Methods

The Cubic-Spline Interpolation Method

The cubic-spline interpolation method, as proposed by Li and Randhawa, is used to estimate the missing intensity values in the green channel only. The interpolation processes seven contiguous pixels, as shown in Figure 2, where four of the pixels represent a valid green value and the remaining three pixels represent valid red or blue values. Three cubic polynomials, s_0, s_1 and s_2, are constructed such that s_0 passes through G_0 and G_2, s_1 passes through G_2 and G_4 and s_2 passes through G_4 and G_6:

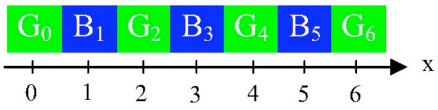


Figure 2. An example 1x7 vector upon which an interpolation is computed.

$$s_0 = a_0x^3 + b_0x^2 + c_0x + d_0, \quad x \in [0, 2]. \qquad s_0(0) = G_0 \qquad s_0(2) = G_2$$
$$s_1 = a_1x^3 + b_1x^2 + c_1x + d_1, \quad x \in [2, 4]. \qquad s_1(2) = G_2 \qquad s_1(4) = G_4$$
$$s_2 = a_2x^3 + b_2x^2 + c_2x + d_2, \quad x \in [4, 6]. \qquad s_2(4) = G_4 \qquad s_2(6) = G_6$$

The curves are also constrained to have continuous first and second order derivatives at their intersections:

$$s_0'(2) = s_1'(2) \qquad s_0''(2) = s_1''(2)$$
$$s_1'(4) = s_2'(4) \qquad s_1''(4) = s_2''(4)$$

Finally, to satisfy the hue assumption [4], the difference between the green and blue values must remain constant across the local 1x7 pixel window. These stated conditions yield the twelve simultaneous equations shown in Figure 3. These equations can be solved by a matrix multiplication, also shown in Figure 4. This matrix multiplication can be computed efficiently in hardware.

$$S_0(0) = G_0 = d_0$$
$$S_0(2) = G_2 = 8a_0 + 4b_0 + 2c_0 + d_0$$
$$S_1(2) = G_2 = 8a_1 + 4b_1 + 2c_1 + d_1$$
$$S_1(4) = G_4 = 64a_1 + 16b_1 + 4c_1 + d_1$$
$$S_2(4) = G_4 = 64a_2 + 16b_2 + 4c_2 + d_2$$
$$S_2(6) = G_6 = 216a_2 + 36b_2 + 6c_2 + d_2$$
$$12a_0 + 4b_0 + c_0 - 12a_1 - 4b_1 - c_1 = 0$$
$$48a_1 + 8b_1 + c_1 - 48a_2 - 8b_2 - c_2 = 0$$
$$12a_0 + 2b_0 - 12a_1 - 2b_1 = 0$$
$$24a_1 + 2b_1 - 24a_2 - 2b_2 = 0$$
$$B_2 - B_1 = \hat{G}_2 - \hat{G}_1$$
$$B_5 - B_3 = \hat{G}_5 - \hat{G}_3$$

Figure 3. The 12 constraints of the cubic spline curves.

$$\text{Let } \mathbf{M} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 8 & 4 & 2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 8 & 4 & 2 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 64 & 16 & 4 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 64 & 16 & 4 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 216 & 36 & 6 & 1 \\ 12 & 4 & 1 & 0 & -12 & -4 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 48 & 8 & 1 & 0 & -48 & -8 & -1 & 0 \\ 12 & 2 & 0 & 0 & -12 & -2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 24 & 2 & 0 & 0 & -24 & -2 & 0 & 0 \\ -1 & -1 & -1 & -1 & 27 & 9 & 3 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -27 & -9 & -3 & -1 & 125 & 25 & 5 & 1 \end{bmatrix}; \quad \mathbf{C} = \begin{bmatrix} b_0 \\ c_0 \\ d_0 \\ a_1 \\ b_1 \\ c_1 \\ d_1 \\ a_2 \\ b_2 \\ c_2 \end{bmatrix}; \quad \mathbf{V} = \begin{bmatrix} G_0 \\ G_2 \\ G_2 \\ G_4 \\ G_4 \\ G_6 \\ 0 \\ 0 \\ 0 \\ 0 \\ B_3 - B_1 \\ B_5 - B_3 \end{bmatrix}$$

Figure 4. The matrices can be used to solve the equations from Figure 3 with minimum computational complexity: $MC = V$; $C = M^{-1}V$.

Once the equations have been solved, the estimated green value at x=3 is stored, and the 1x7 vector is shifted by two pixels in the positive x direction. The cubic spline interpolation process then restarts on the new 1x7 vector. A sample pixel vector and its interpolation curves are plotted in Figure 5.

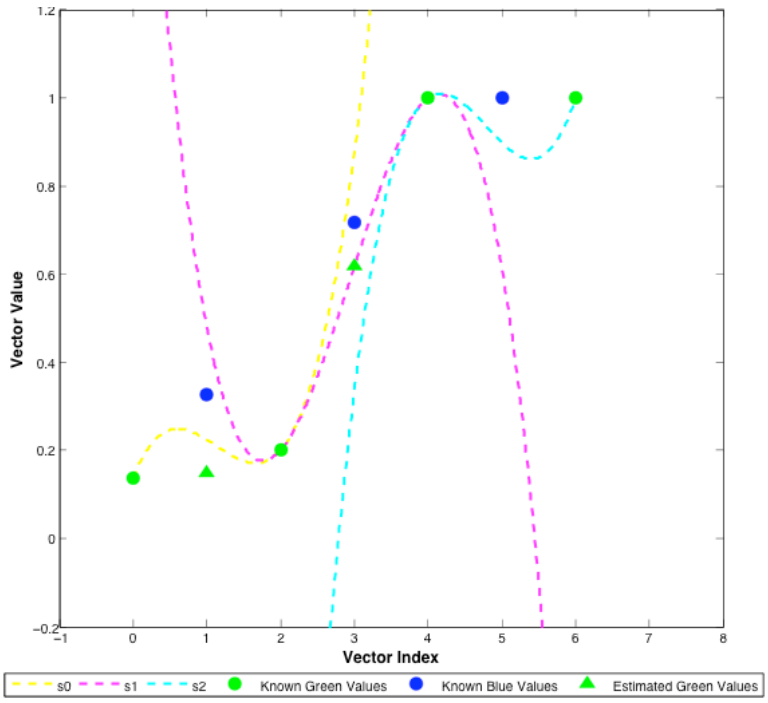


Figure 5. A graphical example of the cubic spline interpolation method.

Since the method interpolates through a linear vector of pixels, it can be used to interpolate across a row or down a column of pixels in an image. Two intermediate images are computed using cubic spline interpolation: one is taken from only horizontal interpolation across rows; a second is taken from only vertical interpolation down columns. To prevent from blurring over edges in the image, an edge orientation map is also computed to determine from which intermediate image a pixel estimation should be chosen.

The Edge Orientation Map

Edges are detected using a simple 2-dimensional difference algorithm. According to the Bayer pattern in figure 1, the pixels directly above and below a given pixel will contain a valid value for the same color channel. Similarly, the pixels directly to the left and right of a given pixel will contain a valid value for the same color channel. If the difference between a pixel's vertical neighbors is less than the difference between a pixel's horizontal neighbors, a potential vertical edge exists at the pixel. Inversely, if the vertical difference is greater than the horizontal difference, a potential horizontal edge exists. A sample edge orientation computation is shown in Figure 6.

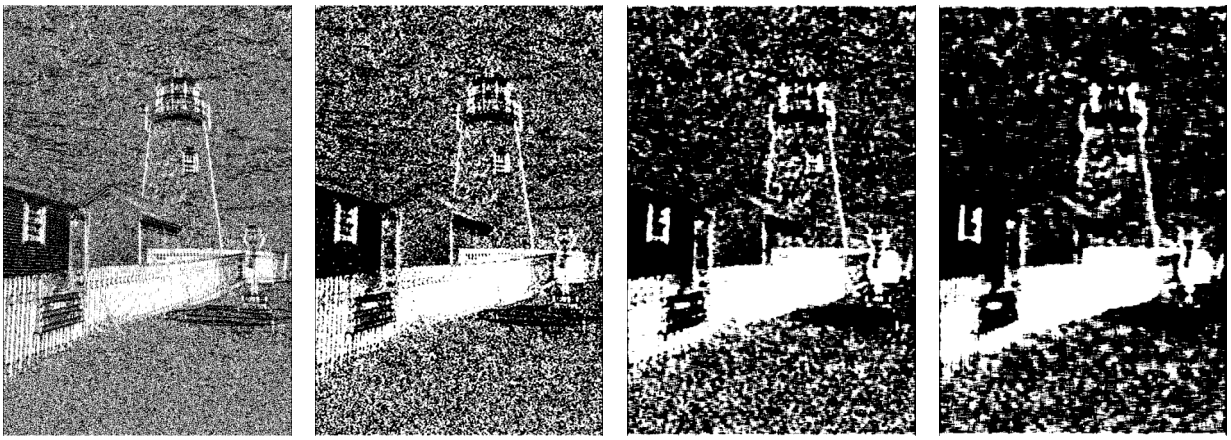
For a red or blue pixel:
$V_{54} = |G_{44}-G_{64}|$, $H_{54} = |G_{53}-G_{55}|$

For a green pixel:
$V_{44} = |R_{34}-R_{54}|$, $H_{44} = |B_{43}-B_{45}|$

$$f_{xy} = \begin{cases} 1, \text{ if } V_{xy} < H_{xy} \\ 0, \text{ otherwise} \end{cases}$$

Figure 6. A sample edge orientation calculation.

Since each pixel is assigned to either a vertical or horizontal edge, with no intermediate values assigned, a great deal of noise will be present in a raw orientation map. Thus odd-window width median filtering is applied to the raw orientation map to reduce noise while preserving true edges. Figure 7 shows the raw orientation map computed from the lighthouse test image and the orientation maps after median filtering with window widths of 3, 7 and 11.



(a) (b) (c) (d)

Figure 7. Edge orientation maps without median filtering (a) and with median filtering width = 3 (b), 7 (c), and 11 (d) pixels.

Extensions to Li and Randhawa's Method

Li and Randhawa present the cubic-spline interpolation method for only the green channel of an image due to the higher sampling frequency as compared to the red and blue channels. They suggest using bilinear interpolation for the red and blue channels. Due to the errors produced by pure bilinear interpolation, we decided to implement cubic-spline interpolation on the blue and red channels. Due to the Bayer sampling pattern, edge orientation cannot be used for estimating blue and red values, and therefore edge blurring cannot be prevented using the cubic-spline interpolation method on the red and blue channels. Nonetheless, applying bidirection, non-adaptive cubic-spline interpolation to the red and blue channels allowed us to decrease mean-square error by 39-62% compared to bilinear interpolation.

We also explored optimizations on the Li and Randhawa method with regard to the median filtering step. We studied the effect of changing the window width of the median filtering on final image mean-square error. For the lighthouse and white fence image, the optimal median filter had a 7-pixel window width. For the river buildings image, the optimal

median filter had a 5-pixel window width. For the spatial frequency test image, the optimal median filter had a 3-pixel window width. These optimizations are illustrated by the plots in Figures 8 through 10.

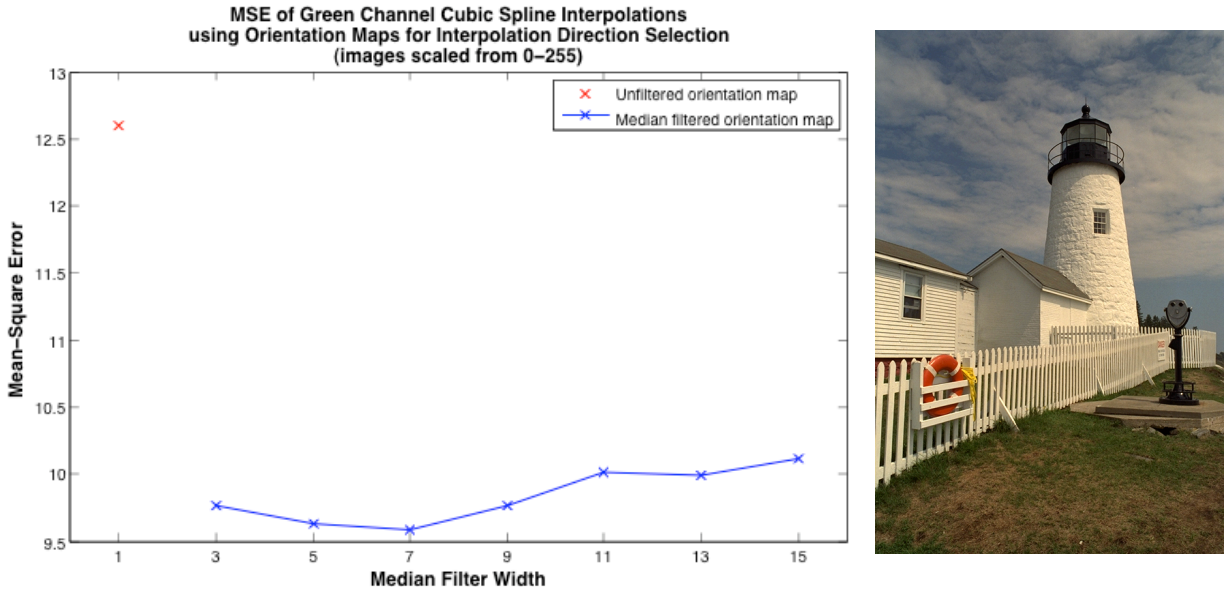


Figure 8. Optimization of median filter width for the lighthouse test image.

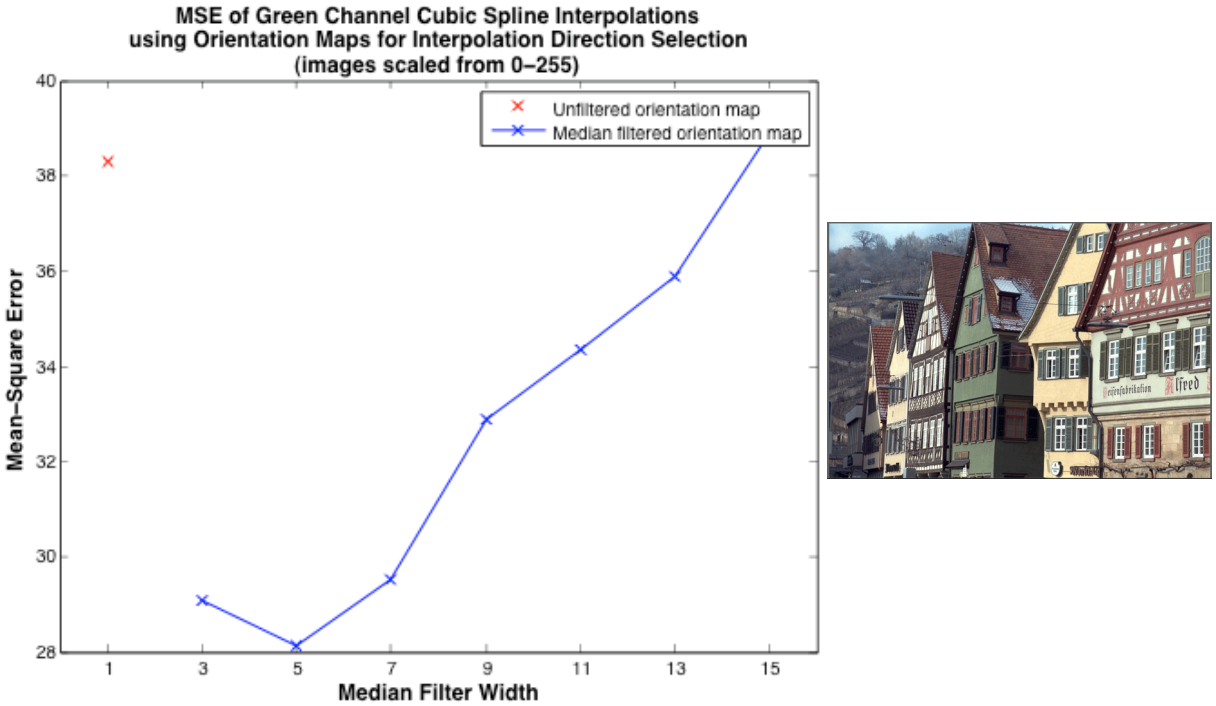


Figure 9. Optimization of median filter width for the buildings test image.

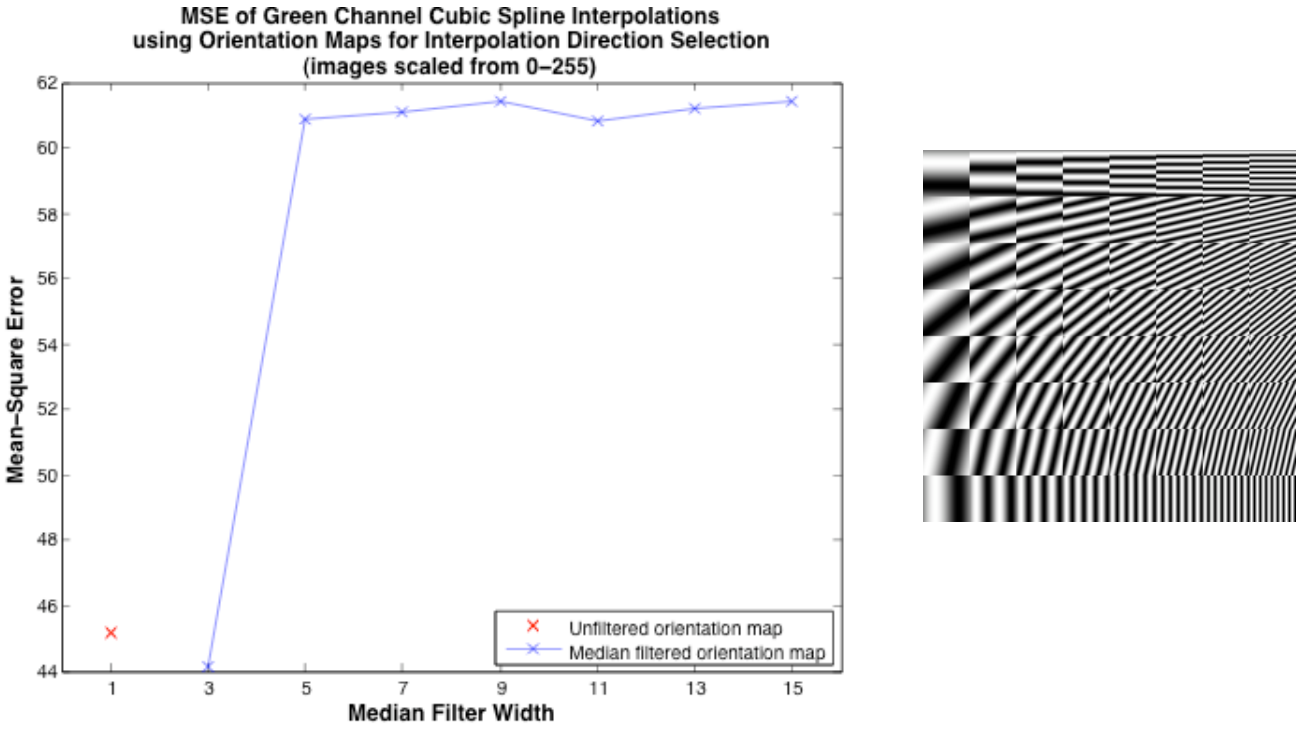


Figure 10. Optimization of median filter width for the spatial frequency test image.

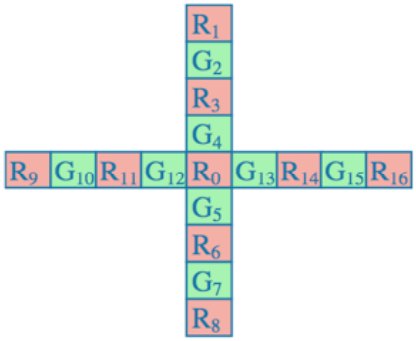
The Color Difference Space Method

The color difference space method proposed by Yuk, Au, Li, and Lam interpolates pixels in green-red and green-blue color difference spaces as opposed to interpolating on the original red, green, and blue channels. The underlying assumption is that due to the correlation between color channels, taking the difference between two channels yields a color difference channel with less contrast and edges that are less sharp. Figure 11 compares images from the original RGB color space with those from the color difference space. Demosaicking an image with less contrast yields fewer glaring errors, as sharp edges are what cause most of the interpolation errors in reconstructing an image. The color difference space method creates K_R (green minus red) and K_B (green minus blue) difference channels and interpolates them; the method then reconstructs the red, green, and blue channels for a fully demosaicked image.

(a) (b) (c) (d)

Figure 11. Lighthouse image color difference channels: original red channel (a), green minus red difference channel (b), original blue channel (c), and green minus blue difference channel (d) .

Because of the color filter array, however, getting the K_R and K_B channels is not trivial and requires a multi-step process. The first step is to find K_R values at red pixel locations and K_B values at blue pixel locations. For every red pixel, an average is made of the surrounding green pixels in either the vertical or horizontal direction. From this value, the average of the surrounding red pixels in the vertical or horizontal direction is subtracted, yielding the K_R difference value at the red pixel. The same is done for finding K_B values at blue pixel locations. The directionality (vertical or horizontal) of the computation is determined based on a simple edge comparison, which determines whether vertical or horizontal interpolation will yield the smoothest result. Figure 12 summarizes this first step in the procedure.



$$K_{R0}^{V} = \frac{1}{2}\left(G_4 + G_5\right) - \frac{1}{4}\left(R_3 + 2R_0 + R_6\right)$$

$$K_{R0}^{H} = \frac{1}{2}\left(G_{12} + G_{13}\right) - \frac{1}{4}\left(R_{11} + 2R_0 + R_{14}\right)$$

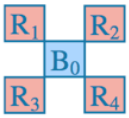
$$V_1 = \left|K_{R0}^{V} - K_{R3}^{V}\right| + \left|K_{R0}^{V} - K_{R6}^{V}\right|$$

$$H_1 = \left|K_{R0}^{H} - K_{R11}^{H}\right| + \left|K_{R0}^{H} - K_{R14}^{H}\right|$$

Choose K_{R0}^{V} if $V_1 < H_1$ else choose K_{R0}^{H}

Figure 12. Computing K_R0 for red pixel locations.

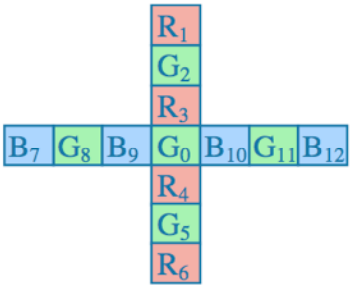
The second step is to find K_R values at blue pixel locations and K_B values at blue pixel locations. For K_R values, this is as simple as averaging the known K_R to the upper-left, upper-right, lower-left, and lower-right of the current blue pixel location. The same is done for finding K_B values at red pixel locations. Figure 13 shows how this is done.



$$K_{R0} = \frac{K_{R1} + K_{R2} + K_{R3} + K_{R4}}{4}$$

Figure 13. Computing K_R0 for blue pixel locations.

Finally, K_R and K_B values are calculated for the green pixel locations. Since at this point every other pixel has been filled in for the K_R and K_B channels, finding the values at the green pixels is as simple as taking the average of the nearest K_R or K_B values in either the vertical or horizontal direction. Again this directional aspect of the computation is based on an edge comparison and ensures smoothness of the interpolation. Figure 14 summarizes this final step.



$$K_{R0}^{V} = \frac{K_{R3} + K_{R4}}{2}$$

$$K_{R0}^{H} = \frac{K_{R9} + K_{R10}}{2}$$

$$V_2 = \left|K_{R0}^{V} - K_{R2}^{V}\right| + \left|K_{R0}^{V} - K_{R5}^{V}\right|$$

$$H_2 = \left|K_{R0}^{H} - K_{R8}^{H}\right| + \left|K_{R0}^{H} - K_{R11}^{H}\right|$$

Choose K_{R0}^{V} if $V_2 < H_2$ else choose K_{R0}^{H}

Figure 14. Computing K_R0 for green pixel locations.

With fully interpolated K_R and K_B channels we can now recompute the RGB channels to get our fully demosaicked image. This can be done using the equations in Figure 15.

At original R pixel locations:

$$R_{new} = R_{orig}$$
$$G_{new} = K_R + R_{new}$$
$$B_{new} = G_{new} - K_B$$

At original G pixel locations:

$$G_{new} = G_{orig}$$
$$R_{new} = G_{new} - K_R$$
$$B_{new} = G_{new} - K_B$$

At original B pixel locations

$$B_{new} = B_{orig}$$
$$G_{new} = K_B + B_{new}$$
$$R_{new} = G_{new} - K_R$$

Figure 15. Recomputing red, green, and blue channels.