luminance already, we can interpolate these hue values rather than the chrominance values to reduce colour aliasing.

In the smooth hue transition algorithm, we approximate luminance with the green channel, with the red and blue as chrominance components. This is good enough, as half the pixels captured are green, and green is the major component in luminance (approximately 0.6). The green channel is bilinearly interpolated, and then used in the hue interpolation. The red and blue channels have their ration with the green channel interpolated, and are then pointwise multiplied by the green channel.

$$G = \hat{G} * \frac{1}{4}\begin{bmatrix} 0 & 1 & 0 \\ 1 & 4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \tag{11}$$

$$R = G \overset{\text{pt.wise}}{\times} \left(\left(\hat{R} \overset{\text{pt.wise}}{\div} G \right) * \frac{1}{4}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \right) \tag{12}$$

$$B = G \overset{\text{pt.wise}}{\times} \left(\left(\hat{B} \overset{\text{pt.wise}}{\div} G \right) * \frac{1}{4}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \right) \tag{13}$$

1.2.4 Median Filtering

All of the above methods contain ‘speckles’, or noise, near edges. This is caused, as above, by shifted edge information, however discontinuities in hue mean that the smooth hue transition does not reduce these artefacts. Since this noise is caused not by the individual channels themselves, but by differences between them, it should be possible to decrease this noise by applying a despeckling filter to the pairwise differences of the colour channels, and incorporating this despeckled difference back into the original. This is what median filtering does.

Firstly, all the channels are interpolated using another interpolation scheme, usually simply bilinear. Then, their pairwise differences are median filtered. The resulting colour channels are constructed pixel-by-pixel from the CFA, where for each pixel in the CFA image, the corresponding median filtered difference pixel is added or subtracted. Say M_{rg}, M_{rb}, M_{bg} are the median filtered pairwise differences. Then, we take the example of $S_{1,0}$, which is the green pixel $\hat{G}_{1,0}$. In this case:

$$R_{1,0} = S_{1,0} + (M_{rg})_{1,0} \tag{14}$$

$$G_{1,0} = S_{1,0} \tag{15}$$

$$B_{1,0} = S_{1,0} - (M_{gb})_{1,0} \tag{16}$$

The resulting image has fewer speckles on edges, and the image could be median filtered again to further remove speckles. Indeed, `dcraw` (an open-source program dealing with, among other things, CFA interpolation) has a separate option of how many times to median filter its interpolated result.

1.2.5 Gradient-Based interpolation

A problem with all the algorithms described above is that they all interpolate across edges: this means that edges are less defined in the final image, and also causes artefacts along edges. The gradient-based interpolation scheme aims to avoid interpolating across edges by adaptively interpolating the green channel based on local horizontal and vertical gradients. A second derivative can be taken in both directions, as a second derivative near zero is indicative of a locally linear gradient, whereas a non-zero second derivative is indicative of local gradient change. Hence, we can compare the absolute values of horizontal and vertical second derivatives. Where there is local linearity in a certain direction but not in the other, it is a safe assumption that we should interpolate linearly only in the locally linear direction. This means that, for example, if the absolute horizontal second derivative is smaller than the vertical, we should only interpolate horizontally.

In gradient-based interpolation, the horizontal and vertical second derivatives are approximated using central difference. Hence, we get H and V as the derivative estimators using:

$$H_{x,y} = \left| \frac{S_{x,y-2} + S_{x,y+2}}{2} - S_{x,y} \right| \tag{17}$$

$$V_{x,y} = \left| \frac{S_{x-2,y} + S_{x+2,y}}{2} - S_{x,y} \right| \tag{18}$$

Notice that, for each possible (x, y), all three S values will be from the same colour pixel. The green channel is then adaptively interpolated as:

$$G_{x,y} = \begin{cases} \hat{G}_{x,y} & \text{if } S_{x,y} \text{ is green} \\ \frac{\hat{G}_{x,y-1}+\hat{G}_{x,y+1}}{2} & \text{if } H_{x,y} < V_{x,y} \\ \frac{\hat{G}_{x-1,y}+\hat{G}_{x+1,y}}{2} & \text{if } H_{x,y} > V_{x,y} \\ \frac{\hat{G}_{x,y-1}+\hat{G}_{x,y+1}+\hat{G}_{x-1,y}+\hat{G}_{x+1,y}}{4} & \text{if } H_{x,y} = V_{x,y} \end{cases} \tag{19}$$

To interpolate the red and blue channels, we take advantage of the fact that human perception notices loss of edge definition in chrominance much less than in luminance. If we once again assume that G is the luminance, then we can define the two chrominance channels as $\hat{R} - G$ and $\hat{B} - G$, not unlike the definition of the YUV colour space. These chrominance channels can now be interpolated in the usual way, without worrying about gradients, as the interpolation across edges will be imperceptible. The resulting interpolated chrominance can be added back to G to recover the colour channels. Hence:

$$R = G + \left(\left(\hat{R} - G \right) * \frac{1}{4} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \right) \tag{20}$$

$$B = G + \left(\left(\hat{B} - G \right) * \frac{1}{4} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \right) \tag{21}$$

1.2.6 Other methods

There are many more methods of interpolation than those described above, however most share the same techniques and principles: identifying edges/gradients in the image, operating on the green channel as if it were luminance, and use linear interpolation were data can be assumed to be linear. Popescu and Farid[3] describe two more algorithms, however both are complications of the gradient-based method, attempting to achieve better treatment of edges in the image.

2 Interpolation Detection

The real aim of both Popescu and Farid[3] and Gallagher and Chen[1] is to detect if an image has undergone a similar interpolation to one of the above. This is useful for several reasons. Firstly, it can prove an image to be digitally tampered, if it does not exhibit these interpolations, and offers evidence to verify its authenticity. If one could identify the exact interpolation used, this could provide evidence about the camera used to take the photograph. Images could also be analysed locally, to identify forged regions in an otherwise untampered photograph. In addition to these forensic applications, such analysis could be used to automatically distinguish photorealistic computer generated (PRCG) images from real images.

2.1 Methods

The principle behind interpolation detection is that interpolation will create a correlation between pixels. The majority of CFA interpolation algorithms are regular and approximately linear, in particular for the green channel which is often first interpolated linearly as a basis for the remainder of the algorithm. Therefore, if an algorithm can determine some regular correlation between pixels, then it is likely that the image is the result of an interpolation.

2.1.1 EM Algorithm

Popescu and Farid[3] use the Expectation-Maximisation (EM) algorithm to attempt to simultaneously find which pixels are correlated, and the parameters of their correlation—that is, what type of interpolation is used. The EM algorithm is an two step iterative algorithm. In each iteration, it estimates the probability for each pixel of being correlated by a model M, and then estimates the parameters of M using minimisation based on these probabilities.

A very strong advantage of the EM algorithm creates a pairwise separable 24-D parameter space for the model, where this separability allows one to determine the interpolation algorithm used. However, its convergence can be slow.

2.1.2 Gallagher and Chen Algorithm

Gallagher and Chen[1] propose a different method of detection. They notice that correlation causes a decrease in variance—indeed, for a bilinear interpolation, the non-interpolated green pixels have four times the variance of interpolated green pixels (figure 8). Therefore one can detect regular correlation by finding periodicity in the variance.

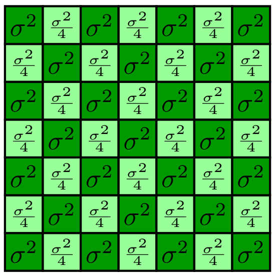

σ^2 $\frac{\sigma^2}{4}$ σ^2 $\frac{\sigma^2}{4}$ σ^2 $\frac{\sigma^2}{4}$ σ^2
$\frac{\sigma^2}{4}$ σ^2 $\frac{\sigma^2}{4}$ σ^2 $\frac{\sigma^2}{4}$ σ^2 $\frac{\sigma^2}{4}$
σ^2 $\frac{\sigma^2}{4}$ σ^2 $\frac{\sigma^2}{4}$ σ^2 $\frac{\sigma^2}{4}$ σ^2
$\frac{\sigma^2}{4}$ σ^2 $\frac{\sigma^2}{4}$ σ^2 $\frac{\sigma^2}{4}$ σ^2 $\frac{\sigma^2}{4}$
σ^2 $\frac{\sigma^2}{4}$ σ^2 $\frac{\sigma^2}{4}$ σ^2 $\frac{\sigma^2}{4}$ σ^2
$\frac{\sigma^2}{4}$ σ^2 $\frac{\sigma^2}{4}$ σ^2 $\frac{\sigma^2}{4}$ σ^2 $\frac{\sigma^2}{4}$
σ^2 $\frac{\sigma^2}{4}$ σ^2 $\frac{\sigma^2}{4}$ σ^2 $\frac{\sigma^2}{4}$ σ^2

Figure 8: *The variance of interpolated green pixels is a quarter of the variance of the non-interpolated pixels*

In their algorithm, Gallagher and Chen only look at the green channel, though they state that other channels could also be considered. First, they convolve the image with a highpass filter to remove low-frequency information (which is irrelevant). The high-pass filter has the interesting property that it will remove any pixels which have been bilinearly interpolated using the convolution described in section 1.2.2.

To calculate variance, Maximum Likelihood Estimation (MLE) is used on the anti-diagonals of the image, assuming that each pixel on the diagonal is drawn from a normal distribution, and that variance along the anti-diagonals is constant. Diagonals are used because it is clear from the pattern that the variance across diagonals varies periodically. It is not clear why the authors choose to use the anti-diagonals rather than the diagonals, however it is likely that this is only to make the mathematical representation simpler.

Instead of actually computing the variance of each diagonal, the authors instead use a computationally simpler estimate, which is the mean of the absolute values of the diagonal. This calculates the mean absolute deviation, which is a measure of disperisity similar to the standard deviation. Therefore, a one-dimensional signal of 'variances' is formed:

$$m(d) = \frac{\sum\limits_{x+y=d} |(h * i)_{x,y}|}{N_d} \tag{22}$$

where N_d is the number of elements along the diagonal, h is the high-pass filter, and i is the image.

For an interpolated image, we expect to find periodicity in $m(d)$. To find such periodicity, the authors use a DFT, to calculate $\left|M(e^{i\omega})\right|$. This M will exhibit peaks at any frequency corresponding to periodicity in the variance; for CFA interpolation—which is by a factor of two—we expect a peak at $\omega = \pi$. The authors quantify this peak as:

$$s = \frac{\left|M(e^{i\omega})\right|_{\omega=\pi}}{\text{median}_\omega\{|M(e^{i\omega})|\}} \tag{23}$$

where normalising by the median was found to be important in differentiating between interpolation and signals with large energy across the spectrum.

I implemented this algorithm in MATLAB, with the following code:

```
I = im2double(imread(...));
G = I(:,:,2);                % Extract green channel
h = [0 1 0;1 -4 1; 0 1 0];
o = conv2(G, h, 'valid');    % High pass filter
fo = fliplr(o);              % Flip to find anti-diagonals using diag
m = [];
for d=(size(fo,2)-1):-1:(-size(fo,1)+1)
  m(size(fo,2)-d) = mean(abs(diag(fo, d)));
end
M = abs(fft(m));
k = median(M(2:end));
mid = M(floor(size(M,2)/2)+1);
s = mid/k;
```

I ran this algorithm on my bilinearly interpolated image, as well as a PRCG image, and on 256×256 sections of the bilinear and gradient-based images to demonstrate localised interpolation detection on both linear and non-linear interpolation algorithms. In each case, each the 'real' image showed a significant peak in the DFT (and significant s value), whereas the PRCG image exhibited no such peak. The images used, and their $m(d)$ and $\left|M(e^{i\omega})\right|$ plots can be found in appendix C.

2.1.3 Identifying Forged Regions

Since the above statistics apply locally as well as over the whole image, we can adapt the above algorithm to perform a local analysis (within a radius n) of every pixel, using local diagonals, a local DFT, and returning a per-pixel peak s_{xy}. The m function would change to a local pixel variance estimation:

$$m(x, y) = \frac{\sum\limits_{i=-n}^{n} |(h * i)_{x+i,y+i}|}{2n + 1} \tag{24}$$

Any pixels with a low value for s_{xy} would be likely to be parts of a forged region. With appropriate thresholding, these forged areas can be segmented, as is the case at the end of [1].

2.2 Possible Problems

There are several possible problems with the above techniques, however these prove not to be as problematic as one may expect. Although most interpolation techniques used in modern cameras are non-linear, the linearity assumptions used in the detection are still ‘good enough’, as the non-linear algorithms end up being sufficiently linear locally to leave traces of interpolation. Also, JPEG compression will interfere with the interpolation watermark, due to its own compression, which includes low-pass filtering. However, the authors of both methods described above found that the techniques degrade gracefully with increased JPEG compression, and that high-quality JPEGs have little effect on the results.

The largest problems is the CFA interpolation patterns can be synthesised, therefore these methods cannot verify authenticity. However, they can verify forgery, and they add a new method of forgery detection which makes convincing forgery more difficult.

3 CFA Pattern Synthesis

In contrast to the other two papers, Kirchner and Böhme[2] describe a method to circumvent the forgery detection described above by restoring CFA-like correlations. The naïve method would be to simply sample the forged image with a CFA filter, and reinterpolate it. However, this form of sub-sampling the image could introduce aliasing if the forged area’s spectrum was too wide. A much better method would be to find the additive tamper error (the difference between tampered and original), low-pass filter it (e.g. Gaussian blur), and add it to the image before sampling the result with a CFA filter. This would work, however it is not guaranteed to give the minimal error from the original.

To guarantee a minimum error, the authors consider this as an algebraic problem. A linear interpolation can be represented as a matrix operation on vectorised forms of the images $\mathbf{y} = \mathbf{Hx}$, and the manipulation can be treated as an additive error:

$$\mathbf{y} = \mathbf{Hx} + \epsilon \tag{25}$$

For a given manipulated image $\mathbf{y}$, Kirchner and Böhme want to find a CFA sensor value $\mathbf{x}$ such that $\|\mathbf{y} - \mathbf{Hx}\| = \|\epsilon\|$ is minimal. This is a least squares problem, with known solution using the Moore-Penrose pseudoinverse:

$$\mathbf{x} = (\mathbf{H}^T\mathbf{H})^{-1}\mathbf{H}^T\mathbf{y} \tag{26}$$

However, calculating the Moore-Penrose pseudoinverse using standard methods (e.g. singular value decomposition) quickly becomes intractable. The remainder of the paper describes an implementation which takes advantage of the sparseness of $\mathbf{H}$, however I will not deal with the implementation in this essay.

References

[1] Andrew C. Gallagher and Tsuhan Chen. Image authentication by detecting traces of demosaicing. In *Proc. CVPR WVU Workshop*, 2008.

[2] Matthias Kirchner and Rainer Böhme. Synthesis of color filter array pattern in digital images. In *Proceedings of SPIE*, volume 7254, page 72540K, 2009.

[3] Alin C. Popescu and Hany Farid. Exposing digital forgeries in color filter array interpolated images. *IEEE Transactions on Signal Processing*, 53(10):3948–3959, 2005.

A Interpolation Implementation

In each of the following implementations, the following values are predefined:

```
% Load sensor data
S = im2double(imread('BoatsSensor.png'));
% Create CFA filter for each of the three colours
Rcfa = repmat([1 0;0 0], size(S)/2);
Gcfa = repmat([0 1;1 0], size(S)/2);
Bcfa = repmat([0 0;0 1], size(S)/2);
% Split data into 'hat' variables
Rh = S .* Rcfa;
Gh = S .* Gcfa;
Bh = S .* Bcfa;
```

A.1 Nearest-Neighbour

```
R = Rh(floor([0:end-1]/2)*2+1, floor([0:end-1]/2)*2+1);
G(floor([0:end-1]/2)*2+1, :) = ...
            Gh(floor([0:end-1]/2)*2+1, floor([0:end-1]/2)*2+2);
G(floor([0:end-1]/2)*2+2, :) = ...
            Gh(floor([0:end-1]/2)*2+2, floor([0:end-1]/2)*2+1);
B = Bh(floor([0:end-1]/2)*2+2, floor([0:end-1]/2)*2+2);
```

A.2 Bilinear

```
R = conv2(Rh, [1 2 1; 2 4 2; 1 2 1]/4, 'same');
G = conv2(Gh, [0 1 0; 1 4 1; 0 1 0]/4, 'same');
B = conv2(Bh, [1 2 1; 2 4 2; 1 2 1]/4, 'same');
```

A.3 Smooth Hue Transition

```
G = conv2(Gh, [0 1 0; 1 4 1; 0 1 0]/4, 'same');
R = G .* conv2(Rh./G, [1 2 1; 2 4 2; 1 2 1]/4, 'same');
B = G .* conv2(Bh./G, [1 2 1; 2 4 2; 1 2 1]/4, 'same');
```

A.4 Median Filter

```
R = conv2(Rh, [1 2 1; 2 4 2; 1 2 1]/4, 'same');
G = conv2(Gh, [0 1 0; 1 4 1; 0 1 0]/4, 'same');
B = conv2(Bh, [1 2 1; 2 4 2; 1 2 1]/4, 'same');
Mrg = R - G;
Mrb = R - B;
Mgb = G - B;
R = S + Mrg.*Gcfa + Mrb.*Bcfa;
G = S - Mrg.*Rcfa + Mgb.*Bcfa;
B = S - Mrb.*Rcfa - Mgb.*Gcfa;
```

A.5 Gradient-Based

```
H = abs((S(:, [1 1 1:end-2]) + S(:, [3:end end end]))/2 - S);
V = abs((S([1 1 1:end-2], :) + S([3:end end end], :))/2 - S);

G = Gh + (Rcfa+Bcfa).*(...
  (H < V) .* ( (Gh(:, [1 1:end-1]) + Gh(:, [2:end end ]))/2 ) + ...
  (H > V) .* ( (Gh([1 1:end-1], :) + Gh([2:end end ], :))/2 ) + ...
  (H == V) .* ( (Gh(:, [1 1:end-1]) + Gh(:, [2:end end ]) ...
               + Gh([1 1:end-1], :) + Gh([2:end end ], :))/4 ) ...
);

R = G + conv2(Rh - Rcfa.*G, [1 2 1; 2 4 2; 1 2 1]/4, 'same');
B = G + conv2(Bh - Bcfa.*G, [1 2 1; 2 4 2; 1 2 1]/4, 'same');
```

B Interpolation Examples

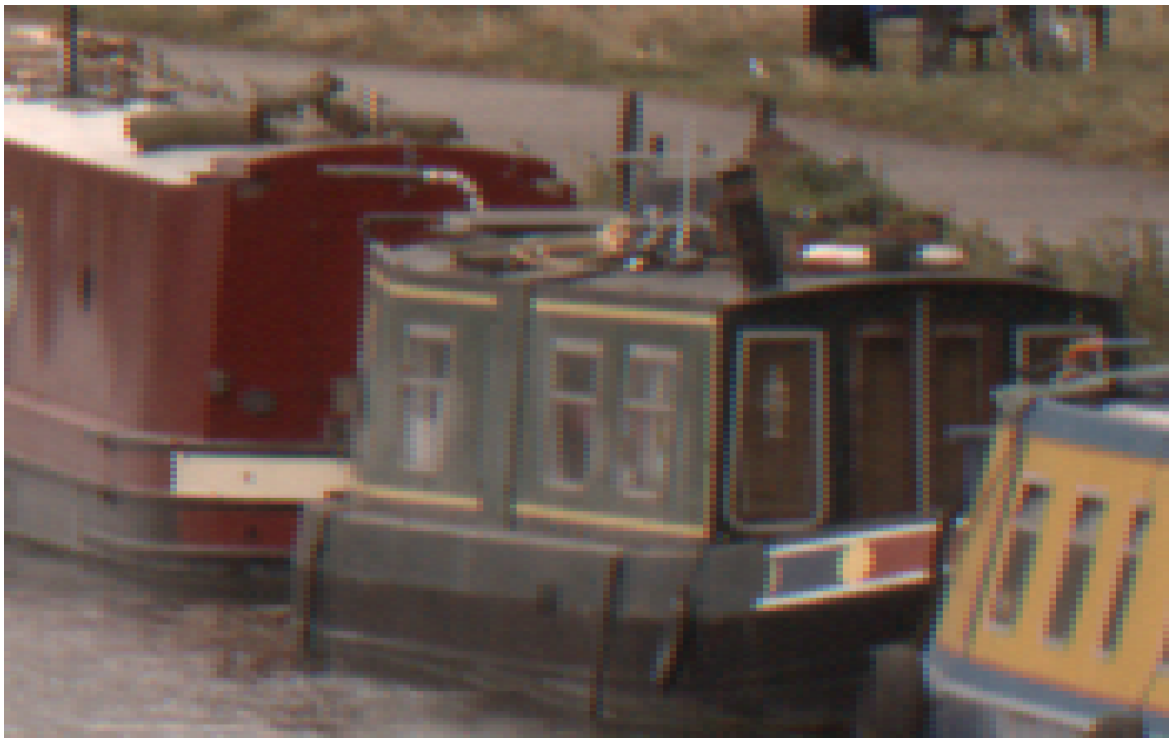


Figure 9: *A nearest neighbour interpolation of the data from figure 5*

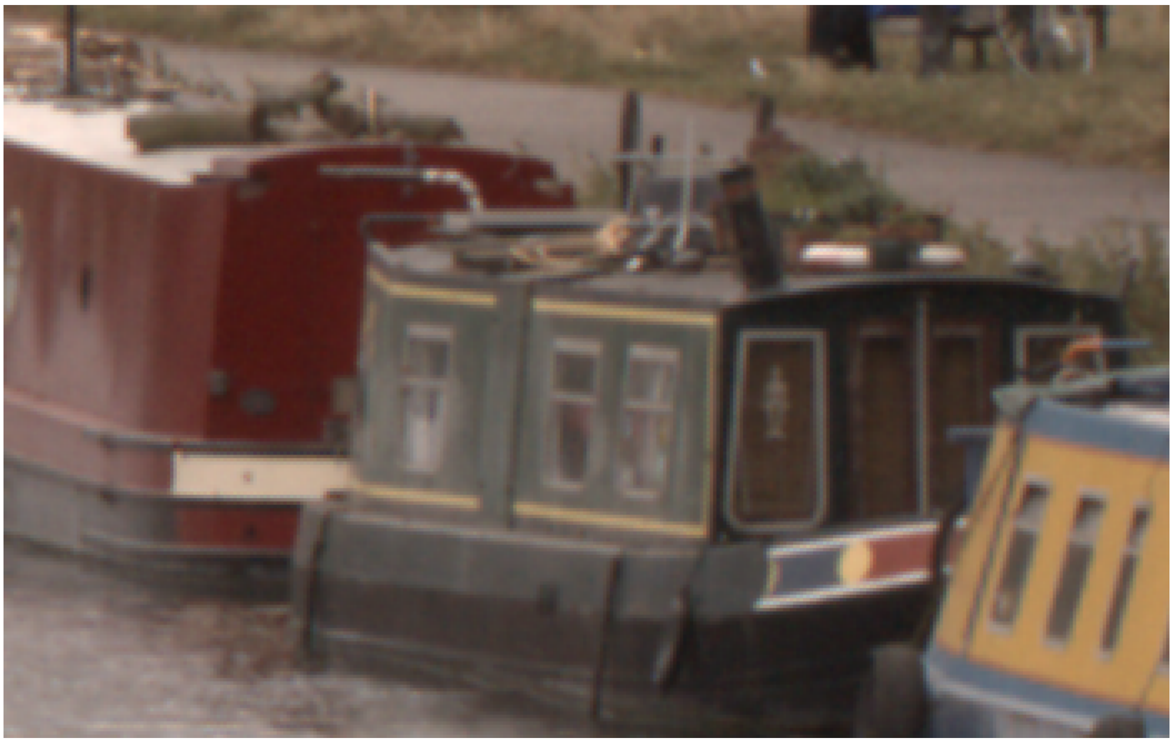


Figure 10: *A bilinear interpolation of the data from figure 5*

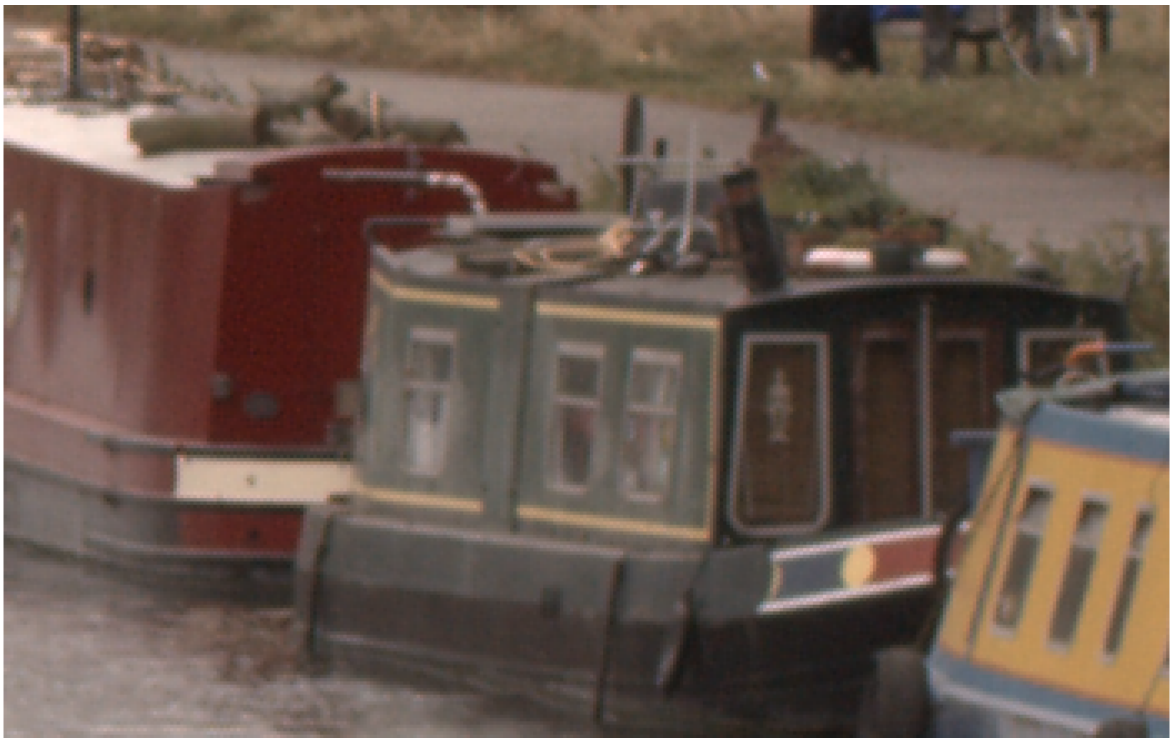


Figure 11: *A smooth hue transition interpolation of the data from figure 5*

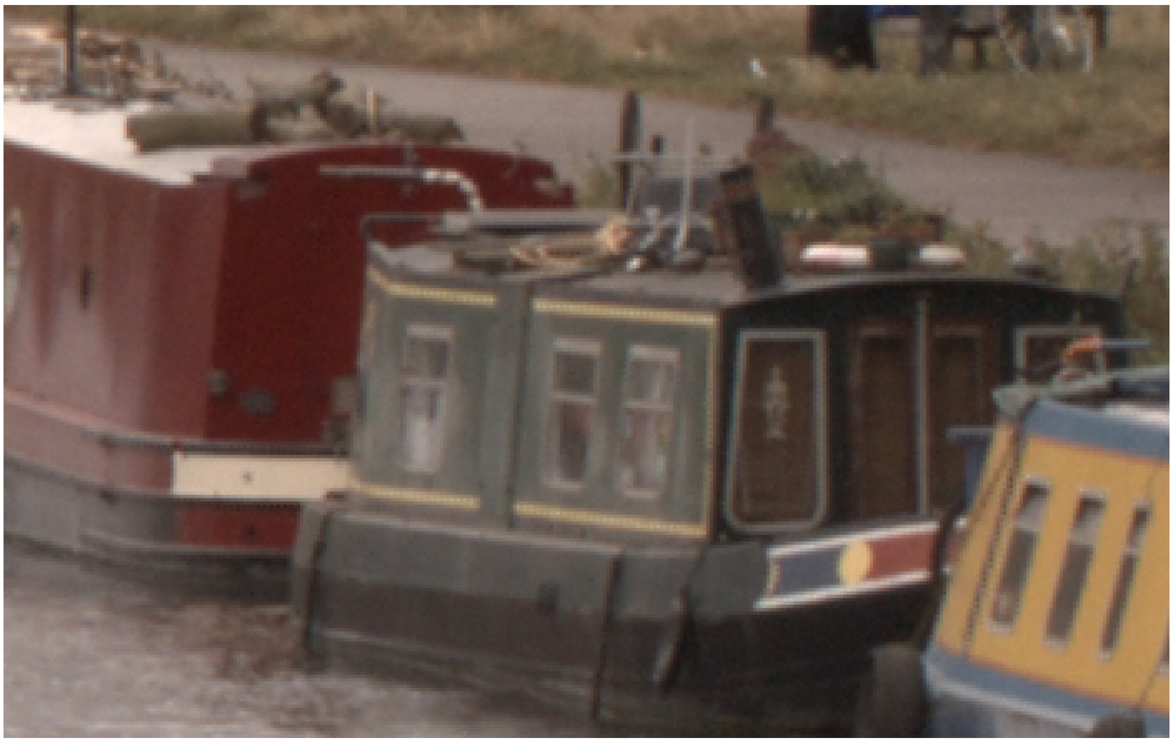


Figure 12: *A median filtered bilinear interpolation of the data from figure 5*

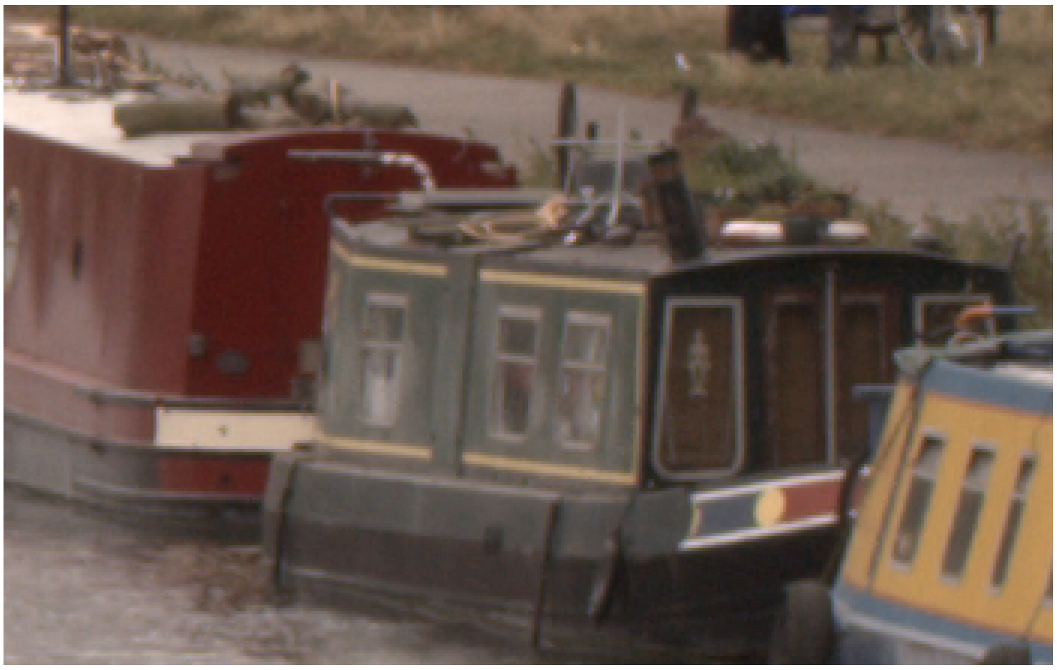
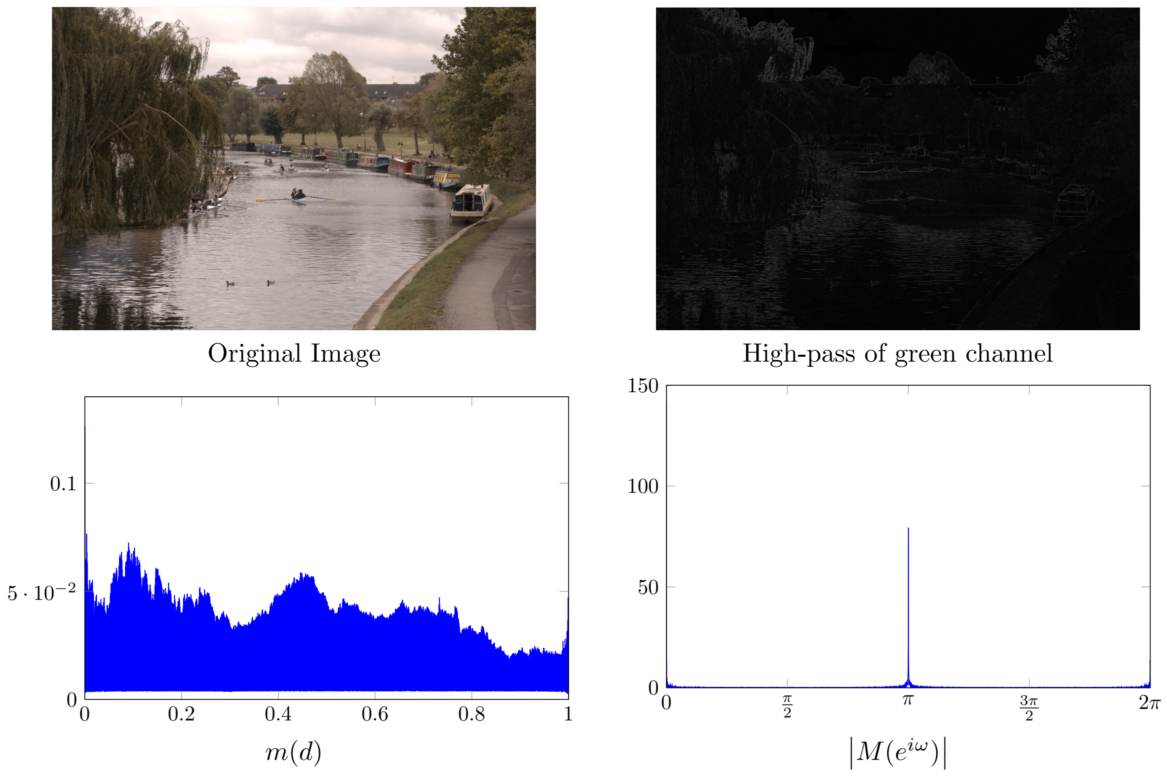


Figure 13: *A gradient-based interpolation of the data from figure 5*

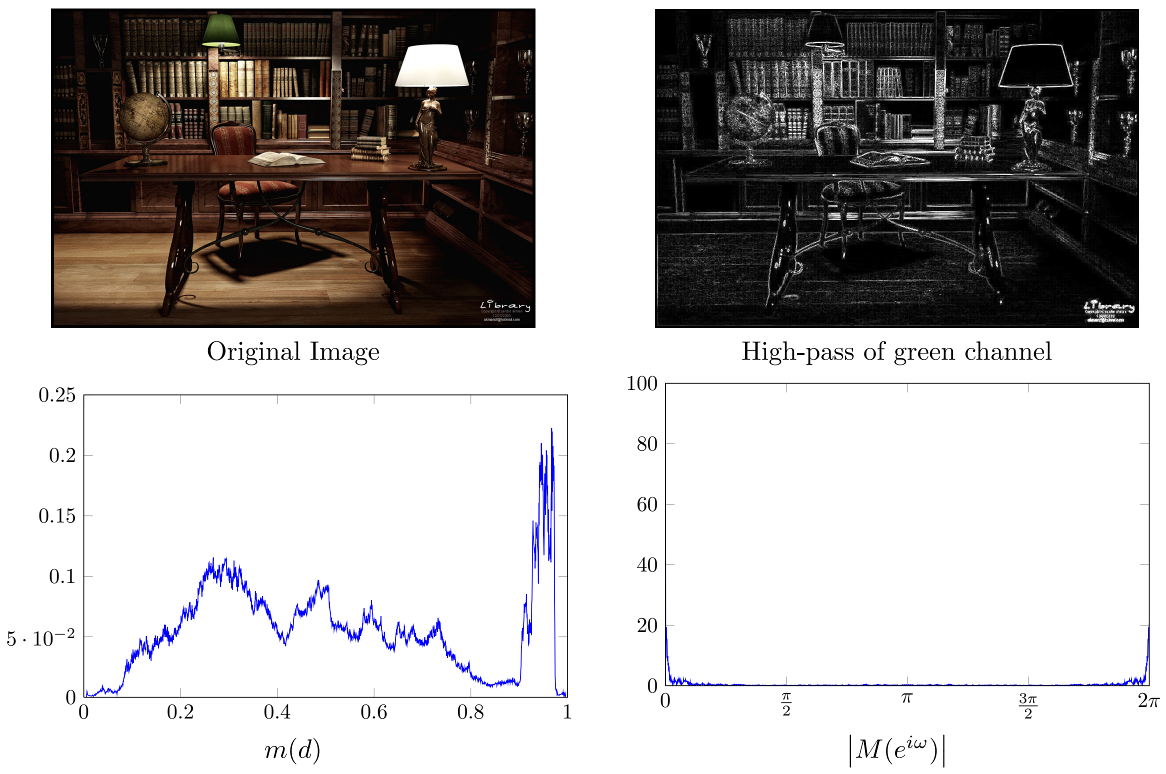
C Detection Examples

This section contains example executions of the CFA interpolation detection algorithm.



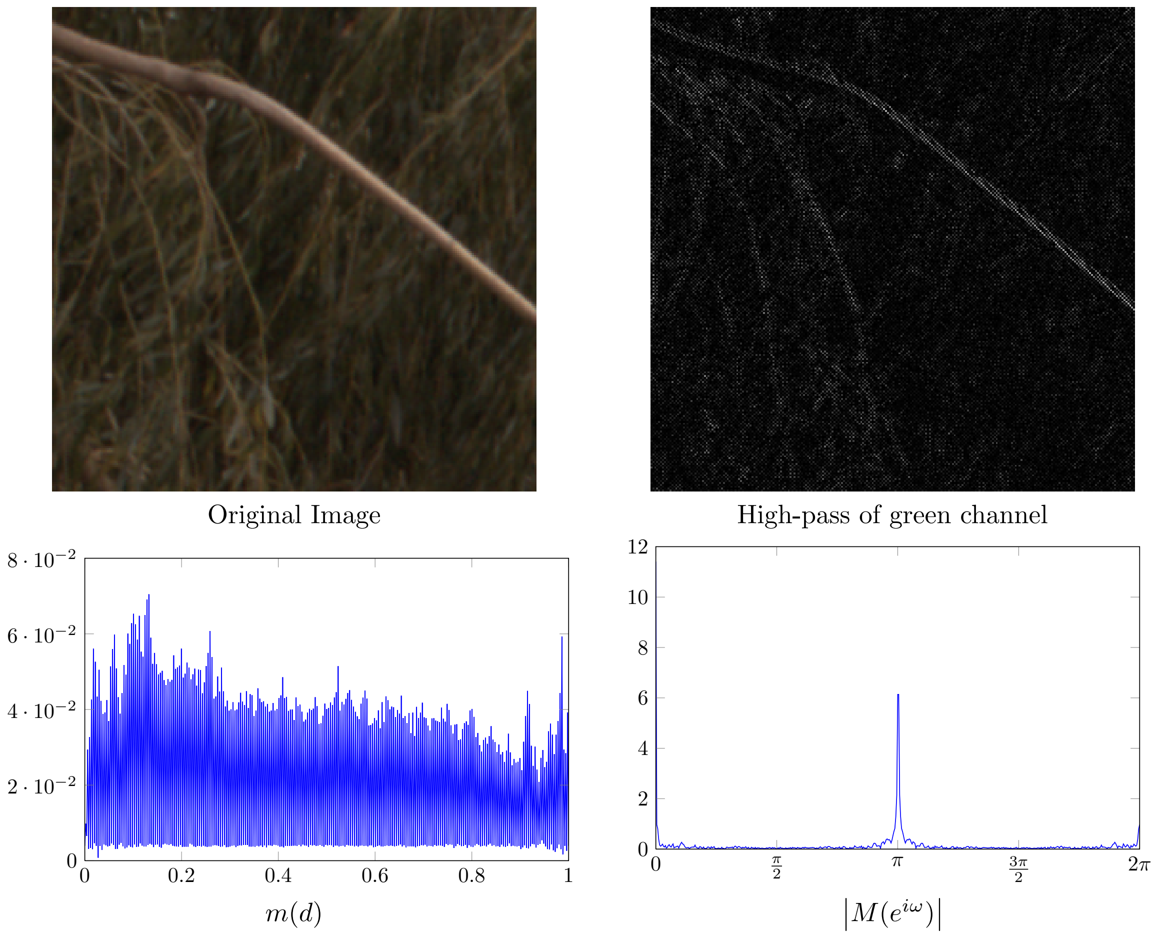
$$\begin{aligned} s &= \frac{\left|M(e^{i\omega})\right|_{\omega=\pi}}{\text{median}_\omega\{|M(e^{i\omega})|\}} \\ &= \frac{79.2337}{0.1191} \\ &= 665.0172 \end{aligned}$$

Figure 14: *A bilinearly interpolated image passing through the interpolation detector*



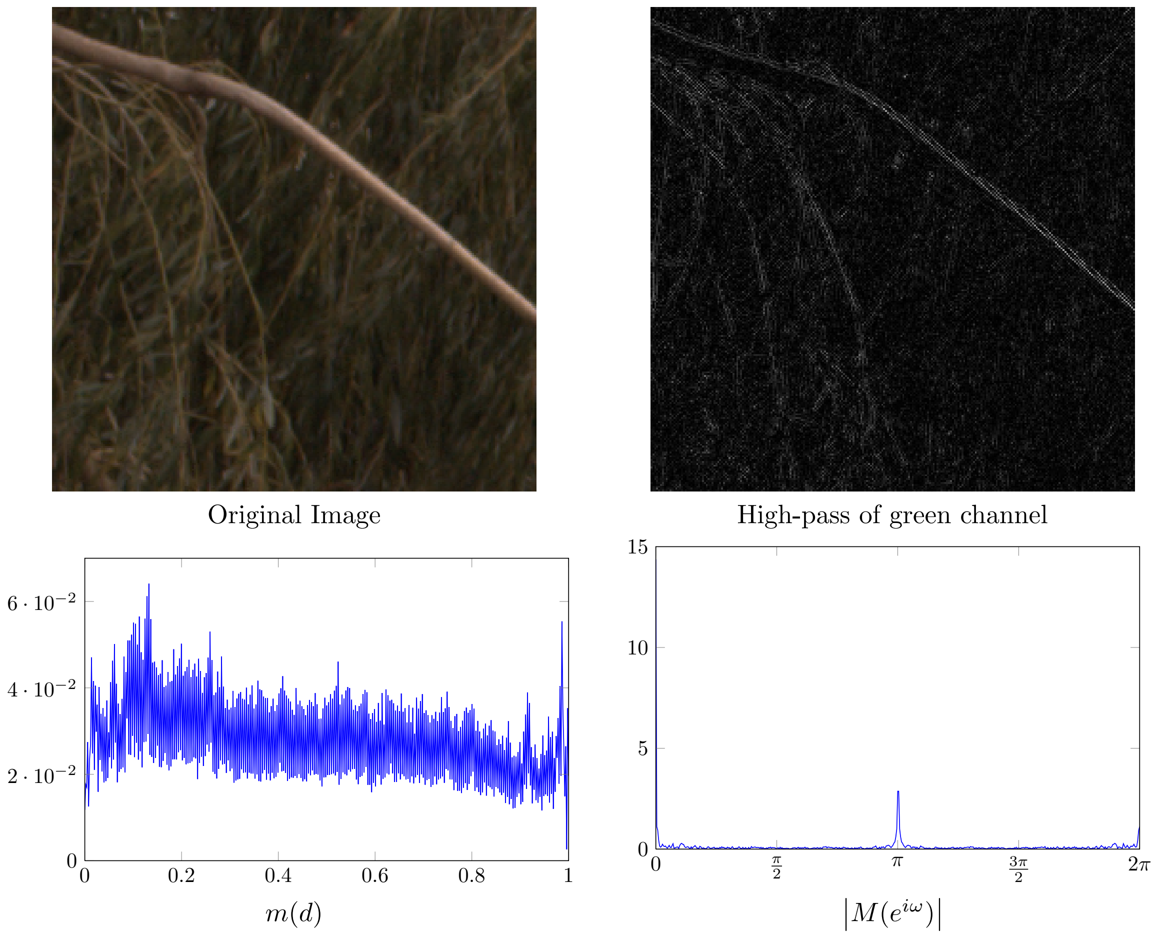
$$
\begin{aligned}
s &= \frac{\left|M(e^{i\omega})\right|_{\omega=\pi}}{\text{median}_{\omega}\{\left|M(e^{i\omega})\right|\}} \\
&= \frac{0.2546}{0.1688} \\
&= 1.5081
\end{aligned}
$$

Figure 15: *A PRCG image passing through the interpolation detector*



$$
\begin{aligned}
s &= \frac{\left|M(e^{i\omega})\right|_{\omega=\pi}}{\text{median}_{\omega}\{|M(e^{i\omega})|\}}\\
&= \frac{6.1377}{0.0574}\\
&= 106.8595
\end{aligned}
$$

Figure 16: *A* 256 × 256 *section of a bilinearly interpolated image passing through the interpolation detector*



$$\begin{aligned} s &= \frac{\left|M(e^{i\omega})\right|_{\omega=\pi}}{\text{median}_{\omega}\{|M(e^{i\omega})|\}} \\ &= \frac{2.8713}{0.0637} \\ &= 45.1039 \end{aligned}$$

Figure 17: *A* 256 × 256 *section of a gradient-interpolated image passing through the interpolation detector*