Image Processing mini project

Diagonal Edge Detection in C++

Gustav Dahl, studienummer

Image Processing and Procedural Programming
Medialogy 3rd semester
Aalborg Universty 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

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, 2012] who writes that an edge in an image is defined as a position where there is a significant change in gray-level values.

1.2 Edge detection

An edge in a gray scale image occurs when there is a transition in gray level over an amount of pixels. A perfect edge would be a transition from black to white over one pixel as shown on figure 1. In many images, edges like these wont occur (unless itâĂŹs a binary image). The transition will be blurred spreading the transition over more pixels, resulting in a slope-like profile of the gray level transition seen on figure 2. Here the edge is spread over more pixels and will show as a wider edge instead of the 1-pixel transition that shows as a 1-pixel edge. We might say the thickness of the edge is defined by the length of the ramp that in the gray level profile.[1]

[1] FiXme Note: DETTE SKAL SKRIVES SELV!!

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 = 30; // 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 void SortValues(uchar* a, int size);
24
25
26
27 int main()
28 {
29   cout << "Edge detection using the Sobel kernel (and OpenCV to load images)" << endl;
30   cout << "By Gustav Dahl - Medialogy 3rd semester 2012 - Aalborg University\n\n";
31   // Load the image
32     Mat colorImage = imread("building.jpg");
33     if (colorImage.empty())
34     {
35         cout << "Cannot load image!" << endl;
36         return -1;
37     }
```

```
38
39  cout << "Processing image. Please wait..." << endl;
40
41  // Convert to black-white
42  Mat gray = ConvertColorImageToBlackWhite(colorImage);
43  cv::imwrite("grayscale.jpg", gray);
44
45  // Mean (black and white only)
46  Mat mean = MeanFilter(gray);
47  imwrite("mean.jpg", mean);
48
49  // Threshold
50  Mat threshold = ThresholdBlackWhiteImage(gray, THRESHOLD_GRAYSCALE)↩
        ;
51  imwrite("threshold.jpg", threshold);
52
53  // Edge detecting
54  Mat edge_diagonal_right = SobelEdgeDetecting(gray, Diagonal_Right, ↩
        true, THRESHOLD_SOBEL);
55  Mat edge_diagonal_left = SobelEdgeDetecting(gray, Diagonal_Left, ↩
        true, THRESHOLD_SOBEL);
56  Mat edge_vertical = SobelEdgeDetecting(gray, Vertical, true, ↩
        THRESHOLD_SOBEL);
57  Mat edge_horizontal = SobelEdgeDetecting(gray, Horizontal, true, ↩
        THRESHOLD_SOBEL);
58
59  Mat verti_plus_horiz = AddTwoMatsTogether(edge_vertical, ↩
        edge_horizontal);
60  Mat diagonal_right_plus_left = AddTwoMatsTogether(↩
        edge_diagonal_right, edge_diagonal_left);
61  Mat diagonal_plus_vertical_horizontal = AddTwoMatsTogether(↩
        verti_plus_horiz, diagonal_right_plus_left);
62
63  imwrite("diagonal_right.jpg", edge_diagonal_right);
64  imwrite("edge_diagonal_left.jpg", edge_diagonal_right);
65  imwrite("edge_vertical.jpg", edge_vertical);
66  imwrite("edge_horizontal.jpg", edge_horizontal);
67  imwrite("verti_plus_horiz.jpg", verti_plus_horiz);
68  imwrite("diagonal_right_plus_left.jpg", diagonal_right_plus_left);
69  imwrite("diagonal_plus_vertical_horizontal.jpg", ↩
        diagonal_plus_vertical_horizontal);
70
71    cv::imshow("Original image", colorImage);
72  cv::imshow("Grayscale image", gray);
73  cv::imshow("Mean filter", mean);
74  cv::imshow("Binary image", threshold);
75  cv::imshow("Edge diagonal right", edge_diagonal_right);
76  cv::imshow("Edge diagonal left", edge_diagonal_left);
77  cv::imshow("Edge vertical", edge_vertical);
78  cv::imshow("Edge horizontal", edge_horizontal);
79  cv::imshow("Diagonal plus vertical and horizontal", ↩
        diagonal_plus_vertical_horizontal);
```

```
80
81      waitKey(0);
82  }
83
84  Mat ConvertColorImageToBlackWhite(Mat colorImage)
85  {
86    // Mat(rows, columns, type)
87    // rows = y = 300
88    // cols = x = 600
89
90    Mat grayScaleImage(colorImage.rows, colorImage.cols, CV_8UC1); // ↩
        new image with only 1 channel
91
92    // Formula for converting from color to grayscale (3.3, p. 30 in ↩
        Introduction to Video and Image Processing book)
93    // I = weightR * R + weightG * G + weightB * B
94
95    // Common weight values used in TV production to calculate to ↩
        grayscale
96    float RedWeight = 0.299;
97    float GreenWeight = 0.587;
98    float BlueWeight = 0.114;
99
100
101   // Iterate through all the pixels and apply the formula for ↩
        grayscale
102   for (int y = 0; y < colorImage.rows; y++) // rows
103   {
104     for (int x = 0; x < colorImage.cols; x++)
105     {
106       // Calculate grayscale value
107       int grayValue = colorImage.at<cv::Vec3b>(y, x)[0] * BlueWeight
108         + colorImage.at<cv::Vec3b>(y, x)[1] * GreenWeight
109         + colorImage.at<cv::Vec3b>(y, x)[2] * RedWeight;
110
111       // Apply the grayscale value (0-255)
112       grayScaleImage.at<uchar>(y, x) = grayValue;
113
114     }
115   }
116   return grayScaleImage;
117 }
118
119 Mat MeanFilter(Mat input)
120 {
121   // 3x3 kernel size
122
123   Mat mean = input.clone();
124
125   // Loop through all pixels
126   for (int y = 0; y < input.rows-2; y++)
127   {
```

```
128     for (int x = 0; x < input.cols-2; x++)
129     {
130       if (x - 2 < 0 || y - 2 < 0) // don't go out of bounds
131         continue;
132 
133       mean.at<uchar>(y, x) = (
134         input.at<uchar>(y-2, x-2) + input.at<uchar>(y-2, x-1)
135         + input.at<uchar>(y-2, x) + input.at<uchar>(y-2, x+1)
136         + input.at<uchar>(y-2, x+2) + input.at<uchar>(y-1, x-2)
137         + input.at<uchar>(y-1, x-1) + input.at<uchar>(y-1, x)
138         + input.at<uchar>(y-1, x+1) + input.at<uchar>(y-1, x+2)
139         + input.at<uchar>(y, x-2) + input.at<uchar>(y, x-1)
140         + input.at<uchar>(y, x) + input.at<uchar>(y, x+1)
141         + input.at<uchar>(y, x+2) + input.at<uchar>(y+1, x-2)
142         + input.at<uchar>(y+1, x-1) + input.at<uchar>(y+1, x)
143         + input.at<uchar>(y+1, x+1) + input.at<uchar>(y+1, x+2)
144         + input.at<uchar>(y+2, x-2) + input.at<uchar>(y+2, x-1)
145         + input.at<uchar>(y+2, x) + input.at<uchar>(y+2, x+1)
146         + input.at<uchar>(y+2, x+2)
147         ) / 25;
148     }
149   }
150 
151   return mean;
152 }
153 
154 Mat ThresholdBlackWhiteImage(Mat blackWhiteImage, int threshold)
155 {
156   Mat image = blackWhiteImage.clone();
157 
158   // Loop through all pixels and set them to either 255 (white) or 0 ↩
        (black) using the threhold value
159   for (int y = 0; y < image.rows; y++)
160   {
161     for (int x = 0; x < image.cols; x++)
162     {
163       if (image.at<uchar>(y, x) >= threshold)
164         image.at<uchar>(y, x) = 255;
165       else
166         image.at<uchar>(y, x) = 0;
167     }
168   }
169 
170   return image;
171 }
172 
173 Mat SobelEdgeDetecting(Mat input, enum SobelDirection direction, bool↩
      useMeanFilterBeforeDoingEdgeDetecting, int threshold)
174 {
175   Mat edge = input.clone();
176 
177   if (useMeanFilterBeforeDoingEdgeDetecting)
```

```
178     edge = MeanFilter(edge);
179
180   // Apply diagonal edge detecting RIGHT
181   if (direction == Diagonal_Right)
182   {
183     for (int y = 0; y < input.rows-1; y++)
184     {
185       for (int x = 0; x < input.cols-1; x++)
186       {
187         if (x-1 < 0 || y-1 < 0) // don't go out of bounds
188           continue;
189
190
191         // temp value is used to not get overflow (value cannot be ↩
                less than 0 or greater than 255)
192         int temp = (
193           (input.at<uchar>(y-1, x-1)) * -2
194           + (input.at<uchar>(y, x-1)) * -1
195           + (input.at<uchar>(y+1, x-1)) * 0
196           + (input.at<uchar>(y-1, x)) * -1
197           + (input.at<uchar>(y, x)) * 0
198           + (input.at<uchar>(y+1, x+0)) * 1
199           + (input.at<uchar>(y-1, x+1)) * 0
200           + (input.at<uchar>(y, x+1)) * 1
201           + (input.at<uchar>(y+1, x+1)) * 2
202           );
203
204
205         // Map values from 0 to 255
206         if (temp < 0)
207           temp = 0;
208         else if (temp > 255)
209           temp = 255;
210
211
212         edge.at<uchar>(y, x) = temp;
213       }
214     }
215   }
216   else if (direction == Diagonal_Left)
217   { // Apply diagonal edge detecting LEFT
218     for (int y = 0; y < input.rows-1; y++)
219     {
220       for (int x = 0; x < input.cols-1; x++)
221       {
222         if (x-1 < 0 || y-1 < 0) // don't go out of bounds
223           continue;
224
225         // temp value is used to not get overflow (value cannot be ↩
                less than 0 or greater than 255)
226         int temp = (
227           (input.at<uchar>(y-1, x-1)) * -2
```

```
228           + (input.at<uchar>(y, x-1)) * -1
229           + (input.at<uchar>(y+1, x-1)) * 0
230           + (input.at<uchar>(y-1, x)) * -1
231           + (input.at<uchar>(y, x)) * 0
232           + (input.at<uchar>(y+1, x+0)) * 1
233           + (input.at<uchar>(y-1, x+1)) * 0
234           + (input.at<uchar>(y, x+1)) * 1
235           + (input.at<uchar>(y+1, x+1)) * 2
236           );
237
238         // Map values from 0 to 255
239         if (temp < 0)
240           temp = 0;
241         else if (temp > 255)
242           temp = 255;
243
244
245         edge.at<uchar>(y, x) = temp;
246       }
247     }
248   }
249   else if (direction == Vertical)
250   {
251     // Apply diagonal edge detecting tVERTICAL
252     for (int y = 0; y < input.rows-1; y++)
253     {
254       for (int x = 0; x < input.cols-1; x++)
255       {
256         if (x-1 < 0 || y-1 < 0) // don't go out of bounds
257           continue;
258
259         // temp value is used to not get overflow (value cannot be ↩
               less than 0 or greater than 255)
260         int temp = (
261           (input.at<uchar>(y-1, x-1)) * -1
262           + (input.at<uchar>(y, x-1)) * -2
263           + (input.at<uchar>(y+1, x-1)) * -1
264           + (input.at<uchar>(y-1, x)) * -0
265           + (input.at<uchar>(y, x)) * 0
266           + (input.at<uchar>(y+1, x+0)) * 0
267           + (input.at<uchar>(y-1, x+1)) * 1
268           + (input.at<uchar>(y, x+1)) * 2
269           + (input.at<uchar>(y+1, x+1)) * 1
270           );
271
272         // Map values from 0 to 255
273         if (temp < 0)
274           temp = 0;
275         else if (temp > 255)
276           temp = 255;
277
278
```

```
279         edge.at<uchar>(y, x) = temp;
280       }
281     }
282   }
283   else if (direction == Horizontal)
284   {
285     // Apply diagonal edge detecting HORIZONTAL
286     for (int y = 0; y < input.rows-1; y++)
287     {
288       for (int x = 0; x < input.cols-1; x++)
289       {
290         if (x-1 < 0 || y-1 < 0) // don't go out of bounds
291           continue;
292 
293         // temp value is used to not get overflow (value cannot be ↩
               less than 0 or greater than 255)
294         int temp = (
295           (input.at<uchar>(y-1, x-1)) * -1
296           + (input.at<uchar>(y, x-1)) * 0
297           + (input.at<uchar>(y+1, x-1)) * 1
298           + (input.at<uchar>(y-1, x)) * -2
299           + (input.at<uchar>(y, x)) * 0
300           + (input.at<uchar>(y+1, x+0)) * 2
301           + (input.at<uchar>(y-1, x+1)) * -1
302           + (input.at<uchar>(y, x+1)) * 0
303           + (input.at<uchar>(y+1, x+1)) * 1
304           );
305 
306         // Map values from 0 to 255
307         if (temp < 0)
308           temp = 0;
309         else if (temp > 255)
310           temp = 255;
311 
312 
313         edge.at<uchar>(y, x) = temp;
314       }
315     }
316   }
317   else
318   {
319     // Error text
320     putText(edge, "ERROR - Sobel type not defined!", Point(10, 50), ↩
          FONT_HERSHEY_PLAIN, 2, Scalar(0, 0, 255), 4, 8, false);
321     putText(edge, "ERROR - Sobel type not defined!", Point(10, 150), ↩
          FONT_HERSHEY_PLAIN, 2, Scalar(0, 0, 255), 4, 8, false);
322     putText(edge, "ERROR - Sobel type not defined!", Point(10, 250), ↩
          FONT_HERSHEY_PLAIN, 2, Scalar(0, 0, 255), 4, 8, false);
323     putText(edge, "ERROR - Sobel type not defined!", Point(10, 300), ↩
          FONT_HERSHEY_PLAIN, 2, Scalar(0, 0, 0), 4, 8, false);
324     putText(edge, "ERROR - Sobel type not defined!", Point(10, 450), ↩
          FONT_HERSHEY_PLAIN, 2, Scalar(0, 0, 0), 4, 8, false);
```

```
325     putText(edge, "ERROR - Sobel type not defined!", Point(10, 600), ↩
            FONT_HERSHEY_PLAIN, 2, Scalar(0, 0, 0), 4, 8, false);
326   }
327
328   // Threshold
329   edge = ThresholdBlackWhiteImage(edge, 30);
330   return edge;
331 }
332
333 Mat AddTwoMatsTogether(Mat matA, Mat matB) // should be same size!
334 {
335   Mat output = matA.clone();
336
337   for (int y = 0; y < matA.rows; y++)
338   {
339     for (int x = 0; x < matA.cols; x++)
340     {
341       output.at<uchar>(y, x) = matA.at<uchar>(y, x) + matB.at<uchar>(↩
              y, x);
342     }
343
344   }
345
346   output = ThresholdBlackWhiteImage(output, THRESHOLD_GRAYSCALE);
347   return output;
348 }
349
350 // Not working yet
351 /*void MedianFilter(Mat input, int kernelSize)
352 {
353   if (kernelSize % 2 == 0) // don't use even numbers
354     kernelSize++;
355
356   int radius = kernelSize / 2; // e.g. kernel is 7X7 --> radius is 3
357
358   uchar neighborPixels[9];
359
360   for (int y = radius; radius < (input.rows - radius); y++)
361   {
362     int counter = 0;
363     for (int x = radius; x < (input.cols - radius); x++)
364     {
365       // Done
366       if (counter >= 9)
367       {
368         //start sort
369         // put pixels back in
370       }
371       neighborPixels[x*y] = input.at<uchar>(y, x);
372       counter++;
373     }
374   }
```

```
}*/

// Not working yet
/*uchar* SortValues(uchar* a, int size)
{
  cout << "Before the bubblesort:" << endl;
  for (int i = 0; i < size; i++)
    cout << a[i] << endl;

  // Bubble list
  bool swapped = true;

  while (swapped)
  {
    swapped = false;
    for (int i = 1; i < size; i++)
    {
      if (a[i-1] > a[i])
      {
        // Sort numbers
        int temp = a[i-1];
        a[i-1] = a[i];
                a[i] = temp;

                swapped = true;
      }
    }
  }

  cout << "After the bubblesort:" << endl;
  for (int i = 0; i < size; i++)
    cout << a[i] << endl;

  uchar* aPointer = a;
  //return aPointer;
}*/
```

Bibliography

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

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