



(a) Original



(b) 2x2 Roberts Cross Operator
Execution Time: 0.2704 seconds



(c) 3x3 Sobel Operator
Execution Time: 0.2663 seconds



(d) Mean Blur, Roberts
Execution Time: 0.2729 seconds



(e) Mean Blur, Sobel
Execution Time: 0.2856 seconds



(f) Gaussian Blur, Roberts
Execution Time: 0.2753 seconds

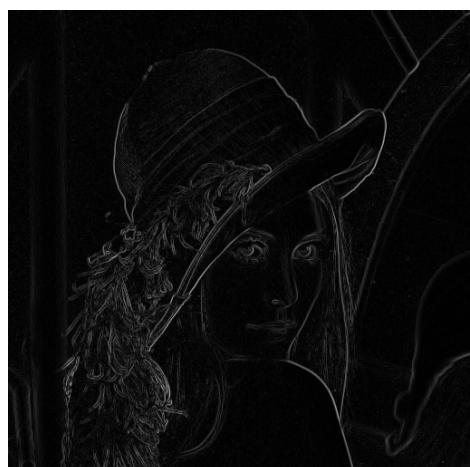


(g) Gaussian Blur, Sobel
Execution Time: 0.3292 seconds

Figure 1: Cameraman



(a) Original



(b) 2x2 Roberts Cross Operator
Execution Time: 0.2463 seconds



(c) 3x3 Sobel Operator
Execution Time: 0.2543 seconds



(d) Mean Blur, Roberts
Execution Time: 0.2796 seconds



(e) Mean Blur, Sobel
Execution Time: 0.3032 seconds

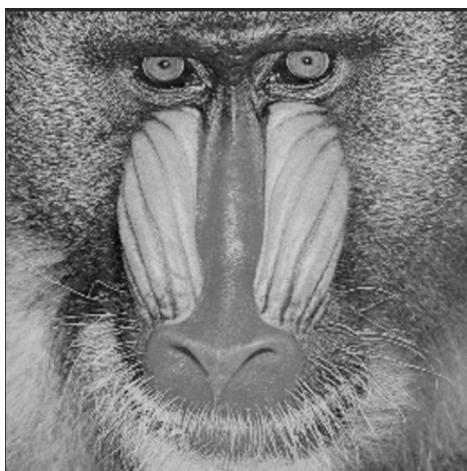


(f) Gaussian Blur, Roberts
Execution Time: 0.2892 seconds

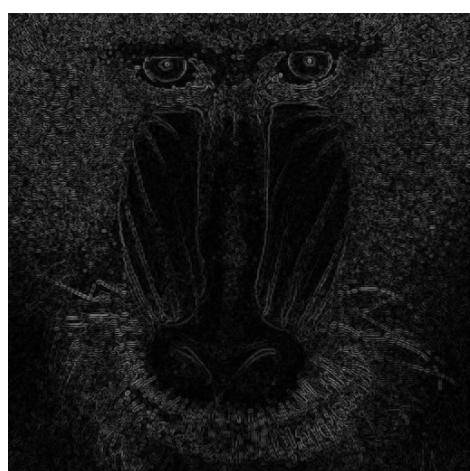


(g) Gaussian Blur, Sobel
Execution Time: 0.2942 seconds

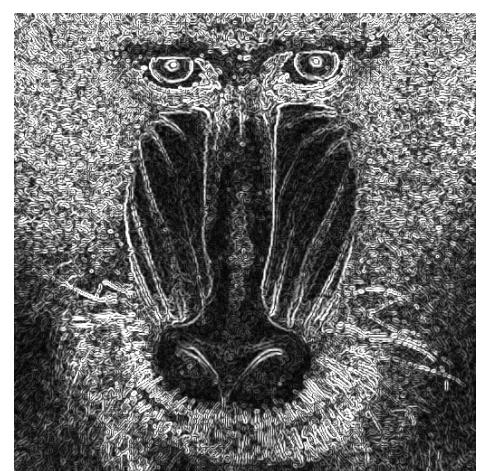
Figure 2: Lena



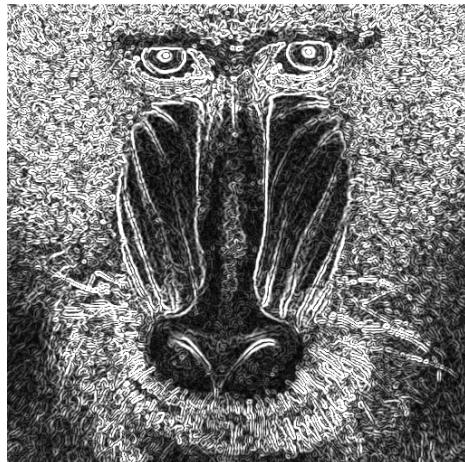
(a) Original



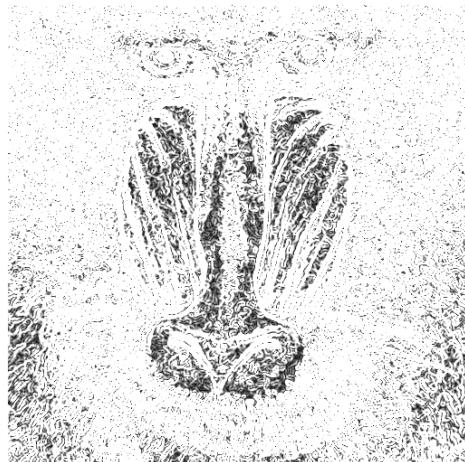
(b) 2x2 Roberts Cross Operator
Execution Time: 0.2424 seconds



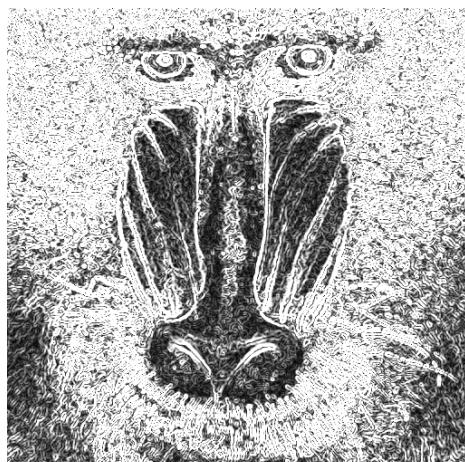
(c) 3x3 Sobel Operator
Execution Time: 0.2733 seconds



(d) Mean Blur, Roberts
Execution Time: 0.3880 seconds



(e) Mean Blur, Sobel
Execution Time: 0.4073 seconds



(f) Gaussian Blur, Roberts
Execution Time: 0.3293 seconds



(g) Gaussian Blur, Sobel
Execution Time: 0.3358 seconds

Figure 3: Mandril



(a) Original



(b) 2x2 Roberts Cross Operator
Execution Time: 0.2403 seconds



(c) 3x3 Sobel Operator
Execution Time: 0.2603 seconds



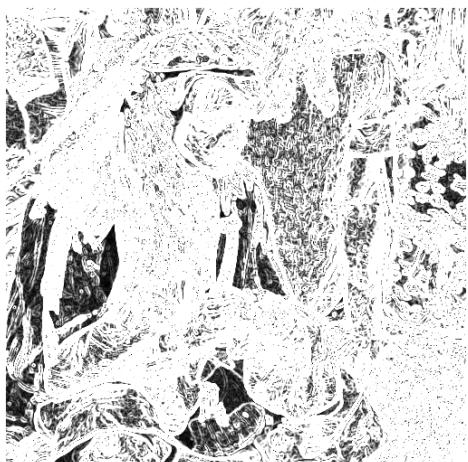
(d) Mean Blur, Roberts
Execution Time: 0.2784 seconds



(e) Mean Blur, Sobel
Execution Time: 0.2994 seconds



(f) Gaussian Blur, Roberts
Execution Time: 0.2838 seconds



(g) Gaussian Blur, Sobel
Execution Time: 0.3158 seconds

Figure 4: Pirate

```
1 /*  
2  * File: edge_detection.h  
3 */  
4  
5 /* C++ Includes */  
6 #include <chrono>  
7 #include <cmath>  
8 #include <ctime>  
9 #include <fstream>  
10 #include <iostream>  
11 #include <string>  
12 #include <vector>  
13 #include <utility>  
14  
15 /* Project Includes */  
16 #include "convert/cameraman.h"  
17 #include "convert/lena.h"  
18 #include "convert/mandril.h"  
19 #include "convert/pirate.h"  
20  
21 /* Output Directory */  
22 static const std::string OUTPUT_DIR = "../output/";  
23  
24 /* Input Image Info */  
25 typedef std::pair<std::string, const unsigned char *> Image;  
26 static const std::vector<Image> IMAGES = {  
27     Image("cameraman", cameraman_image),  
28     Image("lena", lena_image),  
29     Image("mandril", mandril_image),  
30     Image("pirate", pirate_image)  
31 };  
32  
33 /* Input Image Dimensions */  
34 static const int IMAGE_WIDTH = 512;  
35 static const int IMAGE_HEIGHT = 512;  
36 static const int PIXEL_COUNT = IMAGE_WIDTH * IMAGE_HEIGHT;  
37  
38 /* Sobel Operator */  
39 static const int SOBEL_V[3][3] = {{-1, 0, 1}, {-2, 0, 2}, {-1, 0, 1}};  
40 static const int SOBEL_H[3][3] = {{-1, -2, -1}, {0, 0, 0}, {1, 2, 1}};  
41  
42 /* Roberts Cross Operators */  
43 static const int ROBERTS_V[2][2] = {{1, 0}, {0, -1}};  
44 static const int ROBERTS_H[2][2] = {{0, 1}, {-1, 0}};  
45  
46 /* Blur Operators */  
47 static const int MEAN_BLUR[3][3] = {{1, 1, 1}, {1, 1, 1}, {1, 1, 1}};  
48 static const int GAUSSIAN_BLUR[3][3] = {{1, 2, 1}, {2, 4, 2}, {1, 2, 1}};  
49  
50 /* Helper Functions */  
51 int convolve_order_2(const int[2][2], const int, const int, int **);  
52 int convolve_order_3(const int[3][3], const int, const int, int **);  
53 void write_pgm(const std::string&, const std::string&, int **);  
54  
55  
56  
57  
58
```

```
59 /* Execution Timer Class */
60 class Timer {
61 private:
62     std::chrono::time_point<std::chrono::system_clock> start;
63     std::string label;
64 public:
65     Timer(const std::string& l) :
66         start(std::chrono::system_clock::now()), label(l) {}
67     ~Timer() {
68         std::chrono::duration<double> elapsed_seconds =
69         std::chrono::system_clock::now() - start;
70         std::cout << label << " execution time: ";
71         std::cout << elapsed_seconds.count() << " s" << std::endl;
72     }
73 };
74
```

```

1 /*
2  File: main.cpp
3 */
4 #include "edge_detection.h"
5
6 int main(void) { // Start Total Timer
7     Timer total_time("Total");
8
9     /* Allocate memory to hold input/output images. */
10    int **input_image = new int*[IMAGE_HEIGHT];
11    int **blurred_image = new int*[IMAGE_HEIGHT];
12    int **output_image = new int*[IMAGE_HEIGHT - 2];
13    for (int i = 0; i < IMAGE_HEIGHT; i++) {
14        input_image[i] = new int[IMAGE_WIDTH];
15        blurred_image[i] = new int[IMAGE_WIDTH];
16    }
17    for (int i = 0; i < IMAGE_HEIGHT - 2; i++) {
18        output_image[i] = new int[IMAGE_WIDTH - 2];
19    }
20
21    /* Iterate through each image. */
22    for (Image image_pair : IMAGES) {
23        /* Unpack image parameters. */
24        std::string image_label = image_pair.first;
25        const unsigned char *image = image_pair.second;
26
27        /* Read flat 1D image array to 2D array. */
28        for (int i = 0; i < IMAGE_HEIGHT; i++) {
29            for (int j = 0; j < IMAGE_WIDTH; j++) {
30                input_image[i][j] = static_cast<int>(
31                    image[i * IMAGE_HEIGHT + j]);
32            }
33        }
34
35        {Timer sobel_time(image_label + "_sobel");
36
37        /* Detect image's edges using the Sobel operator. */
38        for (int i = 1; i < IMAGE_HEIGHT - 1; i++) {
39            for (int j = 1; j < IMAGE_WIDTH - 1; j++) {
40                int V = convolve_order_3(SOBEL_V, i, j, input_image);
41                int H = convolve_order_3(SOBEL_H, i, j, input_image);
42                output_image[i - 1][j - 1] = static_cast<int>(
43                    round(sqrt(pow(V, 2) + pow(H, 2))));
44            }
45        }
46        write_pgm(image_label, image_label + "_sobel", output_image);
47    } // End Sobel Timer
48
49    {Timer roberts_time(image_label + "_roberts");
50
51        /* Detect image's edges using the Roberts cross. */
52        for (int i = 1; i < IMAGE_HEIGHT - 1; i++) {
53            for (int j = 1; j < IMAGE_WIDTH - 1; j++) {
54                int V = convolve_order_2(ROBERTS_V, i, j, input_image);
55                int H = convolve_order_2(ROBERTS_H, i, j, input_image);
56                output_image[i - 1][j - 1] = static_cast<int>(
57                    round(sqrt(pow(V, 2) + pow(H, 2))));
58            }

```

```

59 }
60 write_pgm(image_label, image_label + "_roberts", output_image);
61 } // End Roberts Timer
62
63 {Timer mean_sobel_time(image_label + "_mean_sobel");
64 /* Blur image with mean blur, then apply Sobel. */
65 for (int i = 0; i < IMAGE_HEIGHT - 0; i++) {
66     for (int j = 0; j < IMAGE_WIDTH - 0; j++) {
67         if (i == 0 || i == IMAGE_HEIGHT - 1 || j == 0 || j == IMAGE_WIDTH - 1) {
68             blurred_image[i][j] = 0;
69         } else {
70             blurred_image[i][j] = convolve_order_3(MEAN_BLUR, i, j, input_image);
71         }
72     }
73 }
74 for (int i = 1; i < IMAGE_HEIGHT - 1; i++) {
75     for (int j = 1; j < IMAGE_WIDTH - 1; j++) {
76         int V = convolve_order_3(SOBEL_V, i, j, blurred_image);
77         int H = convolve_order_3(SOBEL_H, i, j, blurred_image);
78         output_image[i - 1][j - 1] = static_cast<int>(
79             round(sqrt(pow(V, 2) + pow(H, 2))));
80     }
81 }
82 write_pgm(image_label, image_label + "_mean_sobel", output_image);
83 } // End Mean Sobel Timer
84
85 {Timer mean_roberts_time(image_label + "_mean_roberts");
86 /* Blur image with mean blur, then apply Roberts. */
87 for (int i = 0; i < IMAGE_HEIGHT - 0; i++) {
88     for (int j = 0; j < IMAGE_WIDTH - 0; j++) {
89         if (i == 0 || i == IMAGE_HEIGHT - 1 || j == 0 || j == IMAGE_WIDTH - 1) {
90             blurred_image[i][j] = 0;
91         } else {
92             blurred_image[i][j] = convolve_order_3(MEAN_BLUR, i, j, input_image);
93         }
94     }
95 }
96 for (int i = 1; i < IMAGE_HEIGHT - 1; i++) {
97     for (int j = 1; j < IMAGE_WIDTH - 1; j++) {
98         int V = convolve_order_2(ROBERTS_V, i, j, blurred_image);
99         int H = convolve_order_2(ROBERTS_H, i, j, blurred_image);
100        output_image[i - 1][j - 1] = static_cast<int>(
101            round(sqrt(pow(V, 2) + pow(H, 2))));
102    }
103 }
104 write_pgm(image_label, image_label + "_mean_roberts", output_image);
105 } // End Mean Roberts Timer
106
107 {Timer gaussian_sobel_time(image_label + "_gaussian_sobel");
108 /* Blur image with Gaussian blur, then apply Sobel. */
109 for (int i = 0; i < IMAGE_HEIGHT - 0; i++) {
110     for (int j = 0; j < IMAGE_WIDTH - 0; j++) {
111         if (i == 0 || i == IMAGE_HEIGHT - 1 || j == 0 || j == IMAGE_WIDTH - 1) {
112             blurred_image[i][j] = 0;
113         } else {
114             blurred_image[i][j] = convolve_order_3(GAUSSIAN_BLUR, i, j, input_image);
115         }
116     }

```

```

117 }
118     for (int i = 1; i < IMAGE_HEIGHT - 1; i++) {
119         for (int j = 1; j < IMAGE_WIDTH - 1; j++) {
120             int V = convolve_order_3(SOBEL_V, i, j, blurred_image);
121             int H = convolve_order_3(SOBEL_H, i, j, blurred_image);
122             output_image[i - 1][j - 1] = static_cast<int>(
123                 round(sqrt(pow(V, 2) + pow(H, 2))));
124         }
125     }
126     write_pgm(image_label, image_label + "_gaussian_sobel", output_image);
127 } // End Gaussian Sobel Timer
128
129 {Timer mean_roberts_time(image_label + "_gaussian_roberts");
130 /* Blur image with Gaussian blur, then apply Roberts cross. */
131 for (int i = 0; i < IMAGE_HEIGHT - 0; i++) {
132     for (int j = 0; j < IMAGE_WIDTH - 0; j++) {
133         if (i == 0 || i == IMAGE_HEIGHT - 1 || j == 0 || j == IMAGE_WIDTH - 1) {
134             blurred_image[i][j] = 0;
135         } else {
136             blurred_image[i][j] = convolve_order_3(GAUSSIAN_BLUR, i, j, input_image);
137         }
138     }
139 }
140 for (int i = 1; i < IMAGE_HEIGHT - 1; i++) {
141     for (int j = 1; j < IMAGE_WIDTH - 1; j++) {
142         int V = convolve_order_2(ROBERTS_V, i, j, blurred_image);
143         int H = convolve_order_2(ROBERTS_H, i, j, blurred_image);
144         output_image[i - 1][j - 1] = static_cast<int>(
145             round(sqrt(pow(V, 2) + pow(H, 2))));
146     }
147 }
148     write_pgm(image_label, image_label + "_gaussian_roberts", output_image);
149 } // End Gaussian Roberts Timer
150 }
151
152 /* Clean up allocated memory. */
153 for (int i = 0; i < IMAGE_HEIGHT; i++) {
154     delete[] input_image[i];
155     delete[] blurred_image[i];
156 }
157 for (int i = 0; i < IMAGE_HEIGHT - 2; i++) {
158     delete[] output_image[i];
159 }
160 delete[] input_image;
161 delete[] blurred_image;
162 delete[] output_image;
163
164 return 0;
165 } // End Total Timer
166
167 int convolve_order_2(const int mask[2][2], const int i, const int j, int **I) {
168     return I[i][j] * mask[0][0]
169     + I[i][j + 1] * mask[0][1]
170     + I[i + 1][j] * mask[1][0]
171     + I[i + 1][j + 1] * mask[1][1];
172 }
173
174 int convolve_order_3(const int mask[3][3], const int i, const int j, int **I) {

```

```
175 return I[i - 1][j - 1] * mask[0][0]
176 + I[i - 1][j] * mask[0][1]
177 + I[i - 1][j + 1] * mask[0][2]
178 + I[i][j - 1] * mask[1][0]
179 + I[i][j] * mask[1][1]
180 + I[i][j + 1] * mask[1][2]
181 + I[i + 1][j - 1] * mask[2][0]
182 + I[i + 1][j] * mask[2][1]
183 + I[i + 1][j + 1] * mask[2][2];
184 }
185
186 void write_pgm(const std::string& label, const std::string& mask, int **W) {
187     std::ofstream pgm(OUTPUT_DIR + label + "/" + mask + ".pgm");
188
189     if (!pgm) {
190         std::cerr << "Error occurred opening " << label << std::endl;
191         return;
192     }
193
194     /* Write header, dimensions, and maximum value to file. */
195     pgm << "P2" << std::endl;
196     pgm << IMAGE_WIDTH - 2 << " " << IMAGE_HEIGHT - 2 << std::endl;
197     pgm << "255" << std::endl;
198
199     for (int i = 0; i < IMAGE_HEIGHT - 2; i++) {
200         for (int j = 0; j < IMAGE_WIDTH - 2; j++) {
201             pgm << W[i][j] << " ";
202         }
203         pgm << std::endl;
204     }
205 }
```