COMP27112: Visual Computing - Lab 4 Report

Taran Patel (ID: 11430245)

Contents

1 Image Histogram and Segmentation

1.1 3.1: Histogram

For the histogram task we implemented a function (see Listing 1) that counts the occurrences of each gray level and draws a 400 pixel high by 512 pixel wide histogram (with each of the 256 gray levels represented by a 2-pixel wide bar).

```
#include <opencv2/core/core.hpp>
#include <opencv2/imgproc.hpp>
#include <opencv2/highgui.hpp>
using namespace cv;

const int HIST_IMG_HEIGHT = 400;

void createHistogram(Mat& img, Mat& hist) {
    long counts[256] = {}; // Zero-initialized

    // Count pixel intensities
    for (int y = 0; y < img.rows; y++) {
        for (int x = 0; x < img.cols; x++) {
            int val = img.at<uchar>(y, x);
            counts[val]++;
        }
    }

    // Find maximum count
    long max_count = 0;
    for (int i = 0; i < 256; i++) {
        if (counts[i] > max_count)
            max_count = counts[i];
    }

    // Create histogram image with white background
    hist = Mat(HIST_IMG_HEIGHT, 512, CV_8UC1, Scalar(255));

    // Draw each bar (2 pixels wide)
    for (int i = 0; i < 256; i++) {
        int height = static_cast<int>((double)counts[i] / max_count *
            HIST_IMG_HEIGHT);
        if (height == 0)
            continue;
        int x1 = i * 2;
        int x2 = x1 + 1;
        int y1 = HIST_IMG_HEIGHT - 1;
        int y2 = HIST_IMG_HEIGHT - height;
        line(hist, Point(x1, y1), Point(x1, y2), Scalar(0), 1);
        line(hist, Point(x2, y1), Point(x2, y2), Scalar(0), 1);
    }
}
```

Listing 1: Source code for creating a histogram

The report includes histogram images for the following two input images:

- **circuit_board.jpg**
- **science_person.jpg**

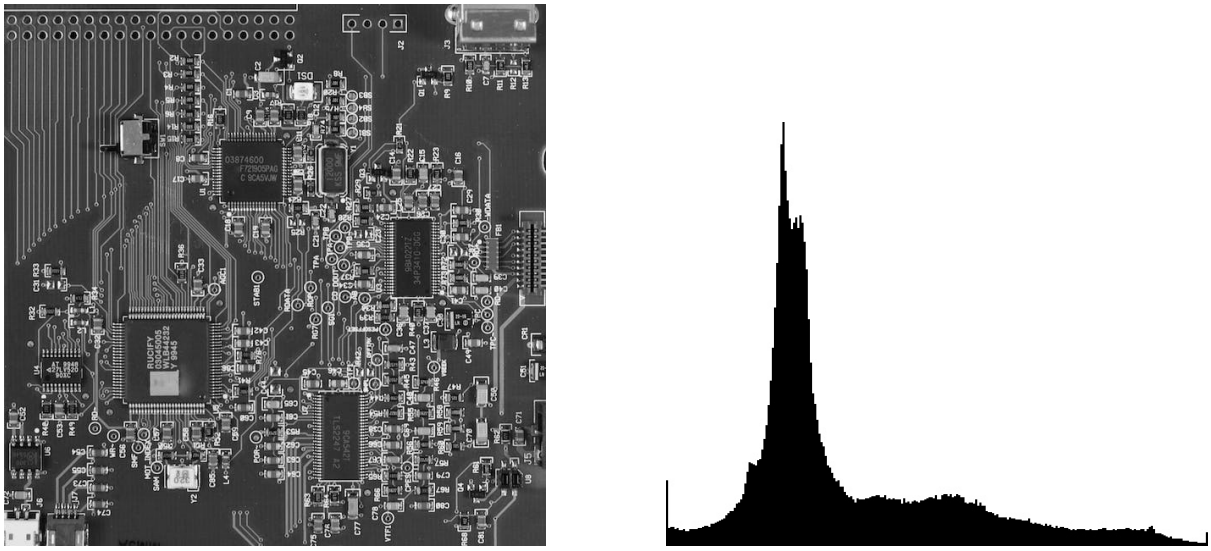

Figure 1: (Left) Original `circuit_board.jpg`; (Right) Generated histogram (`circuit_histo.jpg`).

Figure 2: (Left) Original `science_person.jpg`; (Right) Generated histogram (`science_histo.jpg`).

1.2 3.2: Thresholding

Using the histogram function, we selected appropriate threshold values for segmentation on the following images:

- **fundus.jpg**: 97
- **glaucoma.jpg**: 110
- **optic_nerve_head.jpg**: 100
- **motorway.png**: 170

Below are the output results for each image.

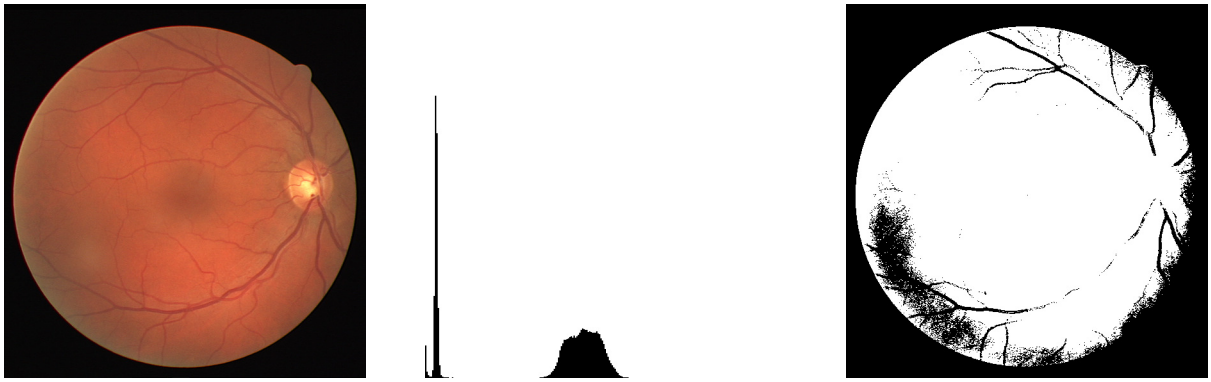

Figure 3: **Fundus:** Original image, histogram, and thresholded result with threshold = 97.

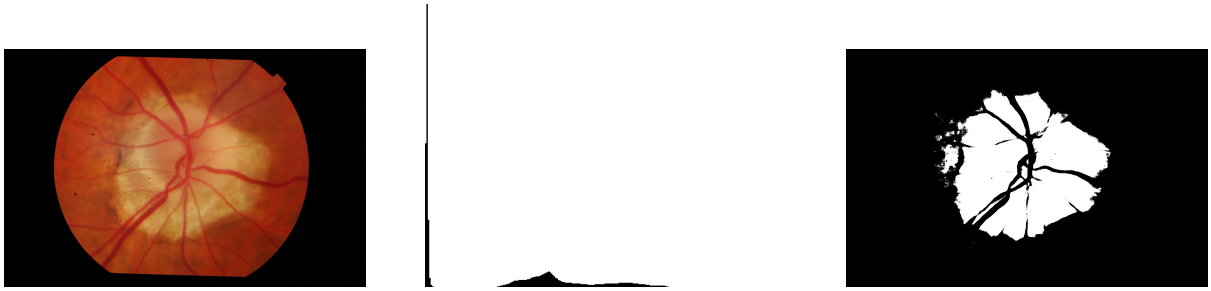

Figure 4: **Glaucoma:** Original image, histogram, and thresholded result with threshold = 110.

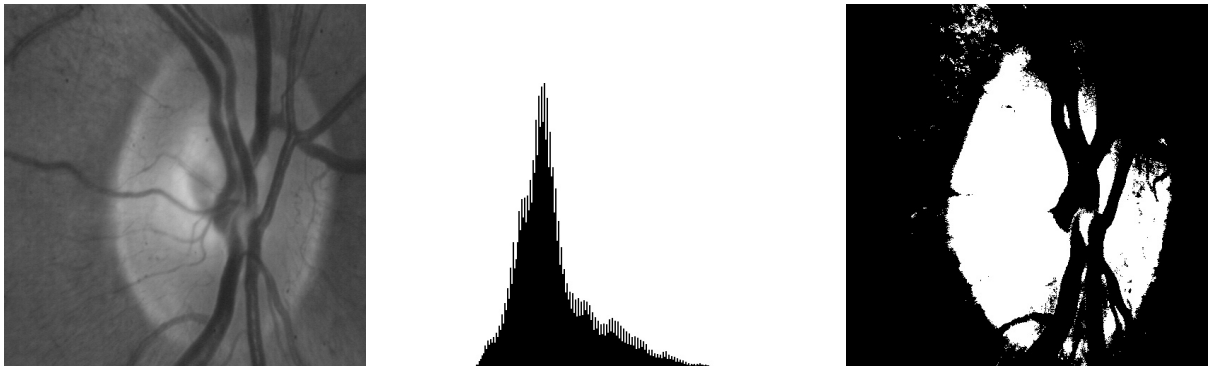

Figure 5: **Optic Nerve Head:** Original image, histogram, and thresholded result with threshold = 100.

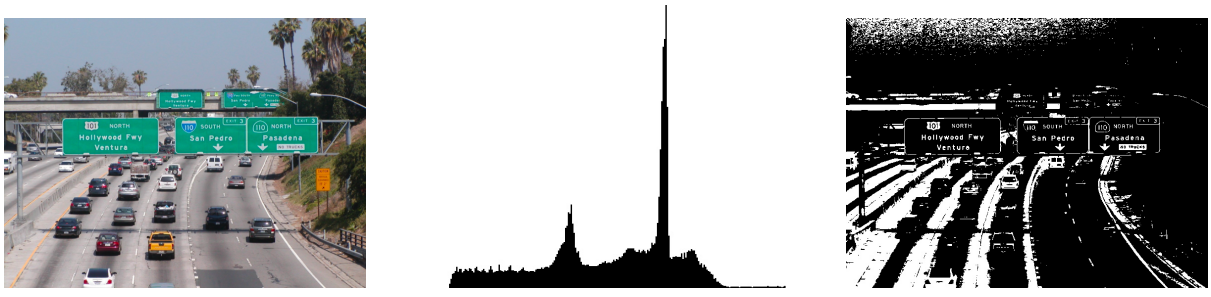

Figure 6: **Motorway:** Original image, histogram, and thresholded result with threshold = 170.

Observation: In each case the selected threshold value successfully segments the features of interest.

2 Horizon Detection

2.1 4.1: Processing Pipeline

```cpp
#include <opencv2/opencv.hpp>
#include <iostream>
#include <cmath>
#include <vector>

// Trackbar-controlled parameters
int g_blurKsize     = 7;
int g_cannyLower    = 50;
int g_cannyUpper    = 150;
int g_houghThresh   = 50;
int g_minLineLen    = 30;
int g_maxLineGap    = 20;
int g_verticalDelta = 10;

cv::Mat g_colorImg;  // Original color image

// Global Mats for each intermediate step
cv::Mat g_edges;
cv::Mat g_allLines;
cv::Mat g_shortLinesRemoved;
cv::Mat g_onlyHorizontalLines;
cv::Mat g_horizonDraw;

// ------------------------------------------------
// 1) fitPoly: polynomial regression (no VLA)
// ------------------------------------------------
std::vector<double> fitPoly(const std::vector<cv::Point>& points, int n)
{
    int nPoints = (int) points.size();
    std::vector<double> xVals(nPoints), yVals(nPoints);

    for(int i = 0; i < nPoints; i++)
    {
        xVals[i] = points[i].x;
        yVals[i] = points[i].y;
    }

    // Build an (n+1) x (n+2) augmented matrix
    std::vector<std::vector<double>> mat(n+1, std::vector<double>(n+2, 0.0));
    for(int row = 0; row < n+1; row++)
    {
        for(int col = 0; col < n+1; col++)
        {
            double sumVal = 0.0;
            for(int i = 0; i < nPoints; i++)
                sumVal += std::pow(xVals[i], row + col);
            mat[row][col] = sumVal;
        }

        double sumVal2 = 0.0;
        for(int i = 0; i < nPoints; i++)
            sumVal2 += std::pow(xVals[i], row) * yVals[i];
        mat[row][n+1] = sumVal2;
    }

    // Solve via Gauss elimination
    std::vector<double> coeffVec(n+1, 0.0);
    for(int i = 0; i < n; i++)
    {
```

```
        for(int k = i+1; k <= n; k++)
        {
            double t = mat[k][i] / mat[i][i];
            for(int j = 0; j <= n+1; j++)
                mat[k][j] -= t * mat[i][j];
        }
    }
    for(int i = n; i >= 0; i--)
    {
        coeffVec[i] = mat[i][n+1];
        for(int j = 0; j < n+1; j++)
        {
            if(j != i)
                coeffVec[i] -= mat[i][j] * coeffVec[j];
        }
        coeffVec[i] /= mat[i][i];
    }
    return coeffVec;
}

// ------------------------------------------------
// 2) pointAtX: evaluate polynomial
// ------------------------------------------------
cv::Point pointAtX(const std::vector<double>& coeff, double x)
{
    double y = 0.0;
    for(int i = 0; i < (int)coeff.size(); i++)
        y += std::pow(x, i) * coeff[i];
    return cv::Point((int)std::round(x), (int)std::round(y));
}

// ------------------------------------------------
// 3) The pipeline that runs each time a trackbar changes
// ------------------------------------------------
void runHorizonDetection()
{
    if(g_colorImg.empty())
    {
        std::cerr << "No image loaded!\n";
        return;
    }

    // 1) Convert to grayscale + blur
    cv::Mat gray;
    cv::cvtColor(g_colorImg, gray, cv::COLOR_BGR2GRAY);

    // Enforce odd kernel size
    if(g_blurKsize < 1) g_blurKsize = 1;
    if(g_blurKsize % 2 == 0) g_blurKsize += 1;
    cv::GaussianBlur(gray, gray, cv::Size(g_blurKsize, g_blurKsize), 0);

    // 2) Canny
    cv::Canny(gray, g_edges, g_cannyLower, g_cannyUpper);
    cv::imshow("Canny Edges", g_edges);

    // 3) Hough
    std::vector<cv::Vec4i> linesP;
    double rho   = 1.0;
    double theta = CV_PI / 180.0;
    cv::HoughLinesP(g_edges, linesP, rho, theta,
                    g_houghThresh, (double)g_minLineLen, (double)g_maxLineGap);

    // 3a) Draw all lines
```

```
123     g_allLines = g_colorImg.clone();
124     for(const auto& ln : linesP)
125     {
126         cv::line(g_allLines, cv::Point(ln[0], ln[1]),
127                              cv::Point(ln[2], ln[3]),
128                              cv::Scalar(0,0,255), 1);
129     }
130     cv::imshow("All Hough Lines", g_allLines);
131 
132     // 4) STEP ONE: Remove short lines only
133     std::vector<cv::Vec4i> linesAfterShortRemoval;
134     g_shortLinesRemoved = g_colorImg.clone();
135     for(const auto& ln : linesP)
136     {
137         int x1 = ln[0], y1 = ln[1];
138         int x2 = ln[2], y2 = ln[3];
139         double dx = (double)(x2 - x1);
140         double dy = (double)(y2 - y1);
141         double length = std::sqrt(dx*dx + dy*dy);
142 
143         if(length >= (double)g_minLineLen)
144         {
145             // keep it
146             linesAfterShortRemoval.push_back(ln);
147             cv::line(g_shortLinesRemoved, cv::Point(x1, y1), cv::Point(x2, y2),
148                      cv::Scalar(255, 0, 0), 2); // blue
149         }
150     }
151     cv::imshow("Short Lines Removed", g_shortLinesRemoved);
152 
153     // 5) STEP TWO: Remove near-vertical from the short-removed set
154     std::vector<cv::Vec4i> finalLines;
155     g_onlyHorizontalLines = g_colorImg.clone();
156 
157     for(const auto& ln : linesAfterShortRemoval)
158     {
159         int x1 = ln[0], y1 = ln[1];
160         int x2 = ln[2], y2 = ln[3];
161 
162         if(std::abs(x2 - x1) >= g_verticalDelta)
163         {
164             // keep
165             finalLines.push_back(ln);
166             cv::line(g_onlyHorizontalLines, cv::Point(x1, y1), cv::Point(x2, y2
                    ),
167                      cv::Scalar(255, 0, 0), 2); // still blue
168         }
169     }
170     cv::imshow("Only Horizontal Lines", g_onlyHorizontalLines);
171 
172     // 6) Fit a polynomial with final lines
173     //    Collect endpoints
174     std::vector<cv::Point> horizonPoints;
175     for(const auto& ln : finalLines)
176     {
177         horizonPoints.push_back(cv::Point(ln[0], ln[1]));
178         horizonPoints.push_back(cv::Point(ln[2], ln[3]));
179     }
180 
181     if(horizonPoints.size() < 4)
182     {
183         // Not enough data
184         g_horizonDraw = g_colorImg.clone();
```

```
185         cv::putText(g_horizonDraw, "Not␣enough␣points!",
186                     cv::Point(50,50), cv::FONT_HERSHEY_SIMPLEX,
187                     1.0, cv::Scalar(0,0,255), 2);
188         cv::imshow("Fitted␣Horizon", g_horizonDraw);
189         return;
190     }
191 
192     std::vector<double> coeffs = fitPoly(horizonPoints, 2);
193 
194     // 7) Draw the polynomial
195     g_horizonDraw = g_colorImg.clone();
196     for(int x = 0; x < g_horizonDraw.cols; x++)
197     {
198         cv::Point pt = pointAtX(coeffs, (double)x);
199         if(pt.y >= 0 && pt.y < g_horizonDraw.rows)
200             cv::circle(g_horizonDraw, pt, 1, cv::Scalar(0,255,0), -1);
201     }
202     cv::imshow("Fitted␣Horizon", g_horizonDraw);
203 }
204 
205 // ------------------------------------------------
206 // 4) onTrackbarChange
207 // ------------------------------------------------
208 static void onTrackbarChange(int, void*)
209 {
210     runHorizonDetection();
211 }
212 
213 // ------------------------------------------------
214 // 5) MAIN
215 // ------------------------------------------------
216 int main(int argc, char** argv)
217 {
218     if(argc < 2)
219     {
220         std::cout << "Usage:␣" << argv[0] << "␣<image>\n";
221         return -1;
222     }
223 
224     // Load color image
225     g_colorImg = cv::imread(argv[1], cv::IMREAD_COLOR);
226     if(g_colorImg.empty())
227     {
228         std::cerr << "Could␣not␣load␣" << argv[1] << "\n";
229         return -1;
230     }
231 
232     // Create windows
233     cv::namedWindow("Canny␣Edges",          cv::WINDOW_NORMAL);
234     cv::namedWindow("All␣Hough␣Lines",      cv::WINDOW_NORMAL);
235     cv::namedWindow("Short␣Lines␣Removed", cv::WINDOW_NORMAL);
236     cv::namedWindow("Only␣Horizontal␣Lines", cv::WINDOW_NORMAL);
237     cv::namedWindow("Fitted␣Horizon",       cv::WINDOW_NORMAL);
238 
239     cv::namedWindow("Controls", cv::WINDOW_NORMAL);
240 
241     // Create trackbars
242     cv::createTrackbar("Blur␣Ksize",        "Controls", &g_blurKsize,     31,
        onTrackbarChange);
243     cv::createTrackbar("Canny␣Lower",       "Controls", &g_cannyLower,    500,
        onTrackbarChange);
244     cv::createTrackbar("Canny␣Upper",       "Controls", &g_cannyUpper,    500,
        onTrackbarChange);
```

```
    cv::createTrackbar("Hough Thresh",     "Controls", &g_houghThresh,   200, onTrackbarChange);
    cv::createTrackbar("Min Line Len",     "Controls", &g_minLineLen,    300, onTrackbarChange);
    cv::createTrackbar("Max Line Gap",     "Controls", &g_maxLineGap,    100, onTrackbarChange);
    cv::createTrackbar("Vert Delta",       "Controls", &g_verticalDelta, 50, onTrackbarChange);

    // Run once
    runHorizonDetection();

    std::cout << "Adjust trackbars to tune parameters.\n";
    std::cout << "Press [s] to save all images. Press [ESC] to quit.\n";

    while(true)
    {
        int key = cv::waitKey(50);
        if(key == 27) // ESC
            break;
        else if(key == 's')
        {
            // Save them all
            cv::imwrite("edges_snapshot.jpg",              g_edges);
            cv::imwrite("all_lines_snapshot.jpg",          g_allLines);
            cv::imwrite("short_lines_removed_snapshot.jpg", g_shortLinesRemoved);
            cv::imwrite("only_horizontal_lines_snapshot.jpg", g_onlyHorizontalLines);
            cv::imwrite("fitted_horizon_snapshot.jpg",      g_horizonDraw);

            std::cout << "Saved images with params:\n"
                      << "   BlurKsize=" << g_blurKsize
                      << ", CannyLower=" << g_cannyLower
                      << ", CannyUpper=" << g_cannyUpper
                      << ", HoughThresh=" << g_houghThresh
                      << ", MinLineLen=" << g_minLineLen
                      << ", MaxLineGap=" << g_maxLineGap
                      << ", VertDelta=" << g_verticalDelta
                      << "\n";
        }
    }

    return 0;
}
```

Listing 2: Horizon Detection Code

2.2 4.2: Processing and Results

Below are the results for the three horizon images:

2.2.1 Horizon 1

Parameters:

- BlurKsize = 21
- CannyLower = 0
- CannyUpper = 105

- HoughThresh = 50
- MinLineLen = 30
- MaxLineGap = 20
- VertDelta = 10

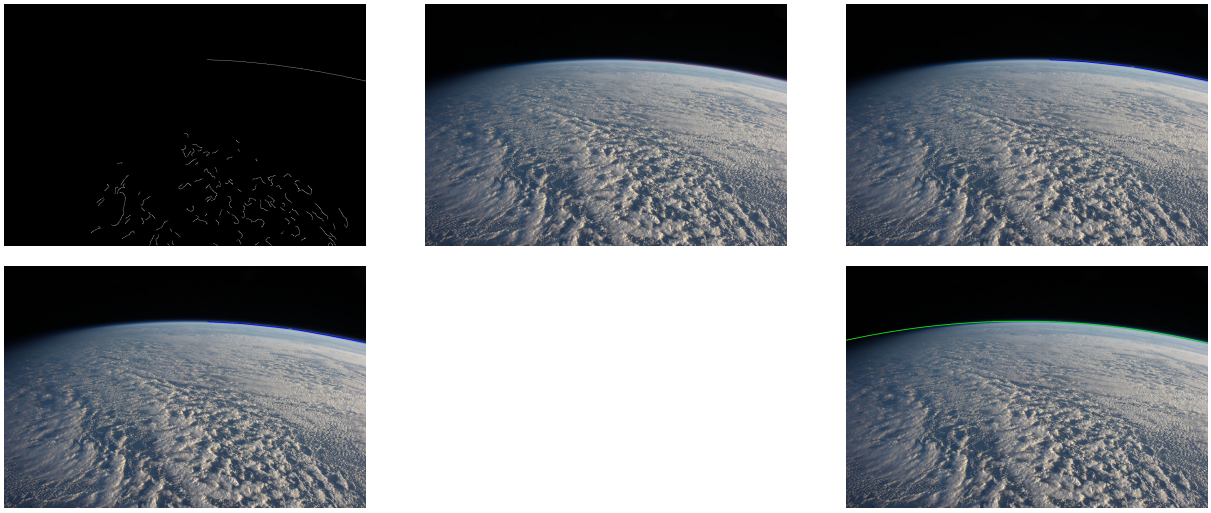

Figure 7: **Horizon 1:** (Top row) Canny Edges, All Hough Lines, Short Lines Removed; (Bottom row) Only Horizontal Lines, Fitted Horizon.

2.2.2 Horizon 2

Parameters:

- BlurKsize = 15
- CannyLower = 50
- CannyUpper = 150
- HoughThresh = 50
- MinLineLen = 30
- MaxLineGap = 20
- VertDelta = 10

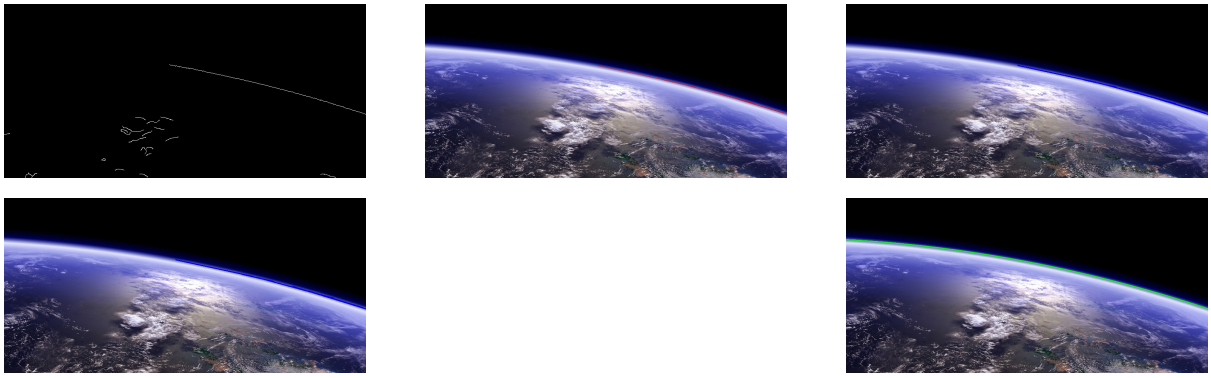

Figure 8: **Horizon 2:** (Top) Canny Edges, All Hough Lines, Short Lines Removed; (Bottom) Only Horizontal Lines, Fitted Horizon.

2.2.3 Horizon 3

Parameters:

- BlurKsize = 7
- CannyLower = 47
- CannyUpper = 150
- HoughThresh = 50
- MinLineLen = 30
- MaxLineGap = 20
- VertDelta = 10

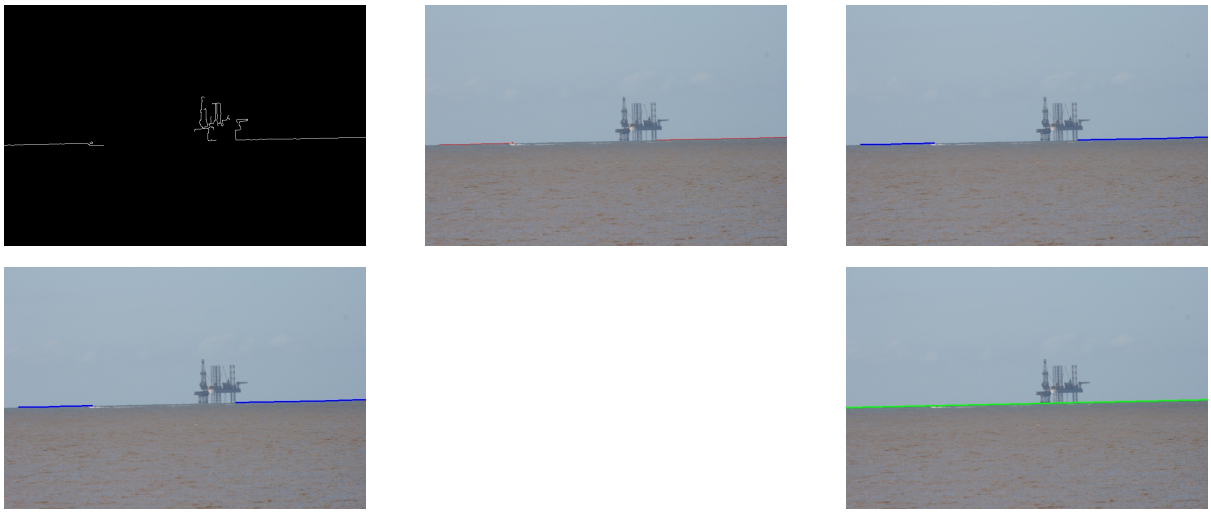

Figure 9: **Horizon 3:** Results with the final fitted horizon curve.

Observations:

- Horizon 1 required a stronger blur to reduce cloud noise.
- Horizon 2 produced distinct edges with moderate blur.
- Horizon 3 parameters were fine-tuned to capture the horizon without excessive extraneous lines.