Sean Webster

Computer Vision

Homework 3

Due 4/4/2016

**Objective:** The objective of this assignment was to explore special-domain convolution while utilizing simulated images to determine the efficiency of the famous “Sobel” edge detection algorithm.

**Background:** The “Sobel” edge detection algorithm is a well-known algorithm for detecting edges. It works by applying a kernel operator along an image, multiplying its neighbor pixels by degrees to detect lines.

**Algorithm:**  To apply the Sobel kernel to the image, a buffering boundary must be created, due to the kernel being of odd dimensions. I did this by creating a buffer of zeros in the image array. The Sobel kernel was then applied by looping through each pixel, and applying the respective kernel with the equation

Gx = |(z3 + 2z6 + z9) – (z1 + 2z4 + z7)|

For the horizontal kernel, and

Gy = |(z7 + 2z8 + z9) – (z1 + 2z2 + z3)|

For the vertical kernel. Adding them together gave the complete Sobel image.

To calculate a threshold, any value above the thresholding value was pushed to 255, and the others to zero.

To calculate noise, a random number generator was applied to the image.

Calculating the Gaussian Blur was similar to the Sobel, in that it was a kernel that needed padding to be applied. The kernel equation was

1/16z1 + 1/8z2 + 1/16z3 + 1/8z4 + 1/4z5 + 1/8z6 + 1/16z7 + 1/8z8 + 1/16z9

The accuracy was calculated by dividing the number of matching pixels by the total amount.

Results:

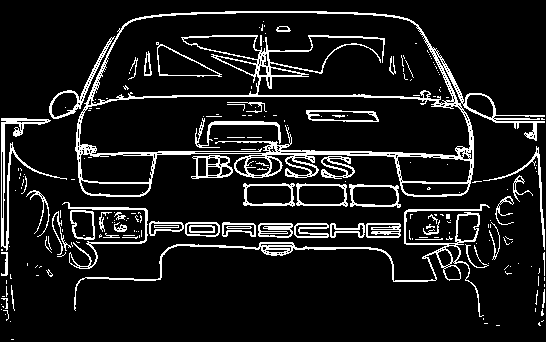
 

Figure 1(left). Sobel Image of Porsche.

Figure 2(right). Threshold of 60, Sobel of Porsche.

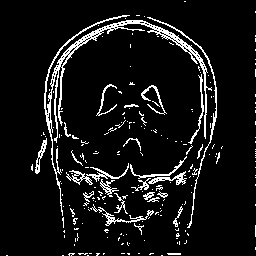
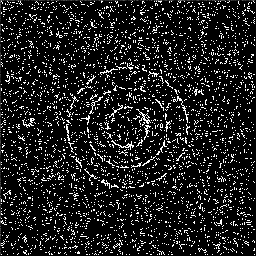
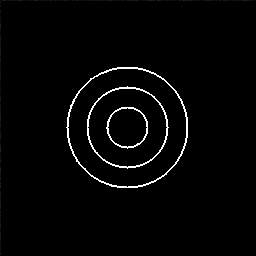
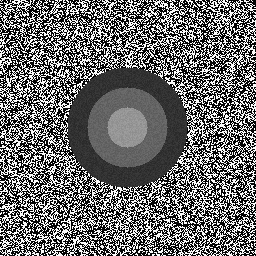


Figure 3(left). Sobel Image of MRI.

Figure 4(right). Threshold of 50, Sobel of MRI.



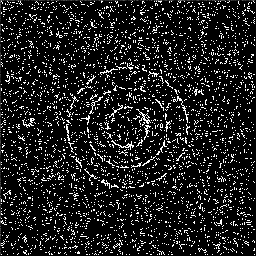
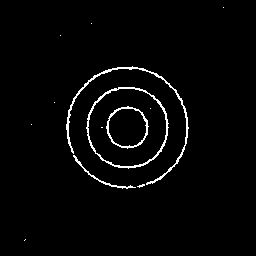
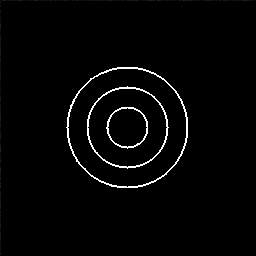
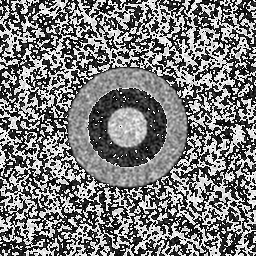
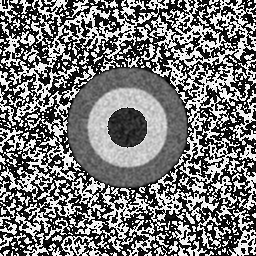
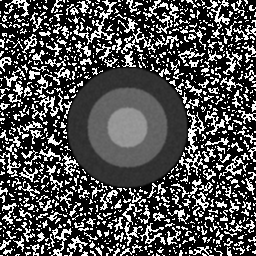
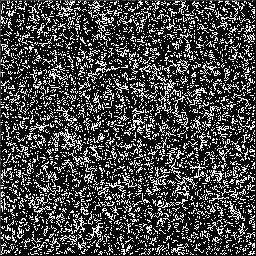
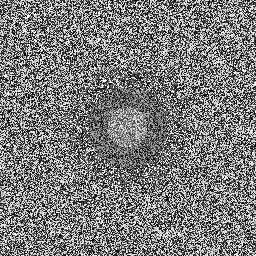
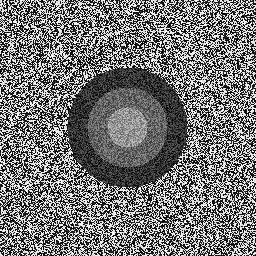
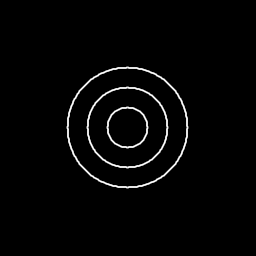
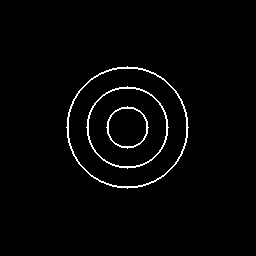
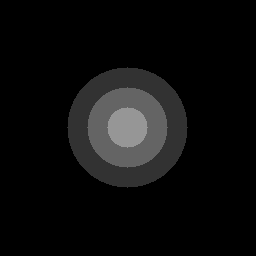


Figure 5-8(top row, left-right). Original Circle Image, Sobel of Circles, Thresholded Image, Circles with Noise at Mag. 10.

Figure 9-12(Second row, left-right). Circles with Noise at Mag. 50, Circles with Noise at Mag. 100, Sobel of Noise 10, Sobel of Noise 50

Figure 13-16(Third row, left-right). Sobel of Noise 100, Gaussian smoothed circles of Noise 10, Gaussian smoothed circles of Noise 50, Gaussian smoothed circles of Noise 100

Figure 17-20(Fourth row, left-right). Sobel of Gaussian smoothed circles of Noise 10, Sobel Of Gaussian smoothed circles of Noise 50, Sobel of Gaussian smoothed circles of Noise 100

\*note: all circle thresholds = 100

Table 1 – Noise Vs Accuracy

|  |  |  |
| --- | --- | --- |
|  | Accuracy | |
| Noise Magnitude | Before Smoothing | After Smoothing |
| 10 | 0.983932 | 0.983917 |
| 50 | 0.866837 | 0.986908 |
| 100 | 0.720505 | 0.972794 |

Chart 1. Noise Vs Accuracy

**Conclusions and Observations:**

Based on the results of the noisy Sobel circles, it was shown that Gaussian Smoothing does indeed improve problems that occur with noise. With little noise, the problem isn’t fixed as well as with lots of noise.

During this assignment, I thought that the laplacian of the image was the Sobel of it, but after further googling, I corrected myself. I also ran into many problems due to data type conversions, mostly being float numbers not working properly.

The Sobel Edge Detector Operator is quite a powerful operator, and the only one of its kind that I have used so far. I can see its shortcomings when it comes to noise, but beyond those, it works rather well.

**Readme:**

Sean Webster

Homework 3 readme

Source Code Files

HW3.c

arrays.c

Spacial.c

Circles.c

Header Files

arrays.h

Spacial.h

Circles.h

Executables

HW3

Instructions for running

Run program, then convert images out of raw format

Inputs

mri 256 x 256 65.5 KB

Porsche 546 x 342 183 KB

Outputs

mri\_sobel 256 x 256 65.5 KB

mri\_thresh50 256 x 256 65.5 KB

porsche\_sobel 546 x 342 183 KB

porsche\_thresh60 546 x 342 183 KB

circles 256 x 256 64 KB

circles\_sobel 256 x 256 64 KB

circles\_thresh100 256 x 256 64 KB

circles\_noise10 256 x 256 64 KB

circles\_noise50 256 x 256 64 KB

circles\_noise100 256 x 256 64 KB

circles\_noise\_gauss10 256 x 256 64 KB

circles\_noise\_gauss50 256 x 256 64 KB

circles\_noise\_gauss100 256 x 256 64 KB

circles\_noise\_gauss\_thresh10 256 x 256 64 KB

circles\_noise\_gauss\_thresh50 256 x 256 64 KB

circles\_noise\_gauss\_thresh100 256 x 256 64 KB

XV Commands

(echo P5; echo 256 256; echo 255; cat mri\_sobel) xv -

(echo P5; echo 256 256; echo 255; cat mri\_thresh50) xv -

(echo P5; echo 546 342; echo 255; cat porsche\_sobel) xv -

(echo P5; echo 546 342; echo 255; cat porsche\_thresh60) xv -

(echo P5; echo 256 256; echo 255; cat circles) xv -

(echo P5; echo 256 256; echo 255; cat circles\_sobel) xv -

(echo P5; echo 256 256; echo 255; cat circles\_thresh100) xv -

(echo P5; echo 256 256; echo 255; cat circles\_noise10) xv -

(echo P5; echo 256 256; echo 255; cat circles\_noise50) xv -

(echo P5; echo 256 256; echo 255; cat circles\_noise100) xv -

(echo P5; echo 256 256; echo 255; cat circles\_noise\_gauss10) xv -

(echo P5; echo 256 256; echo 255; cat circles\_noise\_gauss50) xv -

(echo P5; echo 256 256; echo 255; cat circles\_noise\_gauss100) xv -

(echo P5; echo 256 256; echo 255; cat circles\_noise\_gauss\_thresh10) xv -

(echo P5; echo 256 256; echo 255; cat circles\_noise\_gauss\_thresh50) xv -

(echo P5; echo 256 256; echo 255; cat circles\_noise\_gauss\_thresh100) xv -

**Part 1:**

1. /\*--------------- HW3.c ---------------
2. by: Sean Webster
3. Computer Vision
4. Due 4/4/2016
5. ---------------------------------------------\*/
6. #include <math.h>
7. #include <stdlib.h>
8. #include <stdio.h>
9. #include <string.h>
10. #include "arrays.h"
11. #include "Spacial.h"
12. #include "Circles.h"
13. int main()
14. {
15. /\*----------------------------Part 1------------------------------------\*/
16. int ysize = 342;
17. int xsize = 546;
18. int padding = 1;
19. float \*\*inImage = ImageInit(xsize, ysize);
20. float \*\*padImage = ImageInit(xsize + 2\*padding, ysize + 2\*padding);
21. float \*\*sobelImage = ImageInit(xsize, ysize);
22. float \*\*threshImage = ImageInit(xsize, ysize);
24. OpenImage("porsche", xsize, ysize, inImage);
25. PadImage(inImage, padImage, xsize, ysize, padding);
26. Sobel(padImage, sobelImage, xsize + 2\*padding, ysize + 2\*padding);
27. SaveImage("porsche", "\_sobel", xsize, ysize, sobelImage);
28. BinThresh(sobelImage, threshImage, xsize, ysize, 60);
29. SaveImage("porsche", "\_thresh60", xsize, ysize, threshImage);
30. ysize = 256;
31. xsize = 256;
32. OpenImage("mri", xsize, ysize, inImage);
33. PadImage(inImage, padImage, xsize, ysize, padding);
34. Sobel(padImage, sobelImage, xsize + 2 \* padding, ysize + 2 \* padding);
35. SaveImage("mri", "\_sobel", xsize, ysize, sobelImage);
36. BinThresh(sobelImage, threshImage, xsize, ysize, 50);
37. SaveImage("mri", "\_thresh50", xsize, ysize, threshImage);
38. free(inImage);
39. free(padImage);
40. free(sobelImage);
41. free(threshImage);
42. /\*----------------------------Part 2------------------------------------\*/
43. xsize = ysize = 256;
44. float \*\*circImage = ImageInit(xsize, ysize);
45. float \*\*padCirc = ImageInit(xsize + 2 \* padding, ysize + 2 \* padding);
46. float \*\*sobelCirc = ImageInit(xsize, ysize);
47. float \*\*threshCirc = ImageInit(xsize, ysize);
48. float \*\*noiseCirc = ImageInit(xsize, ysize);
49. float \*\*padNoiseCirc = ImageInit(xsize + 2 \* padding, ysize + 2 \* padding);
50. float \*\*sobelNoiseCirc = ImageInit(xsize, ysize);
51. float \*\*noiseGaussCirc = ImageInit(xsize, ysize);
52. float \*\*padNoiseGaussCirc = ImageInit(xsize + 2 \* padding, ysize + 2 \* padding);
53. float \*\*sobelNoiseGaussCirc = ImageInit(xsize, ysize);
54. float \*\*sobelNoiseGaussThreshCirc = ImageInit(xsize, ysize);
55. float \*\*test = ImageInit(xsize + 2, ysize +2);
56. int c = 256 / 2 - 1;
57. CreateCircle(circImage, ysize, xsize, c, c, 50, 60);
58. CreateCircle(circImage, ysize, xsize, c, c, 50, 40);
59. CreateCircle(circImage, ysize, xsize, c, c, 50, 20);
60. SaveImage("circles", "", xsize, ysize, circImage);
61. PadImage(circImage, padCirc, xsize, ysize, padding);
62. Sobel(padCirc, sobelCirc, xsize + 2 \* padding, ysize + 2 \* padding);
63. SaveImage("circles", "\_sobel", xsize, ysize, sobelCirc);
64. BinThresh(sobelCirc, threshCirc, xsize, ysize, 100);
65. SaveImage("circles", "\_thresh100", xsize, ysize, threshCirc);
66. // Noise = 10
67. AddUniNoise(circImage, noiseCirc, xsize, ysize, 10);
68. SaveImage("circles", "\_noise10", xsize, ysize, noiseCirc);
69. PadImage(noiseCirc, padNoiseCirc, xsize, ysize, padding);
70. Sobel(padNoiseCirc, sobelNoiseCirc, xsize + 2 \* padding, ysize + 2 \* padding);
71. BinThresh(sobelNoiseCirc, sobelNoiseCirc, xsize, ysize, 100);
72. SaveImage("circles", "\_noise\_sobel10", xsize, ysize, sobelNoiseCirc);
73. AccuracyReport(threshCirc, sobelNoiseCirc, xsize, ysize);
74. GaussianSmooth(padNoiseCirc, noiseGaussCirc, xsize + 2 \* padding, ysize + 2 \* padding);
75. SaveImage("circles", "\_noise\_gauss10", xsize, ysize, noiseGaussCirc);
76. PadImage(noiseGaussCirc, padNoiseGaussCirc, xsize, ysize, padding);
77. Sobel(padNoiseGaussCirc, sobelNoiseGaussCirc, xsize + 2 \* padding, ysize + 2 \* padding);
78. BinThresh(sobelNoiseGaussCirc, sobelNoiseGaussThreshCirc, xsize, ysize, 100);
79. SaveImage("circles", "\_noise\_sobel\_gauss\_thresh10", xsize, ysize, sobelNoiseGaussThreshCirc);
80. AccuracyReport(threshCirc, sobelNoiseGaussThreshCirc, xsize, ysize);
81. // Noise = 50
82. AddUniNoise(circImage, noiseCirc, xsize, ysize, 50);
83. SaveImage("circles", "\_noise50", xsize, ysize, noiseCirc);
84. PadImage(noiseCirc, padNoiseCirc, xsize, ysize, padding);
85. Sobel(padNoiseCirc, sobelNoiseCirc, xsize + 2 \* padding, ysize + 2 \* padding);
86. BinThresh(sobelNoiseCirc, sobelNoiseCirc, xsize, ysize, 100);
87. SaveImage("circles", "\_noise\_sobel50", xsize, ysize, sobelNoiseCirc);
88. AccuracyReport(threshCirc, sobelNoiseCirc, xsize, ysize);
89. GaussianSmooth(padNoiseCirc, noiseGaussCirc, xsize + 2 \* padding, ysize + 2 \* padding);
90. SaveImage("circles", "\_noise\_gauss50", xsize, ysize, noiseGaussCirc);
91. PadImage(noiseGaussCirc, padNoiseGaussCirc, xsize, ysize, padding);
92. Sobel(padNoiseGaussCirc, sobelNoiseGaussCirc, xsize + 2 \* padding, ysize + 2 \* padding);
93. BinThresh(sobelNoiseGaussCirc, sobelNoiseGaussThreshCirc, xsize, ysize, 100);
94. SaveImage("circles", "\_noise\_sobel\_gauss\_thresh50", xsize, ysize, sobelNoiseGaussThreshCirc);
95. AccuracyReport(threshCirc, sobelNoiseGaussThreshCirc, xsize, ysize);
96. // Noise = 100
97. AddUniNoise(circImage, noiseCirc, xsize, ysize, 100);
98. SaveImage("circles", "\_noise100", xsize, ysize, noiseCirc);
99. PadImage(noiseCirc, padNoiseCirc, xsize, ysize, padding);
100. Sobel(padNoiseCirc, sobelNoiseCirc, xsize + 2 \* padding, ysize + 2 \* padding);
101. BinThresh(sobelNoiseCirc, sobelNoiseCirc, xsize, ysize, 100);
102. SaveImage("circles", "\_noise\_sobel100", xsize, ysize, sobelNoiseCirc);
103. AccuracyReport(threshCirc, sobelNoiseCirc, xsize, ysize);
104. GaussianSmooth(padNoiseCirc, noiseGaussCirc, xsize + 2 \* padding, ysize + 2 \* padding);
105. SaveImage("circles", "\_noise\_gauss100", xsize, ysize, noiseGaussCirc);
106. PadImage(noiseGaussCirc, padNoiseGaussCirc, xsize, ysize, padding);
107. Sobel(padNoiseGaussCirc, sobelNoiseGaussCirc, xsize + 2 \* padding, ysize + 2 \* padding);
108. BinThresh(sobelNoiseGaussCirc, sobelNoiseGaussThreshCirc, xsize, ysize, 100);
109. SaveImage("circles", "\_noise\_sobel\_gauss\_thresh100", xsize, ysize, sobelNoiseGaussThreshCirc);
110. AccuracyReport(threshCirc, sobelNoiseGaussThreshCirc, xsize, ysize);
111. free(circImage);
112. free(padCirc);
113. free(sobelCirc);
114. free(threshCirc);
115. free(noiseCirc);
116. free(padNoiseCirc);
117. free(sobelNoiseCirc);
118. free(noiseGaussCirc);
119. free(padNoiseGaussCirc);
120. free(sobelNoiseGaussCirc);
121. free(sobelNoiseGaussThreshCirc);
122. return 0;
123. }
124. /\*--------------- Spacial.h ---------------
126. by: Sean Webster
128. PURPOSE
129. function prototypes for Spacial.c
130. ---------------------------------------------\*/
131. #include "arrays.h"
132. #include <math.h>
133. #include <stdlib.h>
134. #include <stdio.h>
135. #include <float.h>
136. #ifndef SPACIAL\_H
137. #define SPACIAL\_H
138. void PadImage(float \*\*oData, float \*\*nData, int xsize, int ysize, int pSize);
139. void Sobel(float \*\*inData, float \*\*lineData, int xsize, int ysize);
140. void BinThresh(float \*\*inData, float \*\*outData, int xsize, int ysize, int thresh);
141. void AddUniNoise(float\*\* inData, float \*\*outData, int xsize, int ysize, int intens);
142. void GaussianSmooth(float \*\*inData, float \*\*outData, int xsize, int ysize);

#endif

1. /\*--------------- Spacial.c ---------------
2. PURPOSE: Holds functions for spacial image manipulation
3. by: Sean Webster
4. Computer Vision
5. Due 4/4/2016
6. ---------------------------------------------\*/
7. #include "Spacial.h"
8. /\*-------------------- PadImage()-------------------------------------
9. PURPOSE
10. Pads image, adding 0's for spacial filtering
11. INPUT PARAMETERS
12. -inData: old 2D array to be input
13. -outData: new 2D array to be input
14. -xsize: size of array in x direction
15. -ysize: size of array in y direction
16. --------------------------------------------------------------------\*/
17. void PadImage(float \*\*inData, float \*\*outData, int xsize, int ysize, int pSize)
18. {
19. int i, j;
20. for (i = 0; i < ysize; i++)
21. {
22. for (j = 0; j < xsize; j++)
23. {
24. outData[i + pSize][j + pSize] = inData[i][j];
25. }
26. }
27. }
28. /\*-------------------- Sobel()-------------------------------------
29. PURPOSE
30. Masks image with sobel operator
31. INPUT PARAMETERS
32. -inData: old 2D array to be input
33. -outData: new 2D array to be input
34. -xsize: size of array in x direction
35. -ysize: size of array in y direction
36. --------------------------------------------------------------------\*/
37. void Sobel(float \*\*inData, float \*\*outData, int xsize, int ysize)
38. {
39. float \*\*temp1 = ImageInit(xsize, ysize);
40. float \*\*temp2 = ImageInit(xsize, ysize);
42. int vertKernel[3][3] = { { -1, 0, 1 },
43. { -2, 0, 2 },
44. { -1, 0, 1 } };
45. int horzKernel[3][3] = { { 1, 2, 1 },
46. { 0, 0, 0 },
47. { -1, -2, -1 } };
48. int x, y, i, j;
49. double pixel\_value;
50. double max = -DBL\_MAX;
51. //Vertical Sobel Mask
52. for (y = 1; y < ysize - 1; y++)
53. {
54. for (x = 1; x < xsize - 1; x++)
55. {
56. pixel\_value = 0.0;
57. for (j = -1; j <= 1; j++) {
58. for (i = -1; i <= 1; i++) {
59. pixel\_value += vertKernel[j + 1][i + 1] \* inData[y + j][x + i];
60. }
61. }
63. temp1[y - 1][x - 1] = pixel\_value;
64. }
65. }
66. // Horizontal Sobel Mask
67. for (y = 1; y < ysize - 1; y++)
68. {
69. for (x = 1; x < xsize - 1; x++)
70. {
71. pixel\_value = 0.0;
72. for (j = -1; j <= 1; j++) {
73. for (i = -1; i <= 1; i++) {
74. pixel\_value += horzKernel[j + 1][i + 1] \* inData[y + j][x + i];
75. }
76. }
77. temp2[y - 1][x - 1] = pixel\_value;
78. }
79. }
80. // Take magnitude
81. for (y = 0; y < ysize - 3; y++)
82. {
83. for (x = 0; x < xsize - 3; x++)
84. {
85. outData[y][x] = sqrt(pow(temp1[y][x], 2) + pow(temp2[y][x], 2));
86. // Get max for normalizing
87. if (outData[y][x] > max) max = outData[y][x];
88. }
89. }
90. // Normalize
91. for (y = 0; y < ysize - 3; y++)
92. {
93. for (x = 0; x < xsize - 3; x++)
94. {
95. outData[y][x] = 255 \* outData[y][x] / max;
96. }
97. }
98. free(temp1);
99. free(temp2);
100. }
101. /\*-------------------- BinThresh()-------------------------------------
102. PURPOSE
103. Takes binary threshold of image
104. INPUT PARAMETERS
105. -inData: old 2D array to be input
106. -outData: new 2D array to be input
107. -xsize: size of array in x direction
108. -ysize: size of array in y direction
109. -thresh: threshold value
110. --------------------------------------------------------------------\*/
111. void BinThresh(float \*\*inData, float \*\*outData, int xsize, int ysize, int thresh)
112. {
113. int x, y;
114. for (y = 1; y < ysize - 1; y++)
115. {
116. for (x = 1; x < xsize - 1; x++)
117. {
118. if (inData[y][x] > thresh)
119. outData[y][x] = 255;
120. else
121. outData[y][x] = 0;
122. }
123. }
124. }
125. /\*-------------------- AddUniNoise()-------------------------------------
126. PURPOSE
127. Adds uniform noise to image
128. INPUT PARAMETERS
129. -inData: old 2D array to be input
130. -outData: new 2D array to be input
131. -xsize: size of array in x direction
132. -ysize: size of array in y direction
133. -intens: intensity of noise
134. --------------------------------------------------------------------\*/
135. void AddUniNoise(float\*\* inData, float \*\*outData, int xsize, int ysize, int intens)
136. {
137. int r;
138. int i = intens;
139. float noise, k;
140. int x, y;
141. for (y = 0; y < ysize; y++)
142. {
143. for (x = 0; x < xsize; x++)
144. {
145. r = rand();
146. k = (2 \* r - RAND\_MAX);
147. k /= RAND\_MAX;
148. noise = k\*i;
149. outData[y][x] = inData[y][x] + noise;
150. }
151. }
152. }
153. /\*-------------------- GaussianSmooth()-------------------------------------
154. PURPOSE
155. Applies gaussian smoothing operator to image
156. INPUT PARAMETERS
157. -inData: old 2D array to be input
158. -outData: new 2D array to be input
159. -xsize: size of array in x direction
160. -ysize: size of array in y direction
161. --------------------------------------------------------------------\*/
162. void GaussianSmooth(float \*\*inData, float \*\*outData, int xsize, int ysize)
163. {
164. float gaussKernel[3][3] = { { 1.0/16, 1.0/8, 1.0/16 },
165. { 1.0/8 , 1.0/4, 1.0/8 },
166. { 1.0/16, 1.0/8, 1.0/16 } };
167. int x, y, i, j;
168. double pixel\_value;
169. double max, min;
170. min = DBL\_MAX;
171. max = -DBL\_MAX;
172. for (y = 1; y < ysize - 1; y++)
173. {
174. for (x = 1; x < xsize - 1; x++)
175. {
176. pixel\_value = 0.0;
177. for (j = -1; j <= 1; j++) {
178. for (i = -1; i <= 1; i++) {
179. pixel\_value += gaussKernel[j + 1][i + 1] \* inData[y + j][x + i];
180. }
181. }
183. outData[y - 1][x - 1] += pixel\_value;
184. }
185. }
186. /\*--------------- Circles.h ---------------
187. by: Sean Webster
188. PURPOSE
189. function prototypes for Circles.c
190. ---------------------------------------------\*/
191. #include <math.h>
192. #ifndef CIRCLES\_H
193. #define CIRCLES\_H
194. void CreateCircles(float \*\*data, int height, int width, int xloc, int yloc, int intensity, int rad);
195. #endif
196. /\*--------------- Circles.c ---------------
197. PURPOSE: Creates Circles
198. by: Sean Webster
199. Computer Vision
200. Due 4/4/2016
201. ---------------------------------------------\*/
202. #include "Circles.h"
203. /\*-------------------- CreateCircle()-------------------------------------
204. PURPOSE
205. Creates a circle in image
206. INPUT PARAMETERS
207. -inData: old 2D array to be input
208. -outData: new 2D array to be input
209. -xsize: size of array in x direction
210. -ysize: size of array in y direction
211. -xloc: x location of circle
212. -yloc: y location of circle
213. -intensity: intensity of circle
214. -radius: radius of circle
215. --------------------------------------------------------------------\*/
216. void CreateCircle(float \*\*data, int ysize, int xsize, int xloc, int yloc, int intensity, int rad)
217. {
218. int x, y;
219. for (y = 0; y < ysize; y++)
220. {
221. for (x = 0; x < xsize; x++)
222. {
223. if (pow(x - xloc, 2) + pow(y - yloc, 2) - pow(rad, 2) <= 0)
224. data[y][x] += intensity;
225. }
226. }
227. }
228. /\*--------------- arrays.h ---------------
229. by: Sean Webster
230. PURPOSE
231. function prototypes for arrays.c
232. ---------------------------------------------\*/
233. #include <math.h>
234. #include <stdlib.h>
235. #include <stdio.h>
236. #include <string.h>
237. #ifndef ARRAYS\_H
238. #define ARRAYS\_H
239. float \*\*ImageInit(int xsize, int ysize);
240. int OpenImage(char \*fn, int xsize, int ysize, float \*\*data);
241. void SaveImage(char \*fn, char\* out, int xsize, int ysize, float \*\*data\_ptr);
242. void SaveImageAsText(float \*fn, char\* out, int xsize, int ysize, float \*\*data\_ptr);
243. void AccuracyReport(float \*\*image1, float \*\*image2, int xsize, int ysize);
244. #endif
245. /\*--------------- arrays.c ---------------
246. by: Sean Webster
247. PURPOSE
248. Array based functions for inputting images
249. ---------------------------------------------\*/
250. #include "arrays.h"
251. /\*-------------------- ImageInit()-------------------------------------
252. PURPOSE
253. Initializes image float array in memory and to 0
254. INPUT PARAMETERS
255. -xsize: size of array in x direction
256. -ysize: size of array in y direction
257. --------------------------------------------------------------------\*/
258. float \*\*ImageInit(int xsize, int ysize)
259. {
260. float \*\*data;
261. int i, j;
262. data = (float \*\*)malloc(sizeof(float \*) \* ysize);
263. for (i = 0; i < ysize; i++)
264. {
265. data[i] = (float \*)malloc(sizeof(float) \* xsize);
266. }
267. for (i = 0; i < ysize; i++)
268. {
269. for (j = 0; j < xsize; j++)
270. {
271. data[i][j] = 0;
272. }
273. }
274. return data;
275. }
276. /\*-------------------- OpenImage()-------------------------------------
277. PURPOSE
278. Reads a raw image file into a 2D array
279. INPUT PARAMETERS
280. -data: 2D array to be input
281. -xsize: size of array in x direction
282. -ysize: size of array in y direction
283. -fn: name of input image
284. --------------------------------------------------------------------\*/
285. int OpenImage(char \*fn, int xsize, int ysize, float \*\*data)
286. {
287. int j;
288. int k;
289. FILE \*fp\_inp;
290. unsigned char pixel;
291. // Read image contents into 2D array
292. if ((fp\_inp = fopen(fn, "rb")) == NULL) return -1;
293. for (j = 0; j < ysize; j++)
294. {
295. for (k = 0; k < xsize; k++)
296. {
297. fread(&pixel, sizeof(char), 1, fp\_inp);
298. data[j][k] = (float)pixel;
299. }
300. }
301. (void)fclose(fp\_inp);
302. return 0;
303. }
304. /\*-------------------- SaveImage()-------------------------------------
305. PURPOSE
306. Writes a 2D array to a raw image file
307. INPUT PARAMETERS
308. -data: 2D array to be input
309. -xsize: size of array in x direction
310. -ysize: size of array in y direction
311. -fn: name of input image
312. -out: string to append to output image
313. --------------------------------------------------------------------\*/
314. void SaveImage(float \*fn, char\* out, int xsize, int ysize, float \*\*data\_ptr)
315. {
316. int j, k;
317. FILE \*out\_file;
318. char out\_name[40];
319. char name[15];
320. unsigned char pixel;
321. strcpy(name, fn);
322. strcpy(out\_name, name);
323. strcat(out\_name, out);
324. out\_file = fopen(out\_name, "wb");
325. for (j = 0; j < ysize; j++)
326. {
327. for (k = 0; k < xsize; k++)
328. {
329. pixel = (unsigned char)data\_ptr[j][k];
330. fwrite(&pixel, sizeof(char), 1, out\_file);
331. }
332. }
333. (void)fclose(out\_file);
334. }
335. void SaveImageAsText(float \*fn, char\* out, int xsize, int ysize, float \*\*data\_ptr)
336. {
337. int j, k;
338. FILE \*out\_file;
339. char out\_name[20];
340. char name[15];
341. unsigned char pixel;
342. strcpy(name, fn);
343. strcpy(out\_name, name);
344. strcat(out\_name, out);
345. strcat(out\_name, ".txt");
346. out\_file = fopen(out\_name, "wb");
347. for (j = 0; j < ysize - 1; j++)
348. {
349. for (k = 0; k < xsize - 1; k++)
350. {
351. pixel = (unsigned char)data\_ptr[j][k];
352. fwrite(&pixel, sizeof(char), 1, out\_file);
353. }
354. fwrite("\n", sizeof(char), 1, out\_file);
355. }
356. (void)fclose(out\_file);
357. }
358. /\*-------------------- SaveImage()-------------------------------------
359. PURPOSE
360. Writes to console report of accuracy
361. INPUT PARAMETERS
362. -image1: 2D image array to be compared
363. -image2: 2D image array to be compared
364. -xsize: size of array in x direction
365. -ysize: size of array in y direction
366. --------------------------------------------------------------------\*/
367. void AccuracyReport(float \*\*image1, float \*\*image2, int xsize, int ysize)
368. {
369. int x, y;
370. double m = xsize\*ysize;
371. float accuracy = 0;
372. for (y = 0; y < ysize - 1; y++)
373. {
374. for (x = 0; x < xsize - 1; x++)
375. {
376. if (image1[y][x] == image2[y][x]) accuracy++;
377. }
378. }
379. printf("Image2 contains %lf of the same pixels as Image2\n", accuracy / m);
380. }