

```

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 using namespace cv;
9
10 void filter(Mat* input, Mat* result) {
11     Size s = input->size();
12     long h, w;
13     long sum;
14
15     //std::cout << "Input type was : " << input->type() << std::endl;
16     uint8_t out[s.height][s.width];
17
18     std::cout << s.height << " " << s.width << std::endl;
19
20     for (w=0; w<s.width; w++){ //loop over the image,
21         for(h=0; h<s.height; h++){
22             sum = 0;
23             std::vector<uint8_t> median;
24
25             for (int _x = -1; _x < 2; ++_x) //for every pixel, loop over every
pixel in a 3x3 kernel
26             {
27                 for (int _y = -1; _y < 2; ++_y)
28                 {
29                     int idx_y = h + _y; //sum the incrementor with the kernel's, so we
can identify borders
30                     int idx_x = w + _x;
31
32                     if (idx_x < 0 || idx_x > s.width) //the kernel goes outside of the
image, therefore break and ignore that pixel.
33                         break;
34
35                     if (idx_y < 0 || idx_y > s.height)
36                         break;
37
38                     median.push_back(input->at<uint8_t>(idx_y,idx_x)); // Add all the
pixels from the kernel into a vector
39                 }
40             }
41             std::sort(std::begin(median), std::end(median)); //sort all pixel
values from high to low
42
43             for (auto it = median.begin(); it != median.end(); ++it) {
44                 sum = median.at(median.size()/2); //The pixel at the y,x coordinate
is now the median from our 3x3 sliding window
45             }
46             out[h][w]=(uint8_t)sum;
47         }
48     }
49     // this did not work, since the out array was never copied to memory,
causing image data pointing to nowhere!
50     //result = Mat(s.height, s.width, CV_8U, out);
51     // Instead, the data from out needs to be copied directly to Mat result
...

```

```

with the correct size
52     std::memcpy(result->data, out, s.height*s.width*sizeof(uint8_t));
53 }
54
55 void gammaCorrection(Mat* input, Mat* output, float gamma) {
56     Size s = input->size();
57     long h, w;
58     float sum;
59
60     std::cout << "Correcting gamma" << std::endl;
61     uint8_t out[s.height][s.width];
62     uint8_t lut[256];
63
64     for (int i = 0; i < 256; i++) { //create a lookup table, with the gamma
correction curve.
65         lut[i] = saturate_cast<uint8_t>(pow((float)(i / 255.0), gamma) * 255.0f);
//saturate cast negative values to 0, and higher values to 255 (uint8_t or
unsigned char)
66         //std::cout << unsigned(lut[i]) << " "; //print the function for testing.
cout prints uint8_t as chars so we cast it.
67     }
68
69
70     for (w=0; w<s.width; w++){ //loop over the image,
71         for(h=0; h<s.height; h++){
72             sum = lut[(input->at<uint8_t>(h,w))]; //the original output value will
be scaled to the value in the LUT.
73             out[h][w]=(uint8_t)sum;
74         }
75     }
76     std::memcpy(output->data, out, s.height*s.width*sizeof(uint8_t)); //copy
our standard 2D array to a new buffer that OpenCV understands
77 }
78
79
80 int main() {
81     // Read the image (in BGR)
82     Mat img = imread("pixerror.png", IMREAD_COLOR);
83     if(img.empty())
84     {
85         std::cout << "Could not read the image: " << std::endl;
86         return 1;
87     }
88     Size imgsize = img.size();
89
90     // Split the image into 3 new images for blue, green and red.
91     std::cout << "Splitting channels: " << std::endl;
92     Mat bands[3];
93     split(img, bands);
94
95     Mat bandsFiltered[3];
96     Mat bandsCorrected[3];
97     bandsFiltered[0] = Mat(imgsize.height, imgsize.width, CV_8U);
98     bandsFiltered[1] = Mat(imgsize.height, imgsize.width, CV_8U);
99     bandsFiltered[2] = Mat(imgsize.height, imgsize.width, CV_8U);
100
101     bandsCorrected[0] = Mat(imgsize.height, imgsize.width, CV_8U);
102     bandsCorrected[1] = Mat(imgsize.height, imgsize.width, CV_8U);
103     bandsCorrected[2] = Mat(imgsize.height, imgsize.width, CV_8U);

```

```

104
105     filter(&bands[0],&bandsFiltered[0]); //filter all channels from noise
individually
106     filter(&bands[1],&bandsFiltered[1]);
107     filter(&bands[2],&bandsFiltered[2]);
108
109     gammaCorrection(&bandsFiltered[0],&bandsCorrected[0],0.33);
110     gammaCorrection(&bandsFiltered[1],&bandsCorrected[1],0.33);
111     gammaCorrection(&bandsFiltered[2],&bandsCorrected[2],0.33);
112
113     Mat merged;
114     std::vector<Mat> channels =
{bandsCorrected[0],bandsCorrected[1],bandsCorrected[2]};
115     merge(channels, merged);
116
117     // Display the image until q is pressed
118     std::cout << "Displaying result: " << std::endl;
119     imshow("Display window", bands[0]);
120     waitKey(0); // Wait for a keystroke in the window
121     imshow("Display window", bands[1]);
122     waitKey(0); // Wait for a keystroke in the window
123     imshow("Display window", bands[2]);
124     waitKey(0); // Wait for a keystroke in the window
125     imshow("Display window", bandsFiltered[0]);
126     waitKey(0); // Wait for a keystroke in the window
127     imshow("Display window", bandsFiltered[1]);
128     waitKey(0); // Wait for a keystroke in the window
129     imshow("Display window", bandsFiltered[2]);
130     waitKey(0); // Wait for a keystroke in the window
131     imshow("Display window", merged);
132     waitKey(0); // Wait for a keystroke in the window
133     return 0;
134 }
135

```