

Image Processing mini project

Diagonal Edge Detection in C++



Gustav Dahl, 20113263

Image Processing and Procedural Programming
Medialogy 3rd semester
Aalborg University October 2012

Task definition

The theme for the 3rd semester of Medialogy is Visual Computing. Here subjects about how humans and machines perceive images were taught. In the course **Image Processing** it was shown how it is possible to process and manipulate digital images in theory and concept, while the course **Procedural Programming** was about learning the C++ programming language, as well as the OpenCV framework. To apply the knowledge about image processing in practice, each student were tasked with writing a small program that could process an image in a certain way. Everything should be implemented from scratch; OpenCV should only be used to load in an image and nothing more.

Each mini project was meant as an individual task, and everybody in the group received a different task. The following is the description of the task I received.

“ Topic #5: Diagonal Edge Detection
Make a C/C++ program that can find diagonal edges in an image.
Input: Greyscale image Output: Binary image where the diagonal edges
are white (255) and the rest of the pixels black (0) ”

Table of content

Chapter 1	Theory about Edge Detection	3
1.1	Edge definition	3
1.2	The usefulness of edges	3
1.3	The concept of edge detection	3
Chapter 2	The code	5
Bibliography		14

Theory about Edge Detection 1

1.1 Edge definition

An edge in an image is basically a place in an image where there is a contrast between two points.

Block [2007] describes an edge as the apparent line around the borders of a two-dimensional object.

Another definition is given by [Moeslund, 2012b] who writes that an edge in an image is defined as a position where there is a significant change in gray-level values.

In other words, **an edge in an image is where the intensity changes dramatically**. A perfect edge would have to be a transition from e.g. black to white over just one pixel, but in the real world this rarely happen, unless it is a binary image where there are only black and white pixels.

1.2 The usefulness of edges

Edges are typically used to define the boundary of an object. This reduces a lot of calculations needed to be done, either by the human brain or a computer,, since it is only necessary to look at the outline and not the whole object. It allows for higher levels of abstraction. This system is used in the way a human perceives the world, using ganglion cell signal changes [Snowden et al., 2012] ¹. In machine vision this system is applied, e.g. if a robot needs to recognize and work with a specific object.

1.3 The concept of edge detection

Edges can be described as the slope of blabla bla as shown on pictures 1.1 and 1.2.

The following is mainly based on [Moeslund, 2012a]. Edge detectors consist of three steps:

- Noise reduction
- Edge enhancement
- Edge localization

The first step, **noise reduction**, can be done using a filter. This could for instance be a median or mean filter. However, there is a dilemma when choosing the size of the filter.

¹Fixme Note: bedre skrive + Perception bog



Figure 1.1. The original image seen in grayscale.

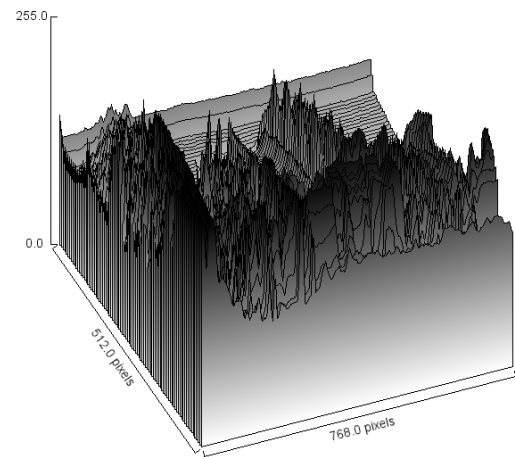


Figure 1.2. Surface plot point created using ImageJ.

A large filter will remove more noise from the image, but it will also remove some of the edges. A smaller filter, on the other hand, keeps more edges but also more noise.

The next step, **edge enhancement**, calculates the possible candidates for edges.

The code 2

```
1 #include <opencv2/highgui/highgui.hpp>
2 #include <iostream>
3
4 using namespace cv;
5 using namespace std;
6
7 enum SobelDirection
8 {
9     Diagonal_Right,
10    Diagonal_Left,
11    Vertical,
12    Horizontal
13 };
14
15 const int THRESHOLD_GRAYSCALE = 133; // optimal value was found using ←
    ImageJ
16 const int THRESHOLD_SOBEL = 100; // found by experimenting
17
18 Mat ConvertColorImageToBlackWhite(Mat colorImage);
19 Mat MeanFilter(Mat input);
20 Mat ThresholdBlackWhiteImage(Mat blackWhiteImage, int threshold);
21 Mat SobelEdgeDetecting(Mat input, enum SobelDirection direction, bool ←
    useMeanFilterBeforeDoingEdgeDetecting, int threshold);
22 Mat AddTwoMatsTogether(Mat matA, Mat matB);
23 Mat Erosion(Mat input, int radius);
24
25
26
27 int main()
28 {
29     // Program description
30     cout << "Edge detection using the Sobel kernel (and OpenCV to load ←
        images)" << endl;
31     cout << "By Gustav Dahl - Medialogy 3rd semester 2012 - Aalborg ←
        University\n\n";
32
33     // Load the original color image
34     Mat colorImage = imread("0_building.jpg");
35
36     if (colorImage.empty())
37     {
```

```
38     cout << "Cannot load image!" << endl;
39     return -1;
40 }
41
42 // "Loading" screen
43 cout << "Processing image. Please wait..." << endl;
44
45
46 // - - - - - APPLY IMAGE PROCESSING - - - - -
47 // Convert color image to grayscale
48 Mat gray = ConvertColorImageToBlackWhite(colorImage);
49
50 // Mean filter applied(black and white only)
51 Mat mean = MeanFilter(gray);
52
53 // Grayscale threshold
54 Mat threshold = ThresholdBlackWhiteImage(gray, THRESHOLD_GRAYSCALE)↵
55     ;
56
57 // Erosion
58 Mat erosion = Erosion(threshold, 1);
59
60 // Finding outline using the eroded image, by subtracting the ↵
61 // original grayscale from the eroded image
62 Mat erosionOutline = threshold - erosion;
63
64 // Edge detecting using the Sobel kernel
65 Mat edge_diagonal_right = SobelEdgeDetecting(gray, Diagonal_Right, ↵
66     true, THRESHOLD_SOBEL);
67 Mat edge_diagonal_left = SobelEdgeDetecting(gray, Diagonal_Left, ↵
68     true, THRESHOLD_SOBEL);
69 Mat edge_vertical = SobelEdgeDetecting(gray, Vertical, true, ↵
70     THRESHOLD_SOBEL);
71 Mat edge_horizontal = SobelEdgeDetecting(gray, Horizontal, true, ↵
72     THRESHOLD_SOBEL);
73
74 // Combine the different kernels
75 Mat vertical_plus_horizontal = AddTwoMatsTogether(edge_vertical, ↵
76     edge_horizontal);
77 Mat diagonal_right_plus_left = AddTwoMatsTogether(↵
78     edge_diagonal_right, edge_diagonal_left);
79 Mat diagonal_plus_vertical_horizontal = AddTwoMatsTogether(↵
80     vertical_plus_horizontal, edge_diagonal_right);
81
82 // - - - - -
83
84 // Save the images
85 imwrite("1_grayscale.jpg", gray);
86 imwrite("2_meanFilter.jpg", mean);
87 imwrite("3_threshold.jpg", threshold);
88 imwrite("4_erosion.jpg", erosion);
89 imwrite("5_erosionOutline.jpg", erosionOutline);
```

```

81  imwrite("6_edge_diagonal_right.jpg", edge_diagonal_right);
82  imwrite("7_edge_diagonal_left.jpg", edge_diagonal_left);
83  imwrite("8_edge_vertical.jpg", edge_vertical);
84  imwrite("9_edge_horizontal.jpg", edge_horizontal);
85  imwrite("10_vertical_plus_horizontal.jpg", vertical_plus_horizontal↵
    );
86  imwrite("11_diagonal_right_plus_left.jpg", diagonal_right_plus_left↵
    );
87  imwrite("12_diagonal_plus_vertical_horizontal.jpg", ↵
    diagonal_plus_vertical_horizontal);
88
89  // Show the images
90  imshow("original color image", colorImage);
91      imshow("grayscale", gray);
92  imshow("meanFilter", mean);
93  imshow("threshold", threshold);
94  imshow("erosion", erosion);
95  imshow("erosionOutline", erosionOutline);
96  imshow("edge_diagonal_right", edge_diagonal_right);
97  imshow("edge_diagonal_left", edge_diagonal_left);
98  imshow("edge_vertical", edge_vertical);
99  imshow("edge_horizontal", edge_horizontal);
100  imshow("vertical_plus_horizontal", vertical_plus_horizontal);
101  imshow("diagonal_right_plus_left", diagonal_right_plus_left);
102  imshow("diagonal_plus_vertical_horizontal", ↵
    diagonal_plus_vertical_horizontal);
103      waitKey(0);
104  }
105
106  Mat ConvertColorImageToBlackWhite(Mat colorImage)
107  {
108      Mat grayScaleImage(colorImage.rows, colorImage.cols, CV_8UC1); // ↵
    new image with only 1 channel
109
110      // Formula for converting from color to grayscale (3.3, p. 30 in ↵
    Introduction to Video and Image Processing book)
111      // I = weightR * R + weightG * G + weightB * B
112
113      // Common weight values used in TV production to calculate to ↵
    grayscale
114      float RedWeight = 0.299;
115      float GreenWeight = 0.587;
116      float BlueWeight = 0.114;
117
118      // Iterate through all the pixels and apply the formula for ↵
    grayscale
119      for (int y = 0; y < colorImage.rows; y++) // rows
120      {
121          for (int x = 0; x < colorImage.cols; x++)
122          {
123              // Calculate grayscale value
124              float grayValue = colorImage.at<cv::Vec3b>(y, x)[0] * ↵

```



```

        BlueWeight
125         + colorImage.at<cv::Vec3b>(y, x)[1] * GreenWeight
126         + colorImage.at<cv::Vec3b>(y, x)[2] * RedWeight;
127
128         // Apply the grayscale value (0-255)
129         grayScaleImage.at<uchar>(y, x) = grayValue;
130
131     }
132 }
133 return grayScaleImage;
134 }
135
136 Mat MeanFilter(Mat input)
137 {
138     // 3x3 kernel size
139
140     Mat mean = input.clone();
141
142     // Loop through all pixels
143     for (int y = 0; y < input.rows-2; y++)
144     {
145         for (int x = 0; x < input.cols-2; x++)
146         {
147             if (x - 2 < 0 || y - 2 < 0) // don't go out of bounds
148                 continue;
149
150             mean.at<uchar>(y, x) = (
151                 input.at<uchar>(y-2, x-2) + input.at<uchar>(y-2, x-1)
152                 + input.at<uchar>(y-2, x) + input.at<uchar>(y-2, x+1)
153                 + input.at<uchar>(y-2, x+2) + input.at<uchar>(y-1, x-2)
154                 + input.at<uchar>(y-1, x-1) + input.at<uchar>(y-1, x)
155                 + input.at<uchar>(y-1, x+1) + input.at<uchar>(y-1, x+2)
156                 + input.at<uchar>(y, x-2) + input.at<uchar>(y, x-1)
157                 + input.at<uchar>(y, x) + input.at<uchar>(y, x+1)
158                 + input.at<uchar>(y, x+2) + input.at<uchar>(y+1, x-2)
159                 + input.at<uchar>(y+1, x-1) + input.at<uchar>(y+1, x)
160                 + input.at<uchar>(y+1, x+1) + input.at<uchar>(y+1, x+2)
161                 + input.at<uchar>(y+2, x-2) + input.at<uchar>(y+2, x-1)
162                 + input.at<uchar>(y+2, x) + input.at<uchar>(y+2, x+1)
163                 + input.at<uchar>(y+2, x+2)
164             ) / 25;
165         }
166     }
167
168     return mean;
169 }
170
171 Mat ThresholdBlackWhiteImage(Mat blackWhiteImage, int threshold)
172 {
173     Mat image = blackWhiteImage.clone();
174
175     // Loop through all pixels and set them to either 255 (white) or 0 ←

```

```

    (black) using the threshold value
176 for (int y = 0; y < image.rows; y++)
177 {
178     for (int x = 0; x < image.cols; x++)
179     {
180         if (image.at<uchar>(y, x) >= threshold)
181             image.at<uchar>(y, x) = 255;
182         else
183             image.at<uchar>(y, x) = 0;
184     }
185 }
186
187 return image;
188 }
189
190 Mat SobelEdgeDetecting(Mat input, enum SobelDirection direction, bool↵
    useMeanFilterBeforeDoingEdgeDetecting, int threshold)
191 {
192     Mat edge = input.clone();
193
194     if (useMeanFilterBeforeDoingEdgeDetecting)
195         edge = MeanFilter(edge);
196
197     // Apply diagonal edge detecting RIGHT
198     if (direction == Diagonal_Right)
199     {
200         for (int y = 0; y < input.rows-1; y++)
201         {
202             for (int x = 0; x < input.cols-1; x++)
203             {
204                 if (x-1 < 0 || y-1 < 0) // don't go out of bounds
205                     continue;
206
207
208                 // temp value is used to not get overflow (value cannot be ↵
                    less than 0 or greater than 255)
209                 int temp = (
210                     (input.at<uchar>(y-1, x-1)) * -2
211                     + (input.at<uchar>(y, x-1)) * -1
212                     + (input.at<uchar>(y+1, x-1)) * 0
213                     + (input.at<uchar>(y-1, x)) * -1
214                     + (input.at<uchar>(y, x)) * 0
215                     + (input.at<uchar>(y+1, x+0)) * 1
216                     + (input.at<uchar>(y-1, x+1)) * 0
217                     + (input.at<uchar>(y, x+1)) * 1
218                     + (input.at<uchar>(y+1, x+1)) * 2
219                 );
220
221                 // Absolute value
222                 if (temp < 0)
223                     temp *= -1;
224

```

```

225     // Map values from 0 to 255
226     if (temp <= threshold)
227         temp = 0;
228     else
229         temp = 255;
230
231
232     edge.at<uchar>(y, x) = temp;
233 }
234 }
235 }
236 else if (direction == Diagonal_Left)
237 { // Apply diagonal edge detecting LEFT
238     for (int y = 0; y < input.rows-1; y++)
239     {
240         for (int x = 0; x < input.cols-1; x++)
241         {
242             if (x-1 < 0 || y-1 < 0) // don't go out of bounds
243                 continue;
244
245             // temp value is used to not get overflow (value cannot be <-
246             // less than 0 or greater than 255)
247             int temp = (
248                 (input.at<uchar>(y-1, x-1)) * -2
249                 + (input.at<uchar>(y, x-1)) * -1
250                 + (input.at<uchar>(y+1, x-1)) * 0
251                 + (input.at<uchar>(y-1, x)) * -1
252                 + (input.at<uchar>(y, x)) * 0
253                 + (input.at<uchar>(y+1, x+0)) * 1
254                 + (input.at<uchar>(y-1, x+1)) * 0
255                 + (input.at<uchar>(y, x+1)) * 1
256                 + (input.at<uchar>(y+1, x+1)) * 2
257             );
258
259             // Absolute value
260             if (temp < 0)
261                 temp *= -1;
262
263             // Map values from 0 to 255
264             if (temp <= threshold)
265                 temp = 0;
266             else
267                 temp = 255;
268
269             edge.at<uchar>(y, x) = temp;
270         }
271     }
272 }
273 else if (direction == Vertical)
274 {
275     // Apply diagonal edge detecting tVERTICAL

```

```

276     for (int y = 0; y < input.rows-1; y++)
277     {
278         for (int x = 0; x < input.cols-1; x++)
279         {
280             if (x-1 < 0 || y-1 < 0) // don't go out of bounds
281                 continue;
282
283             // temp value is used to not get overflow (value cannot be ←
                less than 0 or greater than 255)
284             int temp = (
285                 (input.at<uchar>(y-1, x-1)) * -1
286                 + (input.at<uchar>(y, x-1)) * -2
287                 + (input.at<uchar>(y+1, x-1)) * -1
288                 + (input.at<uchar>(y-1, x)) * -0
289                 + (input.at<uchar>(y, x)) * 0
290                 + (input.at<uchar>(y+1, x+0)) * 0
291                 + (input.at<uchar>(y-1, x+1)) * 1
292                 + (input.at<uchar>(y, x+1)) * 2
293                 + (input.at<uchar>(y+1, x+1)) * 1
294             );
295
296             // Absolute value
297             if (temp < 0)
298                 temp *= -1;
299
300             // Map values from 0 to 255
301             if (temp <= threshold)
302                 temp = 0;
303             else
304                 temp = 255;
305
306
307             edge.at<uchar>(y, x) = temp;
308         }
309     }
310 }
311 else if (direction == Horizontal)
312 {
313     // Apply diagonal edge detecting HORIZONTAL
314     for (int y = 0; y < input.rows-1; y++)
315     {
316         for (int x = 0; x < input.cols-1; x++)
317         {
318             if (x-1 < 0 || y-1 < 0) // don't go out of bounds
319                 continue;
320
321             // temp value is used to not get overflow (value cannot be ←
                less than 0 or greater than 255)
322             int temp = (
323                 (input.at<uchar>(y-1, x-1)) * -1
324                 + (input.at<uchar>(y, x-1)) * 0
325                 + (input.at<uchar>(y+1, x-1)) * 1

```

```

326         + (input.at<uchar>(y-1, x)) * -2
327         + (input.at<uchar>(y, x)) * 0
328         + (input.at<uchar>(y+1, x+0)) * 2
329         + (input.at<uchar>(y-1, x+1)) * -1
330         + (input.at<uchar>(y, x+1)) * 0
331         + (input.at<uchar>(y+1, x+1)) * 1
332     );
333
334     // Absolute value
335     if (temp < 0)
336         temp *= -1;
337
338     // Map values from 0 to 255
339     if (temp <= threshold)
340         temp = 0;
341     else
342         temp = 255;
343
344
345     edge.at<uchar>(y, x) = temp;
346 }
347 }
348 }
349 else
350 {
351     // Error text
352     putText(edge, "ERROR - Sobel type not defined!", Point(10, 50), ←
        FONT_HERSHEY_PLAIN, 2, Scalar(0, 0, 255), 4, 8, false);
353 }
354
355 // Threshold
356 edge = ThresholdBlackWhiteImage(edge, 30);
357 return edge;
358 }
359
360 Mat AddTwoMatsTogether(Mat matA, Mat matB) // should be same size!
361 {
362     Mat output = matA.clone();
363
364     for (int y = 0; y < matA.rows; y++)
365     {
366         for (int x = 0; x < matA.cols; x++)
367         {
368             output.at<uchar>(y, x) = matA.at<uchar>(y, x) + matB.at<uchar>(←
                y, x);
369         }
370     }
371 }
372
373 output = ThresholdBlackWhiteImage(output, THRESHOLD_GRAYSCALE);
374 return output;
375 }

```

```
376
377 // Uses a grayscale image
378 Mat Erosion(Mat input, int radius)
379 {
380     Mat output = input.clone();
381
382     for(int x = radius; x < input.cols-radius; x++)
383     {
384         for(int y = radius; y < input.rows-radius; y++)
385         {
386             bool pixelIsaccepted = true;
387             for(int filterX = x - radius; pixelIsaccepted && filterX <= x + radius; filterX++)
388             {
389                 for(int filterY = y - radius; pixelIsaccepted && filterY <= y + radius; filterY++)
390                 {
391                     if (input.at<uchar>(filterY,filterX) == 0)
392                     {
393                         pixelIsaccepted = false;
394                     }
395                 }
396             }
397             if (pixelIsaccepted == true)
398                 output.at<uchar>(y,x) = 255;
399             else
400                 output.at<uchar>(y,x) = 0;
401         }
402     }
403
404     return output;
405 }
```

Bibliography

Image of building used throughout the report. URL

<http://www.mccullagh.org/db9/1ds2-4/fbi-headquarters-building.jpg>.

Block, 2007. Bruce Block. *The Visual Story, Second Edition: Creating the Visual Structure of Film, TV and Digital Media*. ISBN: 978-0240807799. Focal Press, second edition edition, 2007.

Moeslund, 2012a. Thomas B. Moeslund. *Lecture AAU - Edge detection*, 2012.

Moeslund, 2012b. Thomas B. Moeslund. *Introduction to Video and Image Processing - Building real systems and applications*. ISBN: 978-1447125020, Handbook. Springer, 2012.

Snowden et al., 2012. Robert Snowden, Peter Thompson og Tom Troscianko. *Basic Vision: An Introduction to Visual Perception*. 978-0199572021. Oxford University Press, second edition edition, 2012.