(a) Original

(b) Roberts Cross Operator
Execution Time: 0.2704 seconds

(c) Sobel Operator
Execution Time: 0.2663 seconds

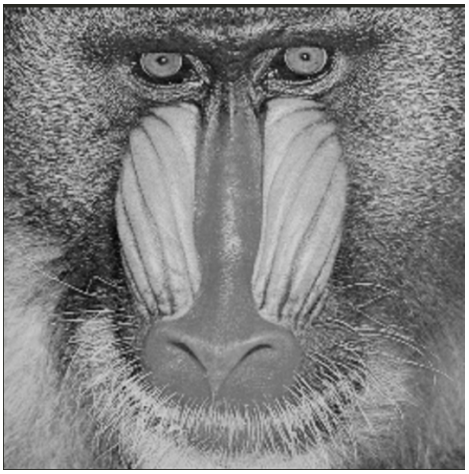
Figure 1: Cameraman

(a) Original

(b) Roberts Cross Operator
Execution Time: 0.2463 seconds

(c) Sobel Operator
Execution Time: 0.2543 seconds

Figure 2: Lena



(a) Original

(b) Roberts Cross Operator
Execution Time: 0.2424 seconds

(c) Sobel Operator
Execution Time: 0.2733 seconds

Figure 3: Mandril

(a) Original

(b) Roberts Cross Operator
Execution Time: 0.2403 seconds

(c) Sobel Operator
Execution Time: 0.2603 seconds

Figure 4: Pirate

```
/*
    File: edge_detection.h
*/

/* C Includes */
#include <stdio.h>

/* C++ Includes */
#include <cmath>
#include <fstream>
#include <iostream>
#include <string>
#include <vector>
#include <utility>

/* Project Includes */
#include "convert/cameraman.h"
#include "convert/lena.h"
#include "convert/mandril.h"
#include "convert/pirate.h"

/* Output Directory */
static const std::string OUTPUT_DIR = "../output/";

/* Input Image Info */
typedef std::pair<std::string, const unsigned char *> Image;
static const std::vector<Image> IMAGES = {
  Image("cameraman", cameraman_image),
  Image("lena", lena_image),
  Image("mandril", mandril_image),
  Image("pirate", pirate_image)
};

/* Input Image Dimensions */
static const int IMAGE_WIDTH = 512;
static const int IMAGE_HEIGHT = 512;
static const int PIXEL_COUNT = IMAGE_WIDTH * IMAGE_HEIGHT;

/* Sobel Operator */
static const int SOBEL_V[3][3] = {{-1, 0, 1},
                  {-2, 0, 2},
                  {-1, 0, 1 }};
static const int SOBEL_H[3][3] = {{-1, -2, -1},
                  {0, 0, 0},
                  {1, 2, 1}};

/* Roberts Cross Operators */
static const int ROBERTS_V[2][2] = {{1, 0},
                  {0, -1}};
static const int ROBERTS_H[2][2] = {{0, 1},
                  {-1, 0}};

/* Helper Functions */
int sobel(const bool, const int, const int, int **);
int roberts(const bool, const int, const int, int **);
void write_pgm(const std::string&, int **);

```

```cpp
/*
    File: main.cpp
*/
#include "edge_detection.h"

int main(void) {
  /* Allocate memory to hold input/output images. */
  int **input_image = new int*[IMAGE_HEIGHT];
  int **output_image = new int*[IMAGE_HEIGHT - 2];
  for (int i = 0; i < IMAGE_HEIGHT; i++) {
    input_image[i] = new int[IMAGE_WIDTH];
    output_image[i] = new int[IMAGE_WIDTH - 2];
  }

  /* Iterate through each image. */
  for (Image image_pair : IMAGES) {
    /* Unpack image parameters. */
    std::string image_label = image_pair.first;
    const unsigned char *image = image_pair.second;

    /* Read flat 1D image array to 2D array. */
    for (int i = 0; i < IMAGE_HEIGHT; i++) {
      for (int j = 0; j < IMAGE_WIDTH; j++) {
        input_image[i][j] = static_cast<int>(image[i * IMAGE_HEIGHT + j]);
      }
    }

    /* Detect image's edges using the Sobel operator. */
    for (int i = 1; i < IMAGE_HEIGHT - 1; i++) {
      for (int j = 1; j < IMAGE_WIDTH - 1; j++) {
        int V = sobel(true, i, j, input_image);
        int H = sobel(false, i, j, input_image);
        output_image[i - 1][j - 1] = static_cast<int>(round(sqrt(pow(V, 2) + pow(H,
2))));
      }
    }
    /* Generate PGM file for Sobel output. */
    write_pgm(image_label + "_sobel", output_image);

    /* Detect image's edges using the Roberts cross. */
    for (int i = 1; i < IMAGE_HEIGHT - 1; i++) {
      for (int j = 1; j < IMAGE_WIDTH - 1; j++) {
        int V = roberts(true, i, j, input_image);
        int H = roberts(false, i, j, input_image);
        output_image[i - 1][j - 1] = static_cast<int>(round(sqrt(pow(V, 2) + pow(H,
2))));
      }
    }
    /* Generate PGM file for Roberts output. */
    write_pgm(image_label + "_roberts", output_image);
  }

    return 0;
}

int sobel(const bool vertical, const int i, const int j, int **I) {
  if (vertical) {
    return I[i - 1][j - 1] * SOBEL_V[0][0]
```

```
57        + I[i - 1][j] * SOBEL_V[0][1]
58        + I[i - 1][j + 1] * SOBEL_V[0][2]
59        + I[i][j - 1] * SOBEL_V[1][0]
60        + I[i][j] * SOBEL_V[1][1]
61        + I[i][j + 1] * SOBEL_V[1][2]
62        + I[i + 1][j - 1] * SOBEL_V[2][0]
63        + I[i + 1][j] * SOBEL_V[2][1]
64        + I[i + 1][j + 1] * SOBEL_V[2][2];
65    }
66    else {
67      return I[i - 1][j - 1] * SOBEL_H[0][0]
68        + I[i - 1][j] * SOBEL_H[0][1]
69        + I[i - 1][j + 1] * SOBEL_H[0][2]
70        + I[i][j - 1] * SOBEL_H[1][0]
71        + I[i][j] * SOBEL_H[1][1]
72        + I[i][j + 1] * SOBEL_H[1][2]
73        + I[i + 1][j - 1] * SOBEL_H[2][0]
74        + I[i + 1][j] * SOBEL_H[2][1]
75        + I[i + 1][j + 1] * SOBEL_H[2][2];
76    }
77  }
78
79  int roberts(const bool vertical, const int i, const int j, int **I) {
80    if (vertical) {
81      return I[i][j] * ROBERTS_V[0][0]
82        + I[i][j + 1] * ROBERTS_V[0][1]
83        + I[i + 1][j] * ROBERTS_V[1][0]
84        + I[i + 1][j + 1] * ROBERTS_V[1][1];
85    }
86    else {
87      return I[i][j] * ROBERTS_H[0][0]
88        + I[i][j + 1] * ROBERTS_H[0][1]
89        + I[i + 1][j] * ROBERTS_H[1][0]
90        + I[i + 1][j + 1] * ROBERTS_H[1][1];
91    }
92  }
93
94  void write_pgm(const std::string& label, int **W) {
95      std::cout << OUTPUT_DIR + label + ".pgm" << std::endl;
96    std::ofstream pgm(OUTPUT_DIR + label + ".pgm");
97
98      if (!pgm) {
99      std::cerr << "Error occurred opening " << label << std::endl;
100     return;
101   }
102
103   pgm << "P2" << std::endl;  // Header
104   pgm << IMAGE_WIDTH - 2 << " " << IMAGE_HEIGHT - 2 << std::endl;  // Dimensions
105   pgm << "255" << std::endl;  // Maximum Value
106
107   for (int i = 0; i < IMAGE_HEIGHT - 2; i++) {
108     for (int j = 0; j < IMAGE_WIDTH - 2; j++) {
109       pgm << W[i][j] << " ";
110     }
111     pgm << std::endl;
112   }
113 }
114
```