

```

1 #include <iostream>
2
3 #include <opencv2/core.hpp>
4 #include <opencv2/imgcodecs.hpp>
5 #include <opencv2/highgui.hpp>
6 #include <opencv2/opencv.hpp>
7
8 #define BGR_BLUE 0
9 #define BGR_GREEN 1
10 #define BGR_RED 2
11
12
13 using namespace cv;
14
15 void deBayer(Mat *rawImg, Mat *outImg) {
16     // the Daheng camera datasheet specifies a GRBG pattern (page 88). Our
17     // input values will be 8 bit
18     // Split 1 channel image into 3 channels according to bayer pattern
19     // 0 1
20     //0 G R
21     //1 B G
22
23     //I will create a new "Mat" with OpenCV that contains three channels. The
24     //splitting and processing is done by hand.
25     if (rawImg->type() != CV_8UC1)
26         throw("Sorry, only 1 8-bit channel should be used");
27
28     (*outImg) = cv::Mat::zeros(rawImg->rows, rawImg->cols, CV_8UC3); // Fill
29     // output buffer with zeros with the correct geometry
30     MAT outImgMID = cv::Mat::zeros(rawImg->rows, rawImg->cols, CV_8UC3); //
31     // Fill output buffer with zeros with the correct geometry
32     //Size s = rawImg->size(); I will be using rows and columns rather than
33     //height and width
34     long row, col;
35
36     std::cout << "Bayer splitting to 3 channels" << std::endl;
37
38     for(row=0; row<rawImg->rows; row++){ //todo: make this evaluation smaller
39     // to increase speed
40         for(col=0; col<rawImg->cols; col++){
41             if (row % 2 == 0 && col % 2 == 0) //odd row, odd column
42                 outImgMID->at<Vec3b>(row, col).val[BGR_GREEN] = rawImg->at<uint8_t>
43                 (row,col);
44             if (row % 2 == 0 && col % 2 == 1) //odd row, even column
45                 outImgMID->at<Vec3b>(row, col).val[BGR_RED] = rawImg->at<uint8_t>
46                 (row,col);
47             if (row % 2 == 1 && col % 2 == 0) //even row, odd column
48                 outImgMID->at<Vec3b>(row, col).val[BGR_BLUE] = rawImg->at<uint8_t>
49                 (row,col);
50             if (row % 2 == 1 && col % 2 == 1) //even row, even column
51                 outImgMID->at<Vec3b>(row, col).val[BGR_GREEN] = rawImg->at<uint8_t>
52                 (row,col);
53             }
54         }
55
56     // Interpolate green channel by taking the value of the nearest neighbour
57     // that is green
58     std::cout << "Interpolating green channel" << std::endl;
59 }

```

```

48
49     for(row=0; row<outImgMID->rows; row++){
50         for(col=0; col<outImgMID->cols; col++){
51             if (outImgMID->at<Vec3b>(row, col).val[BGR_GREEN] == 0){
52                 if (col > 0 && outImgMID->at<Vec3b>(row, col-1).val[BGR_GREEN] != 0)
53                     outImg->at<Vec3b>(row, col).val[BGR_GREEN] = outImgMID->at<Vec3b>
54 (row, col-1).val[BGR_GREEN];
55                 else if (col < outImgMID->cols-1 && outImgMID->at<Vec3b>(row,
56 col+1).val[BGR_GREEN] != 0)
57                     outImg->at<Vec3b>(row, col).val[BGR_GREEN] = outImgMID->at<Vec3b>
58 (row, col+1).val[BGR_GREEN];
59                 else if (row > 0 && outImgMID->at<Vec3b>(row-1, col).val[BGR_GREEN]
60 != 0)
61                     outImg->at<Vec3b>(row, col).val[BGR_GREEN] = outImgMID->at<Vec3b>
62 (row-1, col).val[BGR_GREEN];
63                 else if (row < outImgMID->rows-1 && outImgMID->at<Vec3b>(row+1,
64 col).val[BGR_GREEN] != 0)
65                     outImg->at<Vec3b>(row, col).val[BGR_GREEN] = outImgMID->at<Vec3b>
66 (row+1, col).val[BGR_GREEN];
67             }
68         }
69     }
70
71     // Interpolate red channel by taking the value of the nearest neighbour
72     that is red
73     std::cout << "Interpolating red channels" << std::endl;
74
75     for(row=0; row<outImgMID->rows; row++){
76         for(col=0; col<outImgMID->cols; col++){
77             if (outImgMID->at<Vec3b>(row, col).val[BGR_RED] == 0){
78                 if (col > 0 && outImgMID->at<Vec3b>(row, col-1).val[BGR_RED] != 0)
79                     outImg->at<Vec3b>(row, col).val[BGR_RED] = outImgMID->at<Vec3b>
80 (row, col-1).val[BGR_RED];
81                 else if (col < outImgMID->cols-1 && outImgMID->at<Vec3b>(row,
82 col+1).val[BGR_RED] != 0)
83                     outImg->at<Vec3b>(row, col).val[BGR_RED] = outImgMID->at<Vec3b>
84 (row, col+1).val[BGR_RED];
85                 else if (row > 0 && outImgMID->at<Vec3b>(row-1, col).val[BGR_RED] !=
86 0)
87                     outImg->at<Vec3b>(row, col).val[BGR_RED] = outImgMID->at<Vec3b>
88 (row-1, col).val[BGR_RED];
89                 else if (row < outImgMID->rows-1 && outImgMID->at<Vec3b>(row+1,
90 col).val[BGR_RED] != 0)
91                     outImg->at<Vec3b>(row, col).val[BGR_RED] = outImgMID->at<Vec3b>
92 (row+1, col).val[BGR_RED];
93             }
94         }
95     }
96
97     // Interpolate blue channel by taking the value of the nearest neighbour
98     that is blue
99     std::cout << "Interpolating blue channels" << std::endl;
100
101     for(row=0; row<outImgMID->rows; row++){
102         for(col=0; col<outImgMID->cols; col++){
103             if (outImgMID->at<Vec3b>(row, col).val[BGR_BLUE] == 0){
104                 if (col > 0 && outImgMID->at<Vec3b>(row, col-1).val[BGR_BLUE] != 0)
105                     outImg->at<Vec3b>(row, col).val[BGR_BLUE] = outImgMID->at<Vec3b>
106 (row, col-1).val[BGR_BLUE];

```

```

90     else if (col < outImgMID->cols-1 && outImgMID->at<Vec3b>(row,
91 col+1).val[BGR_BLUE] != 0)
92         outImg->at<Vec3b>(row, col).val[BGR_BLUE] = outImgMID->at<Vec3b>
93 (row, col+1).val[BGR_BLUE];
94     else if (row > 0 && outImgMID->at<Vec3b>(row-1, col).val[BGR_BLUE] !=
95 0)
96         outImg->at<Vec3b>(row, col).val[BGR_BLUE] = outImgMID->at<Vec3b>
97 (row-1, col).val[BGR_BLUE];
98     else if (row < outImgMID->rows-1 && outImgMID->at<Vec3b>(row+1,
99 col).val[BGR_BLUE] != 0)
100         outImg->at<Vec3b>(row, col).val[BGR_BLUE] = outImgMID->at<Vec3b>
101 (row+1, col).val[BGR_BLUE];
102     }
103 }
104
105 // done?
106 std::cout << "Done?" << std::endl;
107
108 }
109
110 int main() {
111     // Read the image (in 8bit grayscale)
112     Mat img = imread("test_RAW.png", CV_8UC1);
113     if(img.empty()) {
114         std::cout << "Could not read the image: " << std::endl;
115         return 1;
116     }
117
118     Mat result;
119     deBayer(&img, &result);
120
121     imshow("Display window", result);
122     waitKey(0); // Wait for a keystroke in the window
123     return 0;
124 }

```