



(a) Original



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



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



(d) 3x3 Mean Blur



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



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



(g) 3x3 Gaussian Blur



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

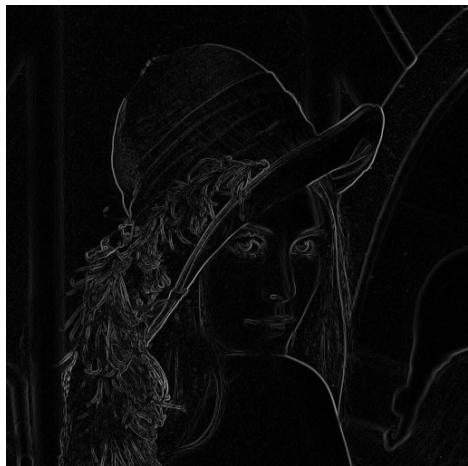


(i) Gaussian Blur, Sobel  
Execution Time: 0.3292 seconds

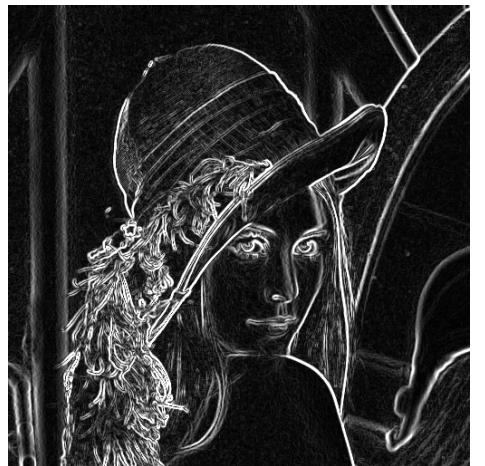
Figure 1: Cameraman



(a) Original



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



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



(d) 3x3 Mean Blur



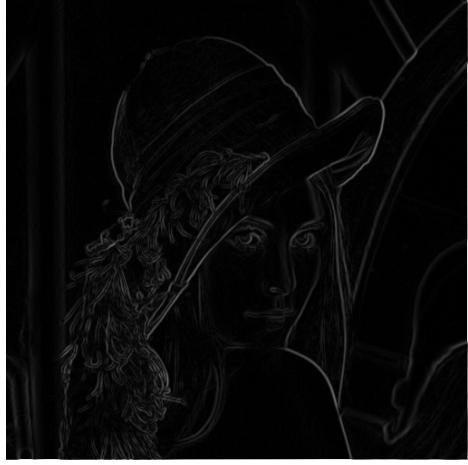
(e) Mean Blur, Roberts  
Execution Time: 0.2796 seconds



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



(g) 3x3 Gaussian Blur

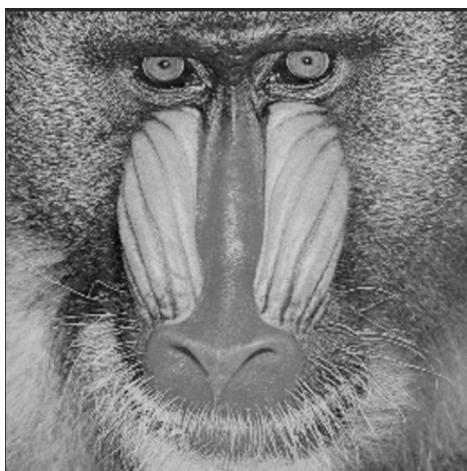


(h) Gaussian Blur, Roberts  
Execution Time: 0.2892 seconds

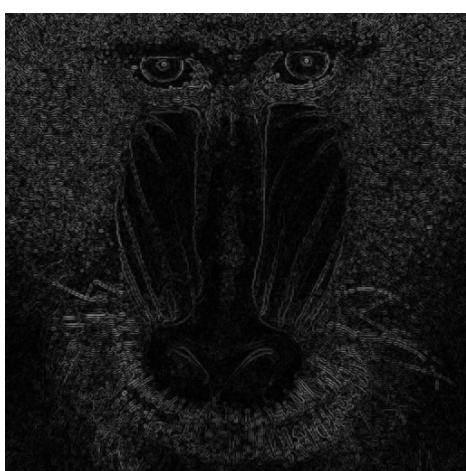


(i) Gaussian Blur, Sobel  
Execution Time: 0.2942 seconds

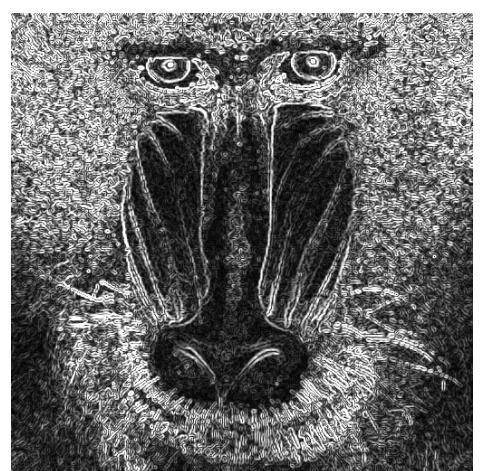
Figure 2: Lena



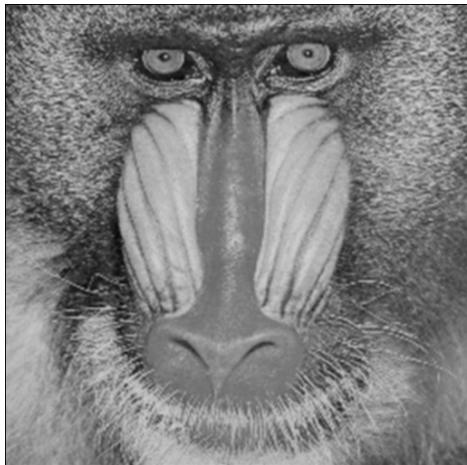
(a) Original



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



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



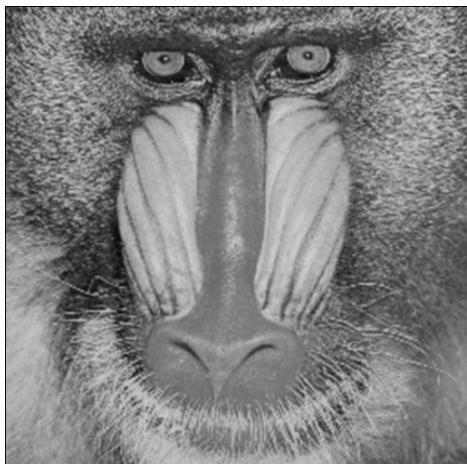
(d) 3x3 Mean Blur



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



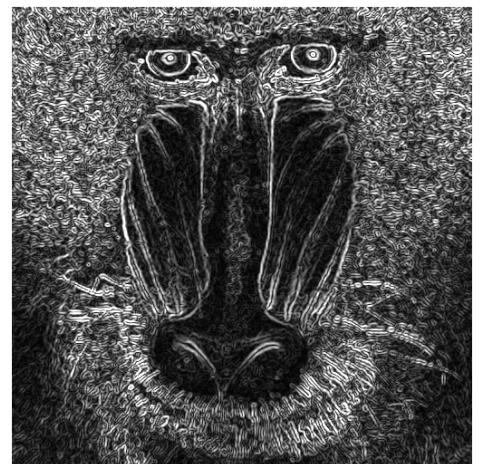
(f) Mean Blur, Sobel  
Execution Time: 0.4073 seconds



(g) 3x3 Gaussian Blur



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



(i) 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) 3x3 Mean Blur



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



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



(g) 3x3 Gaussian Blur



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



(i) 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 /* Execution Timer Class */
56 class Timer {
57 private:
58     std::chrono::time_point<std::chrono::system_clock> start;

```

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

```

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] = static_cast<int>(
71                 round(convolve_order_3(MEAN_BLUR, i, j, input_image) / 9));
72         }
73     }
74 }
75 for (int i = 1; i < IMAGE_HEIGHT - 1; i++) {
76     for (int j = 1; j < IMAGE_WIDTH - 1; j++) {
77         int V = convolve_order_3(SOBEL_V, i, j, blurred_image);
78         int H = convolve_order_3(SOBEL_H, i, j, blurred_image);
79         output_image[i - 1][j - 1] = static_cast<int>(
80             round(sqrt(pow(V, 2) + pow(H, 2))));}
81     }
82 }
83 write_pgm(image_label, image_label + "_mean_sobel", output_image);
84 } // End Mean Sobel Timer
85
86 {Timer mean_roberts_time(image_label + "_mean_roberts");
87 /* Blur image with mean blur, then apply Roberts. */
88 for (int i = 0; i < IMAGE_HEIGHT - 0; i++) {
89     for (int j = 0; j < IMAGE_WIDTH - 0; j++) {
90         if (i == 0 || i == IMAGE_HEIGHT - 1 || j == 0 || j == IMAGE_WIDTH - 1) {
91             blurred_image[i][j] = 0;
92         } else {
93             blurred_image[i][j] = static_cast<int>(
94                 round(convolve_order_3(MEAN_BLUR, i, j, input_image) / 9));
95         }
96     }
97 }
98 write_pgm(image_label, image_label + "_mean_blur", blurred_image);
99 for (int i = 1; i < IMAGE_HEIGHT - 1; i++) {
100     for (int j = 1; j < IMAGE_WIDTH - 1; j++) {
101         int V = convolve_order_2(ROBERTS_V, i, j, blurred_image);
102         int H = convolve_order_2(ROBERTS_H, i, j, blurred_image);
103         output_image[i - 1][j - 1] = static_cast<int>(
104             round(sqrt(pow(V, 2) + pow(H, 2))));}
105     }
106 }
107 write_pgm(image_label, image_label + "_mean_roberts", output_image);
108 } // End Mean Roberts Timer
109
110 {Timer gaussian_sobel_time(image_label + "_gaussian_sobel");
111 /* Blur image with Gaussian blur, then apply Sobel. */
112 for (int i = 0; i < IMAGE_HEIGHT - 0; i++) {
113     for (int j = 0; j < IMAGE_WIDTH - 0; j++) {
114         if (i == 0 || i == IMAGE_HEIGHT - 1 || j == 0 || j == IMAGE_WIDTH - 1) {
115             blurred_image[i][j] = 0;
116         } else {

```

```

117     blurred_image[i][j] = static_cast<int>(
118         round(convolve_order_3(GAUSSIAN_BLUR, i, j, input_image) / 16));
119     }
120   }
121 }
122 for (int i = 1; i < IMAGE_HEIGHT - 1; i++) {
123   for (int j = 1; j < IMAGE_WIDTH - 1; j++) {
124     int V = convolve_order_3(SOBEL_V, i, j, blurred_image);
125     int H = convolve_order_3(SOBEL_H, i, j, blurred_image);
126     output_image[i - 1][j - 1] = static_cast<int>(
127       round(sqrt(pow(V, 2) + pow(H, 2))));}
128   }
129 }
130 write_pgm(image_label, image_label + "_gaussian_sobel", output_image);
131 } // End Gaussian Sobel Timer
132
133 {Timer mean_roberts_time(image_label + "_gaussian_roberts");
134 /* Blur image with Gaussian blur, then apply Roberts cross. */
135 for (int i = 0; i < IMAGE_HEIGHT - 0; i++) {
136   for (int j = 0; j < IMAGE_WIDTH - 0; j++) {
137     if (i == 0 || i == IMAGE_HEIGHT - 1 || j == 0 || j == IMAGE_WIDTH - 1) {
138       blurred_image[i][j] = 0;
139     } else {
140       blurred_image[i][j] = static_cast<int>(
141         round(convolve_order_3(GAUSSIAN_BLUR, i, j, input_image) / 16));
142     }
143   }
144 }
145 write_pgm(image_label, image_label + "_gaussian_blur", blurred_image);
146 for (int i = 1; i < IMAGE_HEIGHT - 1; i++) {
147   for (int j = 1; j < IMAGE_WIDTH - 1; j++) {
148     int V = convolve_order_2(ROBERTS_V, i, j, blurred_image);
149     int H = convolve_order_2(ROBERTS_H, i, j, blurred_image);
150     output_image[i - 1][j - 1] = static_cast<int>(
151       round(sqrt(pow(V, 2) + pow(H, 2))));}
152   }
153 }
154 write_pgm(image_label, image_label + "_gaussian_roberts", output_image);
155 } // End Gaussian Roberts Timer
156 }
157
158 /* Clean up allocated memory. */
159 for (int i = 0; i < IMAGE_HEIGHT; i++) {
160   delete[] input_image[i];
161   delete[] blurred_image[i];
162 }
163 for (int i = 0; i < IMAGE_HEIGHT - 2; i++) {
164   delete[] output_image[i];
165 }
166 delete[] input_image;
167 delete[] blurred_image;
168 delete[] output_image;
169
170 return 0;
171 } // End Total Timer
172
173 int convolve_order_2(const int mask[2][2], const int i, const int j, int **I) {
174   return I[i][j] * mask[0][0]

```

```
175 + I[i][j + 1] * mask[0][1]
176 + I[i + 1][j] * mask[1][0]
177 + I[i + 1][j + 1] * mask[1][1];
178 }
179
180 int convolve_order_3(const int mask[3][3], const int i, const int j, int **I) {
181     return I[i - 1][j - 1] * mask[0][0]
182     + I[i - 1][j] * mask[0][1]
183     + I[i - 1][j + 1] * mask[0][2]
184     + I[i][j - 1] * mask[1][0]
185     + I[i][j] * mask[1][1]
186     + I[i][j + 1] * mask[1][2]
187     + I[i + 1][j - 1] * mask[2][0]
188     + I[i + 1][j] * mask[2][1]
189     + I[i + 1][j + 1] * mask[2][2];
190 }
191
192 void write_pgm(const std::string& label, const std::string& mask, int **W) {
193     std::ofstream pgm(OUTPUT_DIR + label + "/" + mask + ".pgm");
194
195     if (!pgm) {
196         std::cerr << "Error occurred opening " << label << std::endl;
197         return;
198     }
199
200     /* Write header, dimensions, and maximum value to file. */
201     pgm << "P2" << std::endl;
202     pgm << IMAGE_WIDTH - 2 << " " << IMAGE_HEIGHT - 2 << std::endl;
203     pgm << "255" << std::endl;
204
205     for (int i = 0; i < IMAGE_HEIGHT - 2; i++) {
206         for (int j = 0; j < IMAGE_WIDTH - 2; j++) {
207             pgm << W[i][j] << " ";
208         }
209         pgm << std::endl;
210     }
211 }
212 }
```