

Depth Image Processing Journal

Experiments with Python and OpenCV

Siddharth Patel
HTW Berlin

August 20, 2025

Contents

1 Introduction

This journal documents my experiments in depth image processing using Python and OpenCV. The goal is to apply different filters and processing methods on depth frames in order to eventually detect the edge of a table and retrieve the distance to it. For each experiment, I will include:

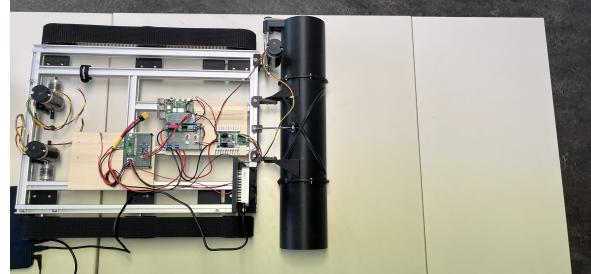
- The Python code used
- Sample output frame(s) for comparison
- Observations about filter performance

2 Experiment Setup

The first step in this project is to define the experimental setup and show the position of the depth camera on the robot. The robot platform is equipped with a depth sensor mounted at the front, as shown in the following images. This setup will be used to capture depth frames while approaching the edge of a table, which forms the basis for the subsequent image processing experiments.



(a) Side view of the robot with mounted depth camera.



(b) Top view of the robot and camera orientation.

Figure 1: Experimental setup: depth camera mounted on robot platform.

3 Depth Frame Subscription (ROS)

Goal: subscribe to the depth topic and convert the ROS image to OpenCV with cv_bridge. We show three basic views: original depth, gray, and colored.

Source

[cv_bridge tutorial \(Python\)](#)

3.1 Code

```

1 #!/usr/bin/env python3
2
3 import rospy
4 from sensor_msgs.msg import Image
5 from cv_bridge import CvBridge
6 import cv2
7
8 class FrameGrabber:
9     def __init__(self):
10         self.bridge = CvBridge()
11         self.depth_image_original = None

```

```

12     rospy.Subscriber("/ascamera/depth0/image_raw", Image, self.depth_callback)
13
14
15
16     def depth_callback(self, msg):
17         self.depth_image_original = self.bridge.imgmsg_to_cv2(msg, desired_encoding="passthrough")
18
19     def show_loop(self):
20         rate = rospy.Rate(30)
21         while not rospy.is_shutdown():
22
23             if self.depth_image_original is not None:
24                 depth_gray = cv2.normalize(self.depth_image_original, None, 0, 255, cv2.NORM_MINMAX)
25                 depth_gray = cv2.convertScaleAbs(depth_gray)
26                 depth_colored = cv2.applyColorMap(depth_gray, cv2.COLORMAP_JET)
27                 cv2.imshow("Depth Original", self.depth_image_original)
28                 cv2.imshow("Depth Gray", depth_gray)
29                 cv2.imshow("Depth Colored", depth_colored)
30
31             key = cv2.waitKey(1) & 0xFF
32             if key == ord('1'):
33                 print("Key '1' pressed. Exiting...")
34                 break
35
36             rate.sleep()
37
38         cv2.destroyAllWindows()
39
40 if __name__ == "__main__":
41     rospy.init_node("rgb_depth_filters_viewer")
42     grabber = FrameGrabber()
43     grabber.show_loop()

```

3.2 Output Frames

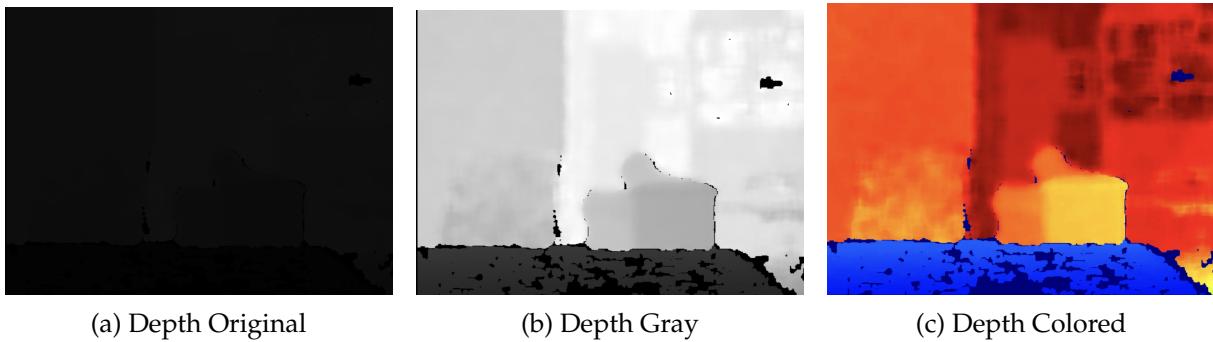


Figure 2: Three fundamental views of the same depth frame.

Notes

Original is the raw depth visualization; Gray is normalized to 8-bit; Colored uses a colormap for contrast.

4 Image Blurring (Image Smoothing)

In this section I illustrate four different blurring techniques that can be applied to depth images.

Source

OpenCV Documentation – Image Filtering

4.1 Averaging

Code

```
1 import cv2
2 import numpy as np
3
4
5 depth_gray = cv2.imread("images/depth_gray.png", cv2.IMREAD_GRAYSCALE)
6
7 depth_colored = cv2.applyColorMap(depth_gray, cv2.COLORMAP_JET)
8
9 # grayscale sequence
10 g1 = depth_gray
11 g2 = cv2.blur(depth_gray, (3,3))
12 g3 = cv2.blur(depth_gray, (5,5))
13 g4 = cv2.blur(depth_gray, (7,7))
14 g5 = cv2.blur(depth_gray, (9,9))
15 g6 = cv2.blur(depth_gray, (11,11))
16 g7 = cv2.blur(depth_gray, (13,13))
17 g8 = cv2.blur(depth_gray, (15,15))
18 g9 = cv2.blur(depth_gray, (17,17))
19 g10 = cv2.blur(depth_gray, (19,19))
20 g11 = cv2.blur(depth_gray, (21,21))
21 g12 = cv2.blur(depth_gray, (23,23))
22 g13 = cv2.blur(depth_gray, (25,25))
23 g14 = cv2.blur(depth_gray, (27,27))
24 g15 = cv2.blur(depth_gray, (29,29))
25 g16 = cv2.blur(depth_gray, (31,31))
26 g17 = cv2.blur(depth_gray, (33,33))
27 g18 = cv2.blur(depth_gray, (35,35))
28 g19 = cv2.blur(depth_gray, (37,37))
29 g20 = cv2.blur(depth_gray, (39,39))
30 g21 = cv2.blur(depth_gray, (41,41))
31
32
33 row1 = np.hstack([g1, g2, g3])
34 row2 = np.hstack([g4, g5, g6])
35 row3 = np.hstack([g7, g8, g9])
36 row4 = np.hstack([g10, g11, g12])
37 row5 = np.hstack([g13, g14, g15])
38 row6 = np.hstack([g16, g17, g18])
39 row7 = np.hstack([g19, g20, g21])
40
41 grid_gray = np.vstack([row1, row2, row3, row4, row5, row6, row7])
42
43 # colored sequence
44 c1 = depth_colored
45 c2 = cv2.blur(depth_colored, (3,3))
46 c3 = cv2.blur(depth_colored, (5,5))
47 c4 = cv2.blur(depth_colored, (7,7))
48 c5 = cv2.blur(depth_colored, (9,9))
49 c6 = cv2.blur(depth_colored, (11,11))
50 c7 = cv2.blur(depth_colored, (13,13))
51 c8 = cv2.blur(depth_colored, (15,15))
52 c9 = cv2.blur(depth_colored, (17,17))
53 c10 = cv2.blur(depth_colored, (19,19))
54 c11 = cv2.blur(depth_colored, (21,21))
```

```
55 c12 = cv2.blur(depth_colored, (23,23))
56 c13 = cv2.blur(depth_colored, (25,25))
57 c14 = cv2.blur(depth_colored, (27,27))
58 c15 = cv2.blur(depth_colored, (29,29))
59 c16 = cv2.blur(depth_colored, (31,31))
60 c17 = cv2.blur(depth_colored, (33,33))
61 c18 = cv2.blur(depth_colored, (35,35))
62 c19 = cv2.blur(depth_colored, (37,37))
63 c20 = cv2.blur(depth_colored, (39,39))
64 c21 = cv2.blur(depth_colored, (41,41))
65
66 row1c = np.hstack([c1, c2, c3])
67 row2c = np.hstack([c4, c5, c6])
68 row3c = np.hstack([c7, c8, c9])
69 row4c = np.hstack([c10, c11, c12])
70 row5c = np.hstack([c13, c14, c15])
71 row6c = np.hstack([c16, c17, c18])
72 row7c = np.hstack([c19, c20, c21])
73
74 grid_colored = np.vstack([row1c, row2c, row3c, row4c, row5c, row6c, row7c])
75
76
77 # cv2.imshow("Gray Grid", grid_gray)
78 # cv2.imshow("Colored Grid", grid_colored)
79
80 # while True:
81 #     key = cv2.waitKey(1) & 0xFF
82 #     if key in (ord('1'), 27): # press '1' or ESC
83 #         break
84
85 # cv2.destroyAllWindows()
86
87 cv2.imwrite("images/04_average_gray_grid.png", grid_gray)
88 cv2.imwrite("images/04_average_colored_grid.png", grid_colored)
```

This blur replaces each pixel with the average of its neighboring pixels under a kernel. **Use:** Simple smoothing, reduces overall noise but also blurs edges.

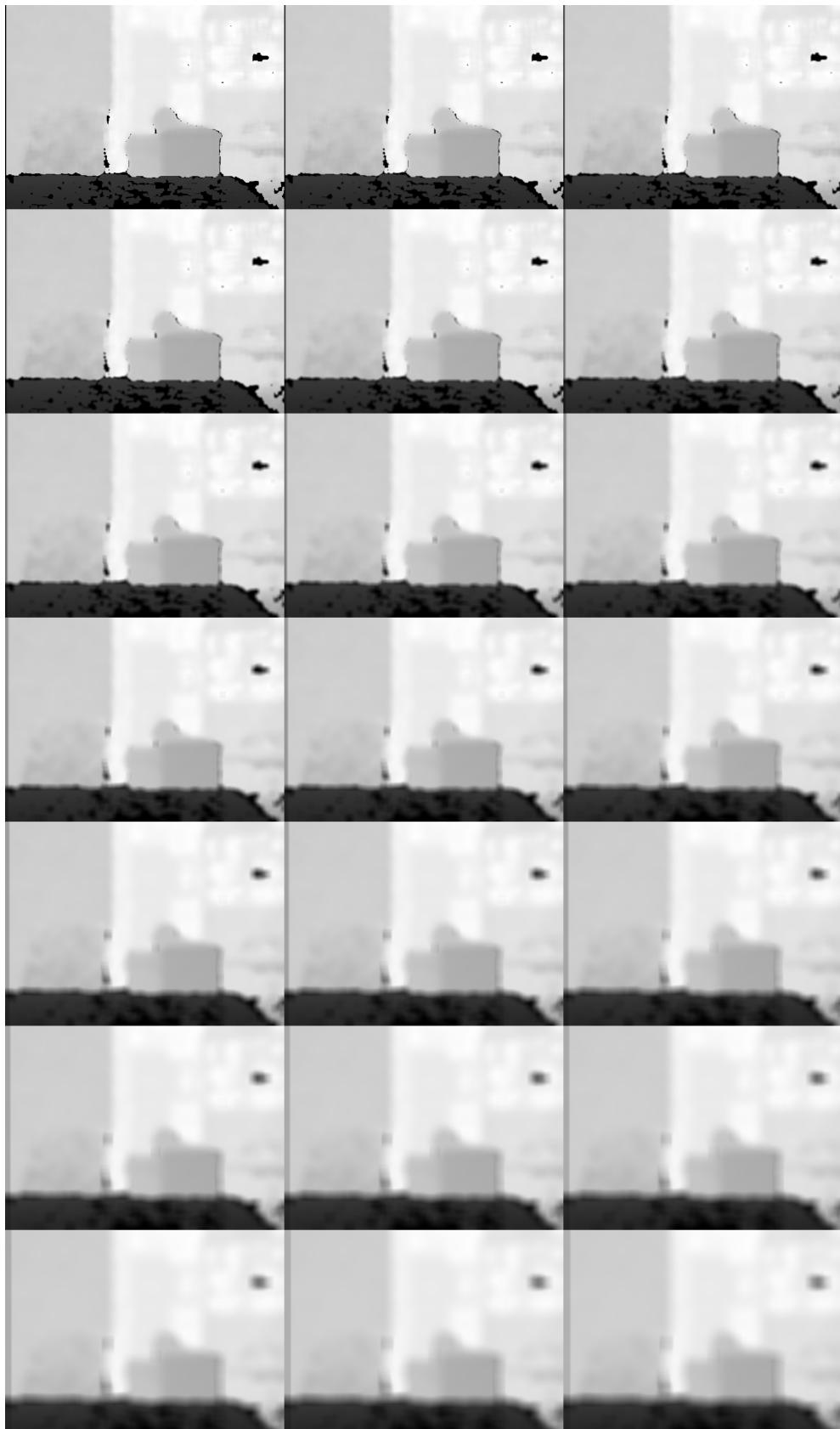
Output (Gray Grid)

Figure 3: Averaging — Gray grid result

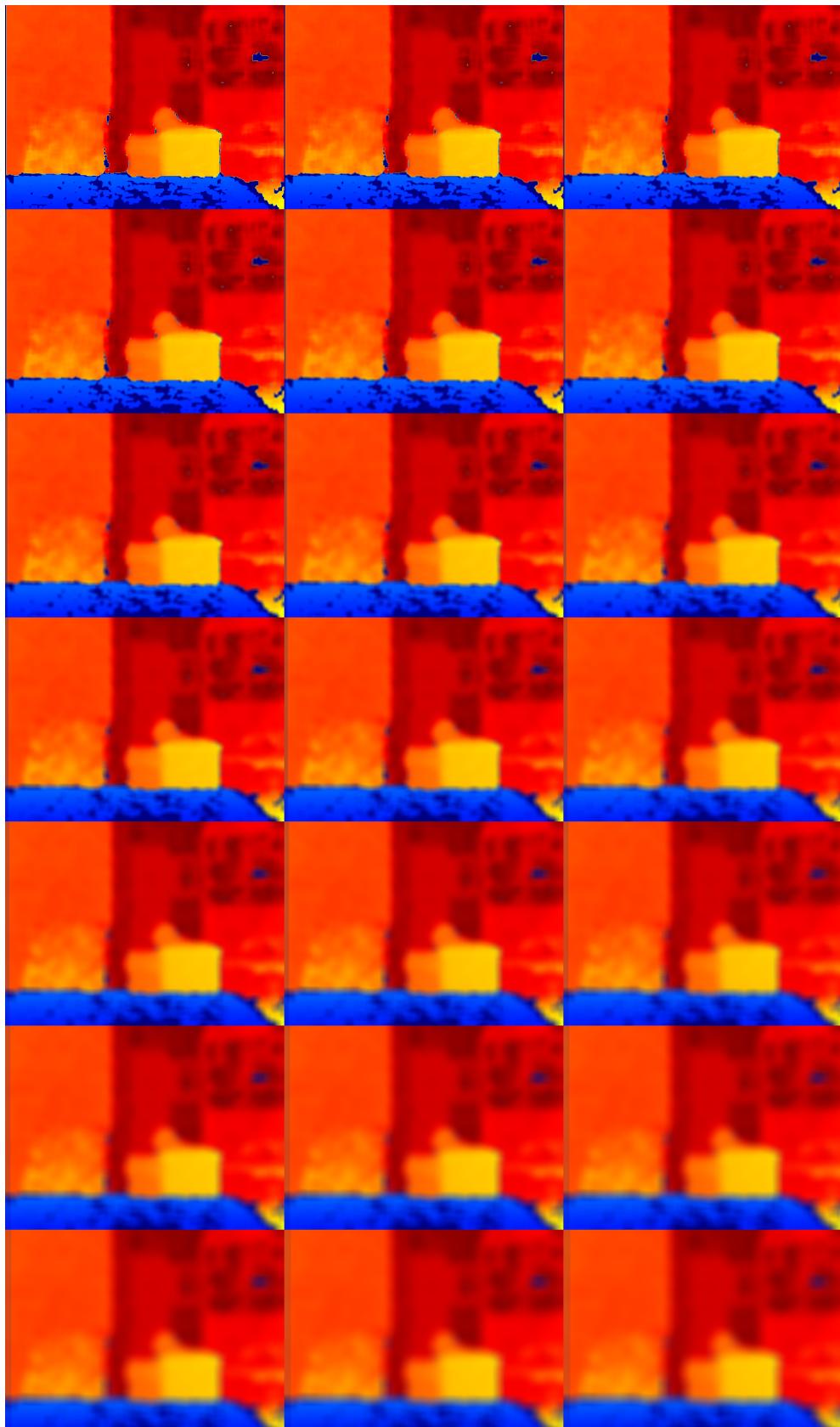
Output (Colored Grid)

Figure 4: Averaging — Colored grid result

4.2 Gaussian Blurring

Code

```
1 import cv2
2 import numpy as np
3
4
5 depth_gray = cv2.imread("images/depth_gray.png", cv2.IMREAD_GRAYSCALE)
6
7 depth_colored = cv2.applyColorMap(depth_gray, cv2.COLORMAP_JET)
8
9 # grayscale sequence
10 g1 = depth_gray
11 g2 = cv2.GaussianBlur(depth_gray, (3,3), 0)
12 g3 = cv2.GaussianBlur(depth_gray, (5,5), 0)
13 g4 = cv2.GaussianBlur(depth_gray, (7,7), 0)
14 g5 = cv2.GaussianBlur(depth_gray, (9,9), 0)
15 g6 = cv2.GaussianBlur(depth_gray, (11,11), 0)
16 g7 = cv2.GaussianBlur(depth_gray, (13,13), 0)
17 g8 = cv2.GaussianBlur(depth_gray, (15,15), 0)
18 g9 = cv2.GaussianBlur(depth_gray, (17,17), 0)
19 g10 = cv2.GaussianBlur(depth_gray, (19,19), 0)
20 g11 = cv2.GaussianBlur(depth_gray, (21,21), 0)
21 g12 = cv2.GaussianBlur(depth_gray, (23,23), 0)
22 g13 = cv2.GaussianBlur(depth_gray, (25,25), 0)
23 g14 = cv2.GaussianBlur(depth_gray, (27,27), 0)
24 g15 = cv2.GaussianBlur(depth_gray, (29,29), 0)
25 g16 = cv2.GaussianBlur(depth_gray, (31,31), 0)
26 g17 = cv2.GaussianBlur(depth_gray, (33,33), 0)
27 g18 = cv2.GaussianBlur(depth_gray, (35,35), 0)
28 g19 = cv2.GaussianBlur(depth_gray, (37,37), 0)
29 g20 = cv2.GaussianBlur(depth_gray, (39,39), 0)
30 g21 = cv2.GaussianBlur(depth_gray, (41,41), 0)
31
32
33 # build 7x3 grid
34 row1 = np.hstack([g1, g2, g3])
35 row2 = np.hstack([g4, g5, g6])
36 row3 = np.hstack([g7, g8, g9])
37 row4 = np.hstack([g10, g11, g12])
38 row5 = np.hstack([g13, g14, g15])
39 row6 = np.hstack([g16, g17, g18])
40 row7 = np.hstack([g19, g20, g21])
41
42 grid_gray = np.vstack([row1, row2, row3, row4, row5, row6, row7])
43
44 # colored sequence
45 c1 = depth_colored
46 c2 = cv2.GaussianBlur(depth_colored, (3,3), 0)
47 c3 = cv2.GaussianBlur(depth_colored, (5,5), 0)
48 c4 = cv2.GaussianBlur(depth_colored, (7,7), 0)
49 c5 = cv2.GaussianBlur(depth_colored, (9,9), 0)
50 c6 = cv2.GaussianBlur(depth_colored, (11,11), 0)
51 c7 = cv2.GaussianBlur(depth_colored, (13,13), 0)
52 c8 = cv2.GaussianBlur(depth_colored, (15,15), 0)
53 c9 = cv2.GaussianBlur(depth_colored, (17,17), 0)
54 c10 = cv2.GaussianBlur(depth_colored, (19,19), 0)
55 c11 = cv2.GaussianBlur(depth_colored, (21,21), 0)
56 c12 = cv2.GaussianBlur(depth_colored, (23,23), 0)
57 c13 = cv2.GaussianBlur(depth_colored, (25,25), 0)
58 c14 = cv2.GaussianBlur(depth_colored, (27,27), 0)
```

```
59 c15 = cv2.GaussianBlur(depth_colored, (29,29), 0)
60 c16 = cv2.GaussianBlur(depth_colored, (31,31), 0)
61 c17 = cv2.GaussianBlur(depth_colored, (33,33), 0)
62 c18 = cv2.GaussianBlur(depth_colored, (35,35), 0)
63 c19 = cv2.GaussianBlur(depth_colored, (37,37), 0)
64 c20 = cv2.GaussianBlur(depth_colored, (39,39), 0)
65 c21 = cv2.GaussianBlur(depth_colored, (41,41), 0)
66
67 row1c = np.hstack([c1, c2, c3])
68 row2c = np.hstack([c4, c5, c6])
69 row3c = np.hstack([c7, c8, c9])
70 row4c = np.hstack([c10, c11, c12])
71 row5c = np.hstack([c13, c14, c15])
72 row6c = np.hstack([c16, c17, c18])
73 row7c = np.hstack([c19, c20, c21])
74
75 grid_colored = np.vstack([row1c, row2c, row3c, row4c, row5c, row6c, row7c])
76
77 # cv2.imshow("Gray Grid", grid_gray)
78 # cv2.imshow("Colored Grid", grid_colored)
79
80 # while True:
81 #     key = cv2.waitKey(1) & 0xFF
82 #     if key in (ord('1'), 27): # press '1' or ESC
83 #         break
84
85 # cv2.destroyAllWindows()
86
87 cv2.imwrite("images/05_gaussian_gray_grid.png", grid_gray)
88 cv2.imwrite("images/05_gaussian_colored_grid.png", grid_colored)
```

This blur uses a Gaussian kernel, giving more weight to closer pixels. **Use:** Effective for removing Gaussian noise while keeping smoother transitions.

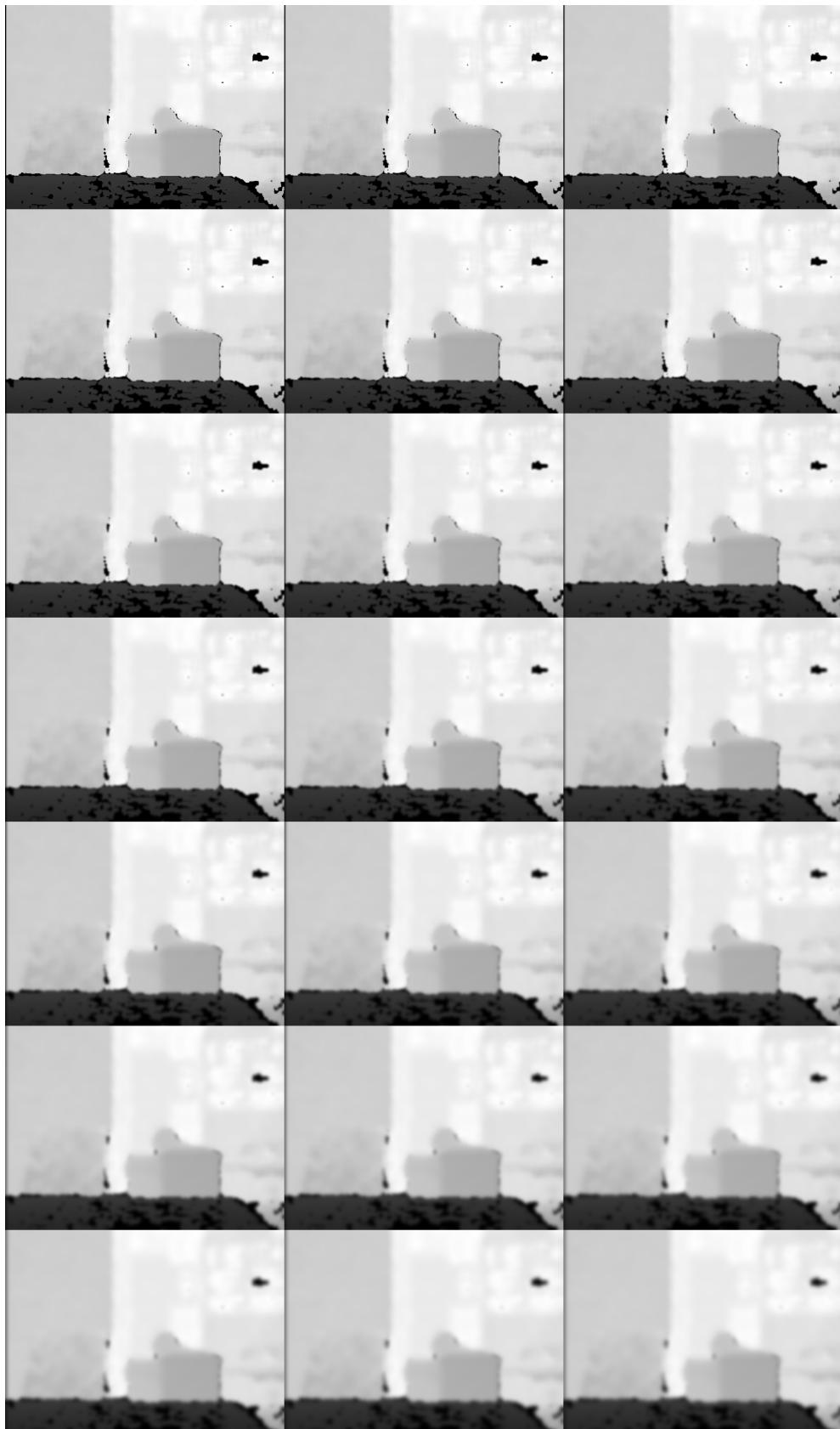
Output (Gray Grid)

Figure 5: Gaussian blurring — Gray grid result

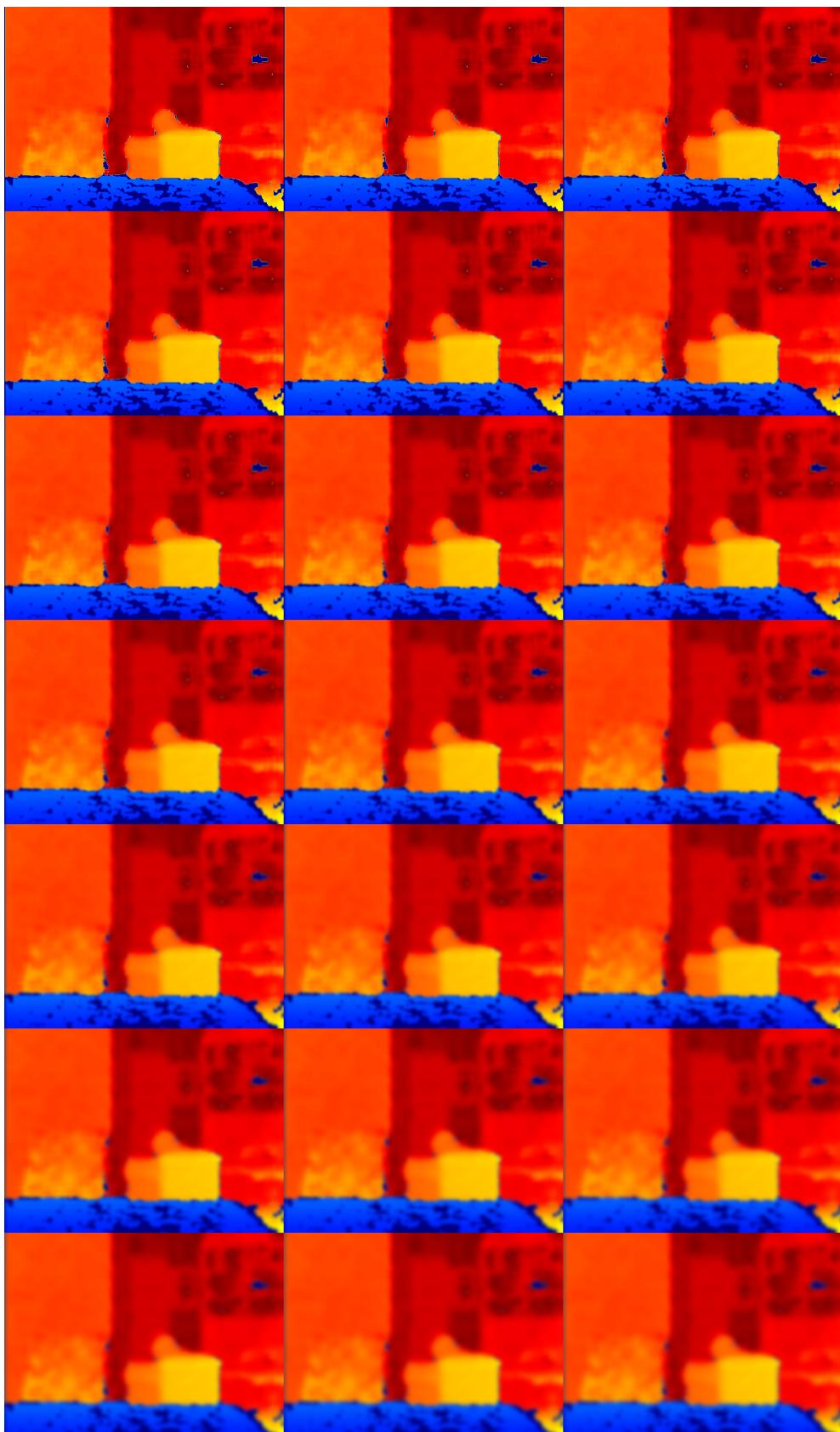
Output (Colored Grid)

Figure 6: Gaussian blurring — Colored grid result

4.3 Median Blurring

Code

```
1 import cv2
2 import numpy as np
3
4
5 depth_gray = cv2.imread("images/depth_gray.png", cv2.IMREAD_GRAYSCALE)
6
7 depth_colored = cv2.applyColorMap(depth_gray, cv2.COLORMAP_JET)
8
9 # grayscale sequence
10 g1 = depth_gray
11 g2 = cv2.medianBlur(depth_gray, 3)
12 g3 = cv2.medianBlur(depth_gray, 5)
13 g4 = cv2.medianBlur(depth_gray, 7)
14 g5 = cv2.medianBlur(depth_gray, 9)
15 g6 = cv2.medianBlur(depth_gray, 11)
16 g7 = cv2.medianBlur(depth_gray, 13)
17 g8 = cv2.medianBlur(depth_gray, 15)
18 g9 = cv2.medianBlur(depth_gray, 17)
19 g10 = cv2.medianBlur(depth_gray, 19)
20 g11 = cv2.medianBlur(depth_gray, 21)
21 g12 = cv2.medianBlur(depth_gray, 23)
22 g13 = cv2.medianBlur(depth_gray, 25)
23 g14 = cv2.medianBlur(depth_gray, 27)
24 g15 = cv2.medianBlur(depth_gray, 29)
25 g16 = cv2.medianBlur(depth_gray, 31)
26 g17 = cv2.medianBlur(depth_gray, 33)
27 g18 = cv2.medianBlur(depth_gray, 35)
28 g19 = cv2.medianBlur(depth_gray, 37)
29 g20 = cv2.medianBlur(depth_gray, 39)
30 g21 = cv2.medianBlur(depth_gray, 41)
31
32
33
34 # build 7x3 grid
35 row1 = np.hstack([g1, g2, g3])
36 row2 = np.hstack([g4, g5, g6])
37 row3 = np.hstack([g7, g8, g9])
38 row4 = np.hstack([g10, g11, g12])
39 row5 = np.hstack([g13, g14, g15])
40 row6 = np.hstack([g16, g17, g18])
41 row7 = np.hstack([g19, g20, g21])
42
43 grid_gray = np.vstack([row1, row2, row3, row4, row5, row6, row7])
44
45 # colored sequence
46 c1 = depth_colored
47 c2 = cv2.medianBlur(depth_colored, 3)
48 c3 = cv2.medianBlur(depth_colored, 5)
49 c4 = cv2.medianBlur(depth_colored, 7)
50 c5 = cv2.medianBlur(depth_colored, 9)
51 c6 = cv2.medianBlur(depth_colored, 11)
52 c7 = cv2.medianBlur(depth_colored, 13)
53 c8 = cv2.medianBlur(depth_colored, 15)
54 c9 = cv2.medianBlur(depth_colored, 17)
55 c10 = cv2.medianBlur(depth_colored, 19)
56 c11 = cv2.medianBlur(depth_colored, 21)
57 c12 = cv2.medianBlur(depth_colored, 23)
58 c13 = cv2.medianBlur(depth_colored, 25)
```

```
59 c14 = cv2.medianBlur(depth_colored, 27)
60 c15 = cv2.medianBlur(depth_colored, 29)
61 c16 = cv2.medianBlur(depth_colored, 31)
62 c17 = cv2.medianBlur(depth_colored, 33)
63 c18 = cv2.medianBlur(depth_colored, 35)
64 c19 = cv2.medianBlur(depth_colored, 37)
65 c20 = cv2.medianBlur(depth_colored, 39)
66 c21 = cv2.medianBlur(depth_colored, 41)
67
68 row1c = np.hstack([c1, c2, c3])
69 row2c = np.hstack([c4, c5, c6])
70 row3c = np.hstack([c7, c8, c9])
71 row4c = np.hstack([c10, c11, c12])
72 row5c = np.hstack([c13, c14, c15])
73 row6c = np.hstack([c16, c17, c18])
74 row7c = np.hstack([c19, c20, c21])
75
76 grid_colored = np.vstack([row1c, row2c, row3c, row4c, row5c, row6c, row7c])
77
78 # cv2.imshow("Gray Grid", grid_gray)
79 # cv2.imshow("Colored Grid", grid_colored)
80
81 # while True:
82 #     key = cv2.waitKey(1) & 0xFF
83 #     if key in (ord('1'), 27): # press '1' or ESC
84 #         break
85
86 # cv2.destroyAllWindows()
87
88 # save each grid separately
89 cv2.imwrite("images/06_median_gray_grid.png", grid_gray)
90 cv2.imwrite("images/06_median_colored_grid.png", grid_colored)
```

This blur replaces each pixel with the median of neighboring pixels. **Use:** Very effective against salt-and-pepper noise, preserves edges better.

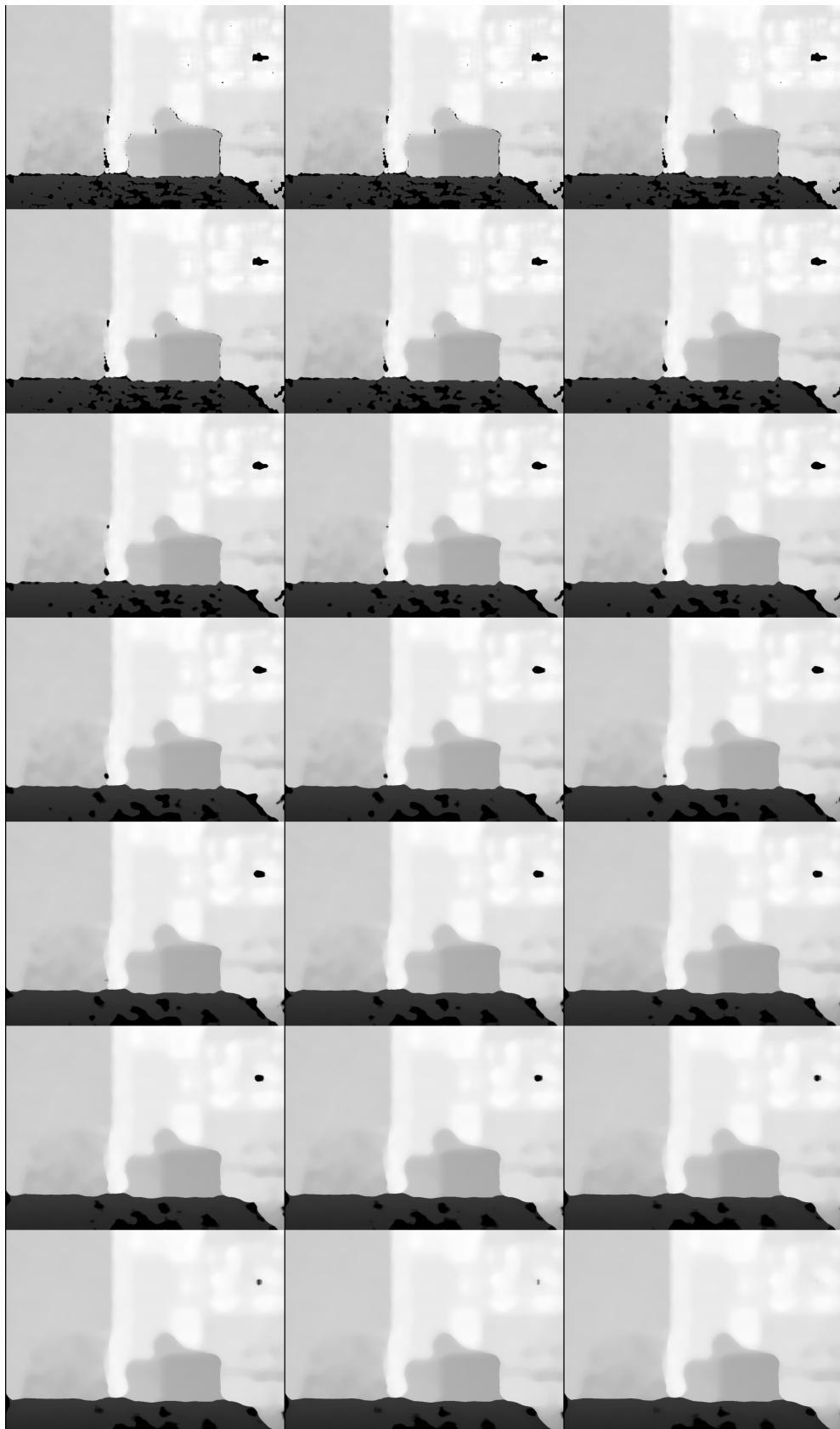
Output (Gray Grid)

Figure 7: Median blurring — Gray grid result

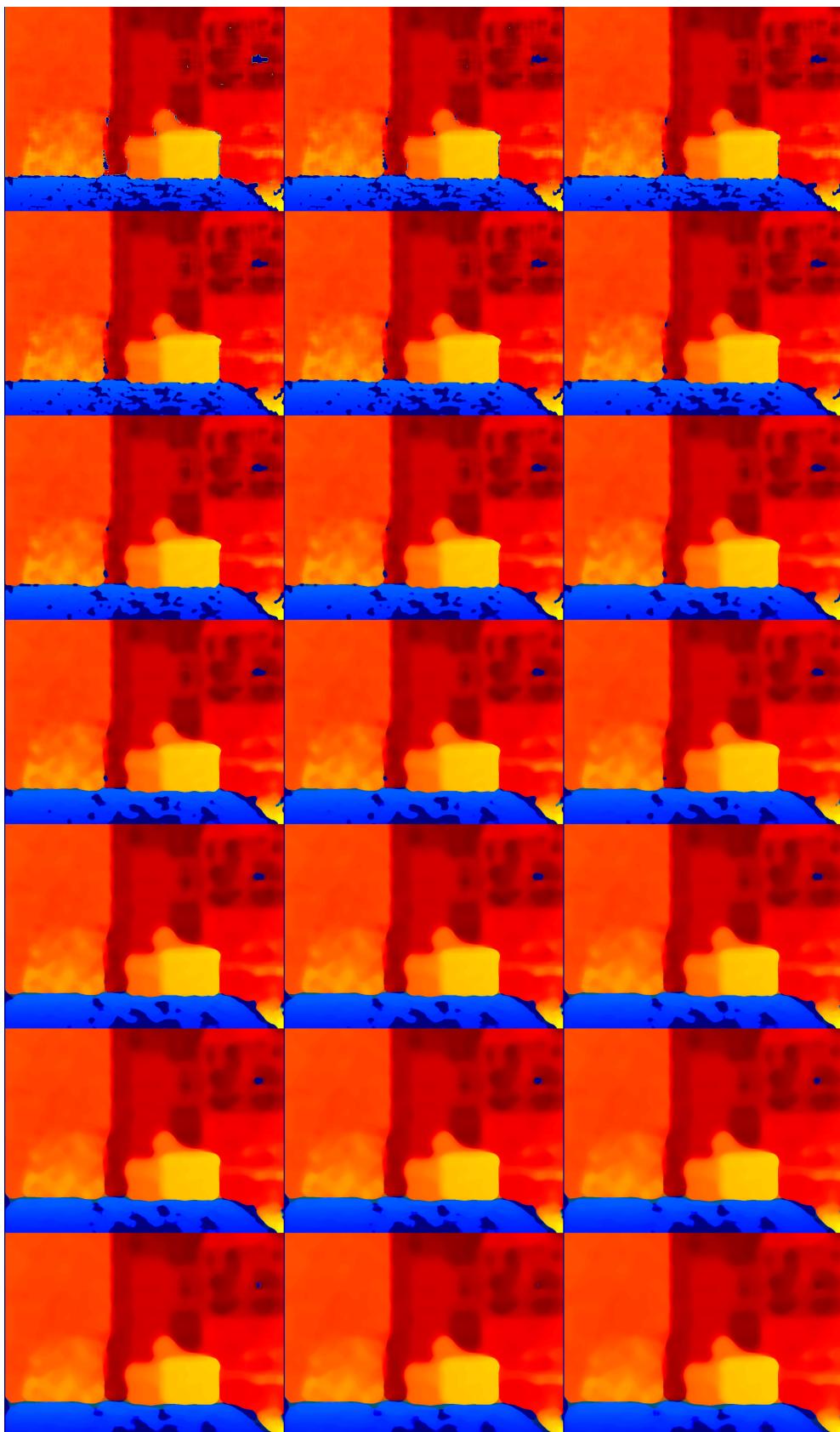
Output (Colored Grid)

Figure 8: Median blurring — Colored grid result

4.4 Bilateral Filtering

Code

```
1 import cv2
2 import numpy as np
3
4
5 depth_gray = cv2.imread("images/depth_gray.png", cv2.IMREAD_GRAYSCALE)
6
7 depth_colored = cv2.applyColorMap(depth_gray, cv2.COLORMAP_JET)
8
9 # grayscale sequence
10 g1 = depth_gray
11 g2 = cv2.bilateralFilter(depth_gray, 3, 75, 75)
12 g3 = cv2.bilateralFilter(depth_gray, 5, 75, 75)
13 g4 = cv2.bilateralFilter(depth_gray, 7, 75, 75)
14 g5 = cv2.bilateralFilter(depth_gray, 9, 75, 75)
15 g6 = cv2.bilateralFilter(depth_gray, 11, 75, 75)
16 g7 = cv2.bilateralFilter(depth_gray, 13, 75, 75)
17 g8 = cv2.bilateralFilter(depth_gray, 15, 75, 75)
18 g9 = cv2.bilateralFilter(depth_gray, 17, 75, 75)
19 g10 = cv2.bilateralFilter(depth_gray, 19, 75, 75)
20 g11 = cv2.bilateralFilter(depth_gray, 21, 75, 75)
21 g12 = cv2.bilateralFilter(depth_gray, 23, 75, 75)
22 g13 = cv2.bilateralFilter(depth_gray, 25, 75, 75)
23 g14 = cv2.bilateralFilter(depth_gray, 27, 75, 75)
24 g15 = cv2.bilateralFilter(depth_gray, 29, 75, 75)
25 g16 = cv2.bilateralFilter(depth_gray, 31, 75, 75)
26 g17 = cv2.bilateralFilter(depth_gray, 33, 75, 75)
27 g18 = cv2.bilateralFilter(depth_gray, 35, 75, 75)
28 g19 = cv2.bilateralFilter(depth_gray, 37, 75, 75)
29 g20 = cv2.bilateralFilter(depth_gray, 39, 75, 75)
30 g21 = cv2.bilateralFilter(depth_gray, 41, 75, 75)
31
32
33
34 # build 7x3 grid
35 row1 = np.hstack([g1, g2, g3])
36 row2 = np.hstack([g4, g5, g6])
37 row3 = np.hstack([g7, g8, g9])
38 row4 = np.hstack([g10, g11, g12])
39 row5 = np.hstack([g13, g14, g15])
40 row6 = np.hstack([g16, g17, g18])
41 row7 = np.hstack([g19, g20, g21])
42
43 grid_gray = np.vstack([row1, row2, row3, row4, row5, row6, row7])
44
45 # colored sequence
46 c1 = depth_colored
47 c2 = cv2.bilateralFilter(depth_colored, 3, 75, 75)
48 c3 = cv2.bilateralFilter(depth_colored, 5, 75, 75)
49 c4 = cv2.bilateralFilter(depth_colored, 7, 75, 75)
50 c5 = cv2.bilateralFilter(depth_colored, 9, 75, 75)
51 c6 = cv2.bilateralFilter(depth_colored, 11, 75, 75)
52 c7 = cv2.bilateralFilter(depth_colored, 13, 75, 75)
53 c8 = cv2.bilateralFilter(depth_colored, 15, 75, 75)
54 c9 = cv2.bilateralFilter(depth_colored, 17, 75, 75)
55 c10 = cv2.bilateralFilter(depth_colored, 19, 75, 75)
56 c11 = cv2.bilateralFilter(depth_colored, 21, 75, 75)
57 c12 = cv2.bilateralFilter(depth_colored, 23, 75, 75)
58 c13 = cv2.bilateralFilter(depth_colored, 25, 75, 75)
```

```
59 c14 = cv2.bilateralFilter(depth_colored, 27, 75, 75)
60 c15 = cv2.bilateralFilter(depth_colored, 29, 75, 75)
61 c16 = cv2.bilateralFilter(depth_colored, 31, 75, 75)
62 c17 = cv2.bilateralFilter(depth_colored, 33, 75, 75)
63 c18 = cv2.bilateralFilter(depth_colored, 35, 75, 75)
64 c19 = cv2.bilateralFilter(depth_colored, 37, 75, 75)
65 c20 = cv2.bilateralFilter(depth_colored, 39, 75, 75)
66 c21 = cv2.bilateralFilter(depth_colored, 41, 75, 75)
67
68 row1c = np.hstack([c1, c2, c3])
69 row2c = np.hstack([c4, c5, c6])
70 row3c = np.hstack([c7, c8, c9])
71 row4c = np.hstack([c10, c11, c12])
72 row5c = np.hstack([c13, c14, c15])
73 row6c = np.hstack([c16, c17, c18])
74 row7c = np.hstack([c19, c20, c21])
75
76 grid_colored = np.vstack([row1c, row2c, row3c, row4c, row5c, row6c, row7c])
77
78 # cv2.imshow("Gray Grid", grid_gray)
79 # cv2.imshow("Colored Grid", grid_colored)
80
81 # while True:
82 #     key = cv2.waitKey(1) & 0xFF
83 #     if key in (ord('1'), 27): # press '1' or ESC
84 #         break
85
86 # cv2.destroyAllWindows()
87
88 # save each grid separately
89 cv2.imwrite("images/07_bilateral_gray_grid.png", grid_gray)
90 cv2.imwrite("images/07_bilateral_colored_grid.png", grid_colored)
```

This blur considers both spatial closeness and pixel intensity similarity. **Use:** Removes noise while preserving sharp edges, but slower than others.

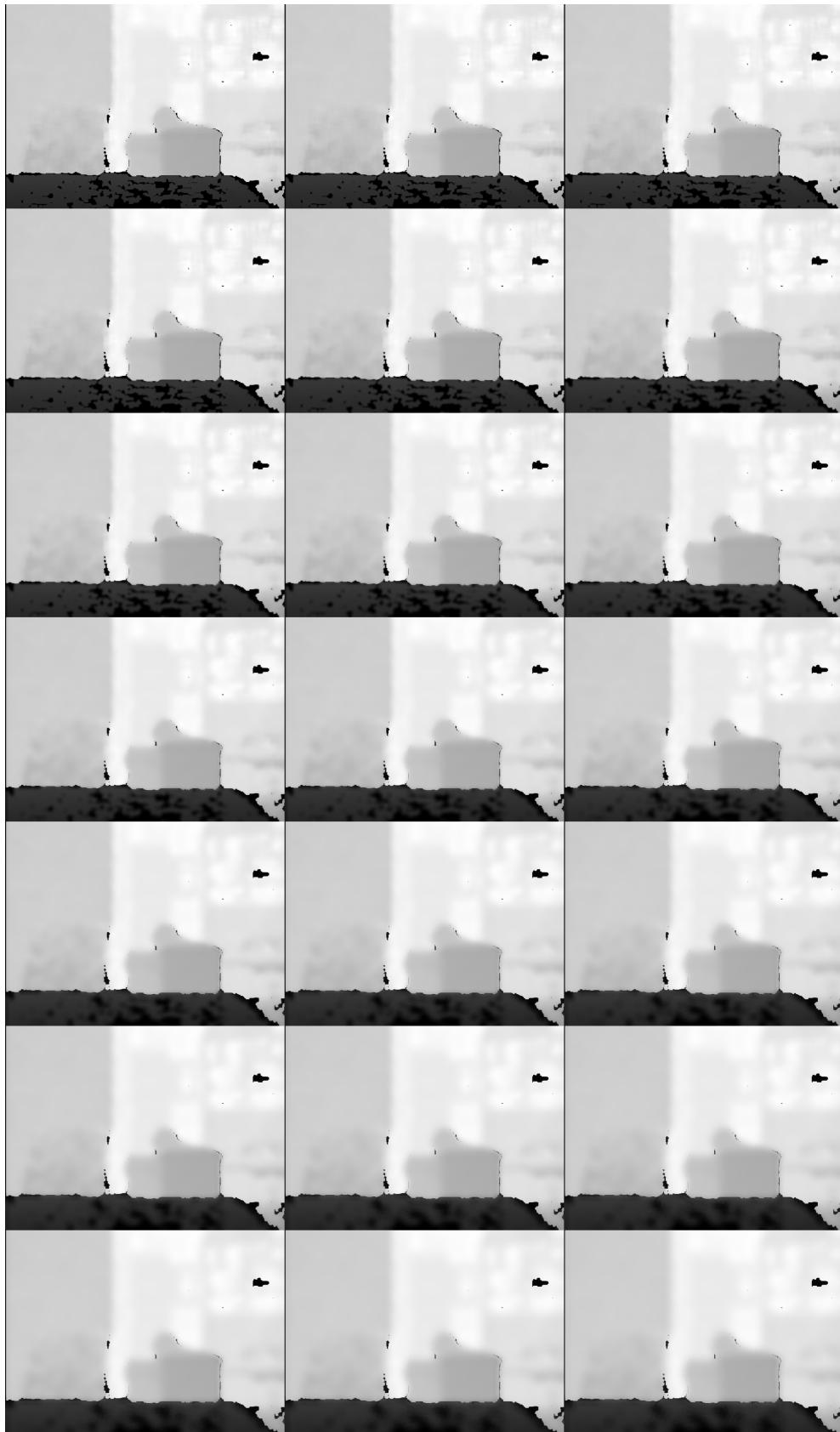


Figure 9: Bilateral filtering — Gray grid result

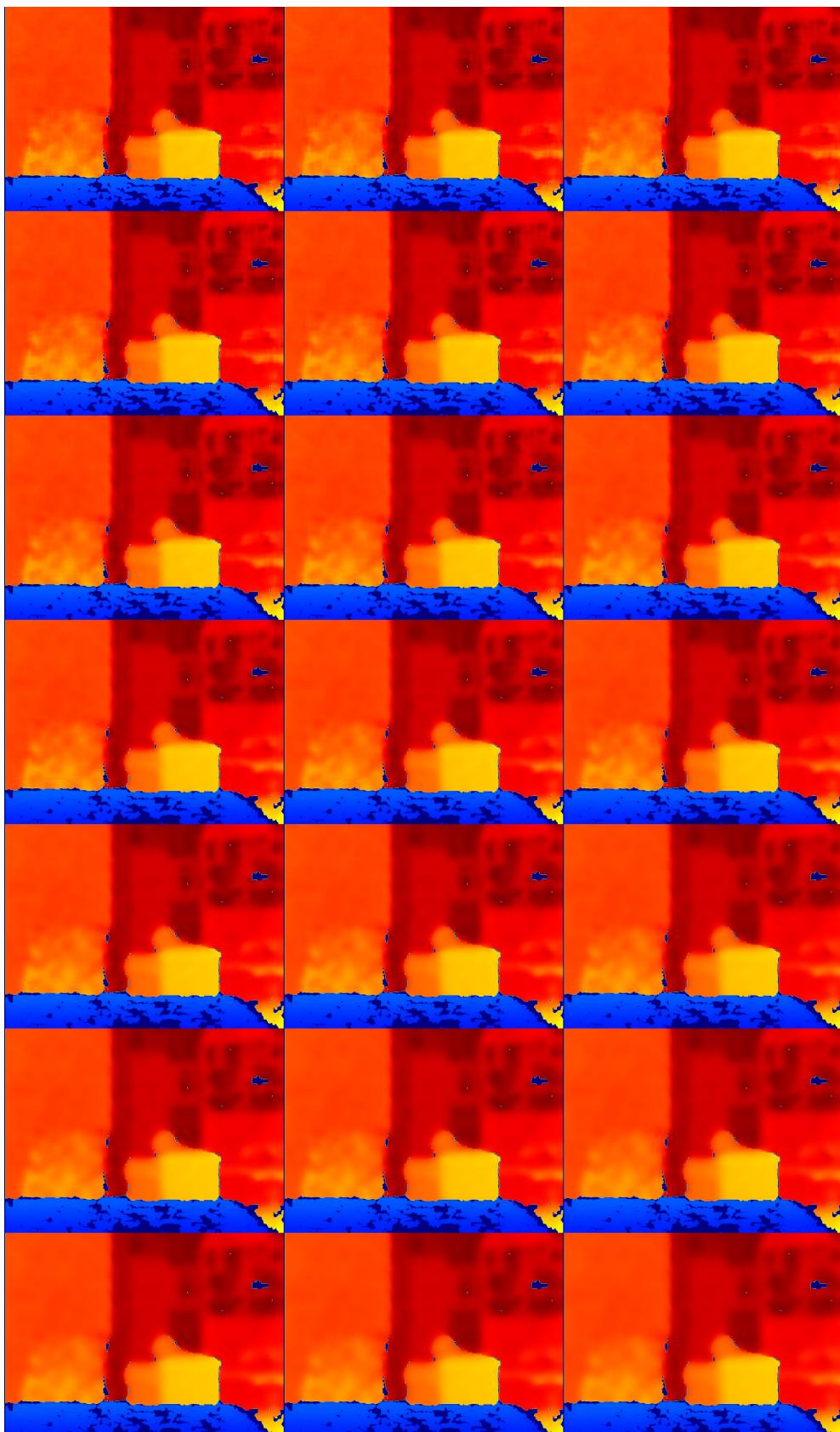
Output (Colored Grid)

Figure 10: Bilateral filtering — Colored grid result

5 Canny Edge Detection on Blur

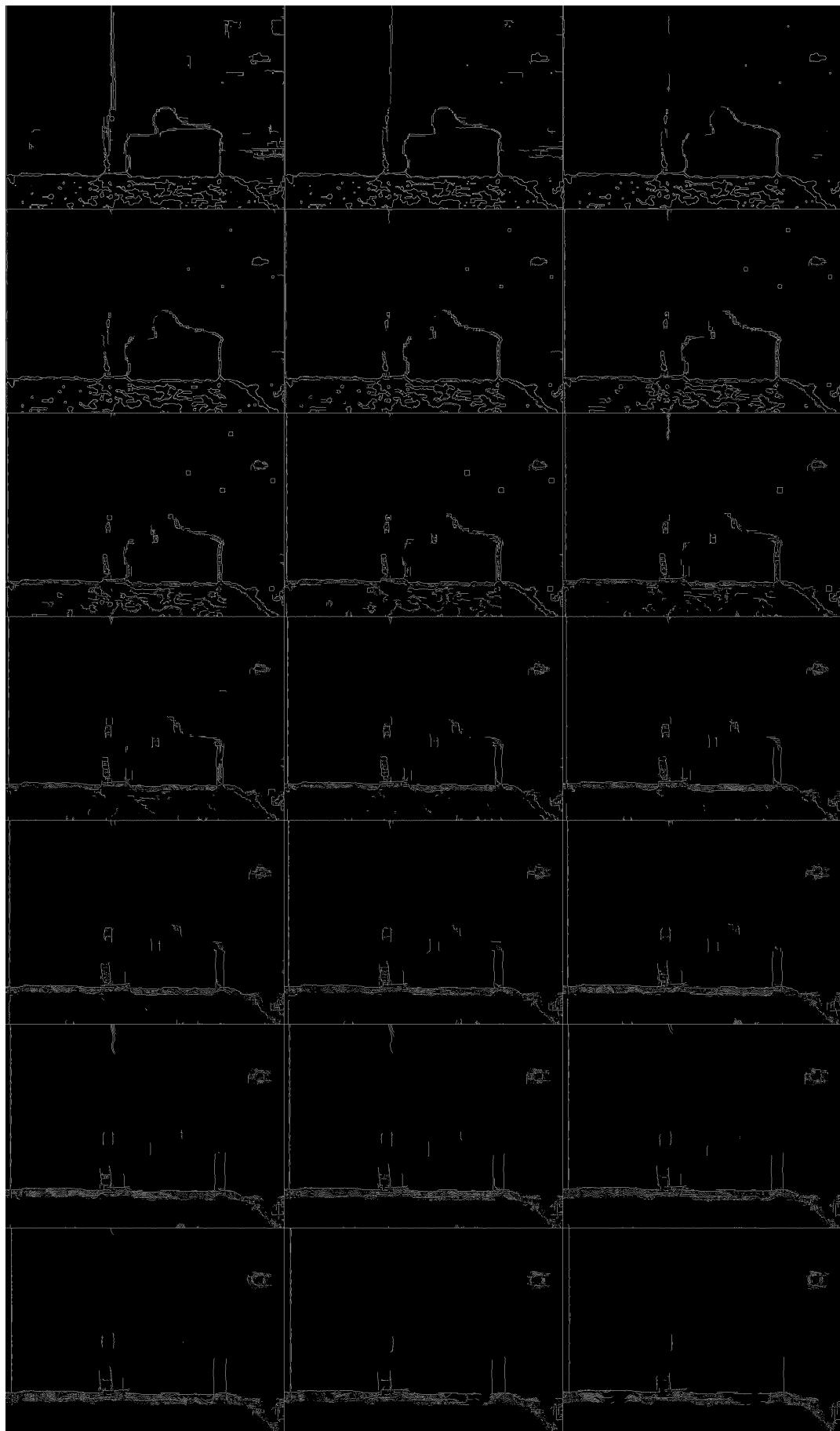


Figure 11: avg_01_canny_10_30

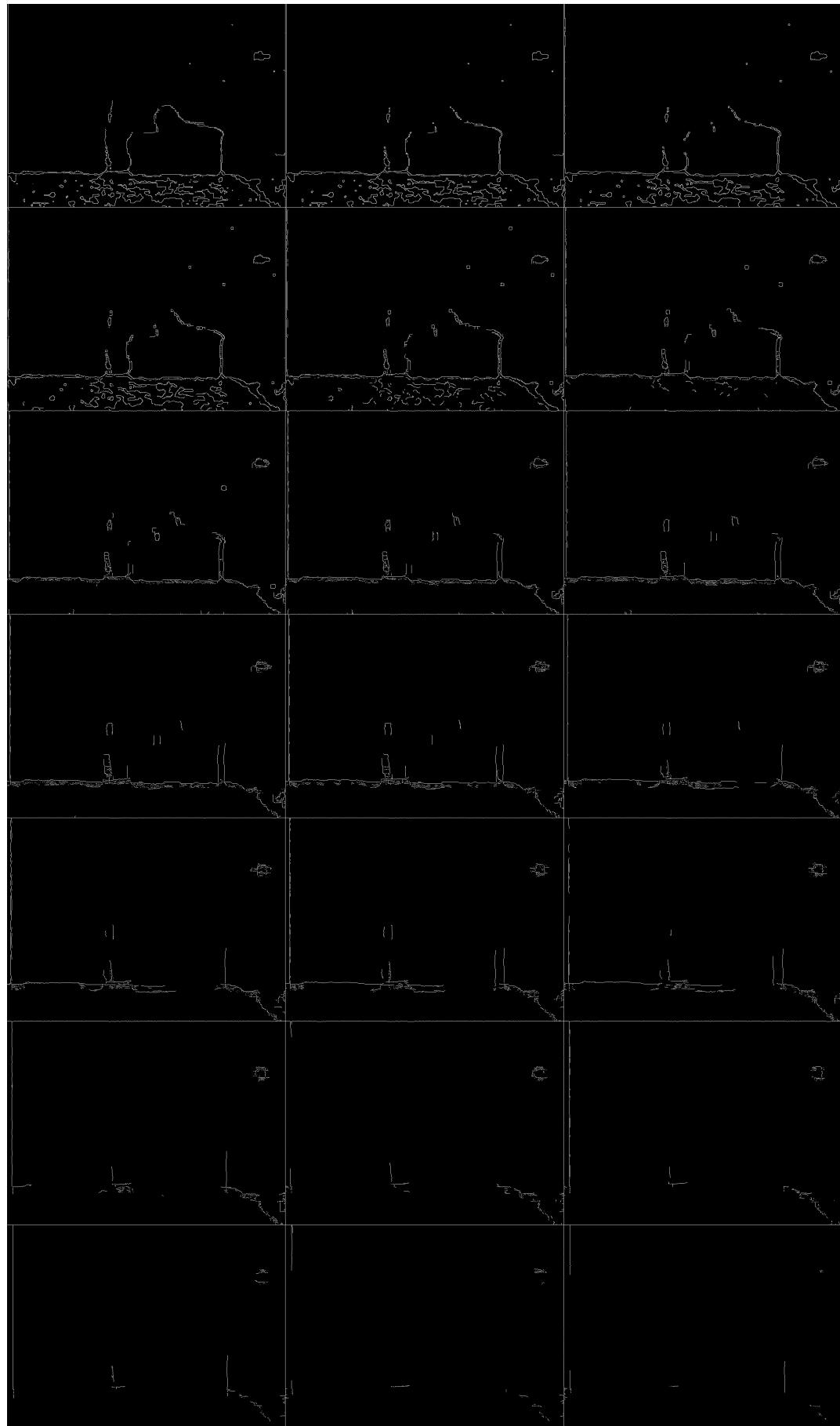


Figure 12: avg_02_canny_20_60

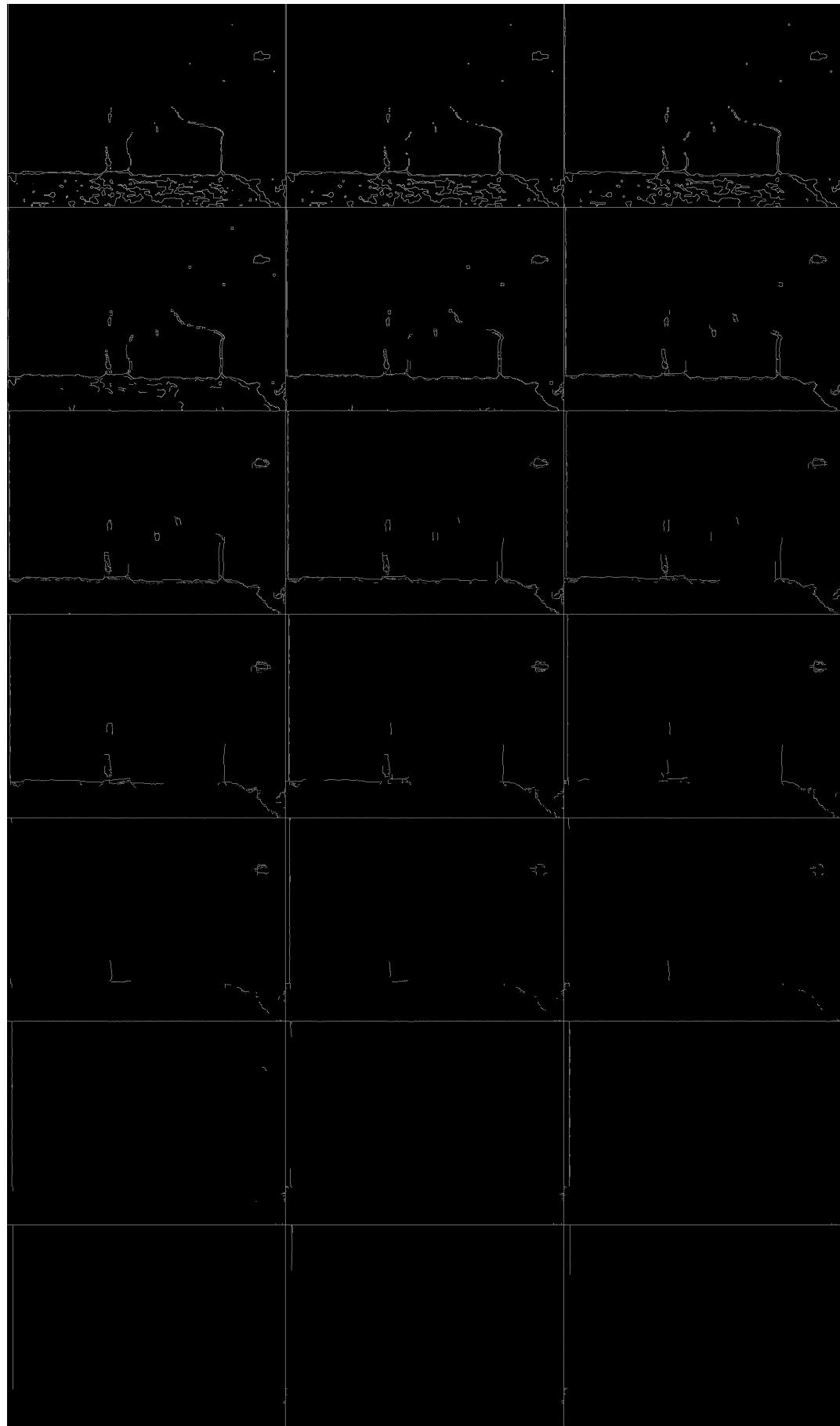


Figure 13: avg_03_canny_30_90

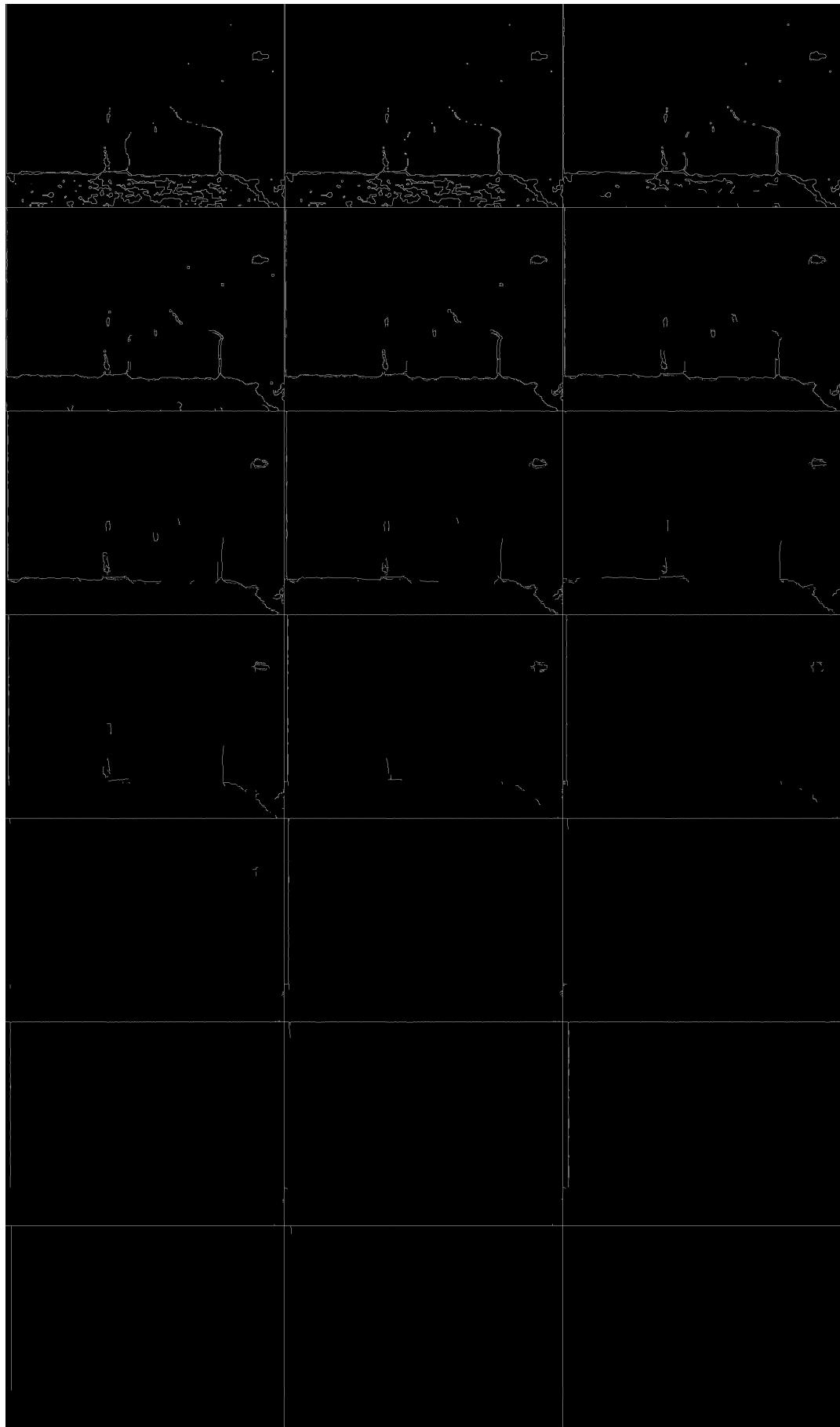


Figure 14: avg_04_canny_40_120

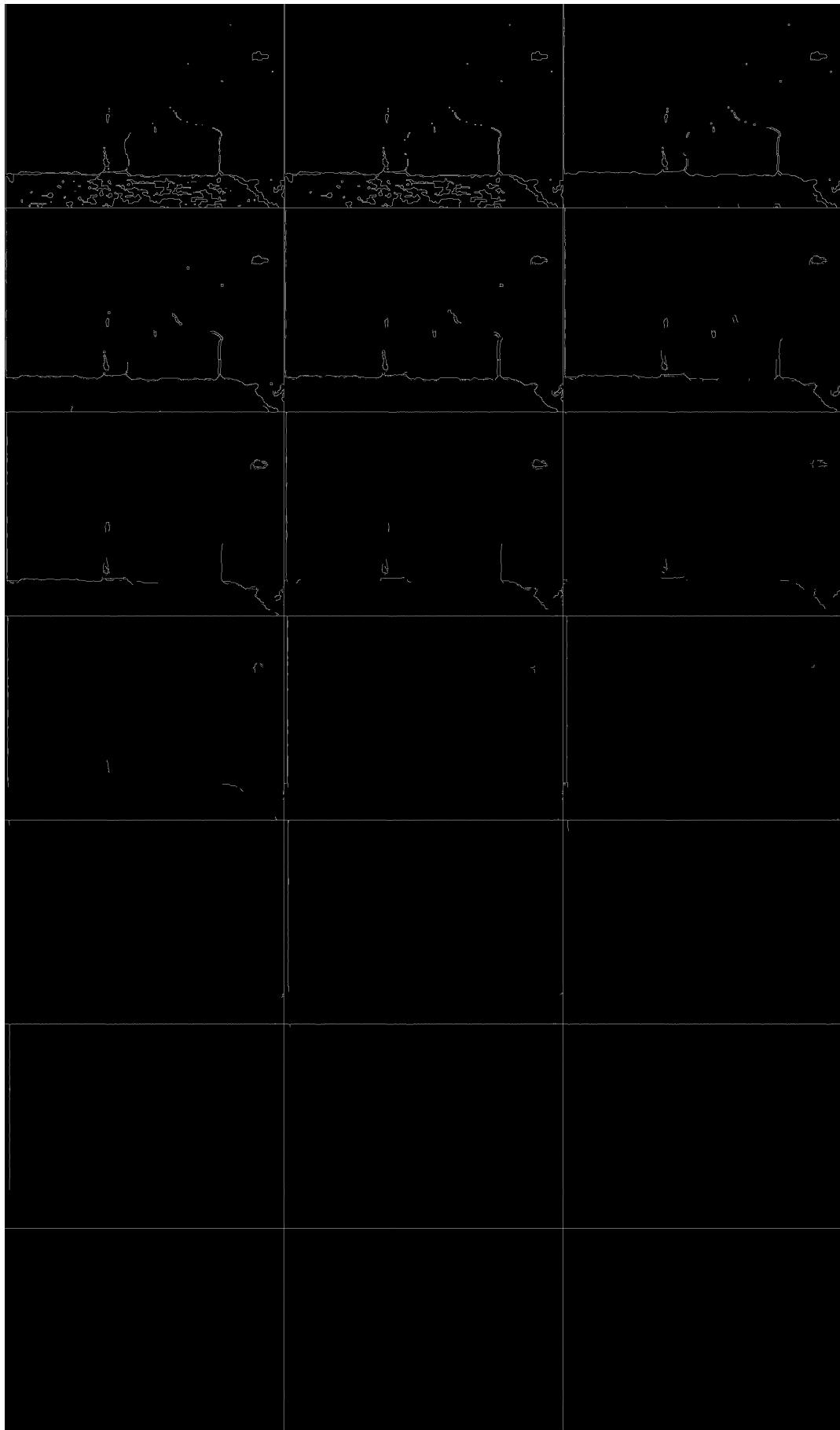


Figure 15: avg_05_canny_50_150

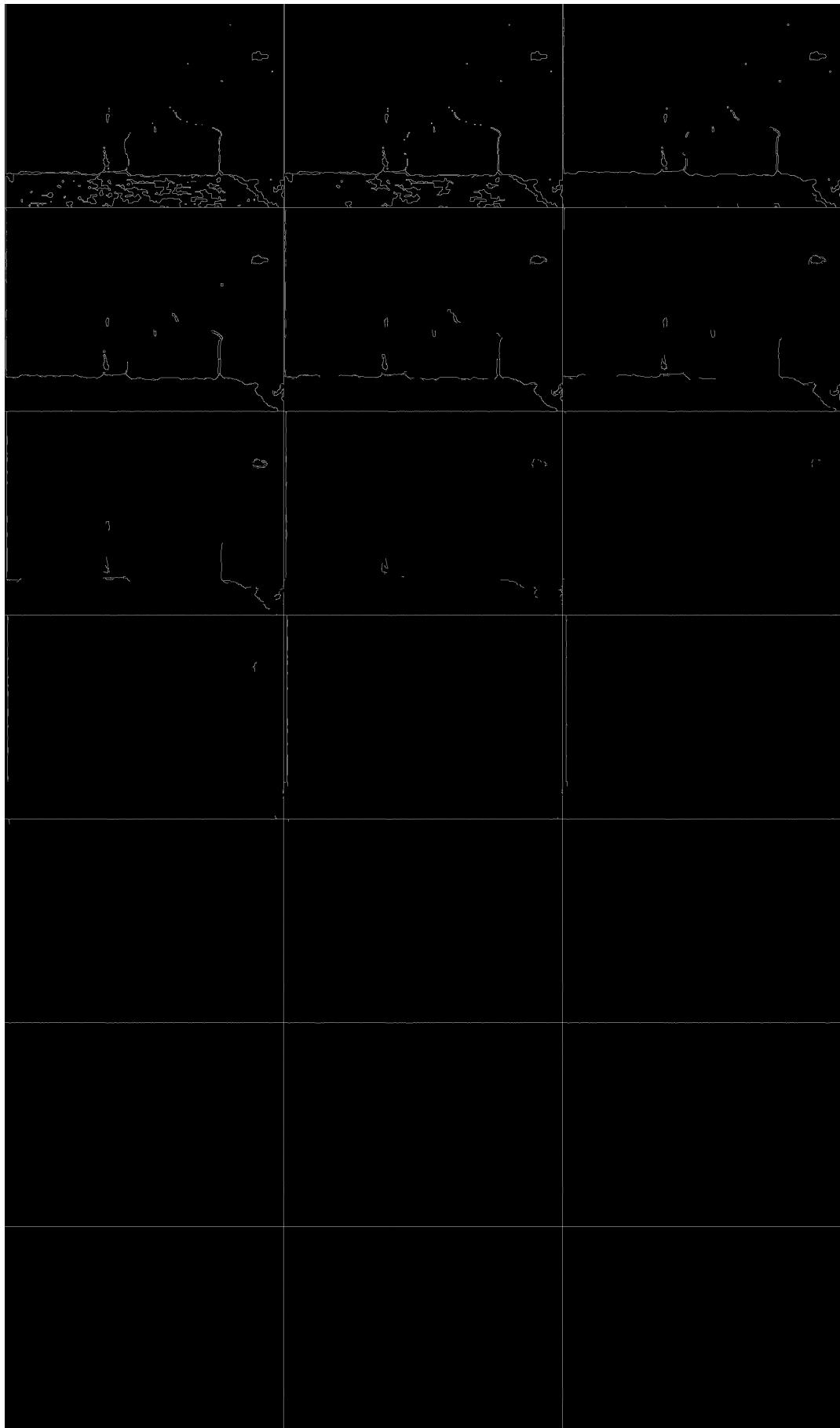


Figure 16: avg_06_canny_60_180

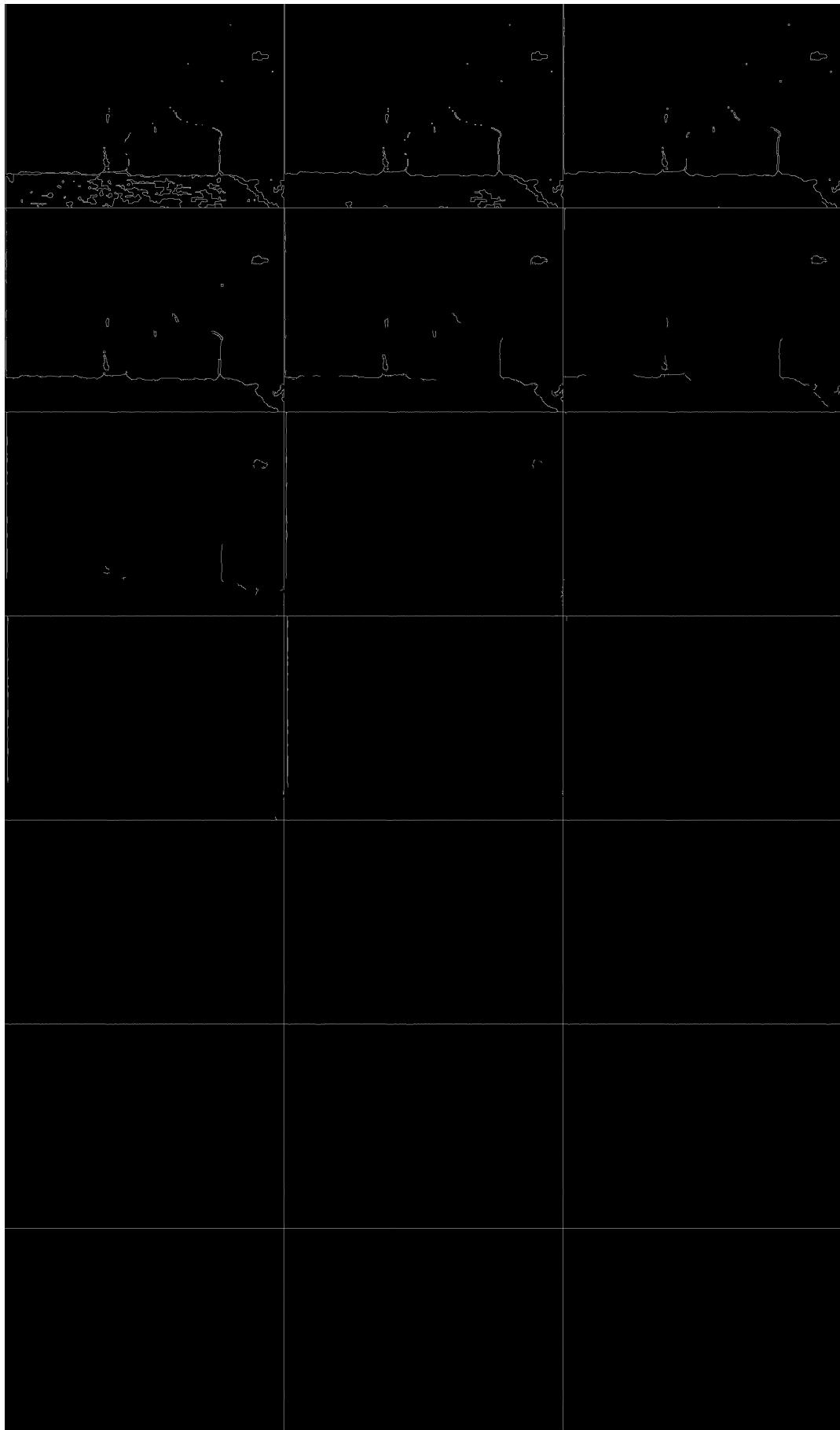


Figure 17: avg_07_canny_70_210

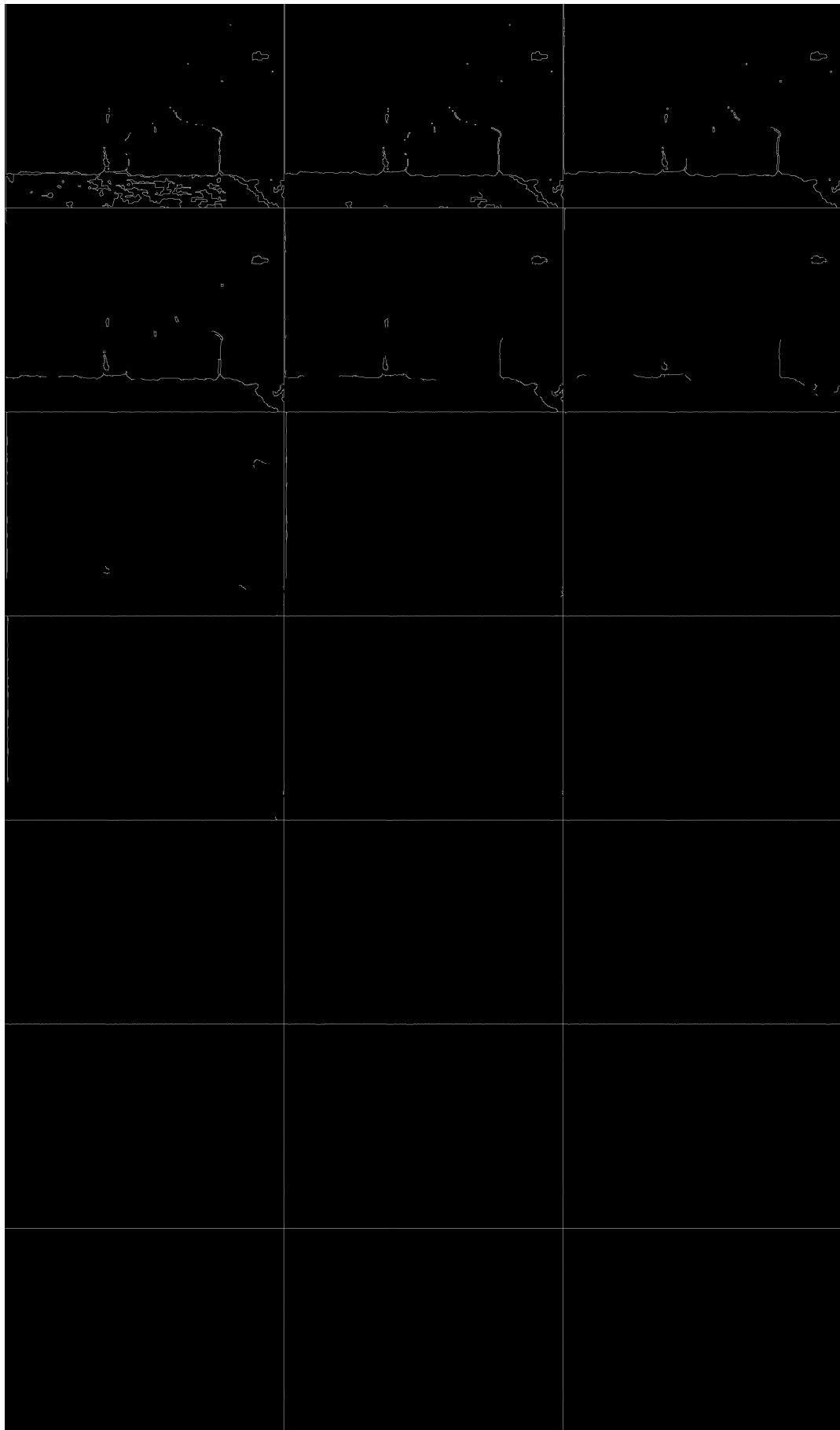


Figure 18: avg_08_canny_80_240

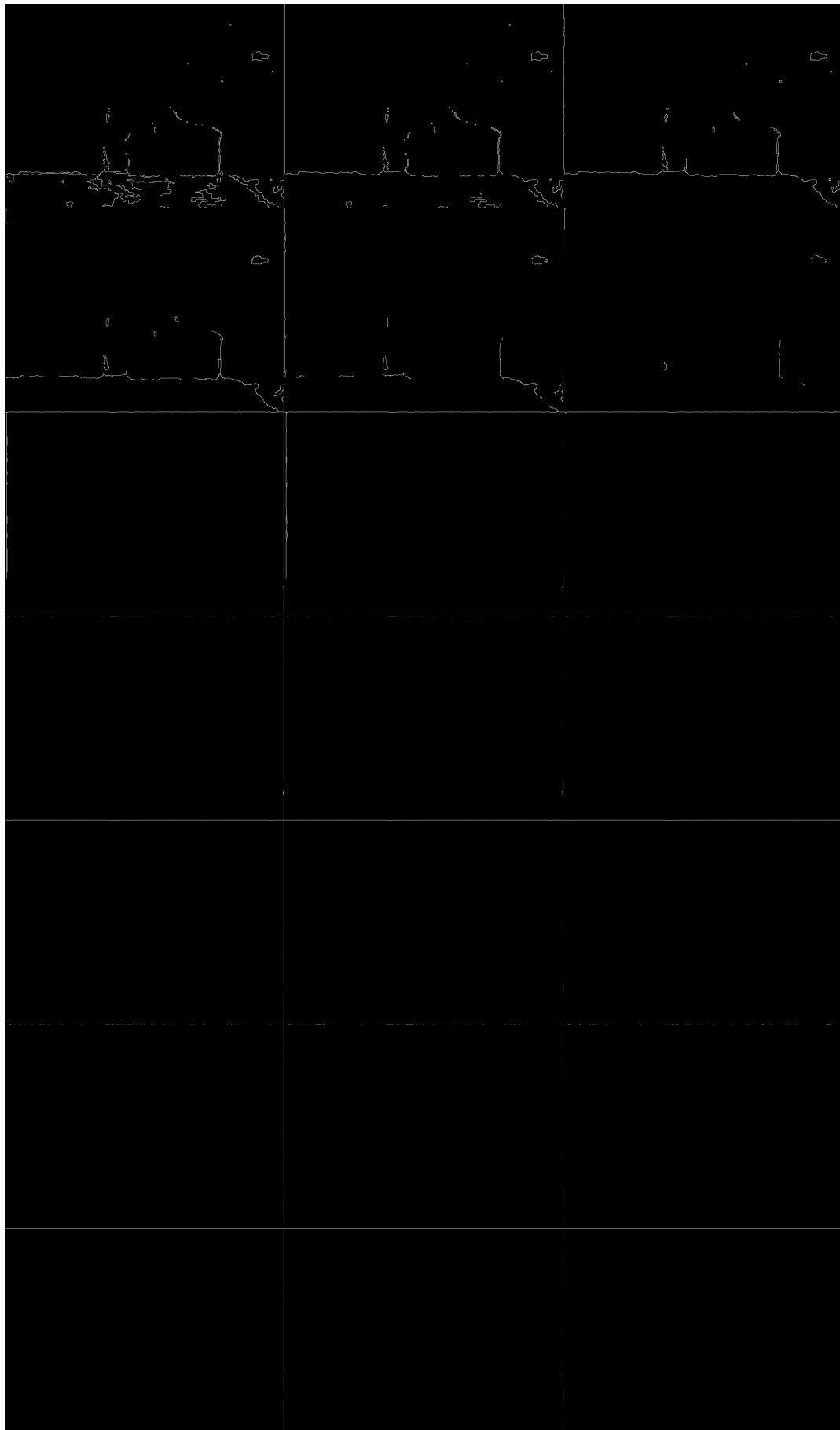


Figure 19: avg_09_canny_90_270

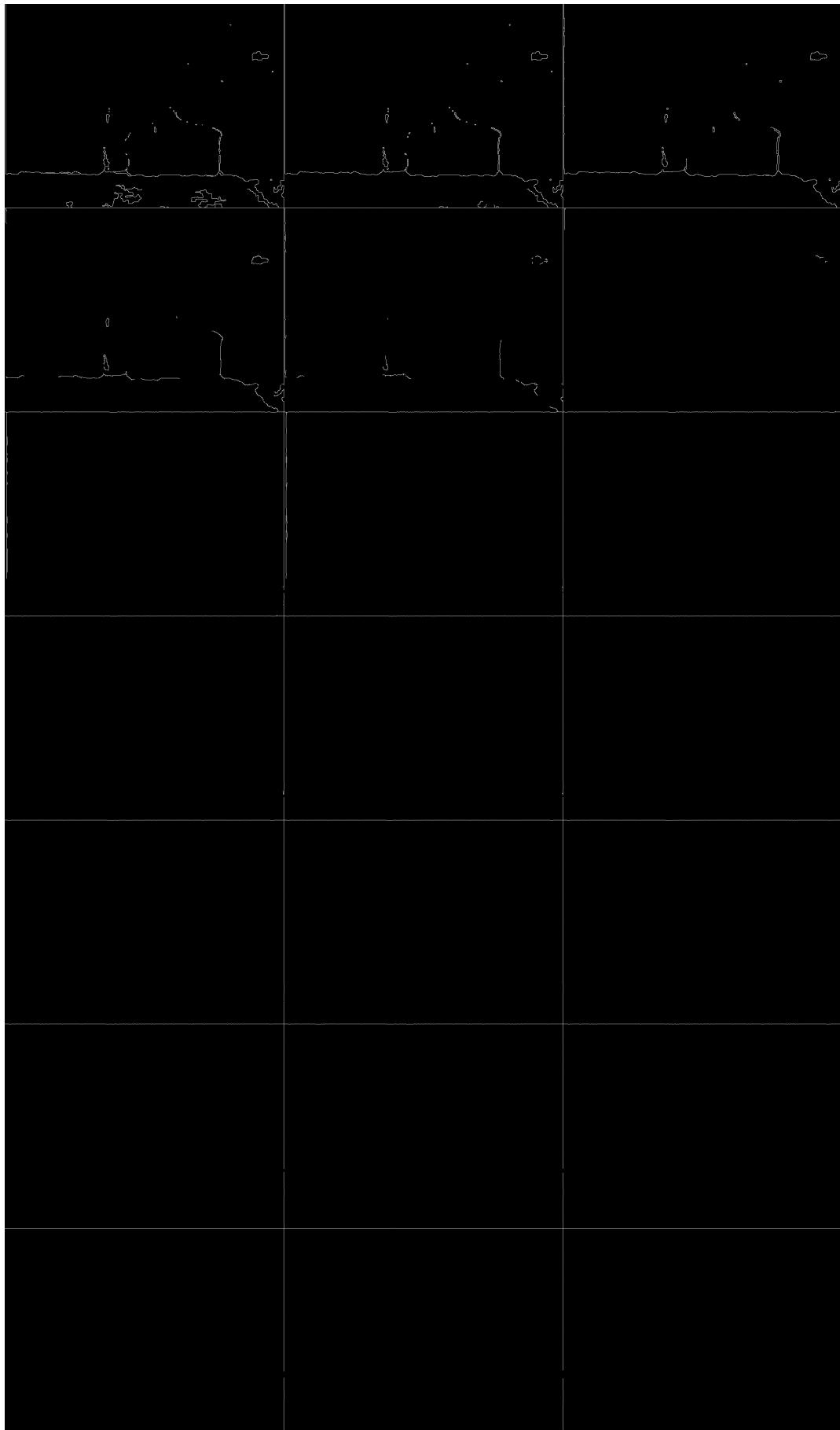


Figure 20: avg_10_canny_100_300



Figure 21: avg_11_canny_110_330



Figure 22: avg_12_canny_120_360



Figure 23: avg_13_canny_130_390



Figure 24: avg_14_canny_140_420

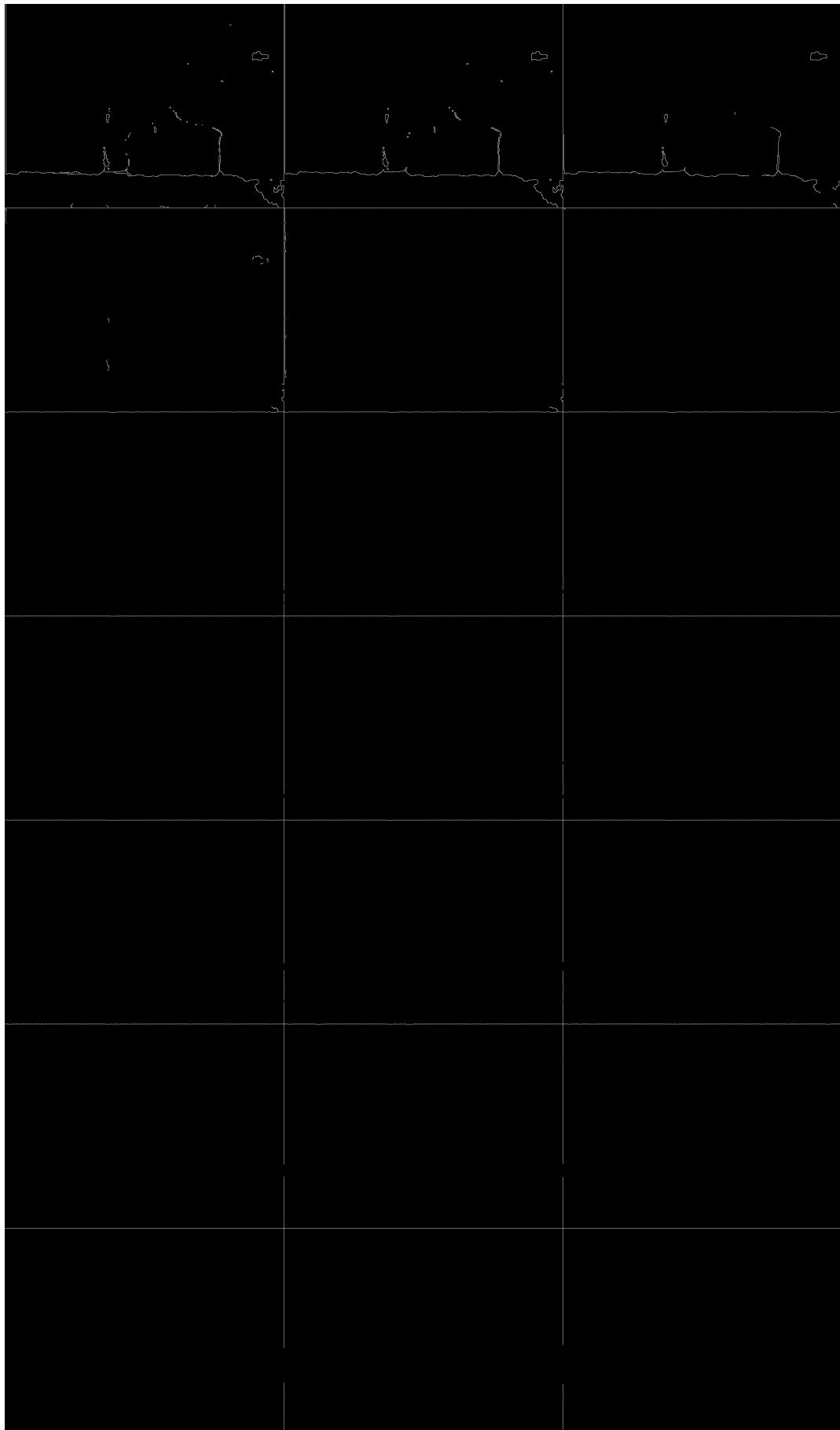


Figure 25: avg_15_canny_150_450

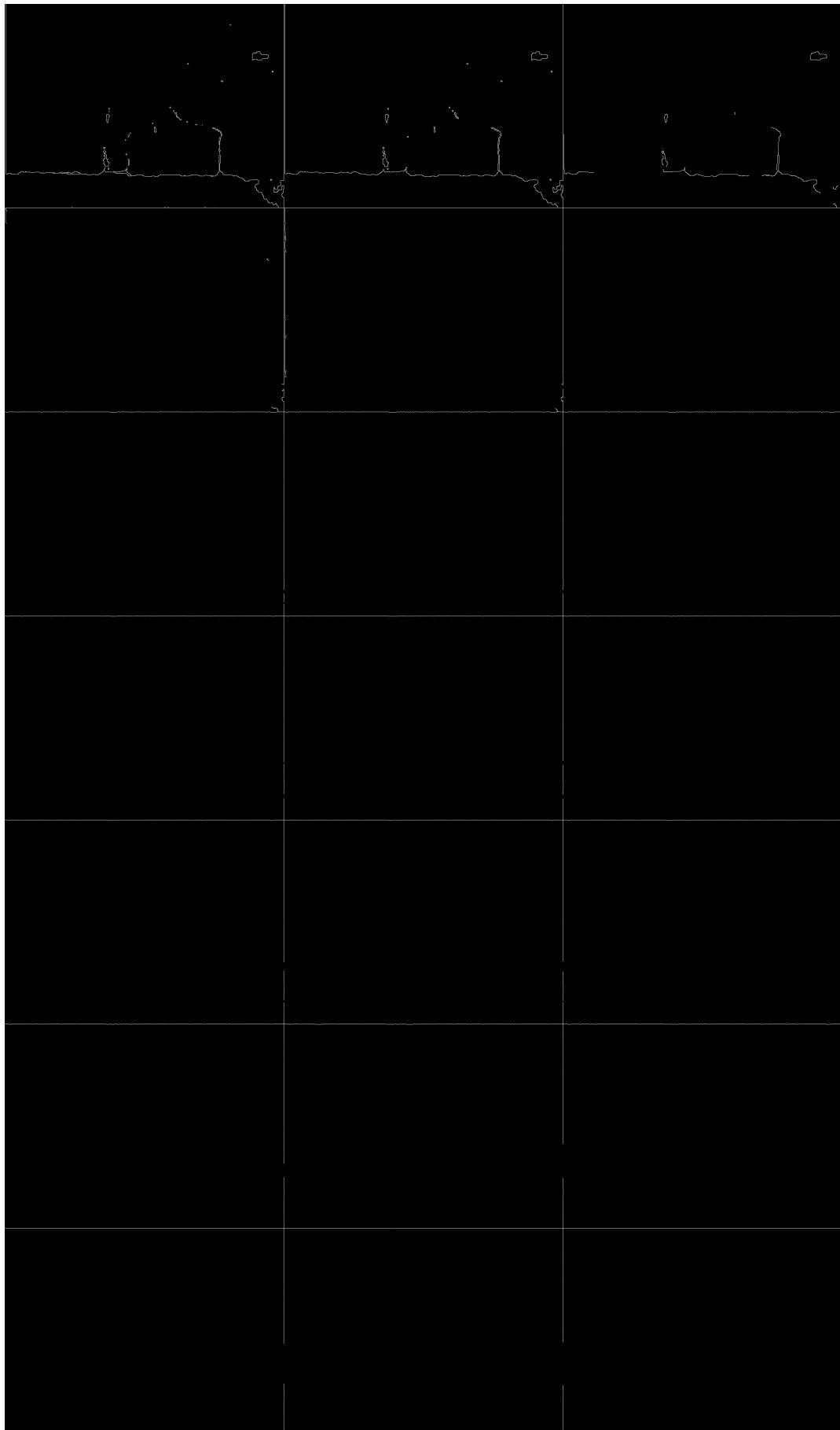


Figure 26: avg_16_canny_160_480

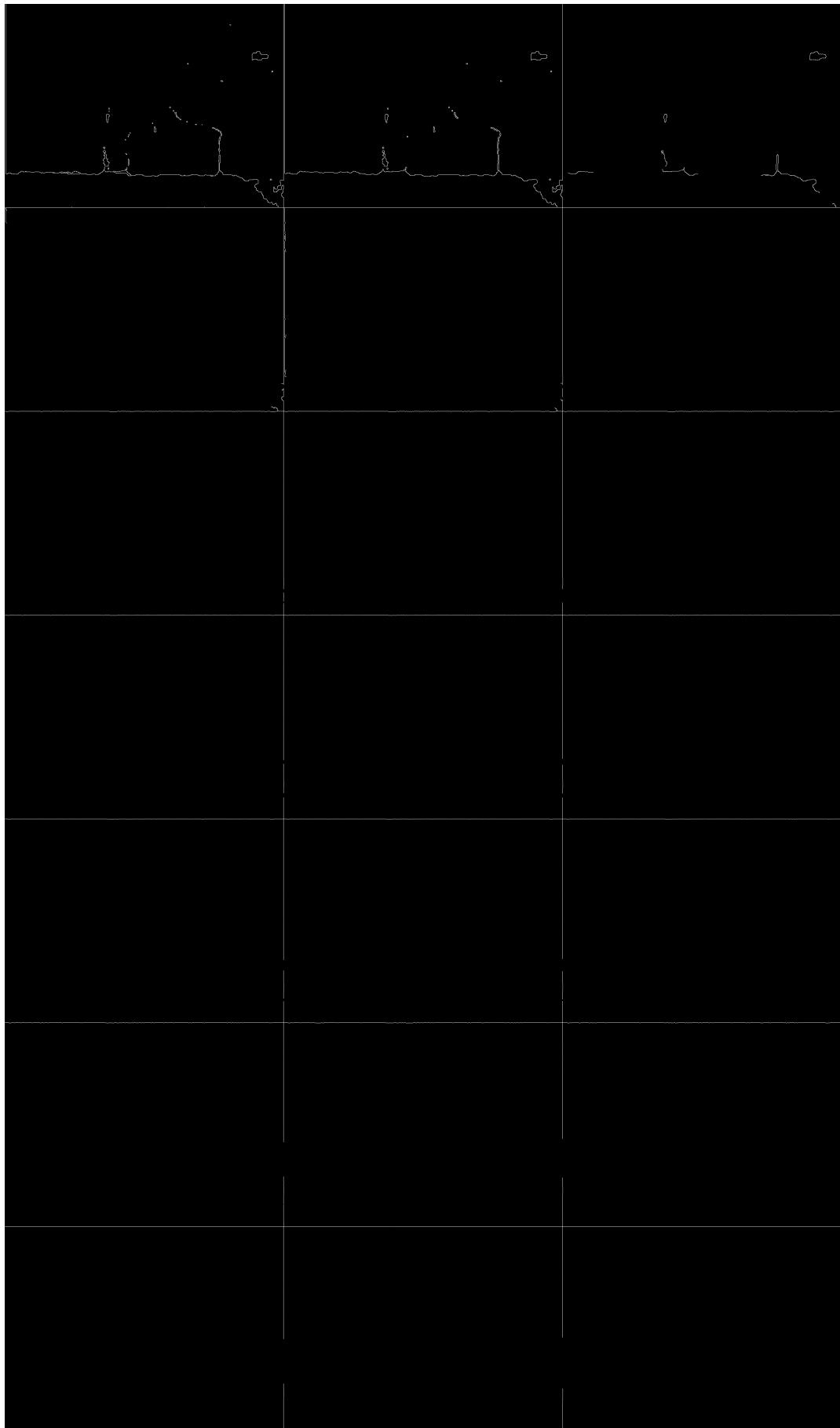


Figure 27: avg_17_canny_170_510

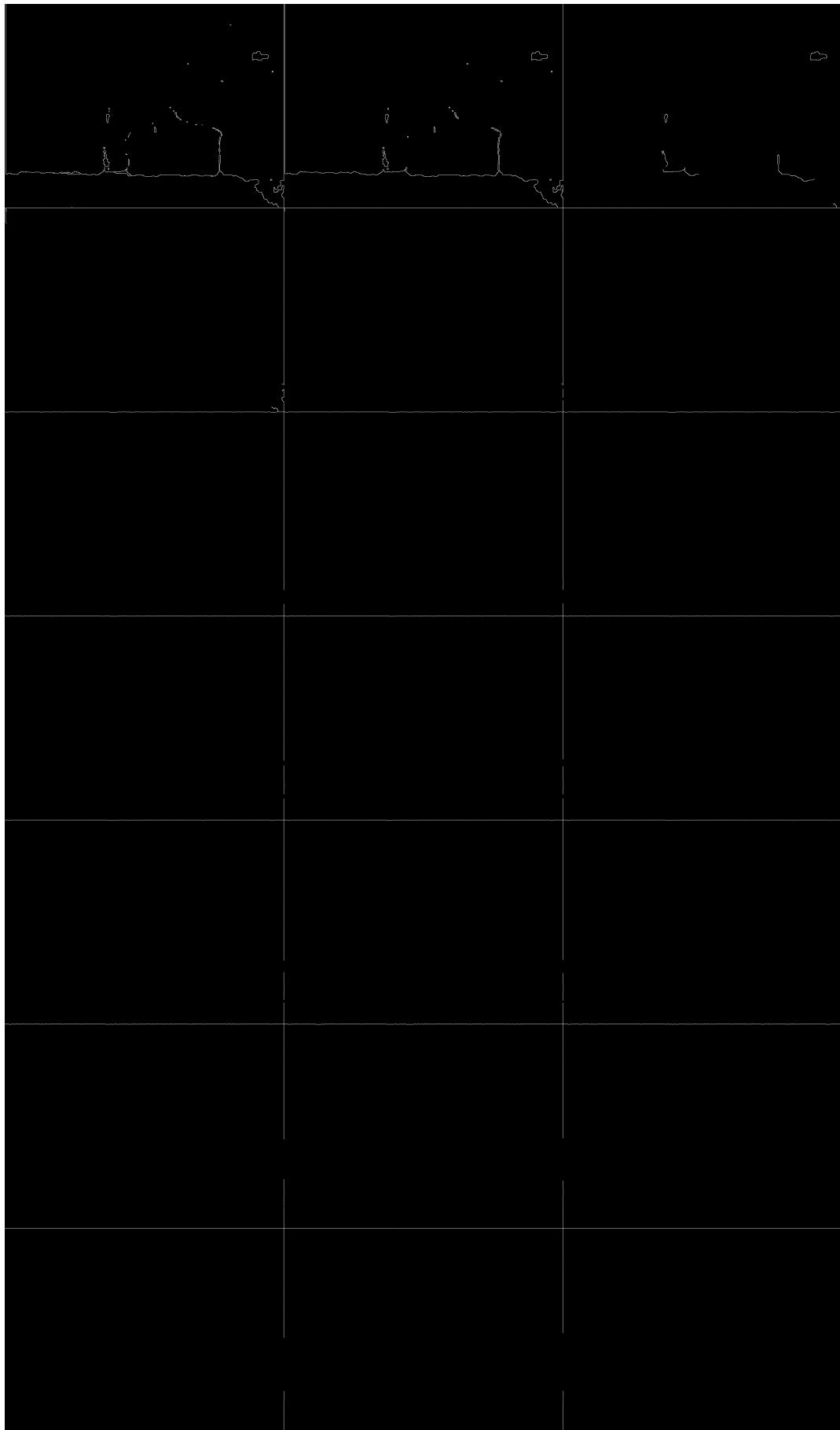


Figure 28: avg_18_canny_180_540

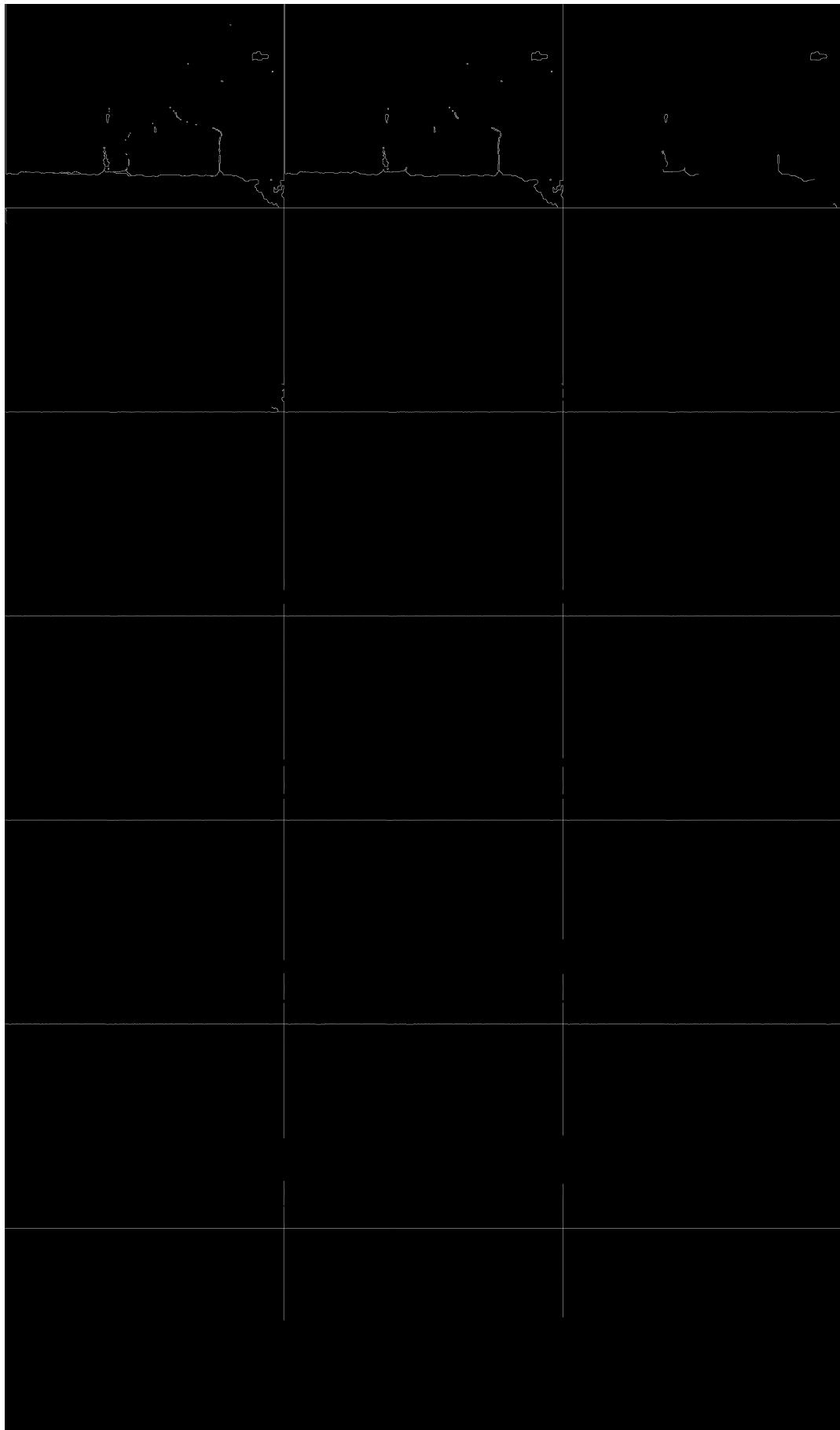


Figure 29: avg_19_canny_190_570

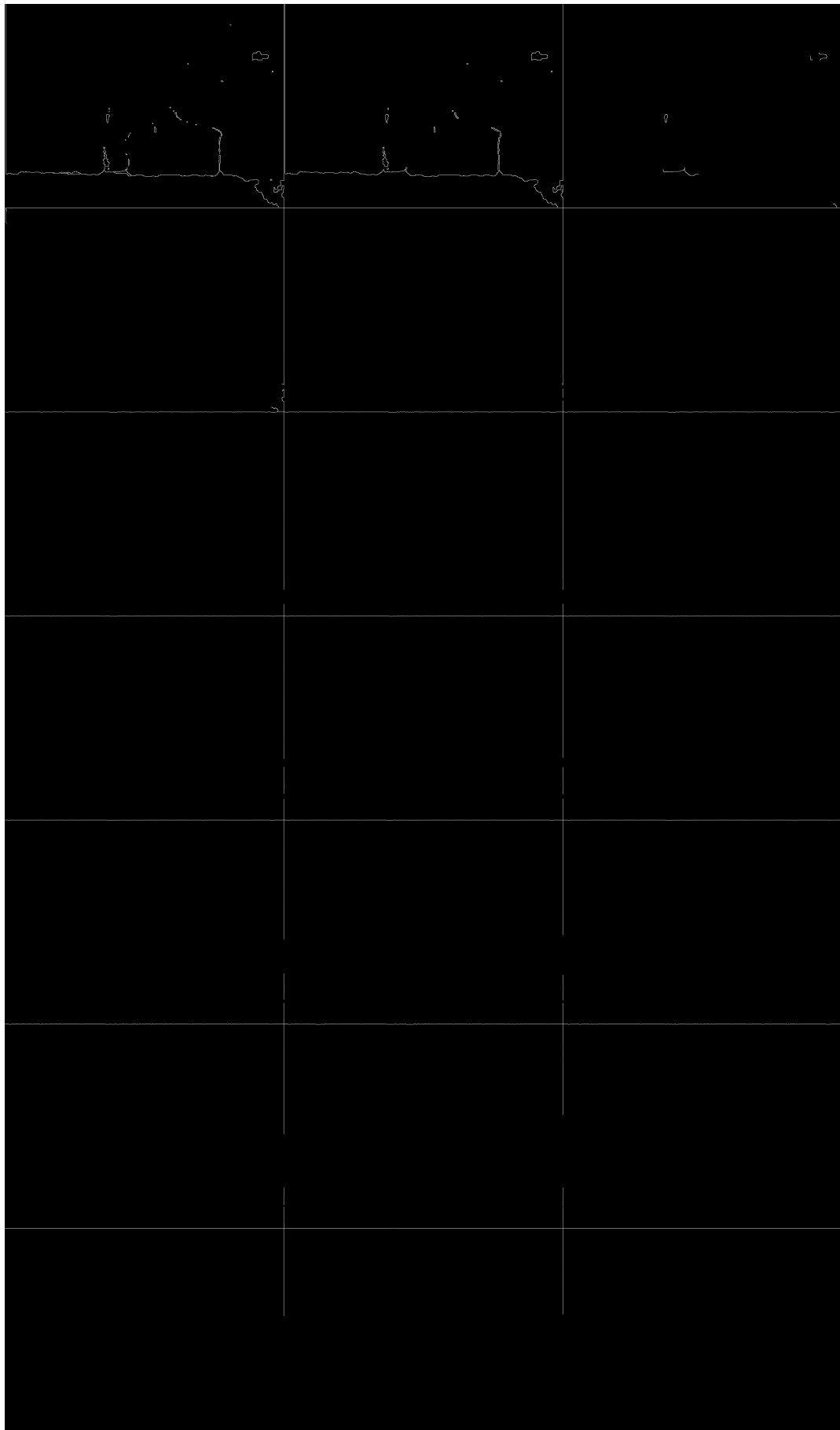


Figure 30: avg_20_canny_200_600

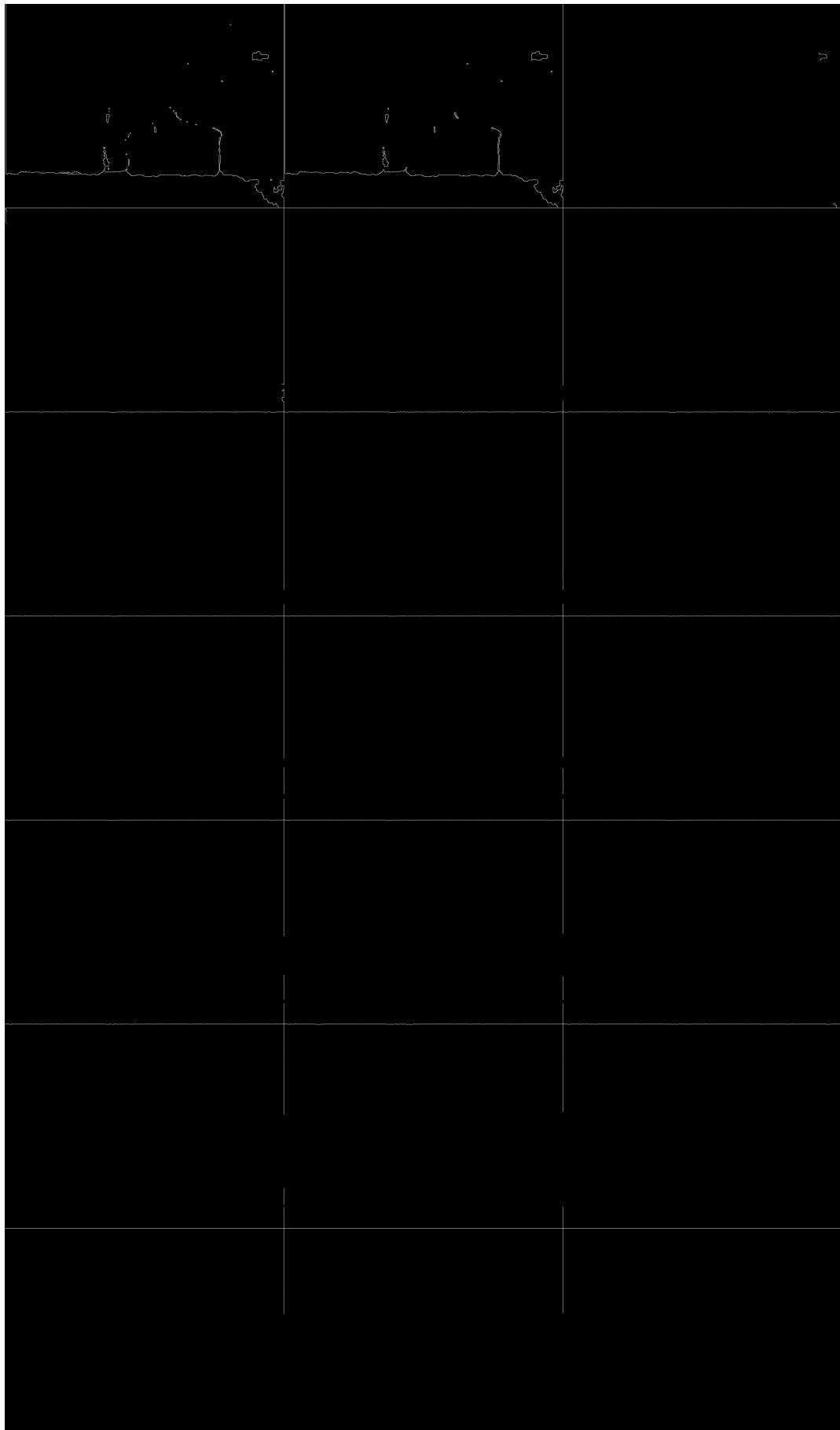


Figure 31: avg_21_canny_210_630

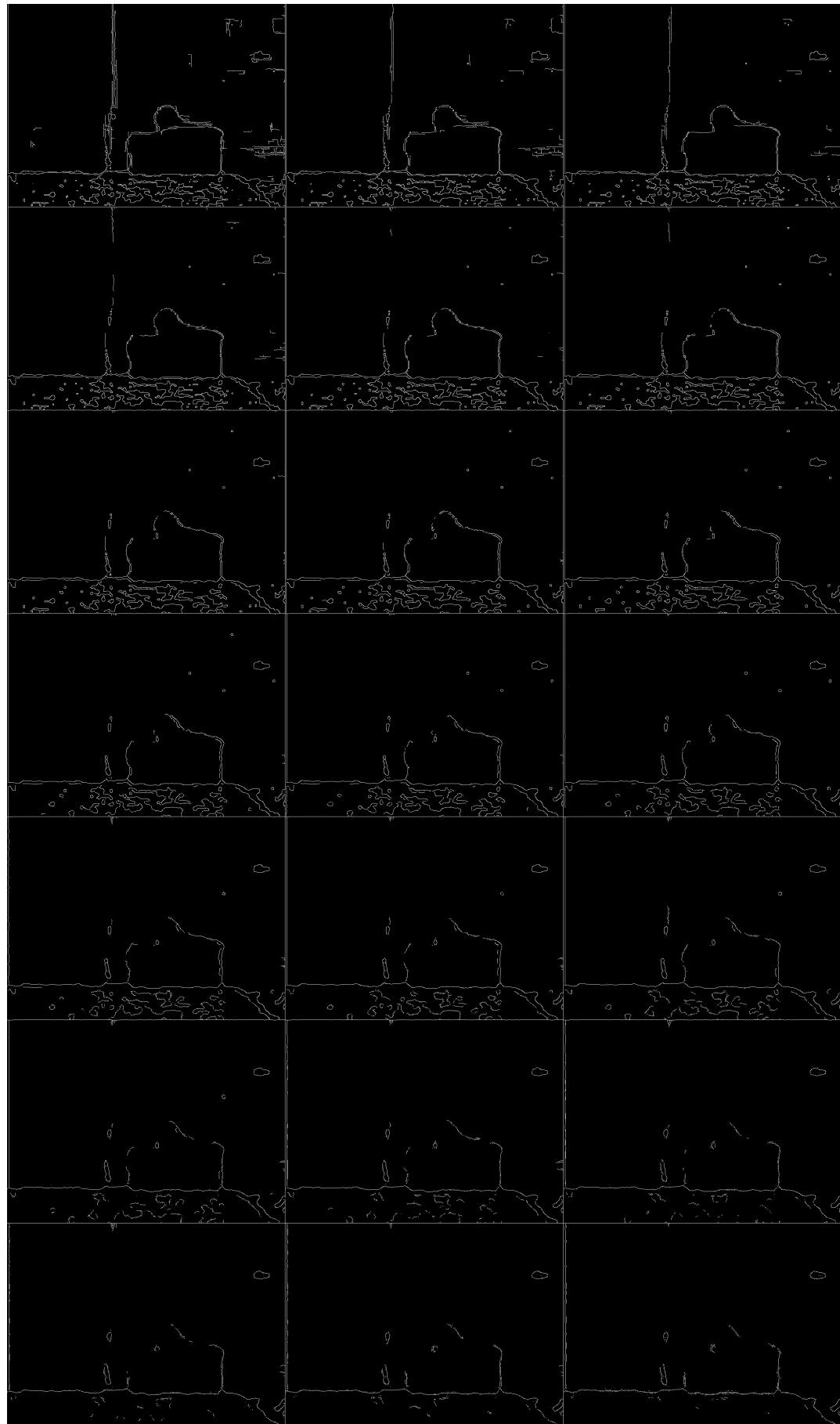


Figure 32: gauss_01_canny_10_30

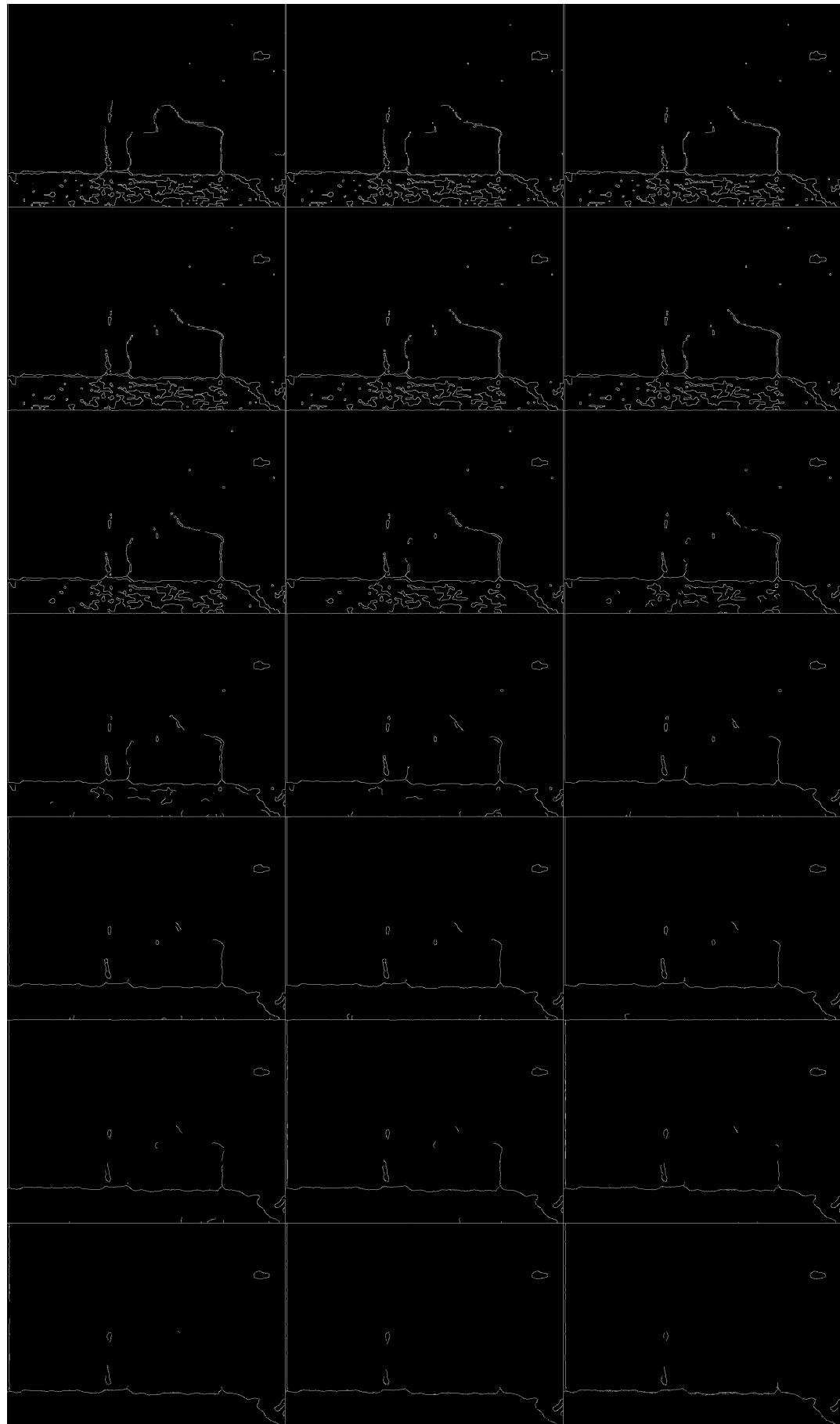


Figure 33: gauss_02_canny_20_60

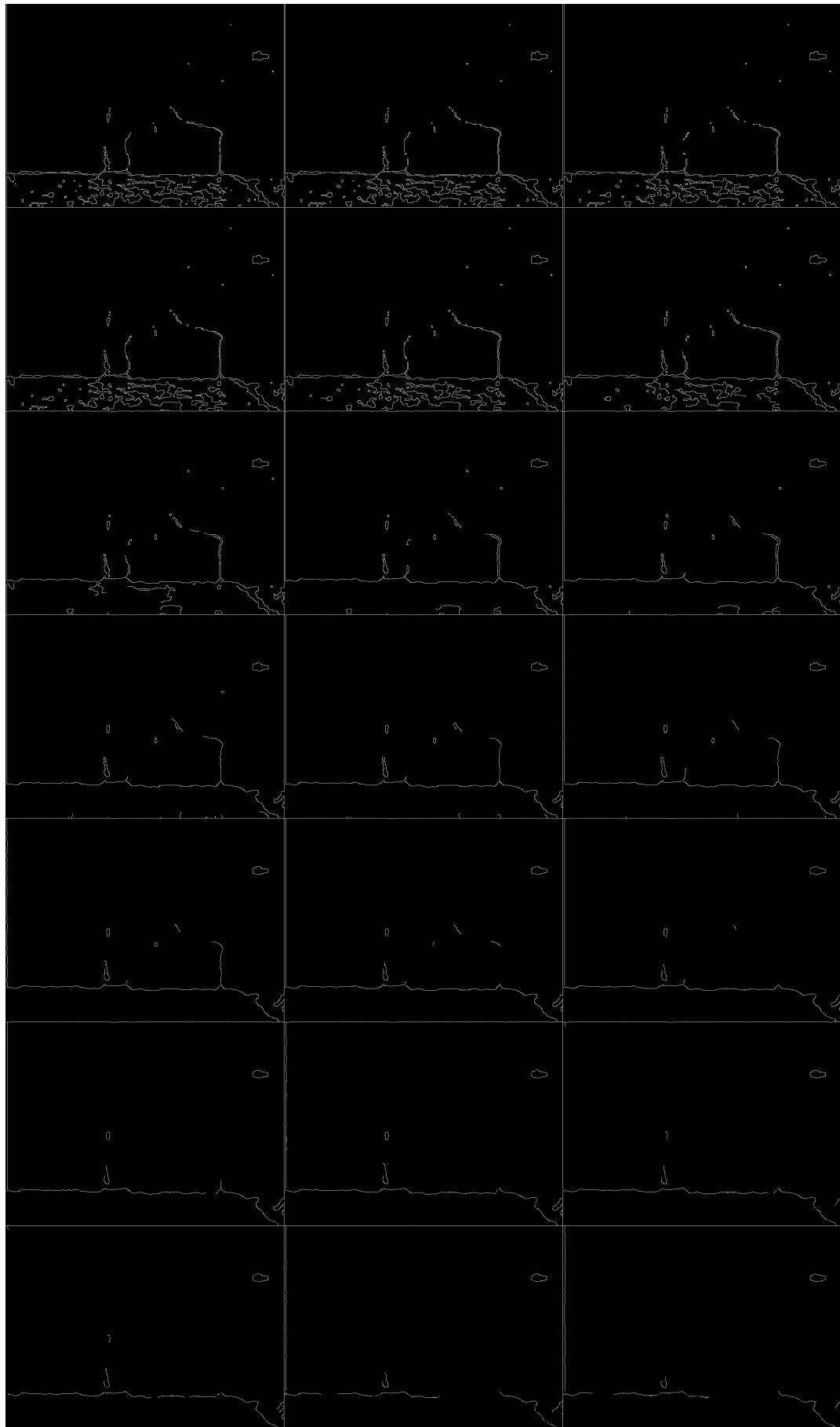


Figure 34: gauss_03_canny_30_90

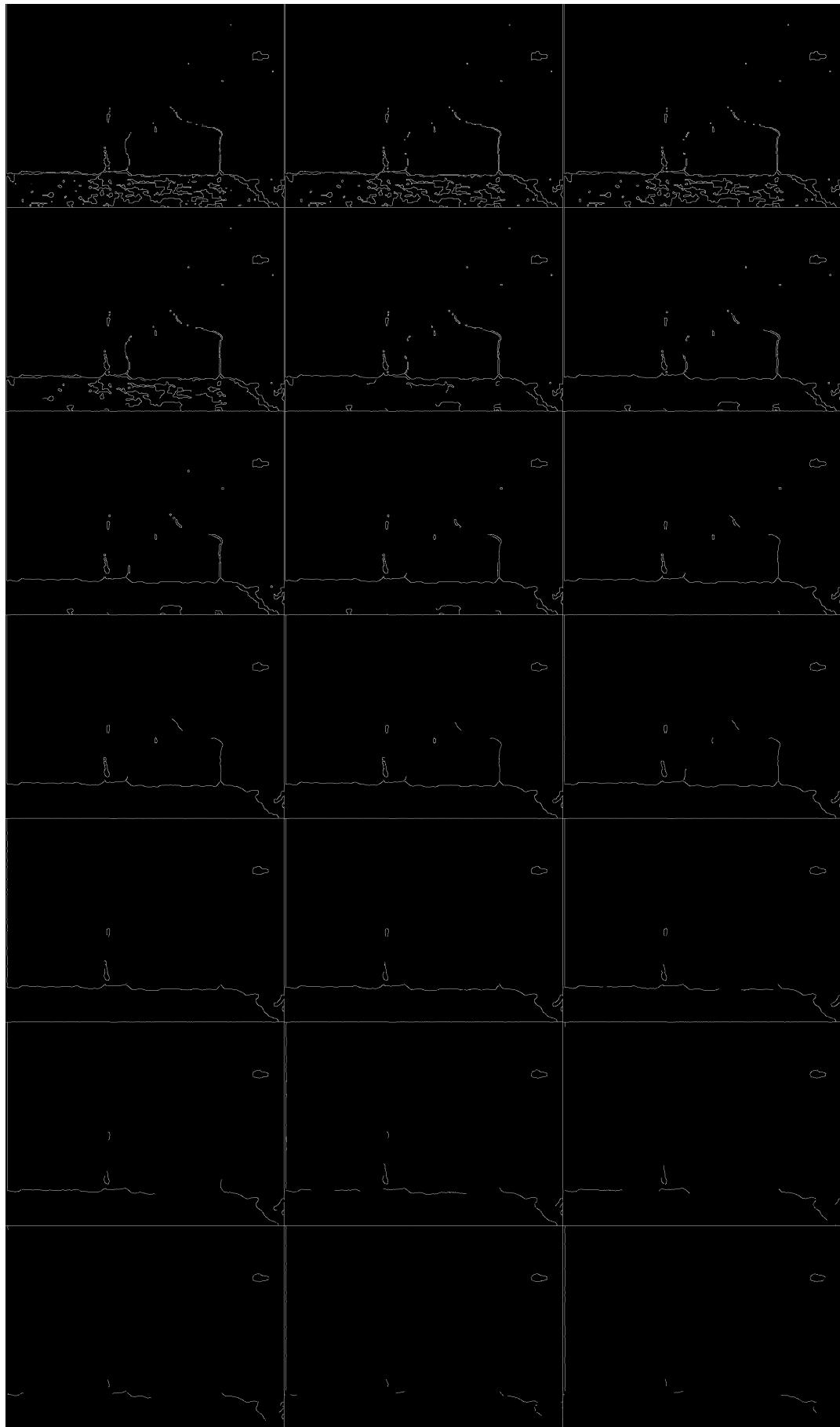


Figure 35: gauss_04_canny_40_120

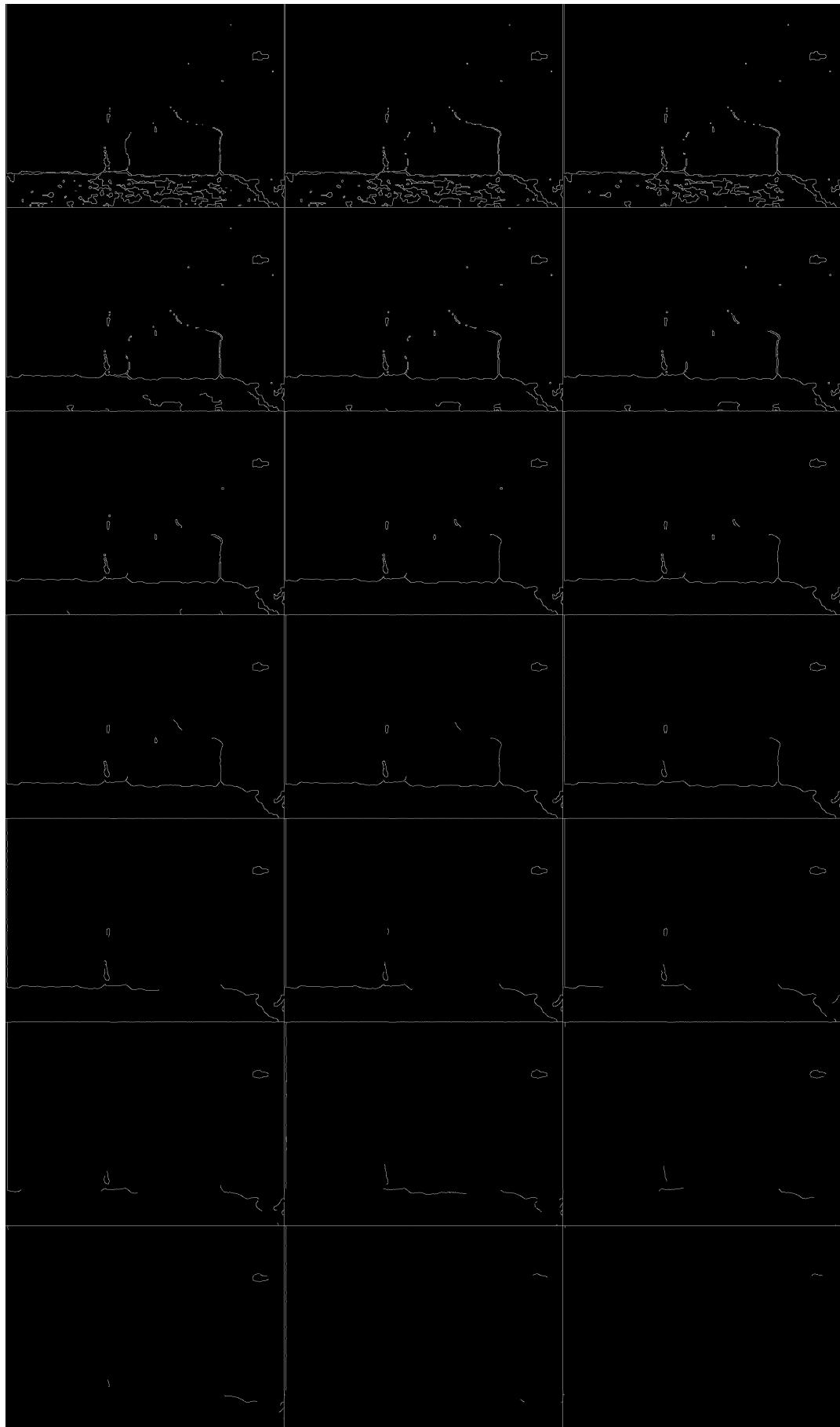


Figure 36: gauss_05_canny_50_150

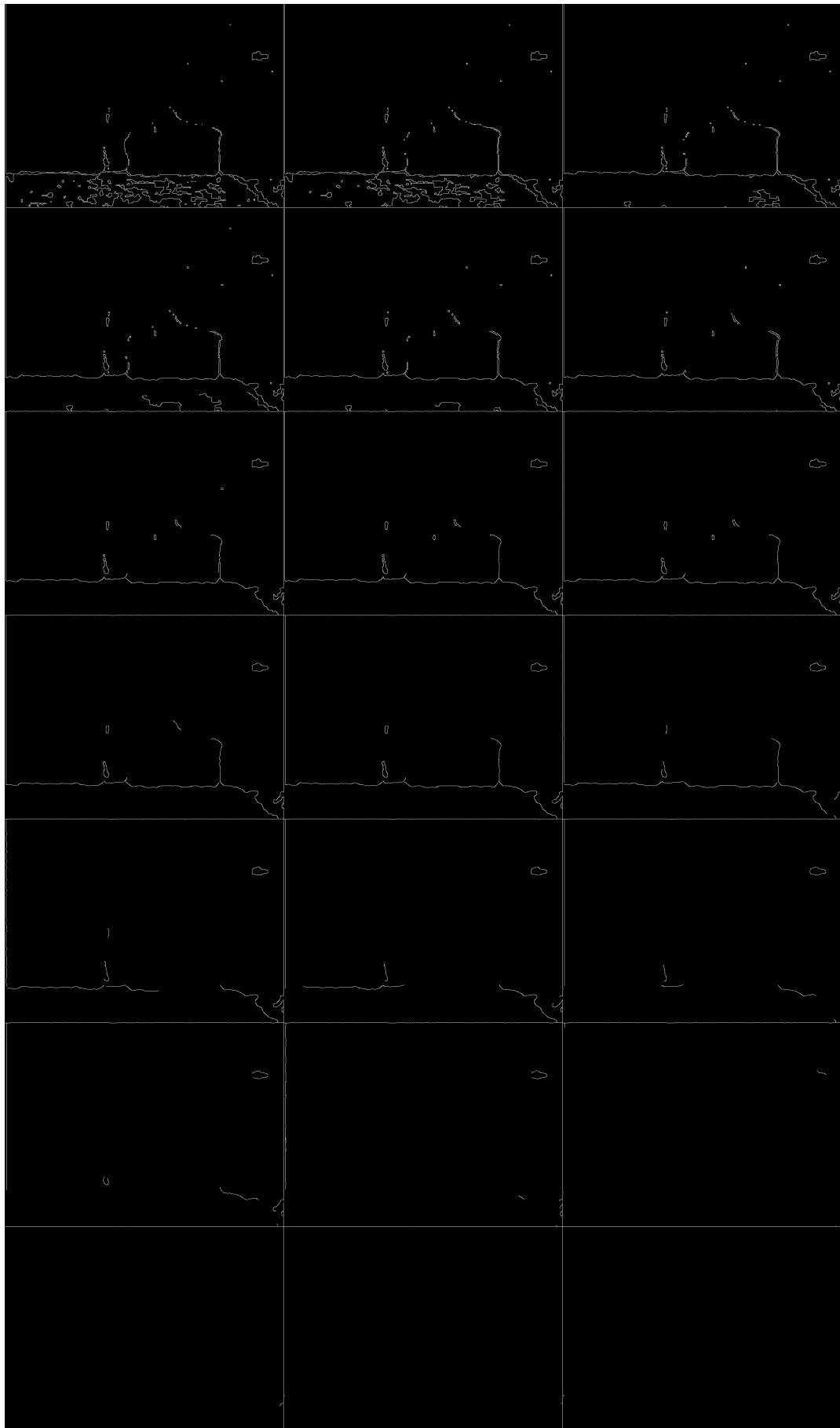


Figure 37: gauss_06_canny_60_180

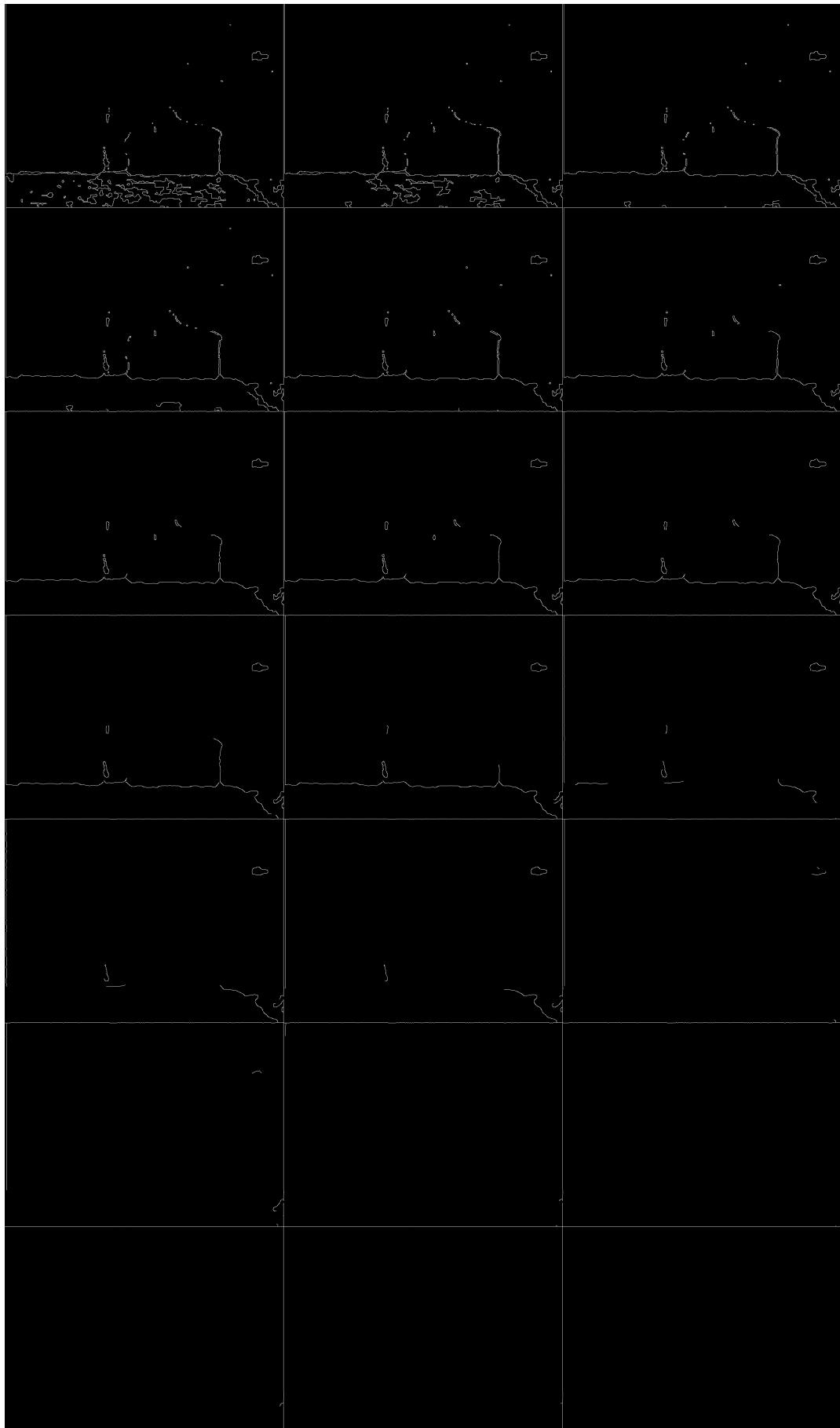


Figure 38: gauss_07_canny_70_210

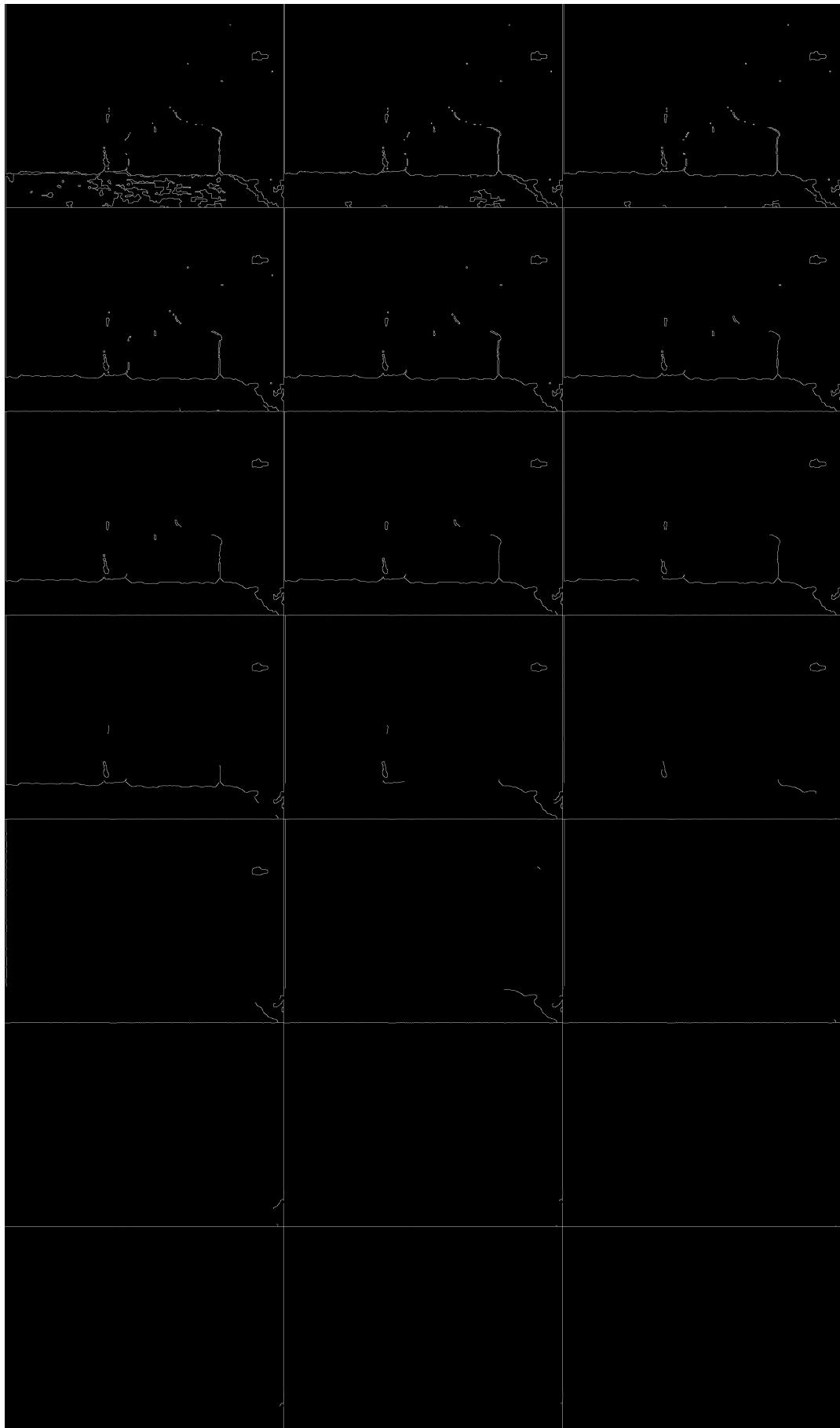


Figure 39: gauss_08_canny_80_240

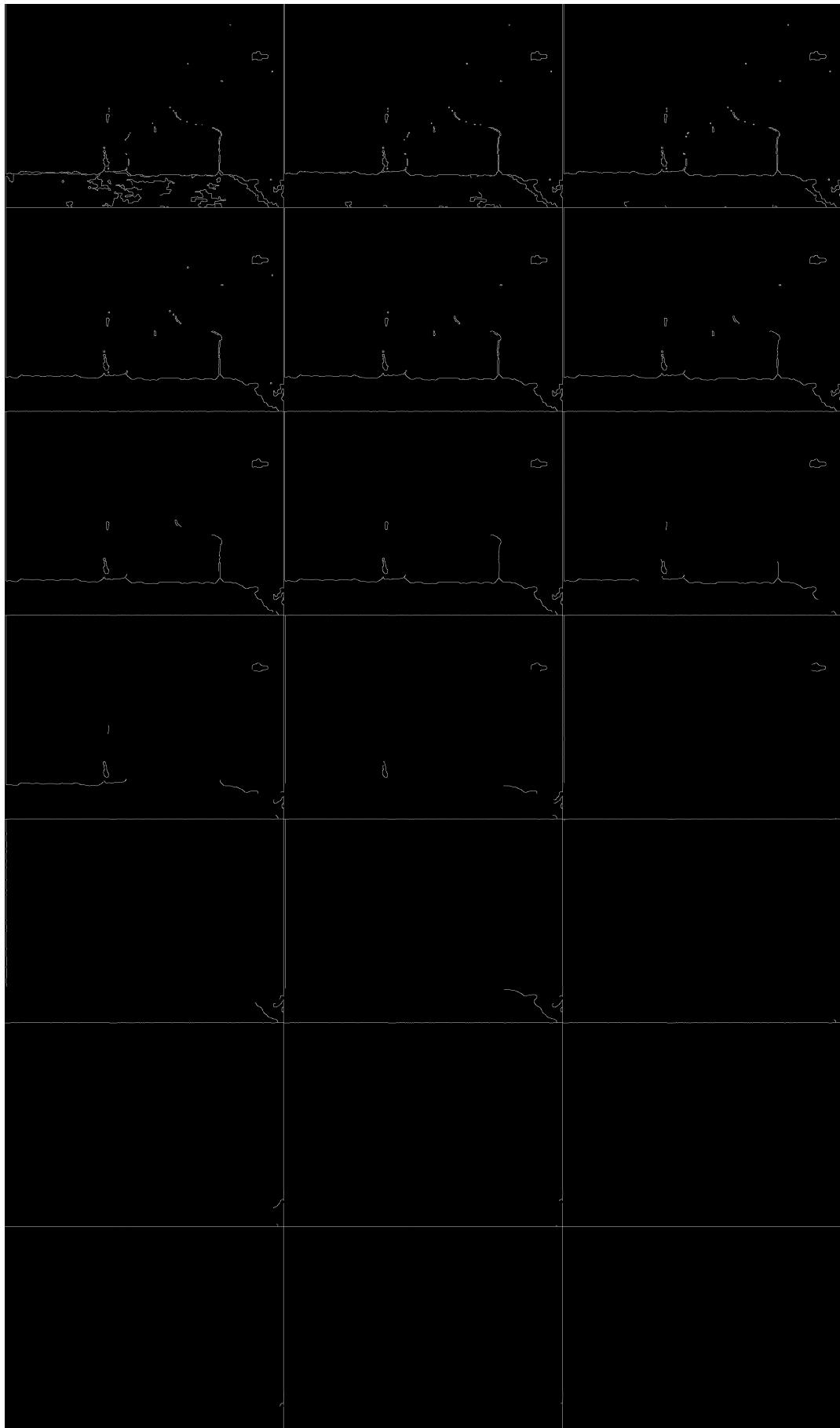


Figure 40: gauss_09_canny_90_270

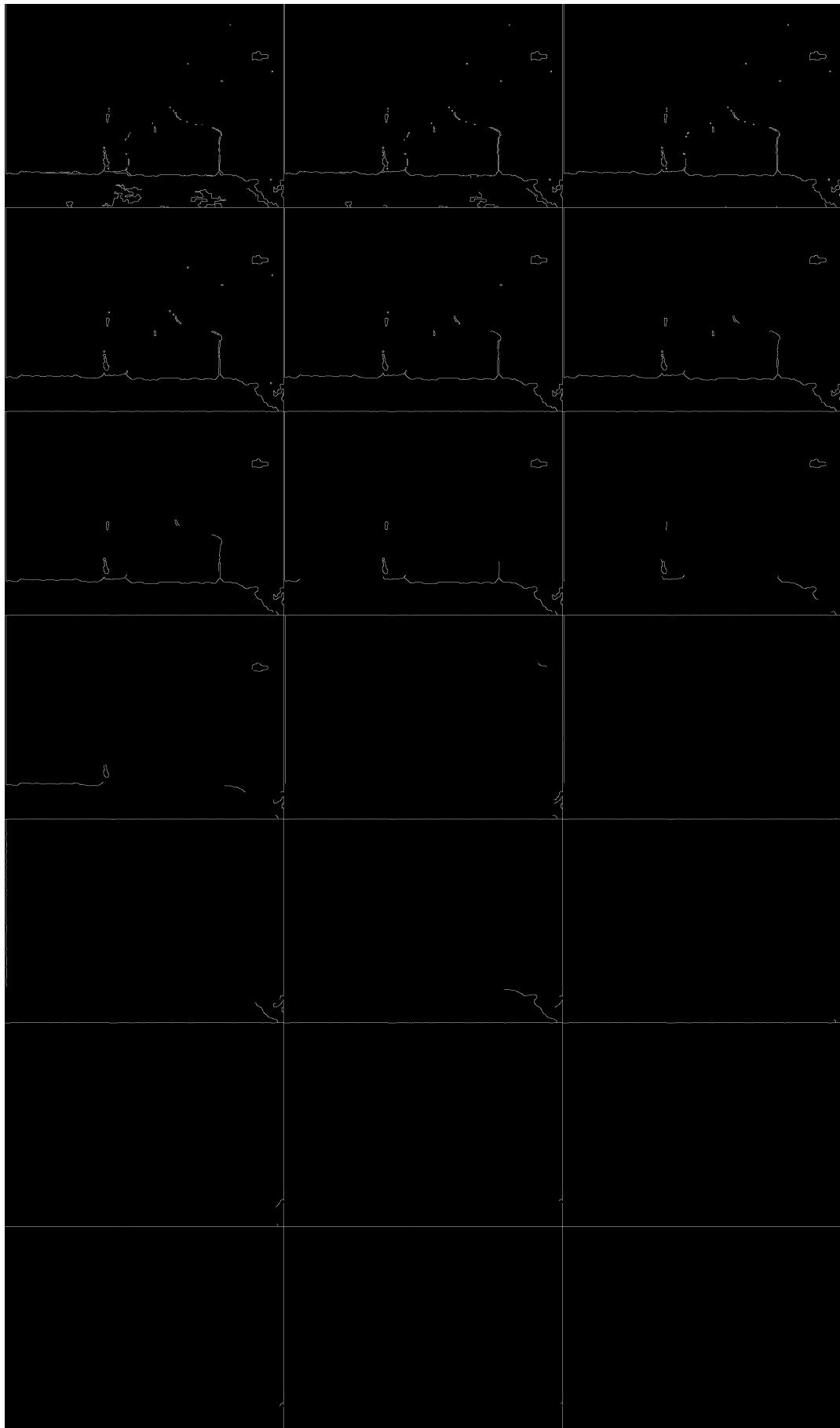


Figure 41: gauss_10_canny_100_300

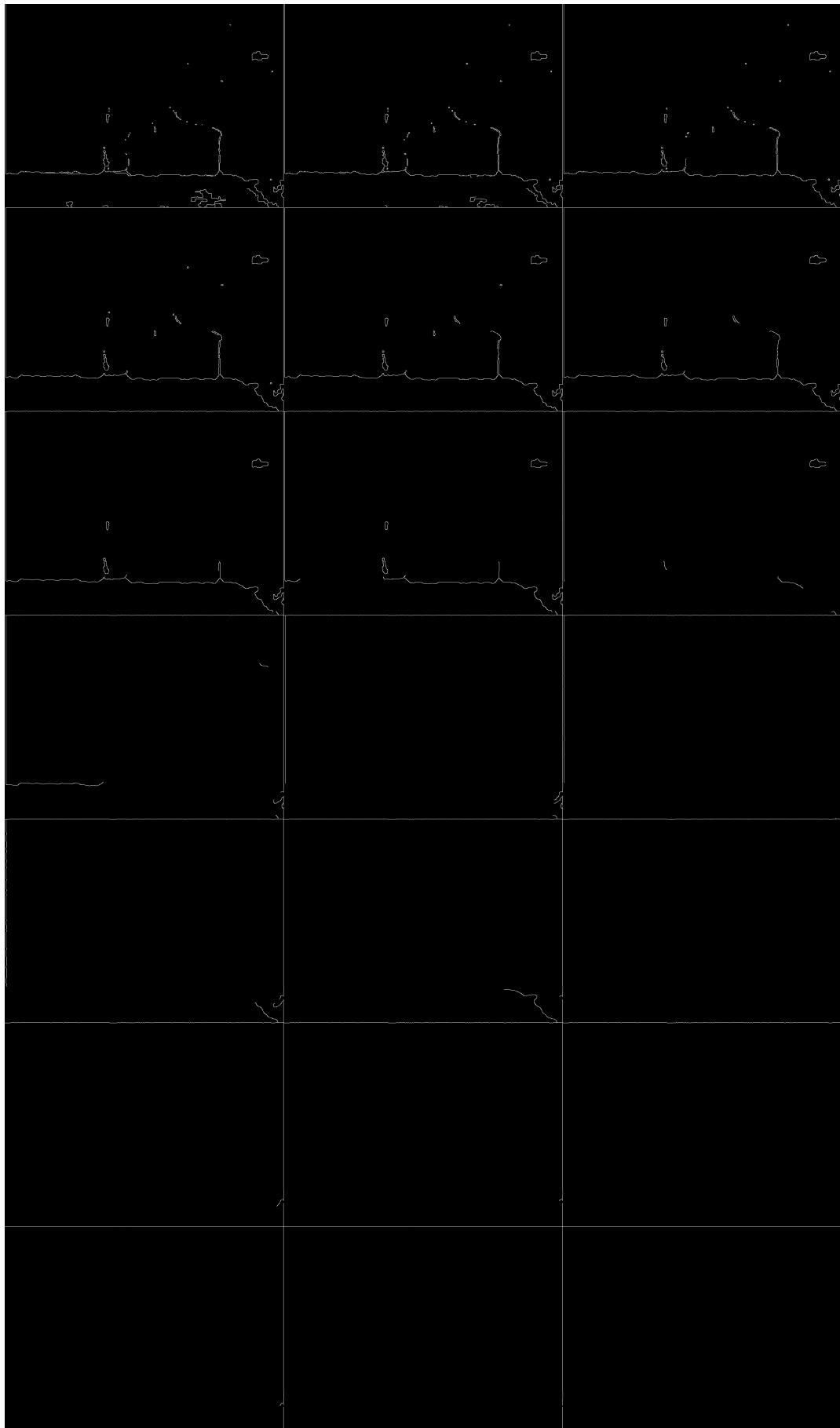


Figure 42: gauss_11_canny_110_330

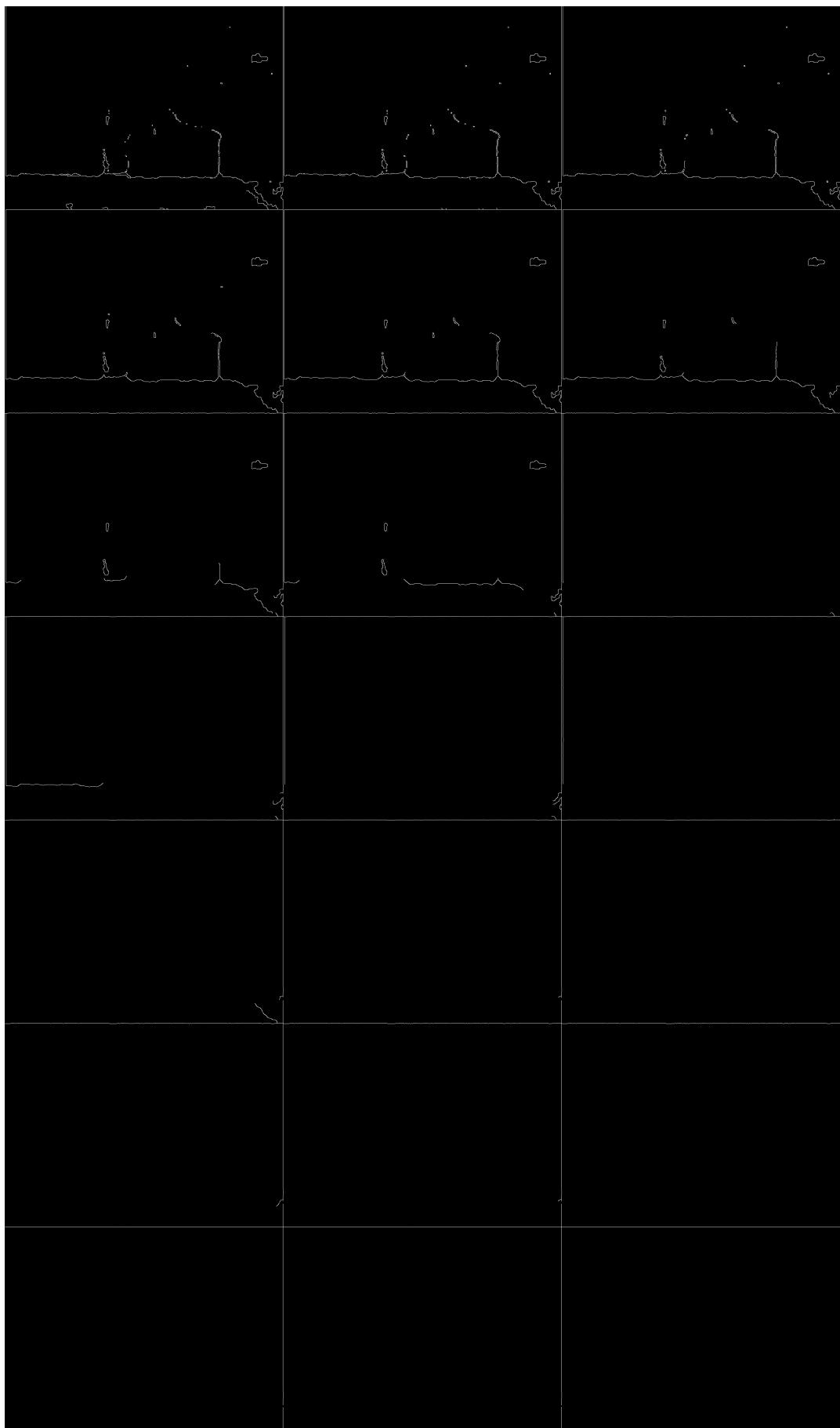


Figure 43: gauss_12_canny_120_360



Figure 44: gauss_13_canny_130_390

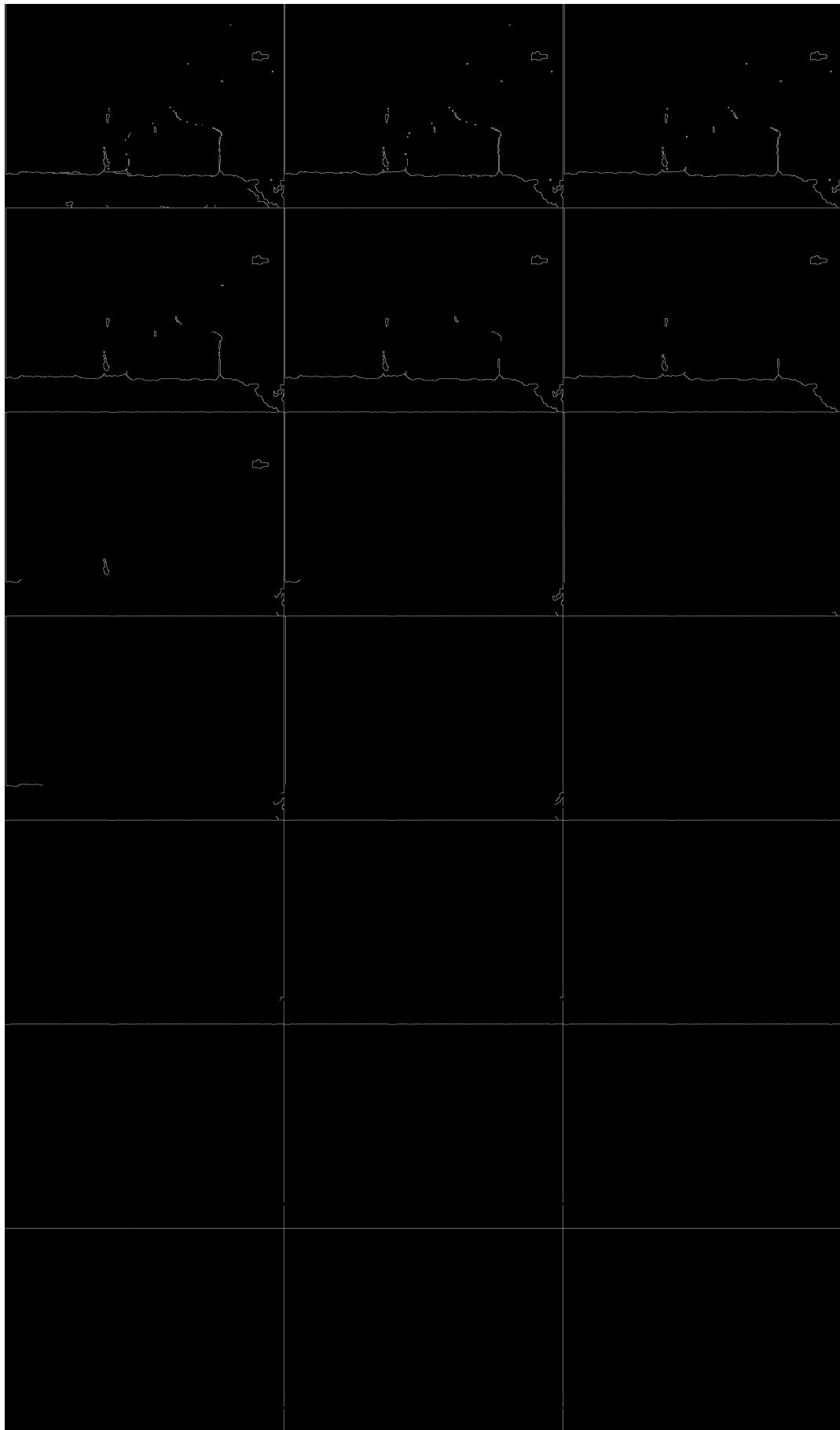


Figure 45: gauss_14_canny_140_420

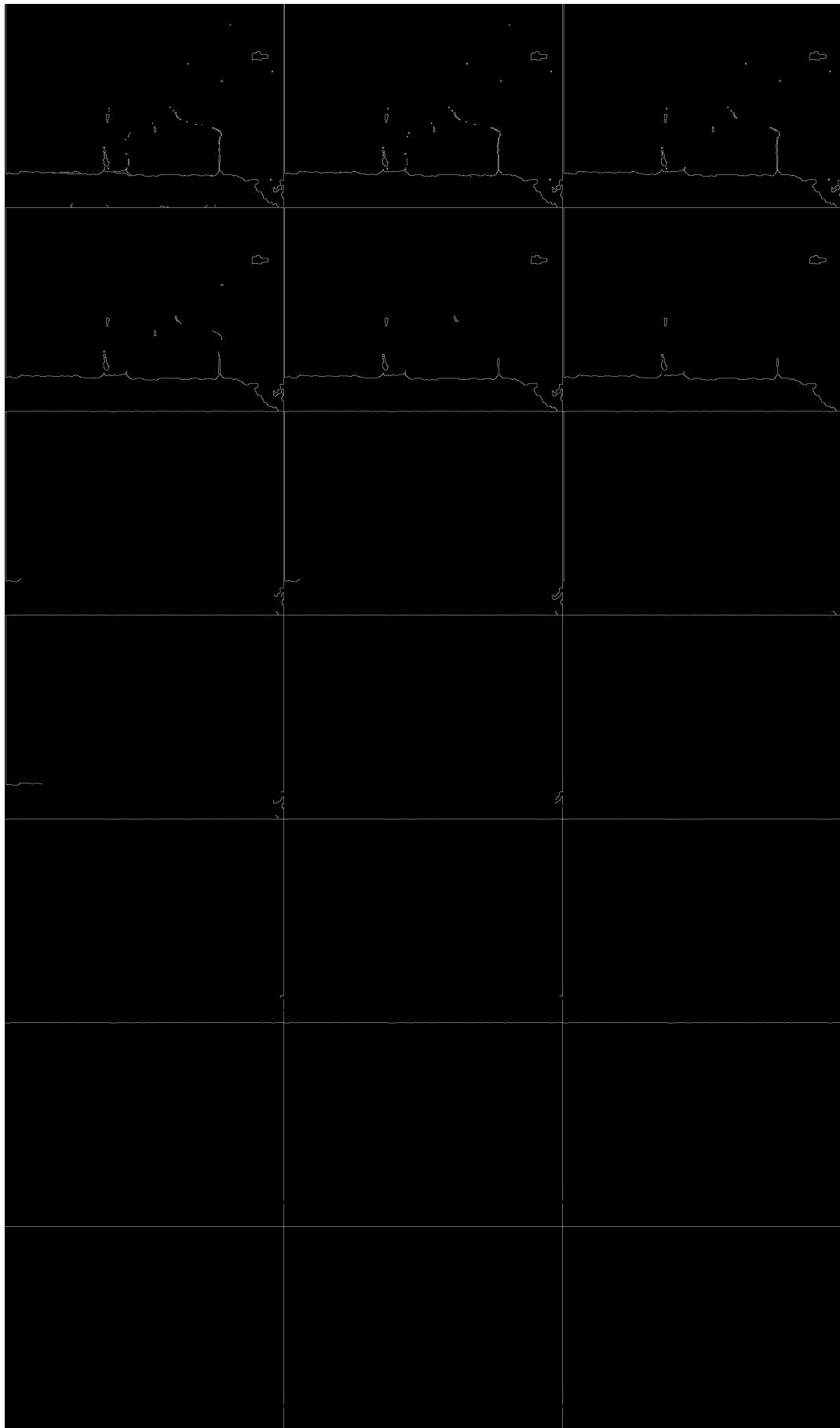


Figure 46: gauss_15_canny_150_450

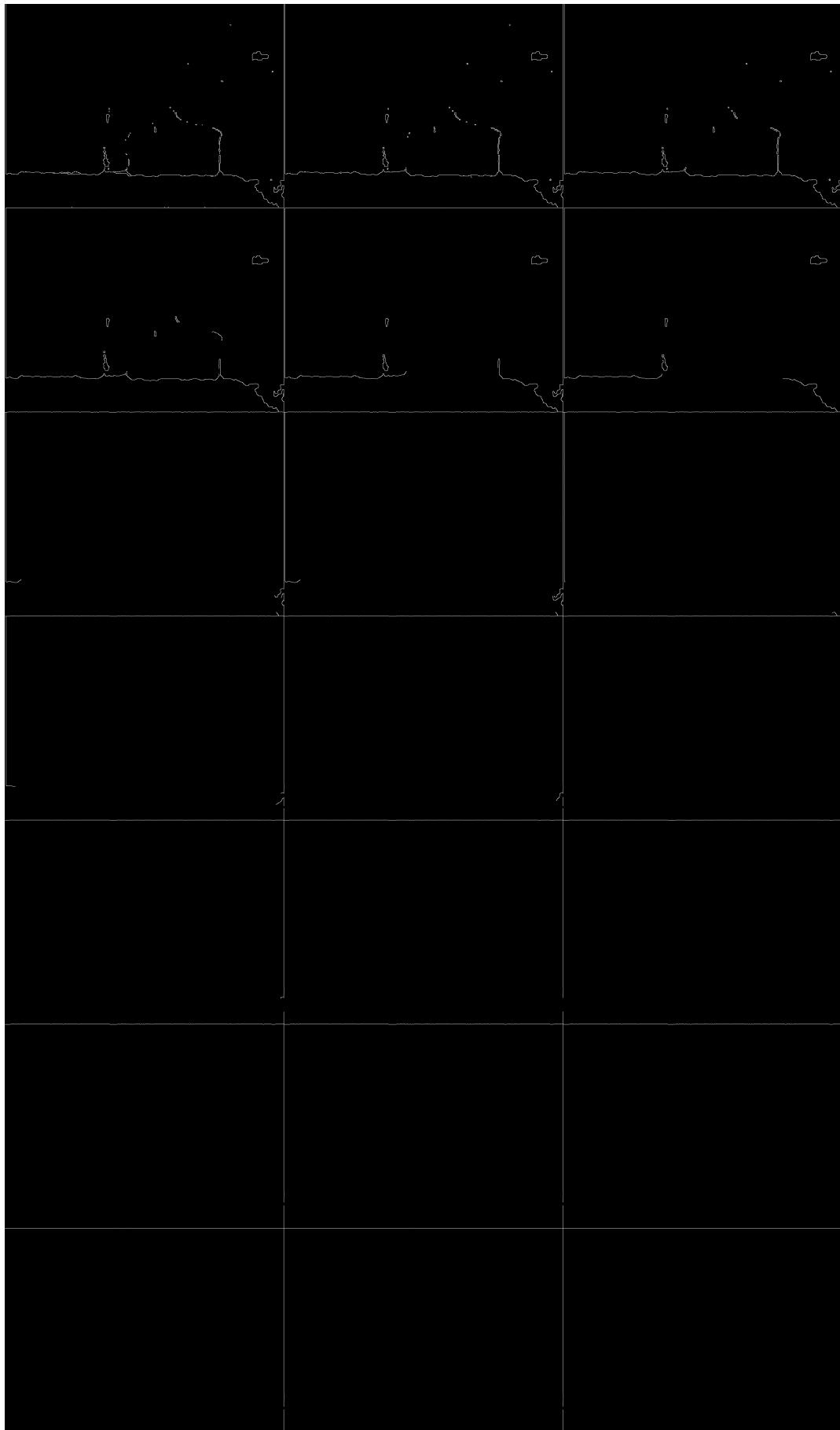


Figure 47: gauss_16_canny_160_480

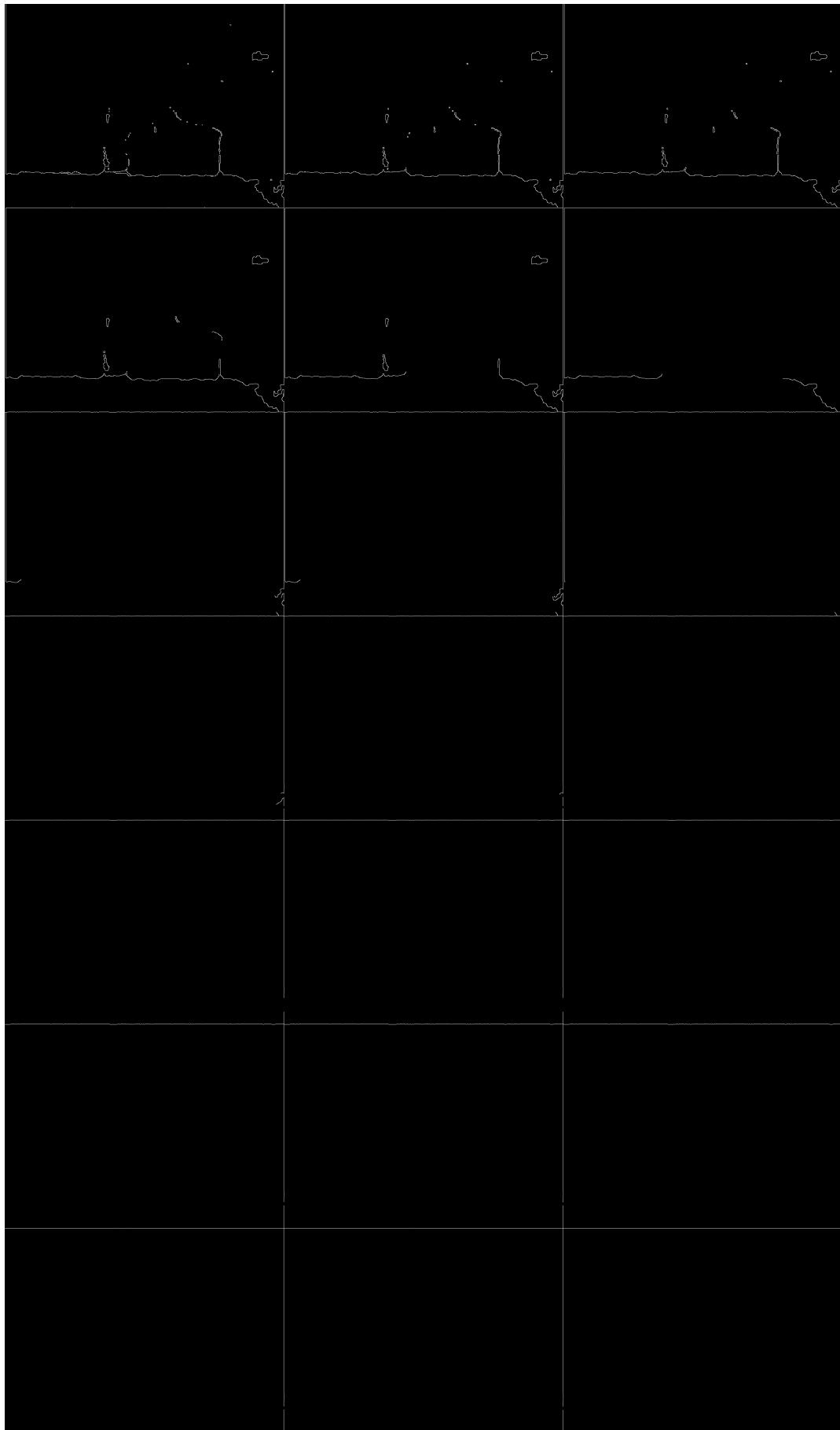


Figure 48: gauss_17_canny_170_510

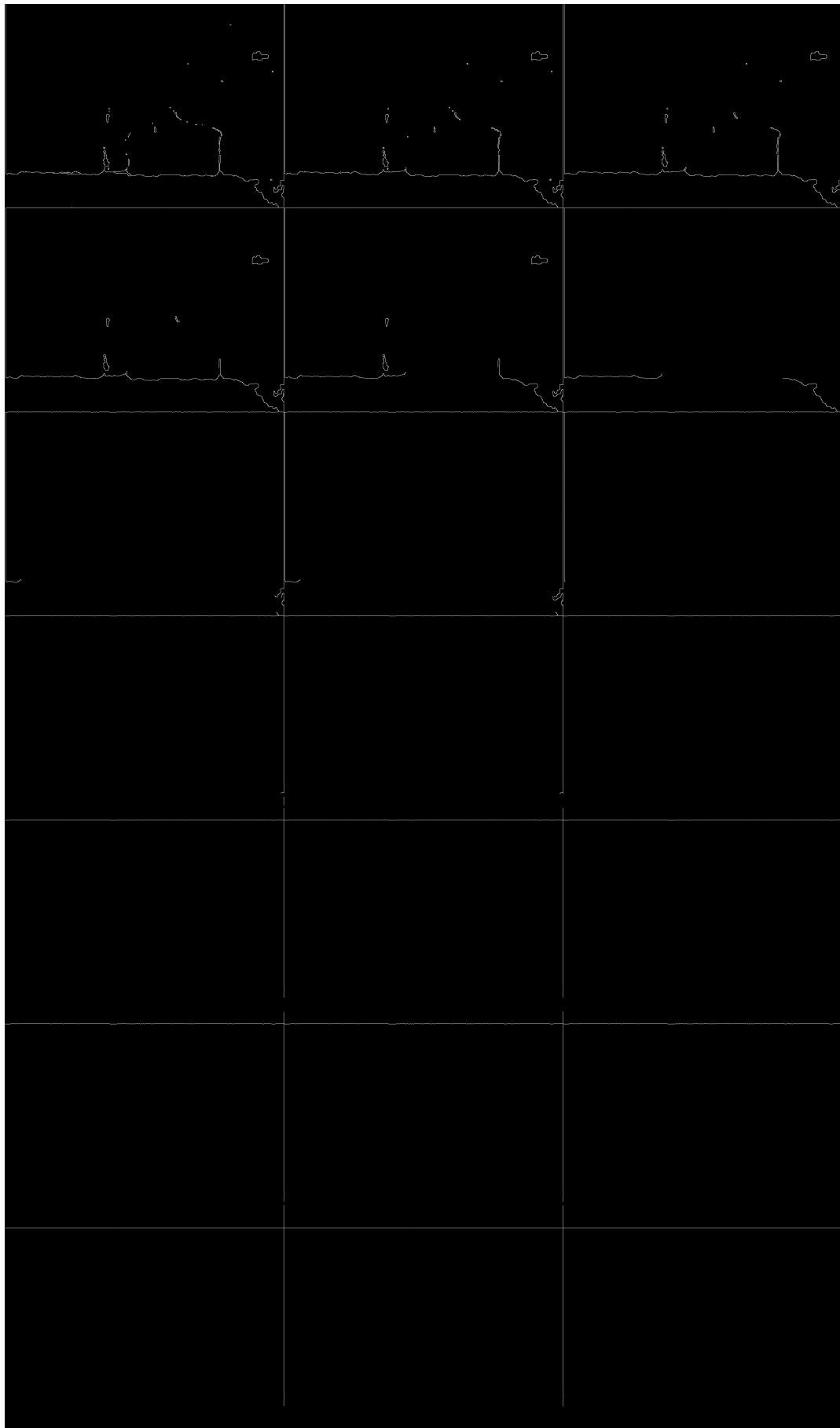


Figure 49: gauss_18_canny_180_540

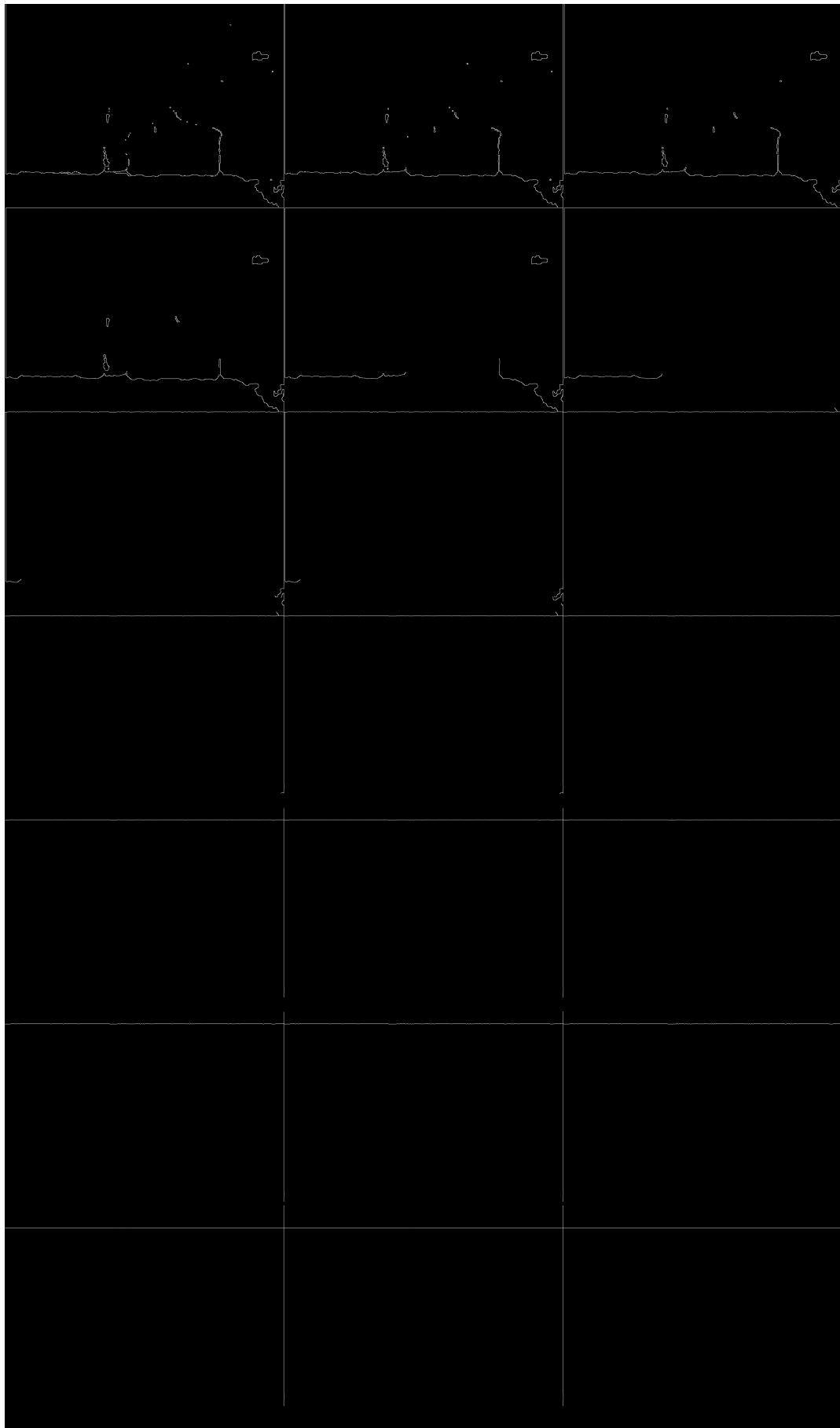


Figure 50: gauss_19_canny_190_570

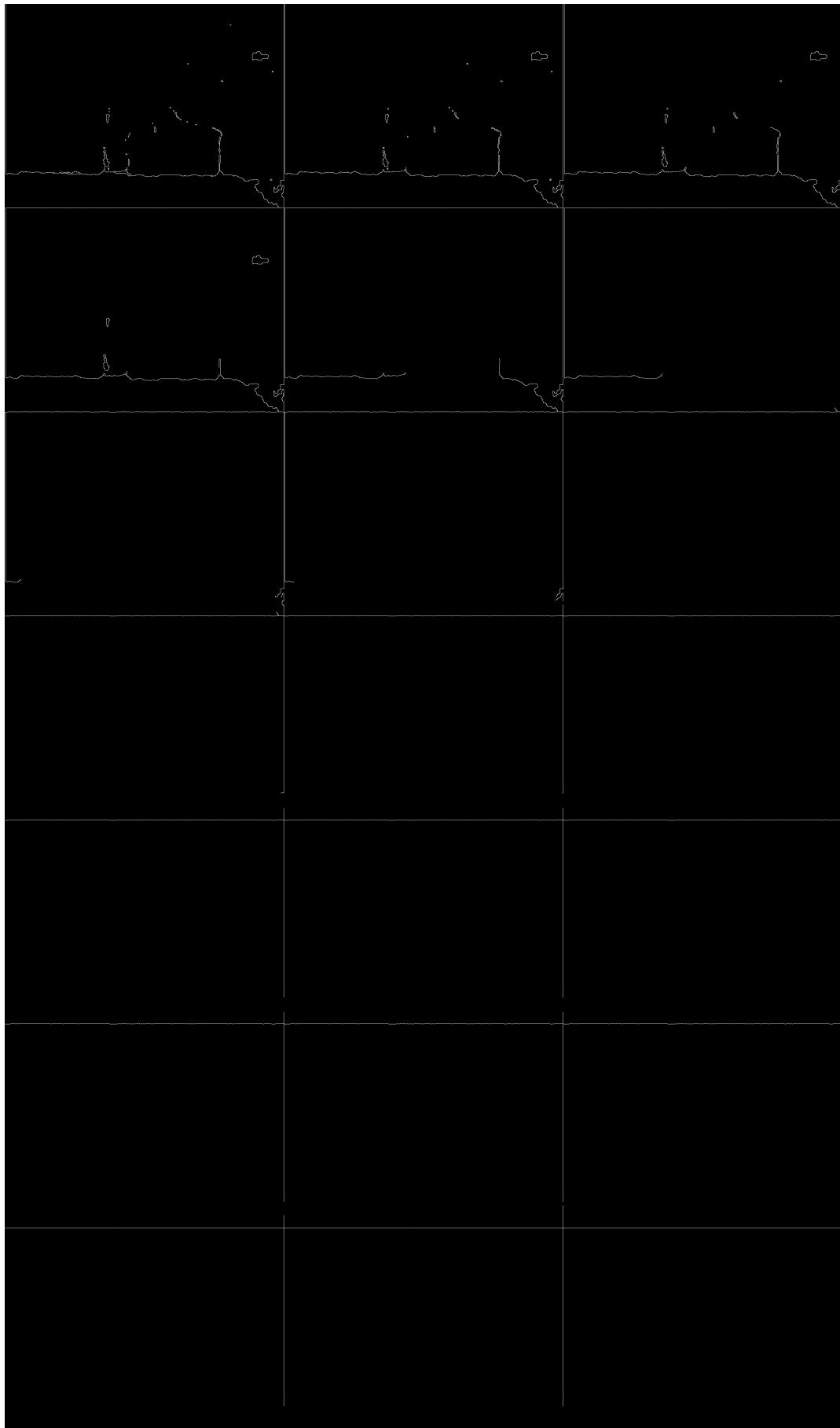


Figure 51: gauss_20_canny_200_600

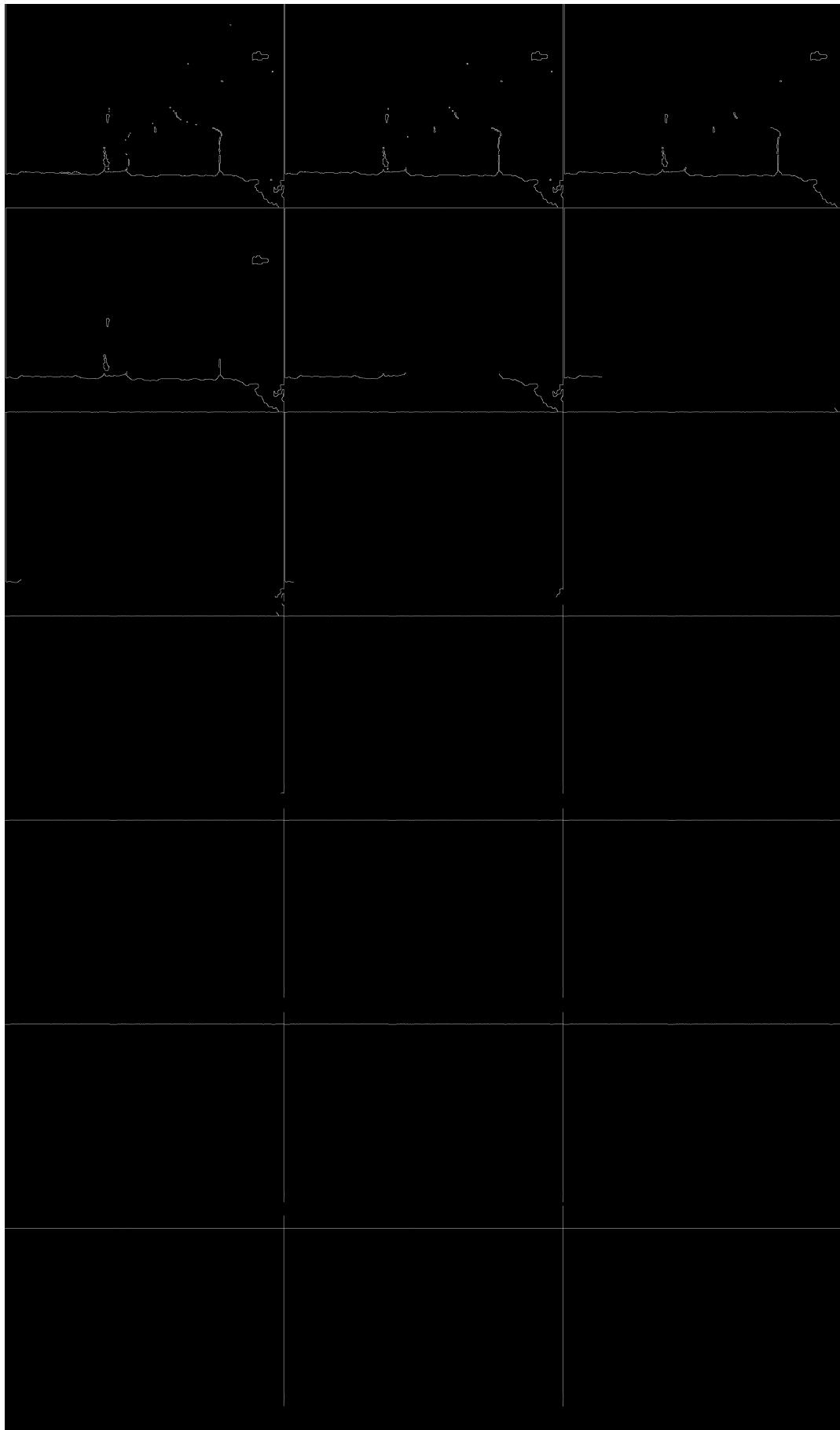


Figure 52: gauss_21_canny_210_630

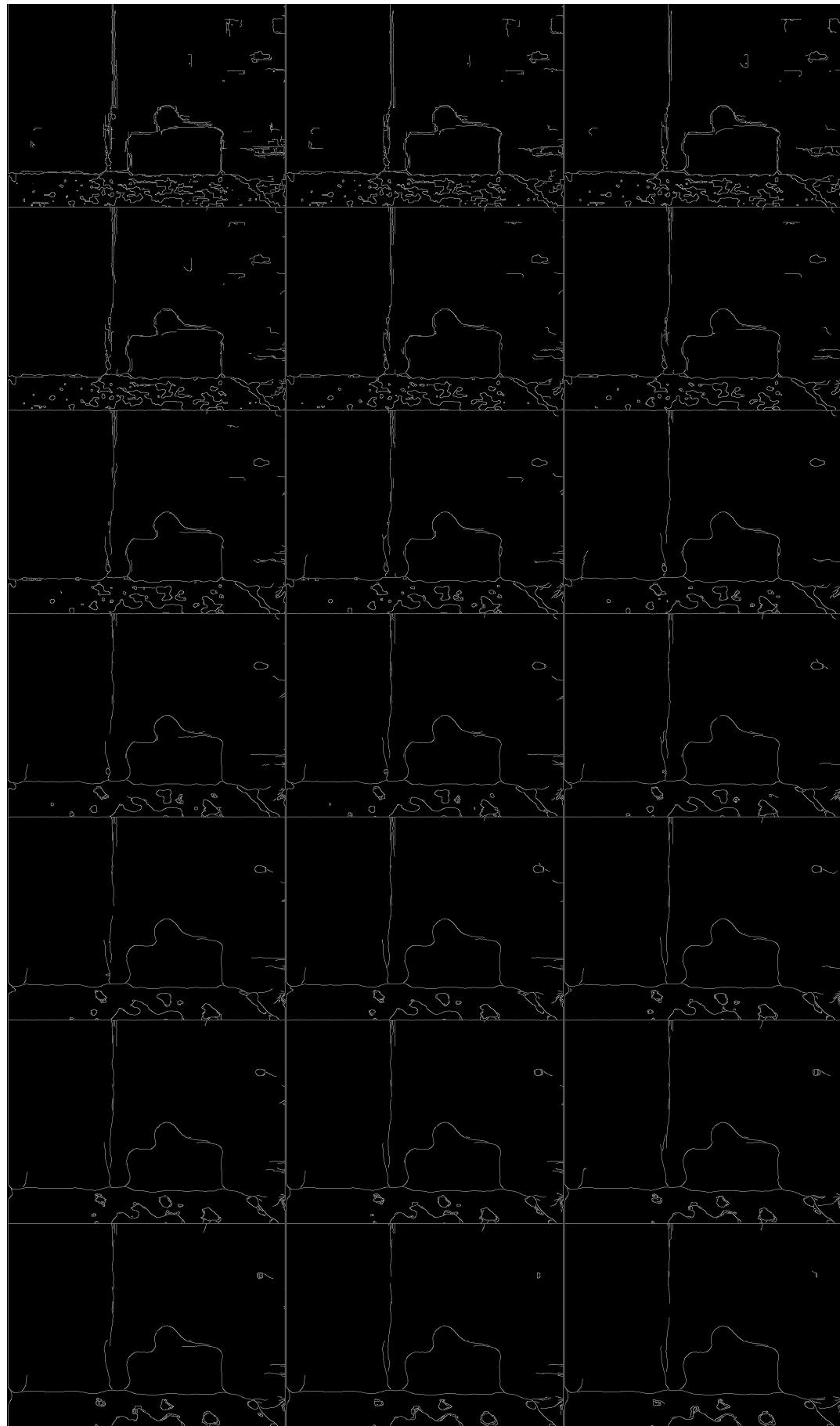


Figure 53: median_01_canny_10_30



Figure 54: median_02_canny_20_60

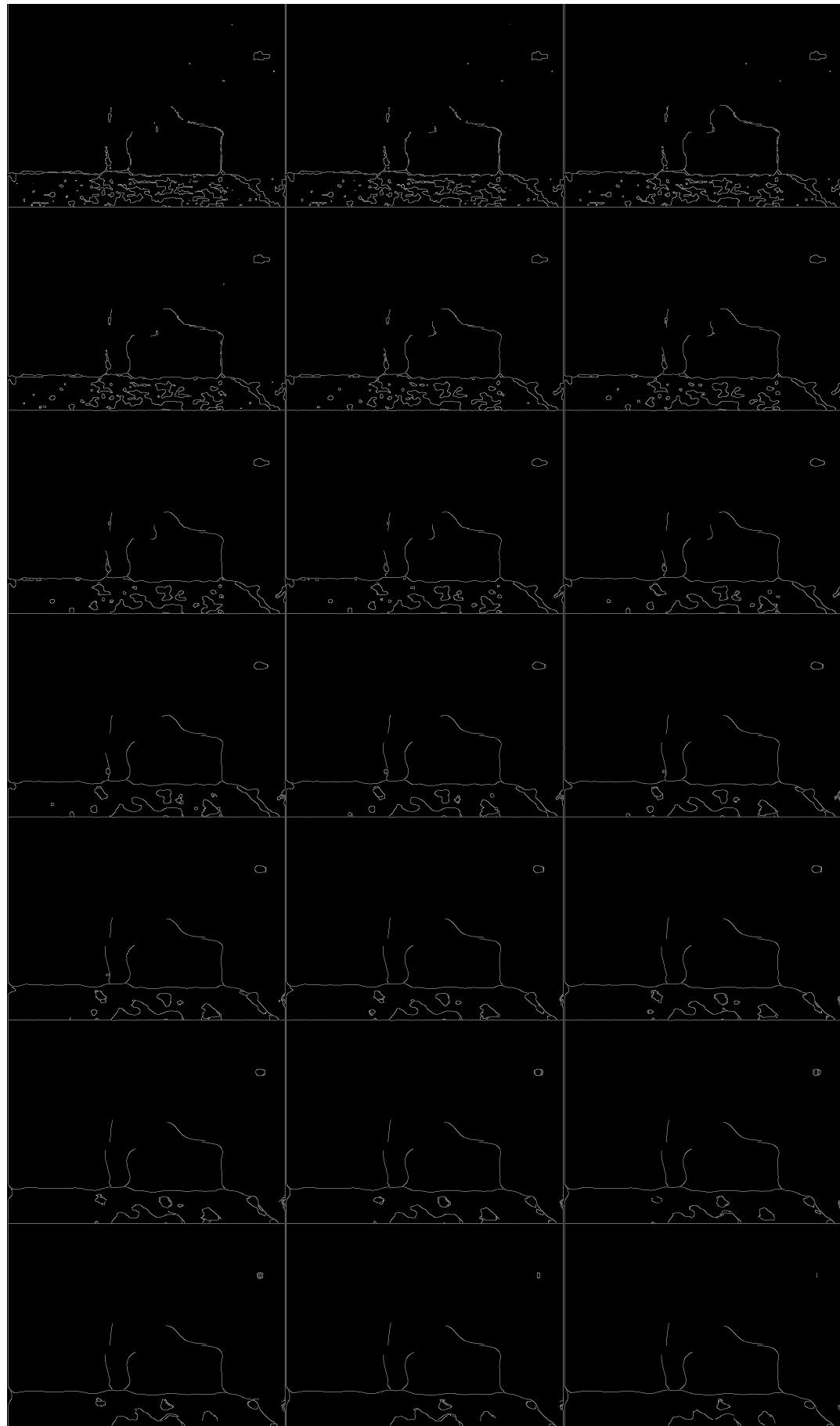


Figure 55: median_03_canny_30_90

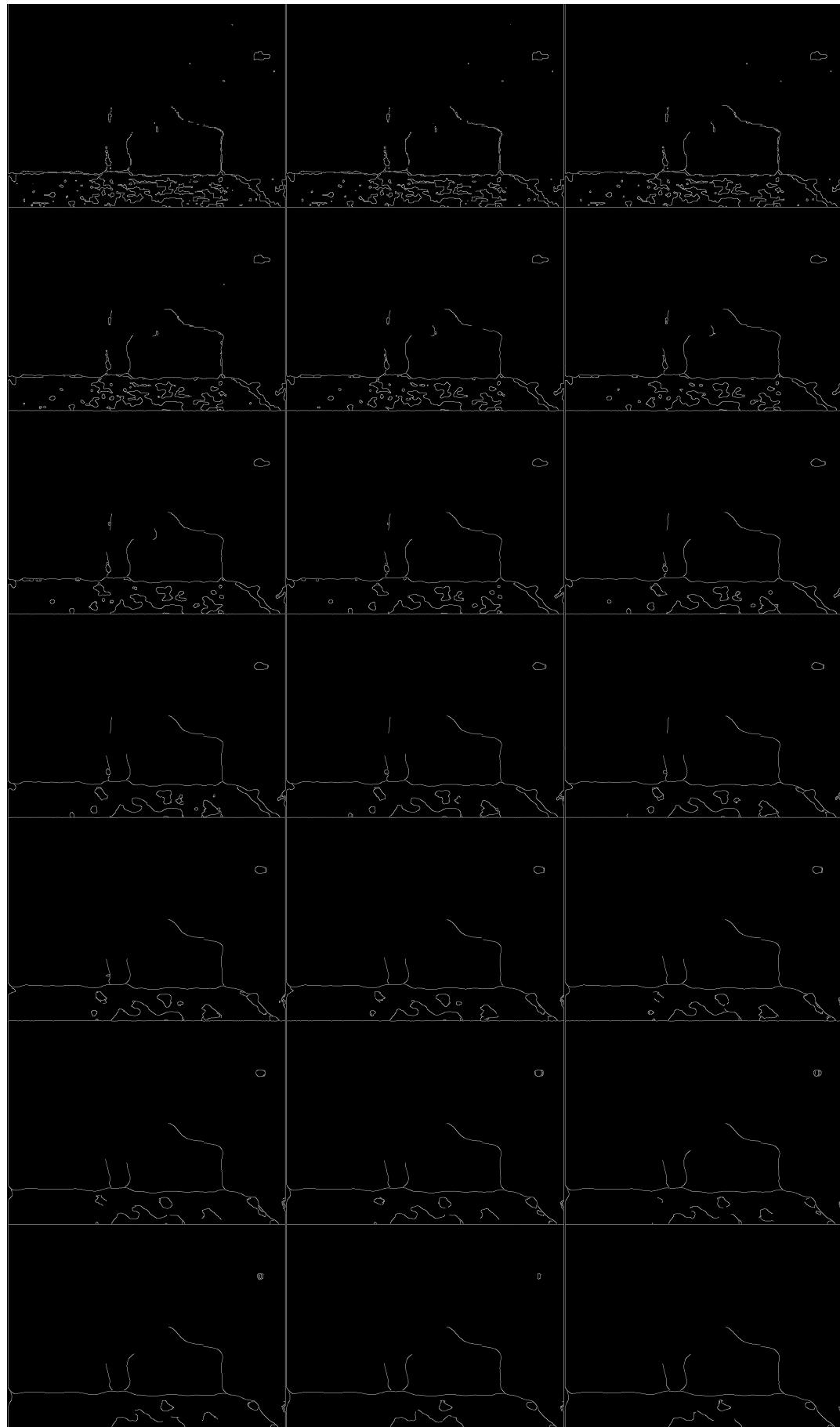


Figure 56: median_04_canny_40_120

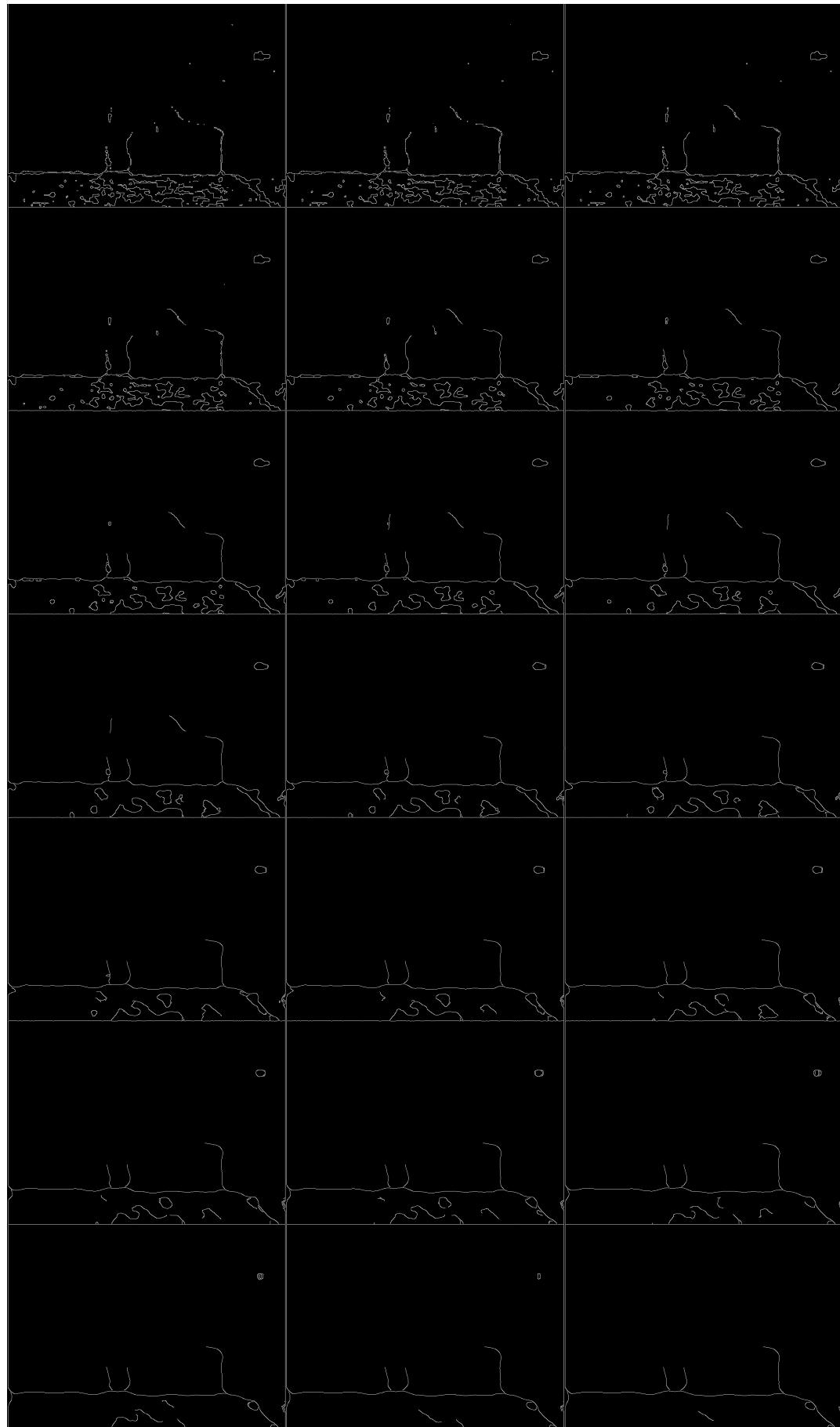


Figure 57: median_05_canny_50_150

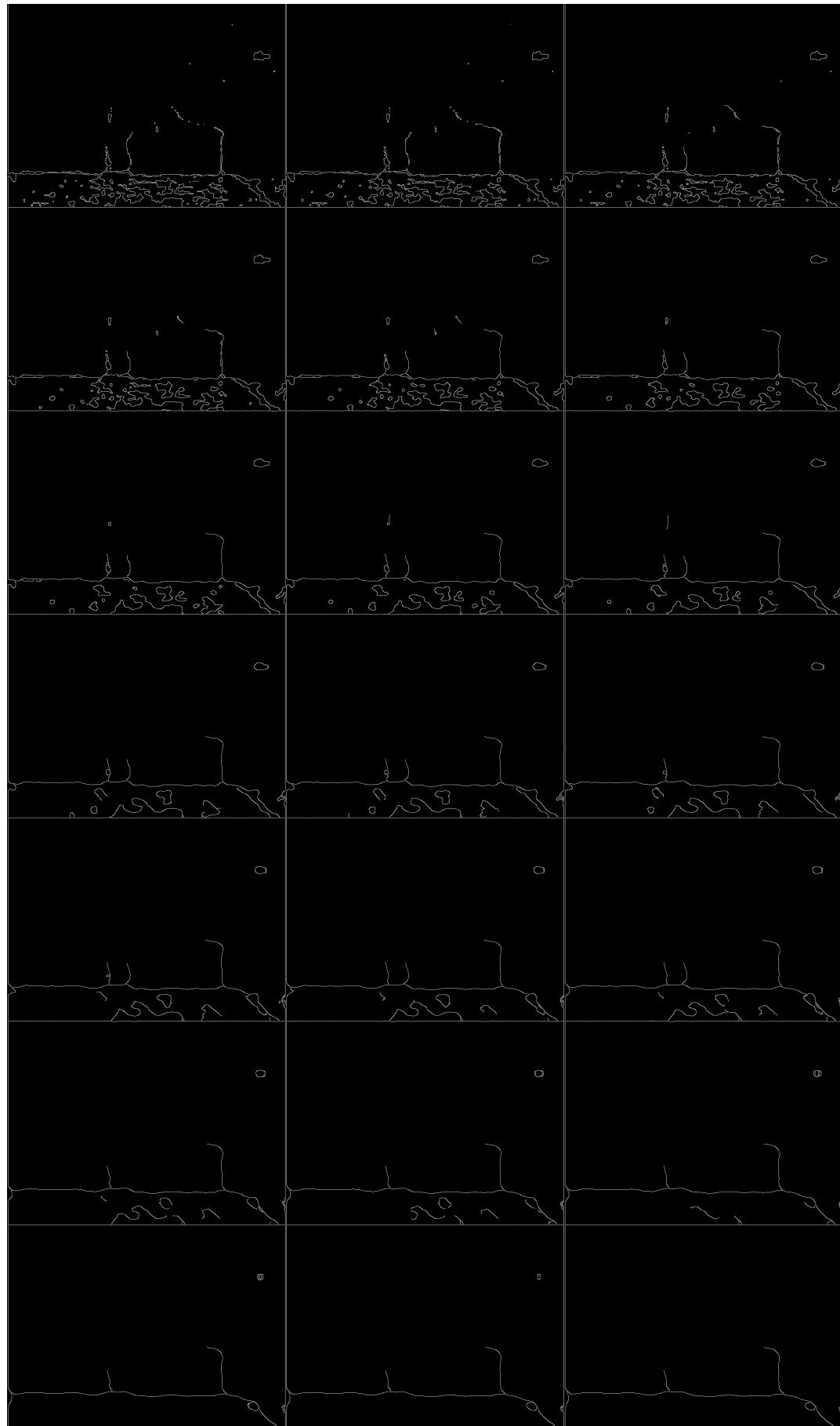


Figure 58: median_06_canny_60_180

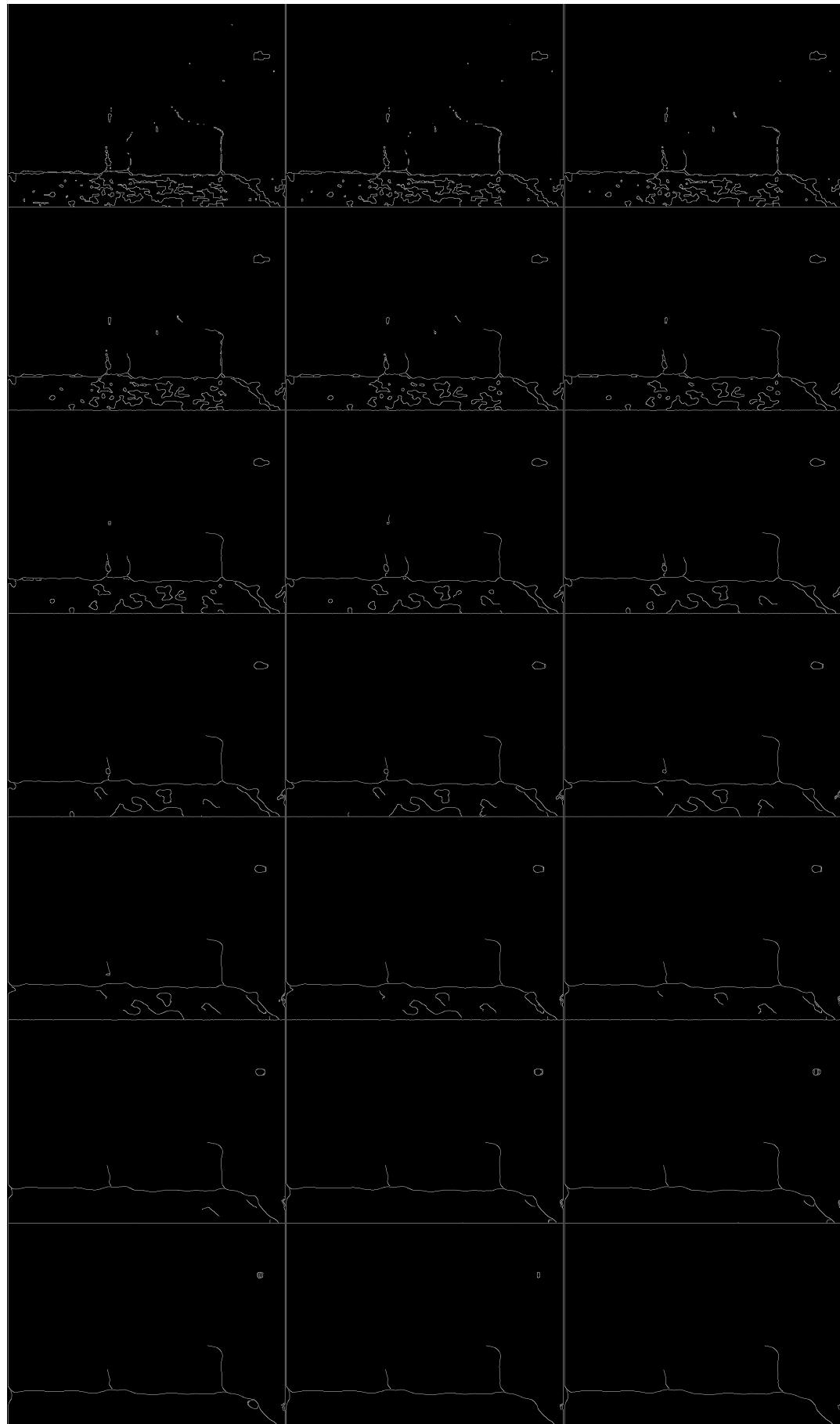


Figure 59: median_07_canny_70_210

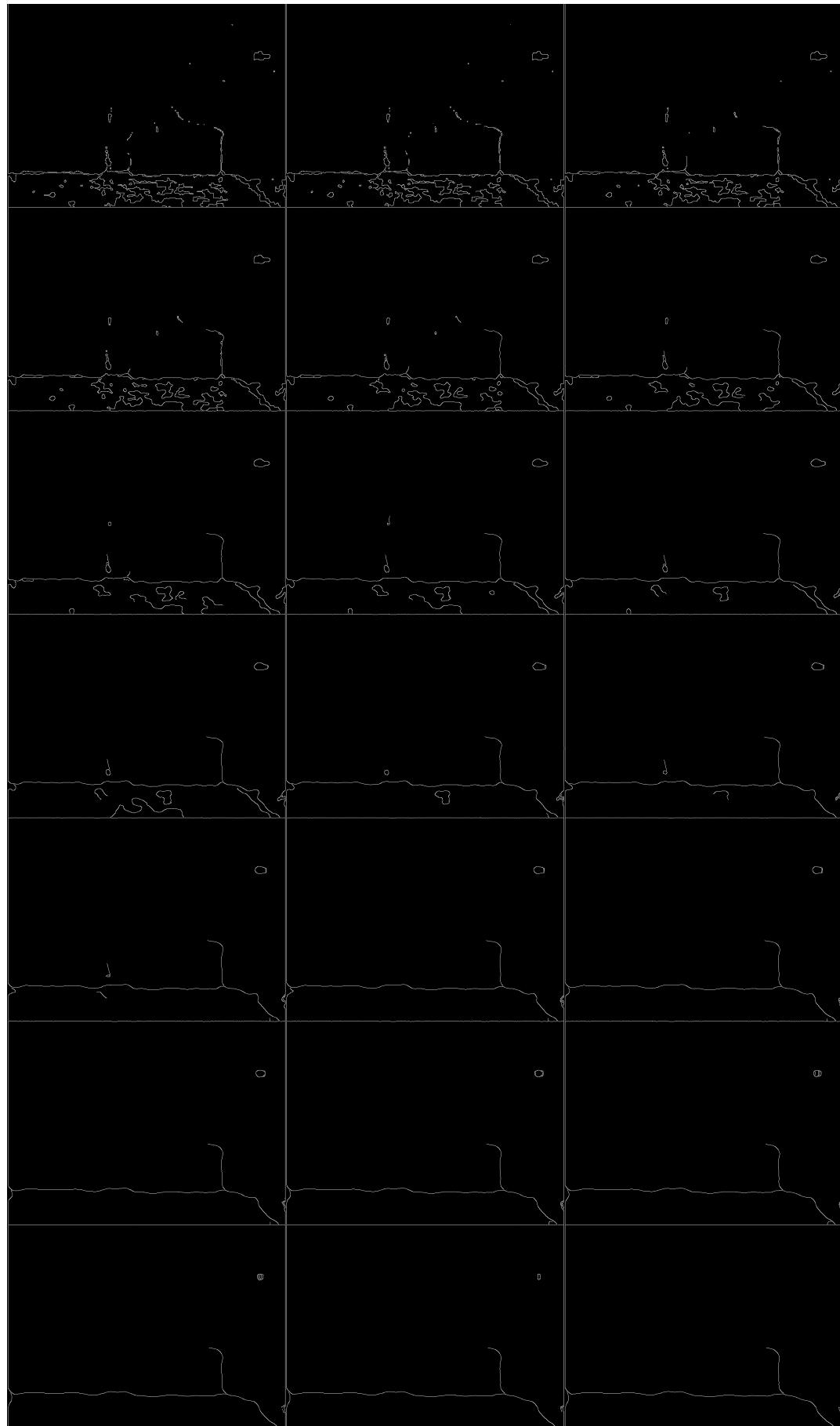


Figure 60: median_08_canny_80_240

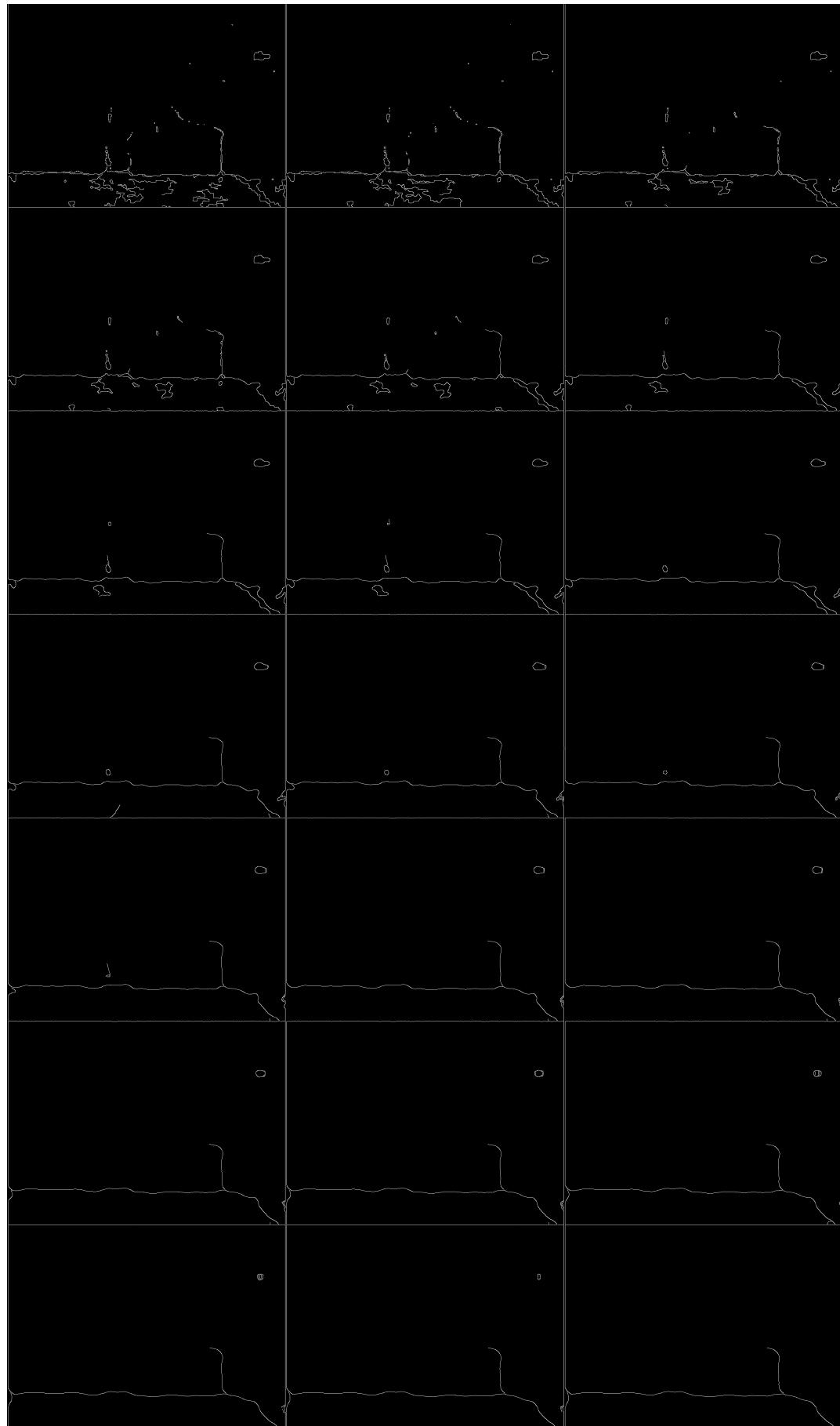


Figure 61: median_09_canny_90_270

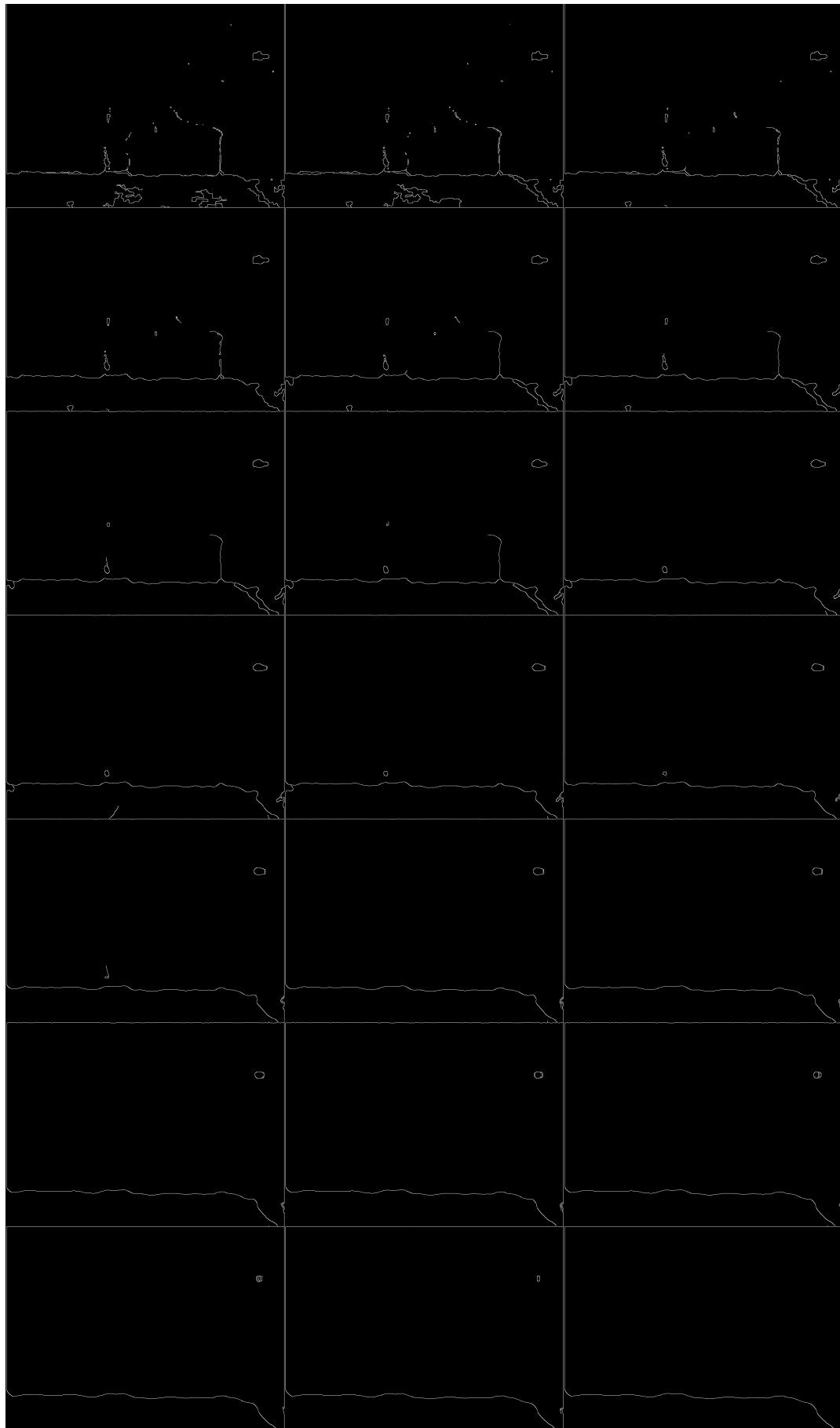


Figure 62: median_10_canny_100_300

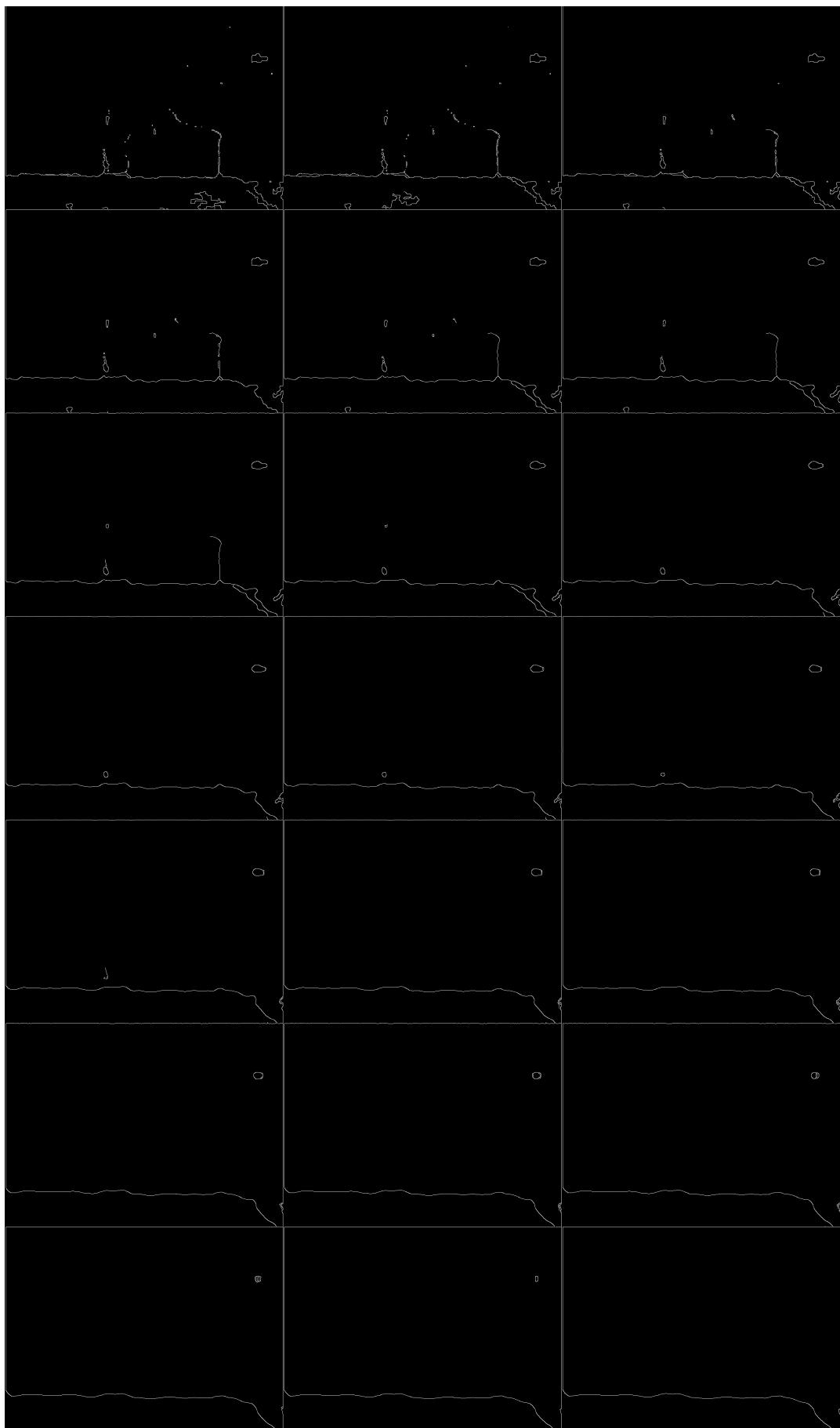


Figure 63: median_11_canny_110_330

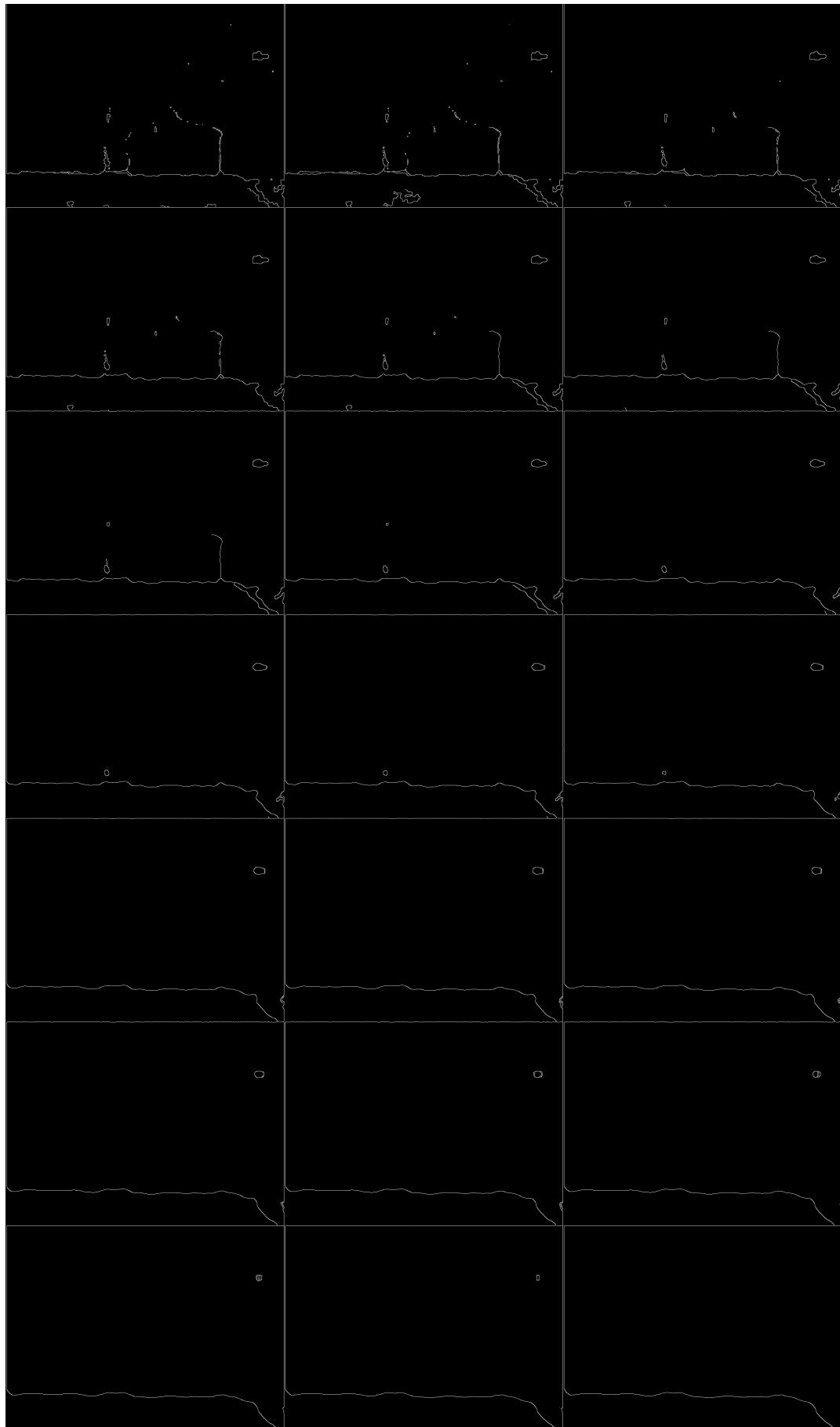


Figure 64: median_12_canny_120_360

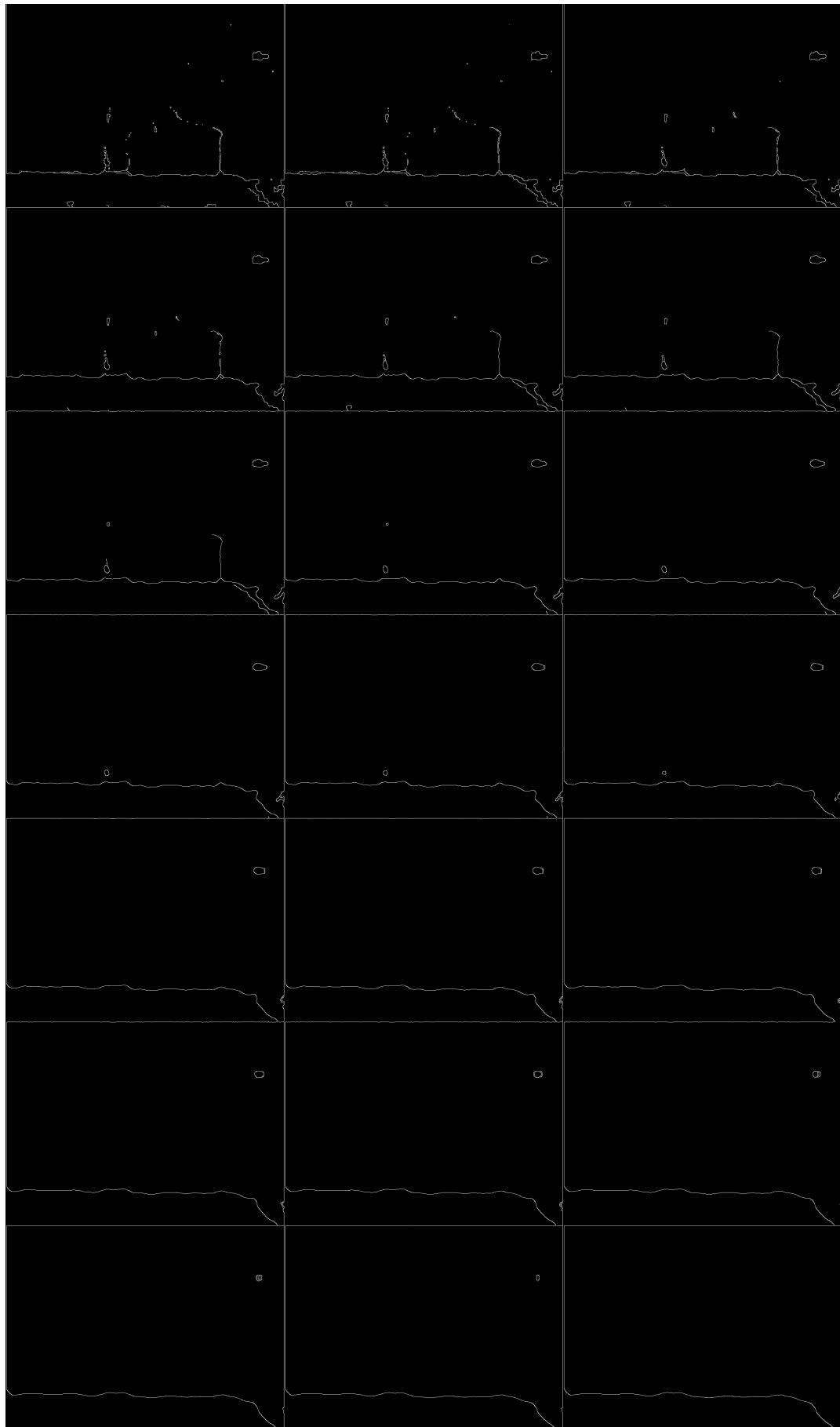


Figure 65: median_13_canny_130_390

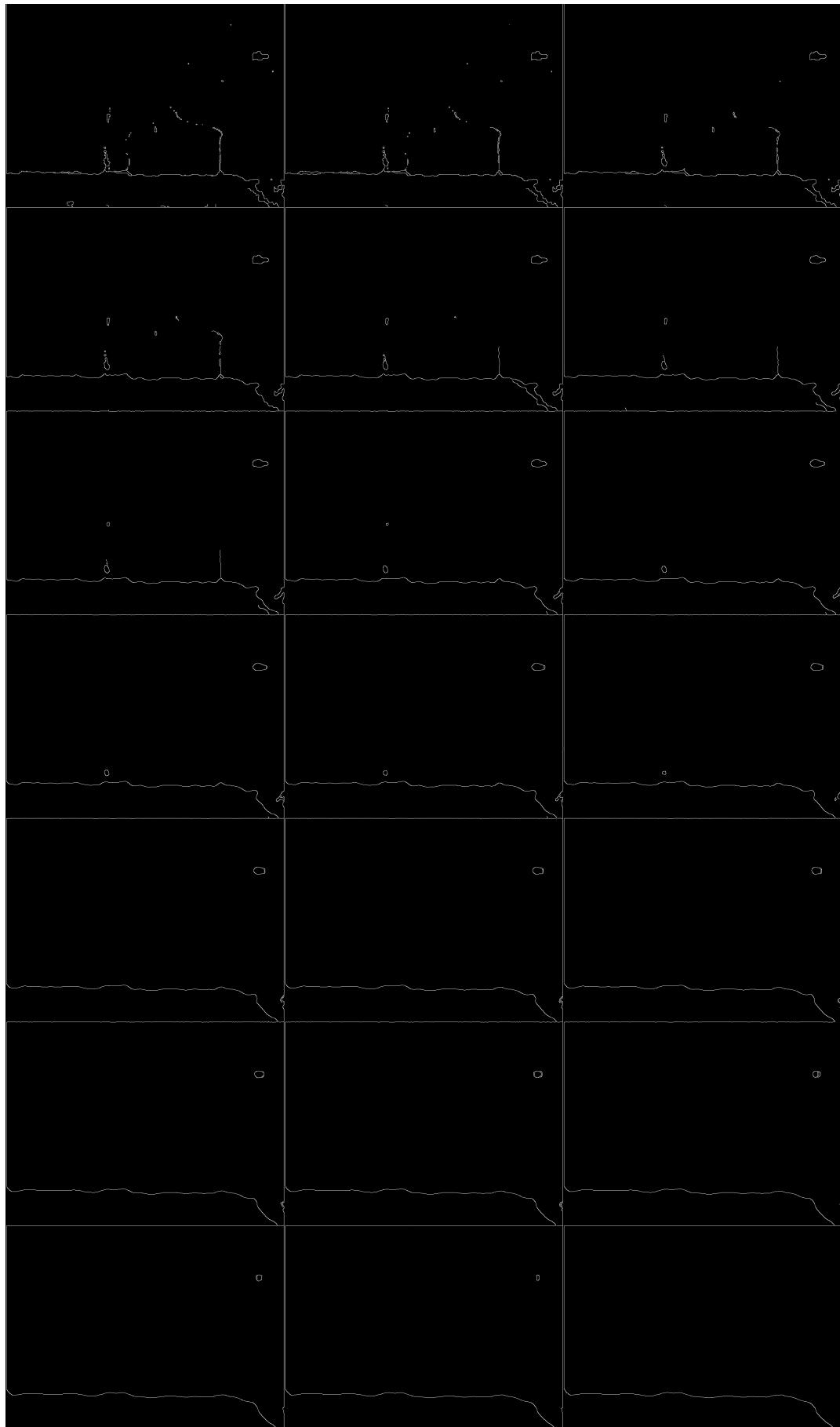


Figure 66: median_14_canny_140_420

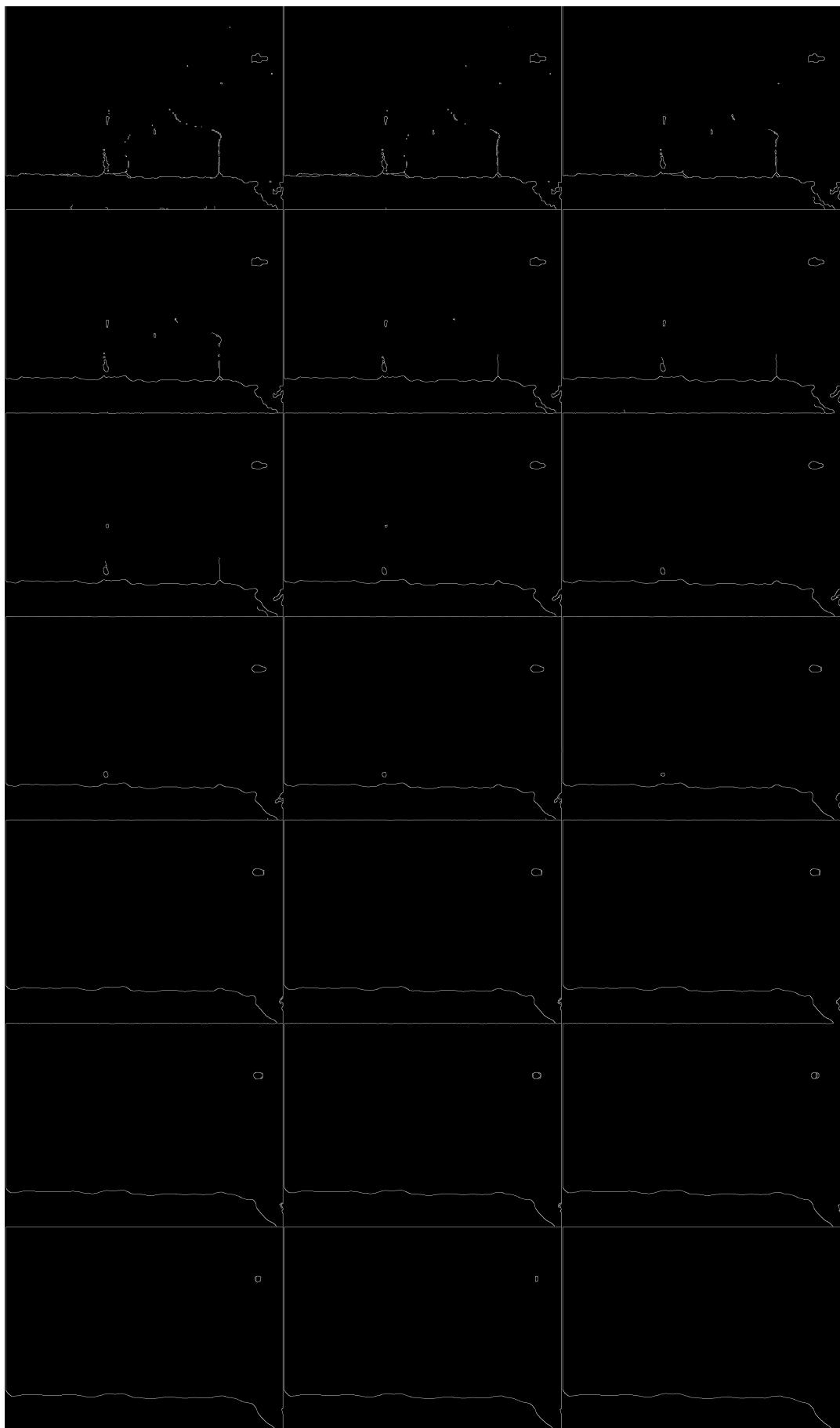


Figure 67: median_15_canny_150_450

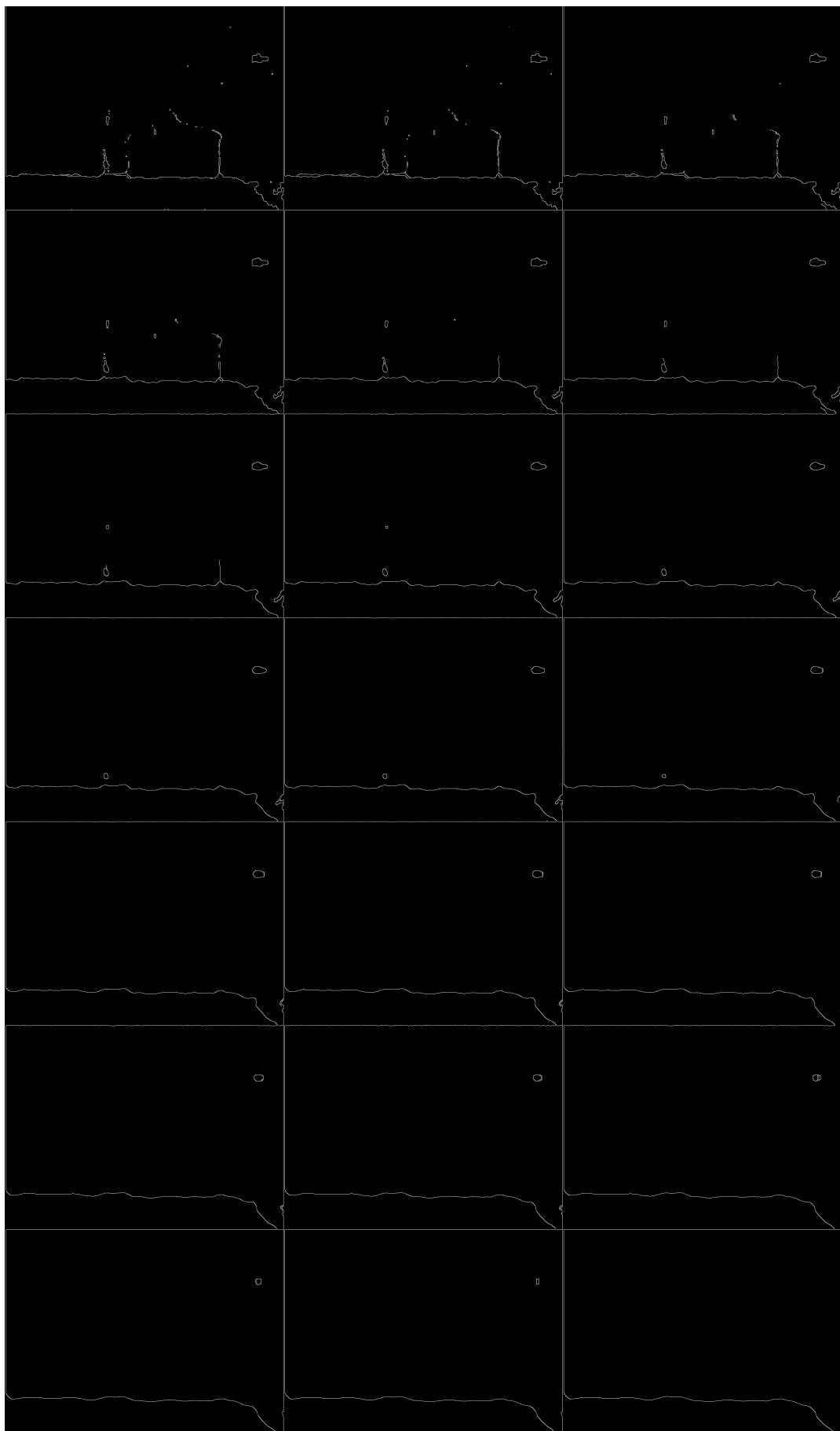


Figure 68: median_16_canny_160_480

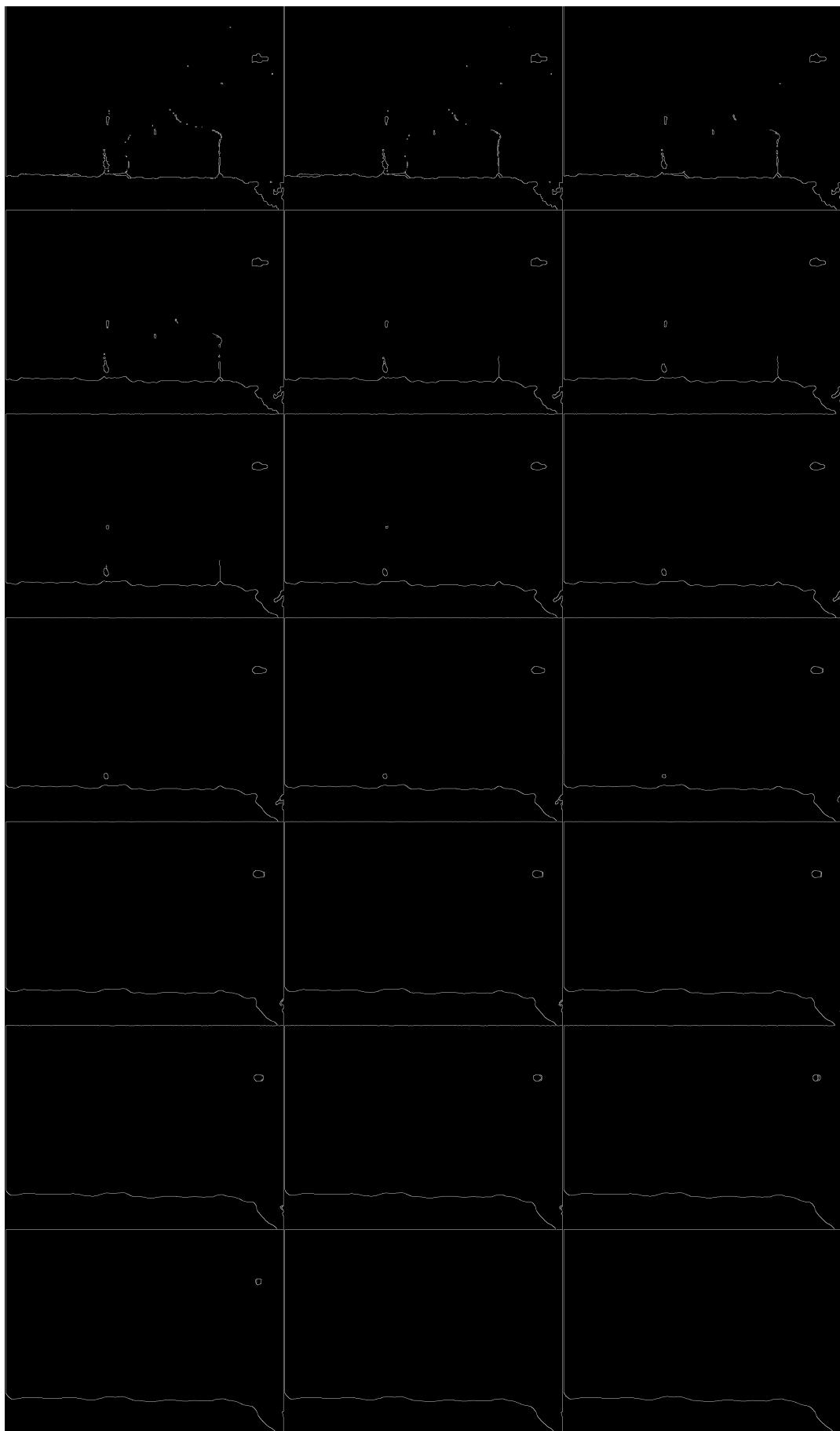


Figure 69: median_17_canny_170_510

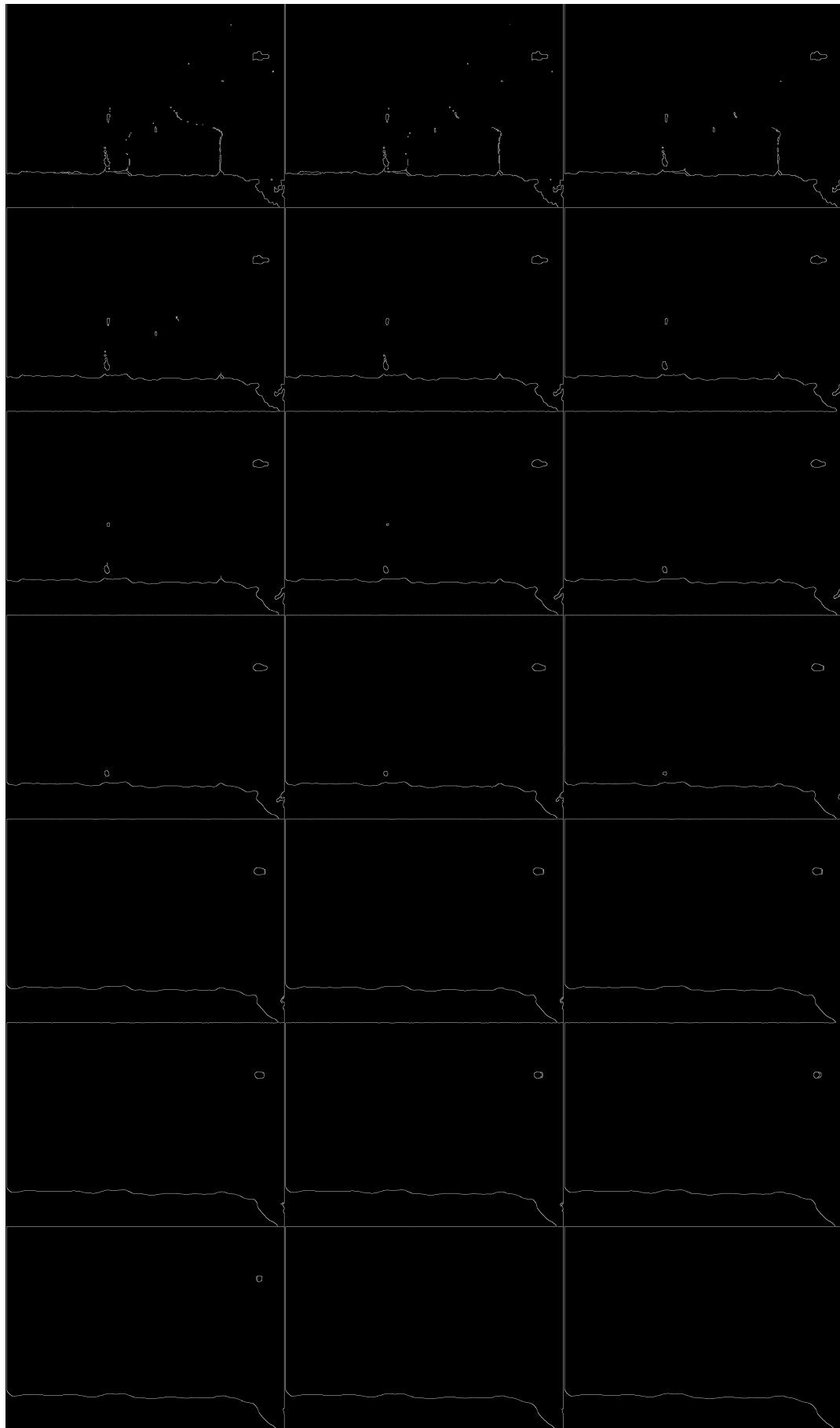


Figure 70: median_18_canny_180_540

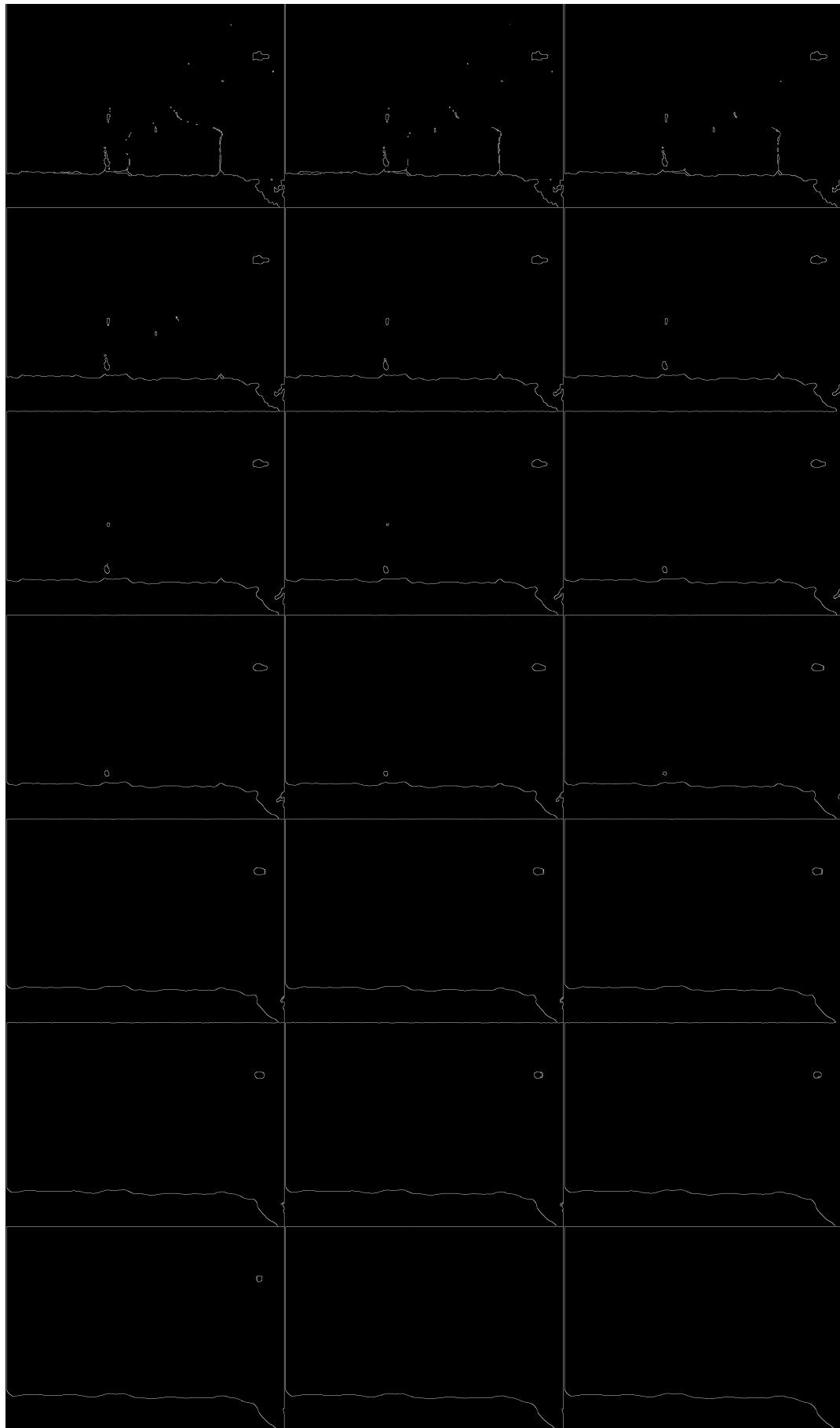


Figure 71: median_19_canny_190_570

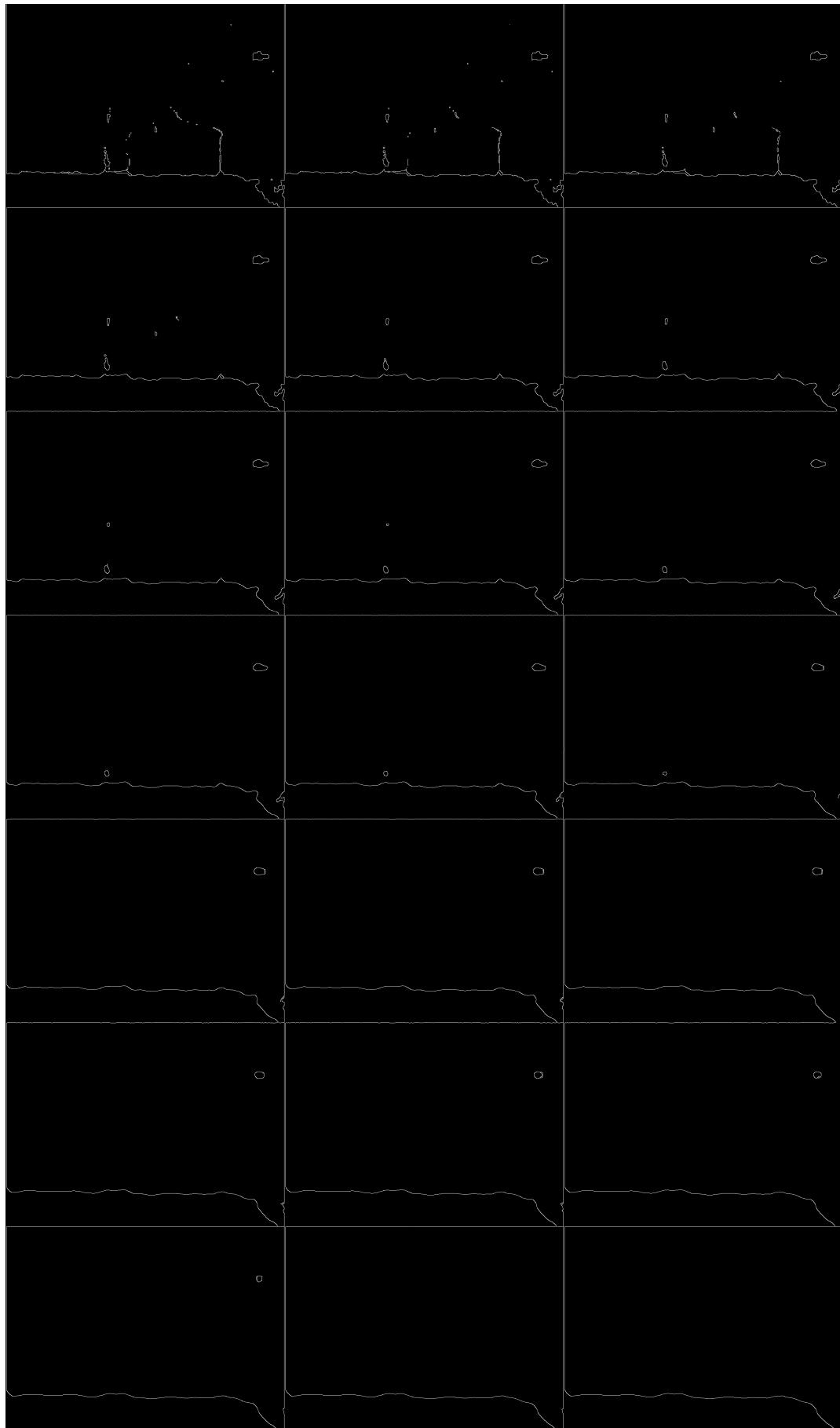


Figure 72: median_20_canny_200_600

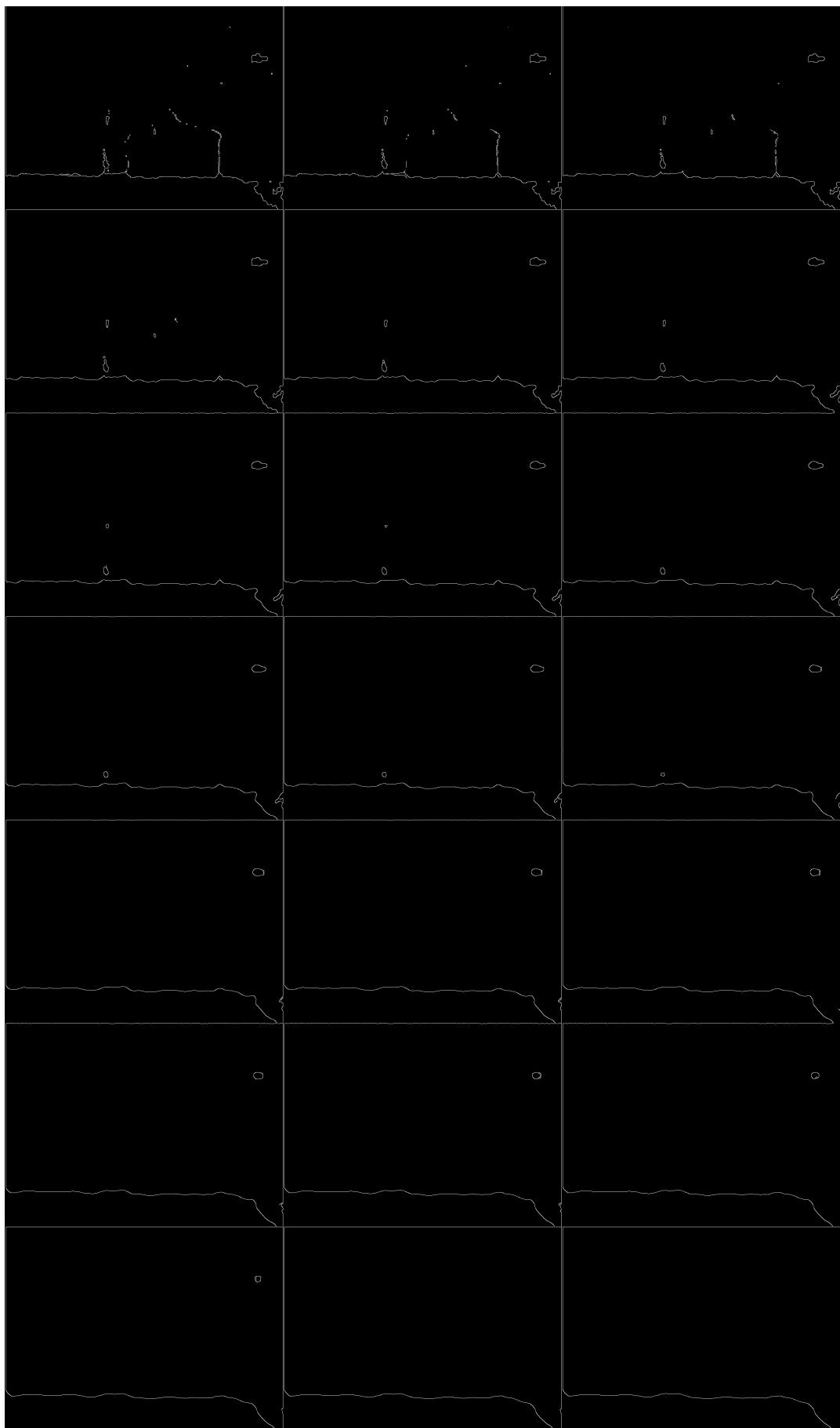


Figure 73: median_21_canny_210_630

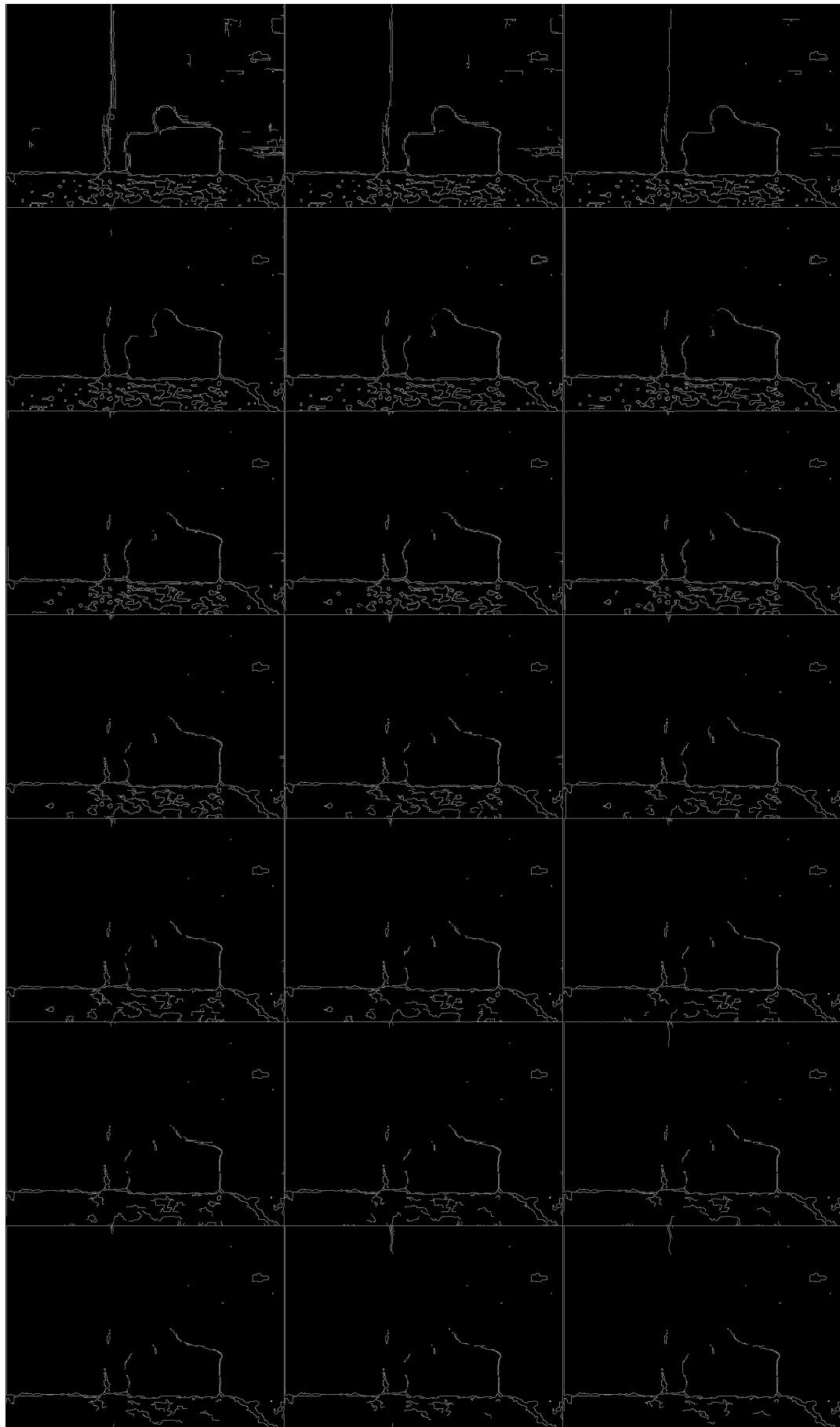


Figure 74: bilateral_01_canny_10_30

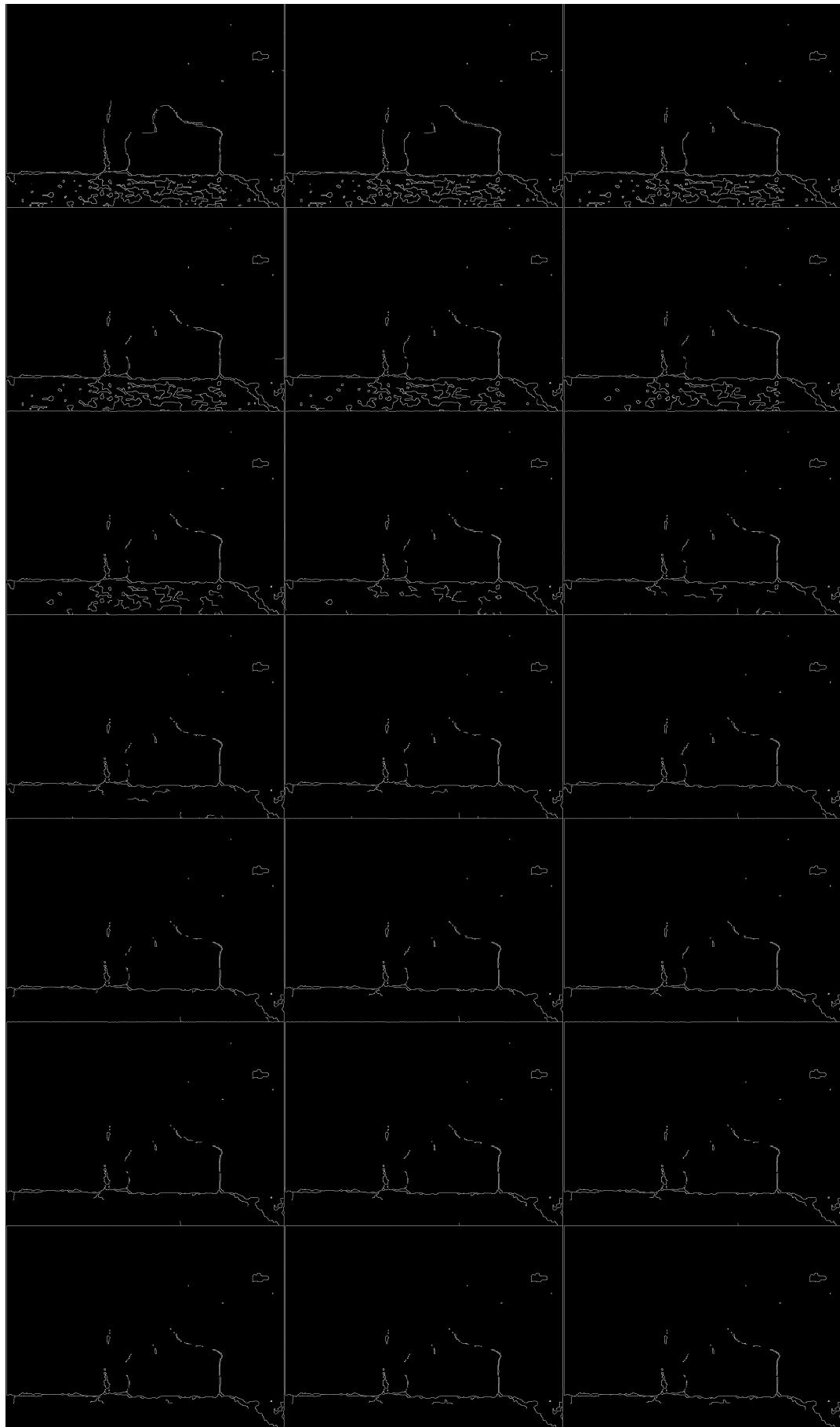


Figure 75: bilateral_02_canny_20_60

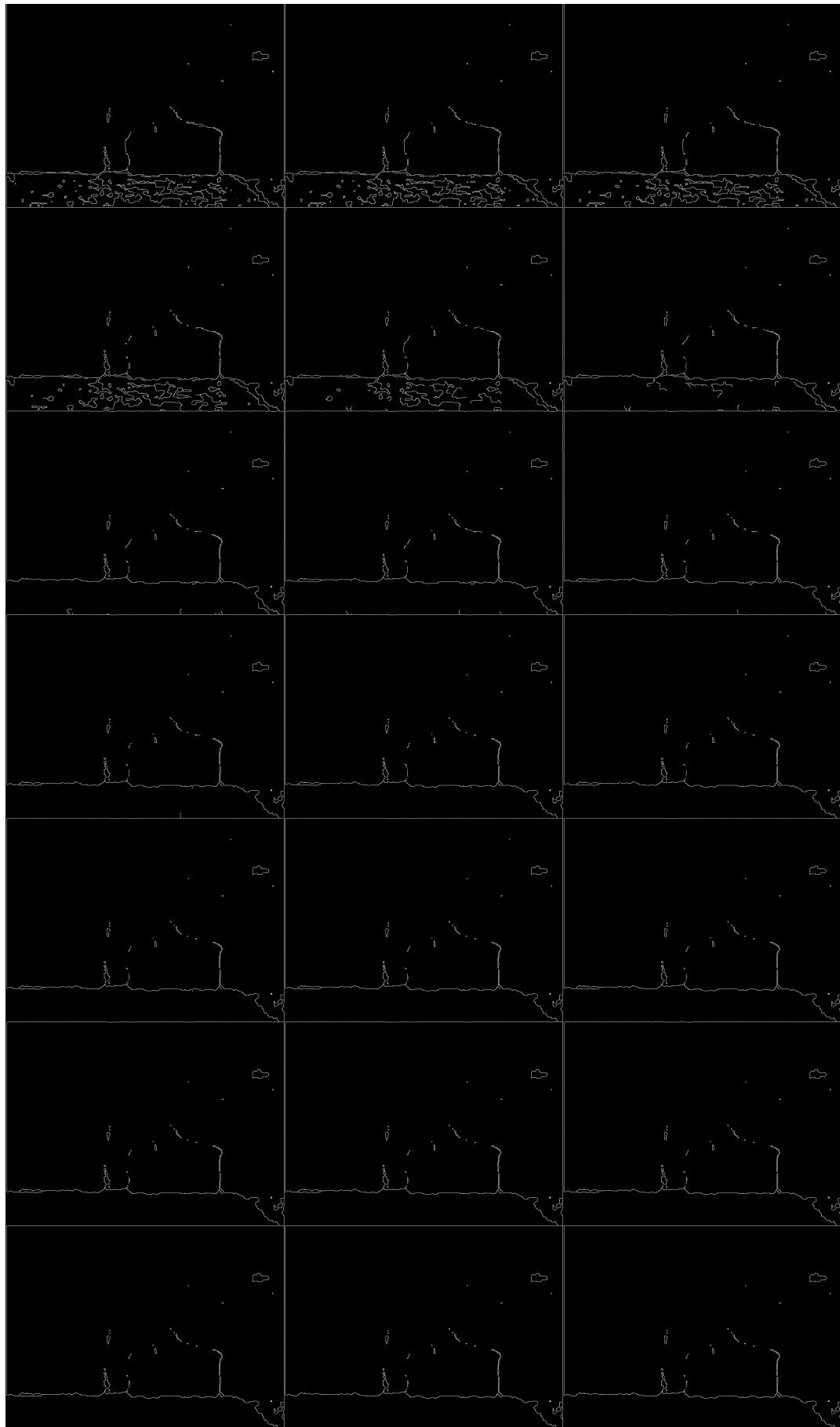


Figure 76: bilateral_03_canny_30_90

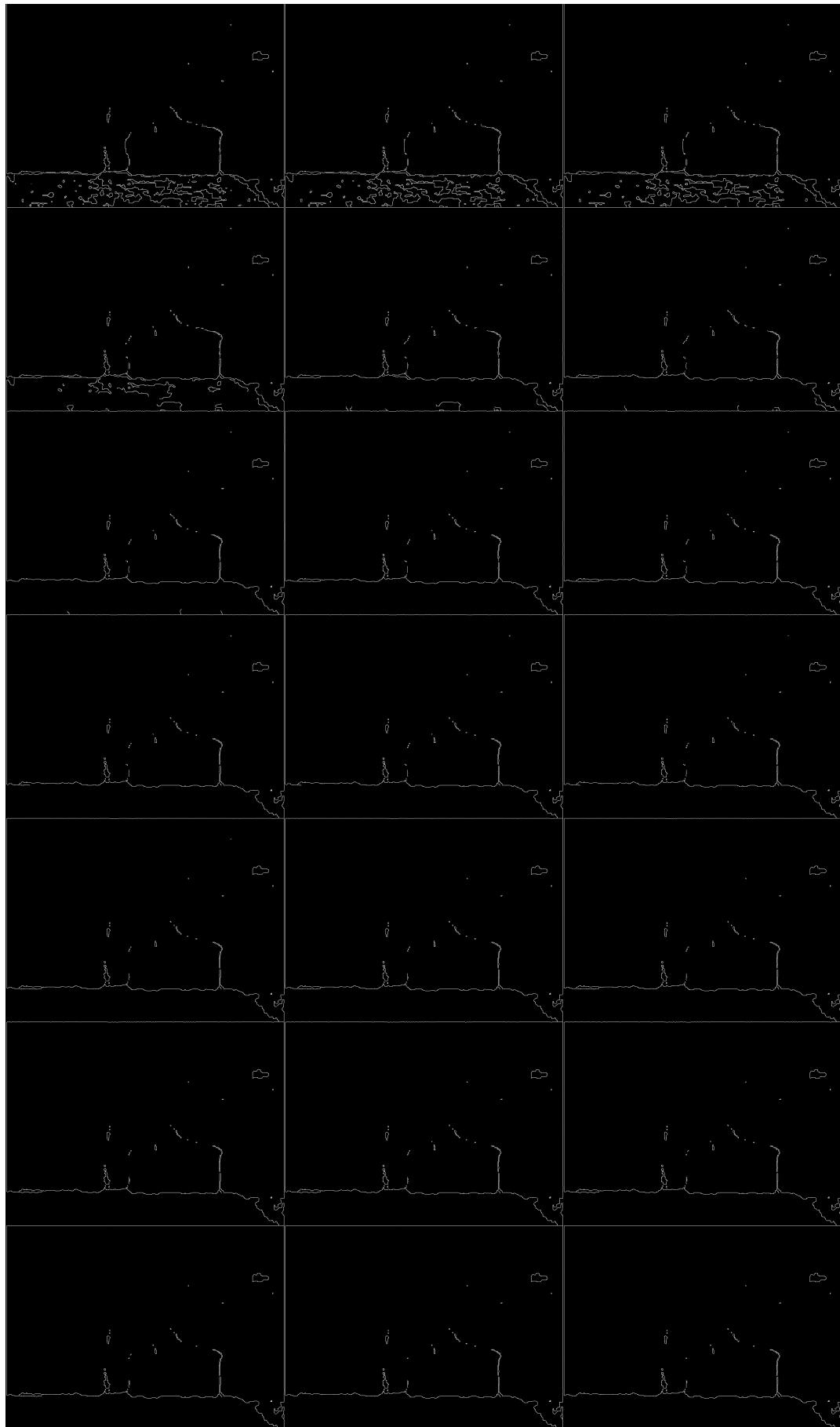


Figure 77: bilateral_04_canny_40_120

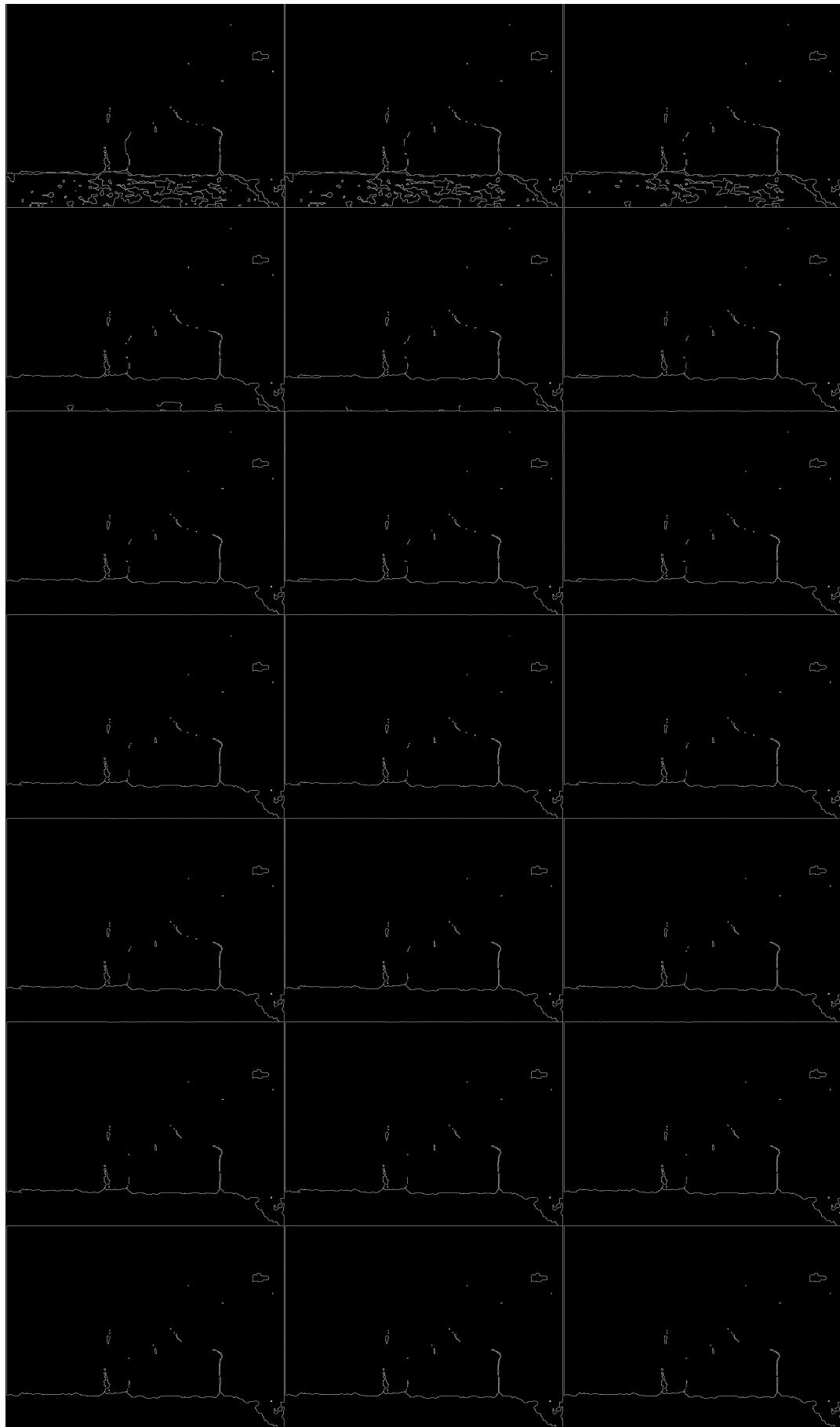


Figure 78: bilateral_05_canny_50_150

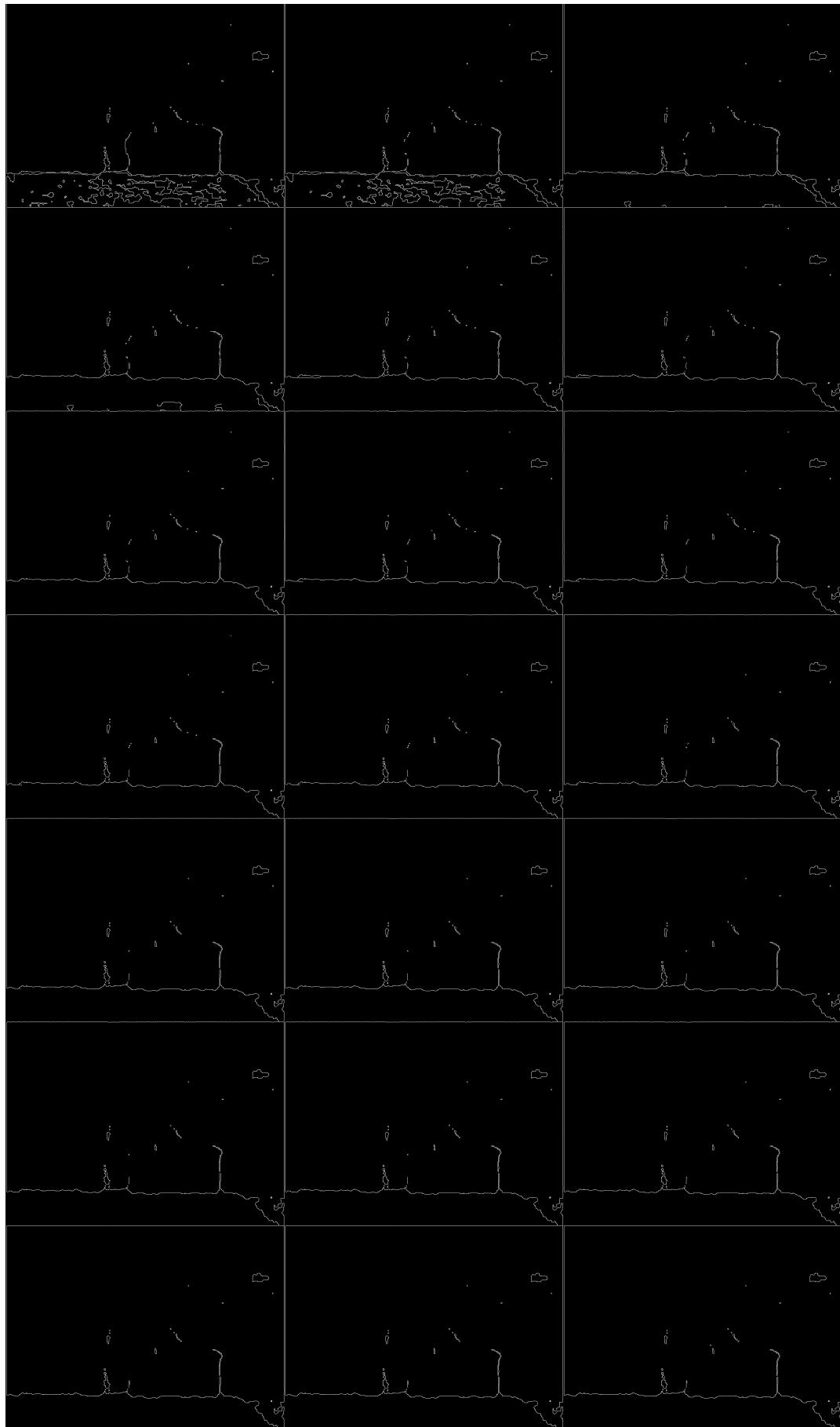


Figure 79: bilateral_06_canny_60_180

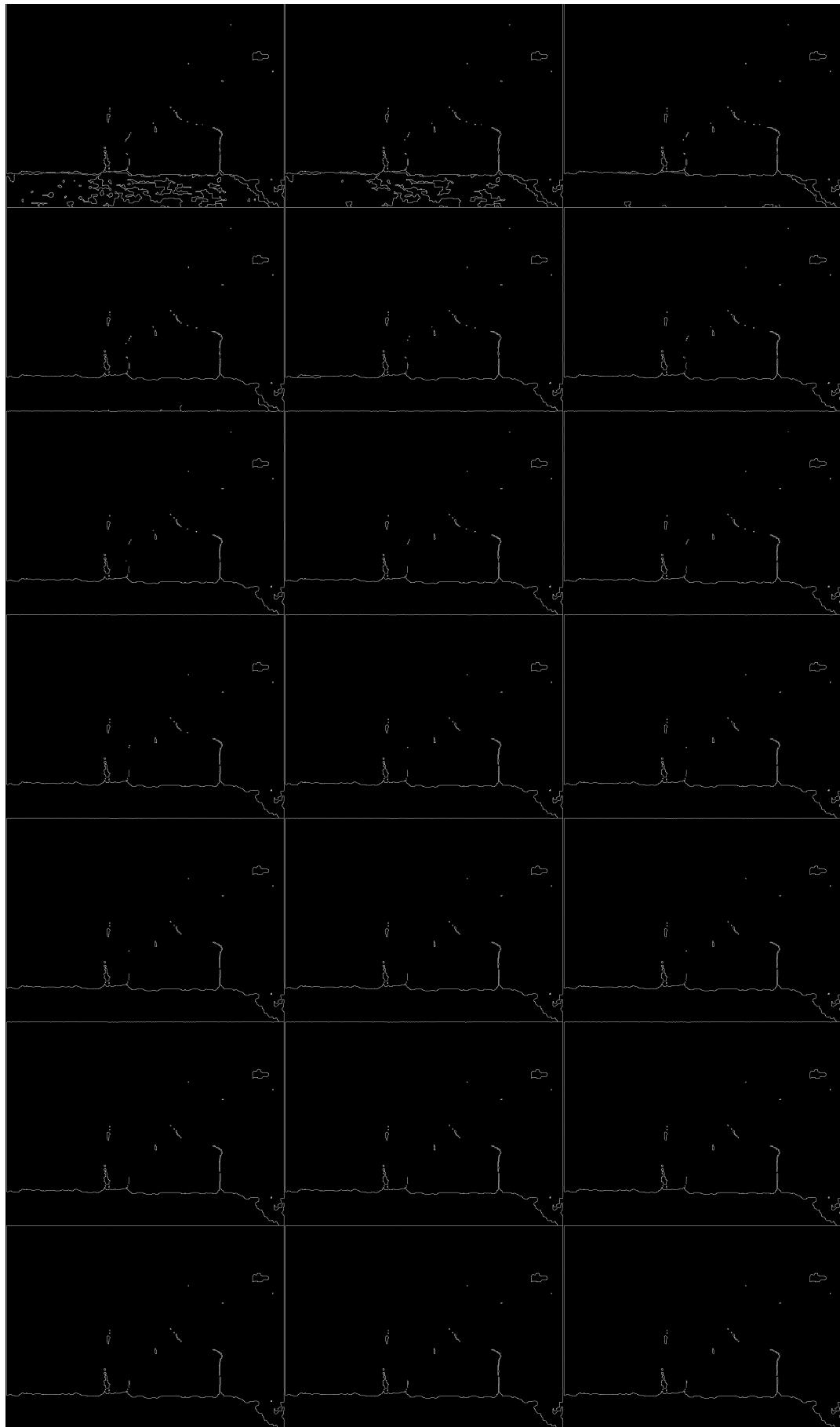


Figure 80: bilateral_07_canny_70_210

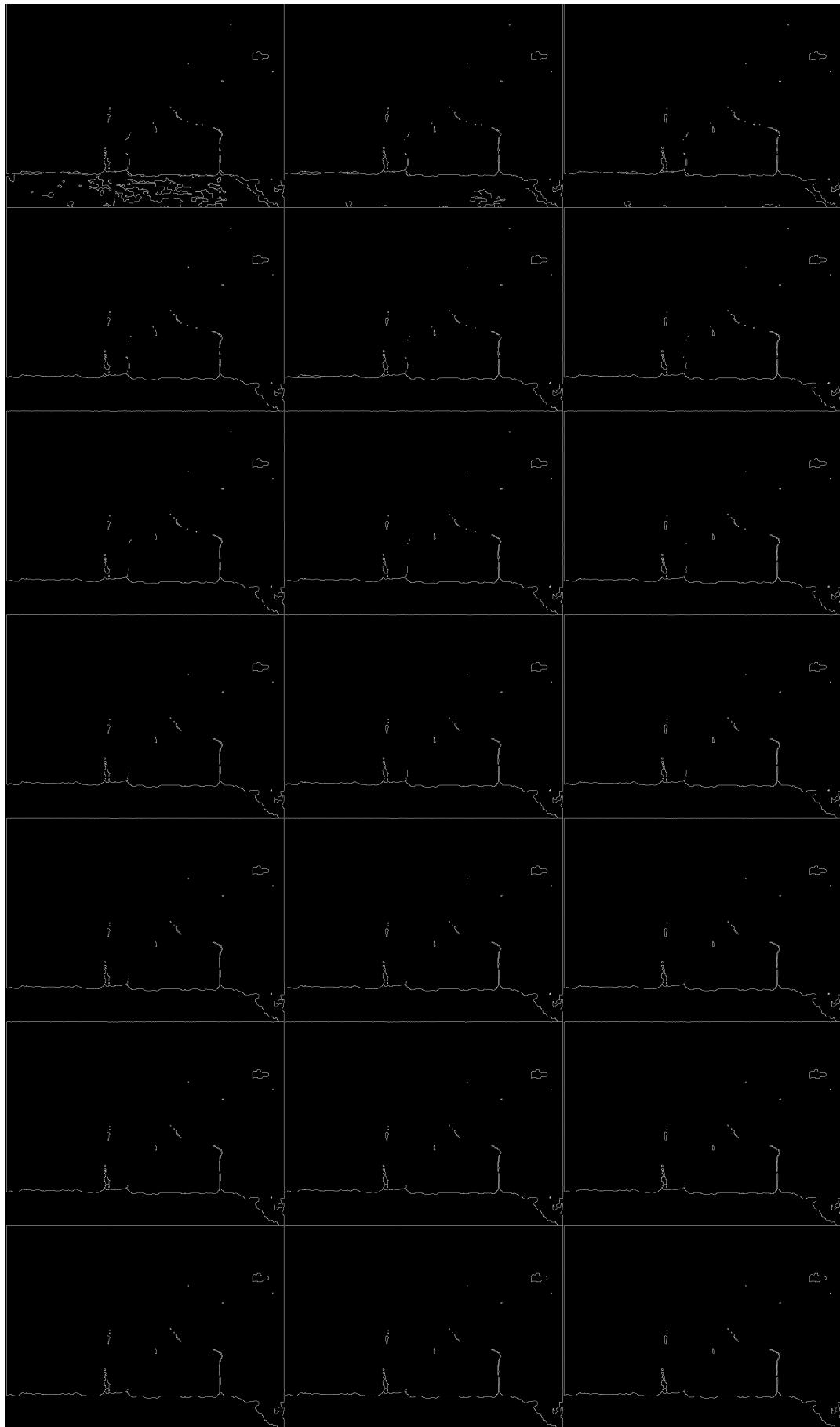


Figure 81: bilateral_08_canny_80_240

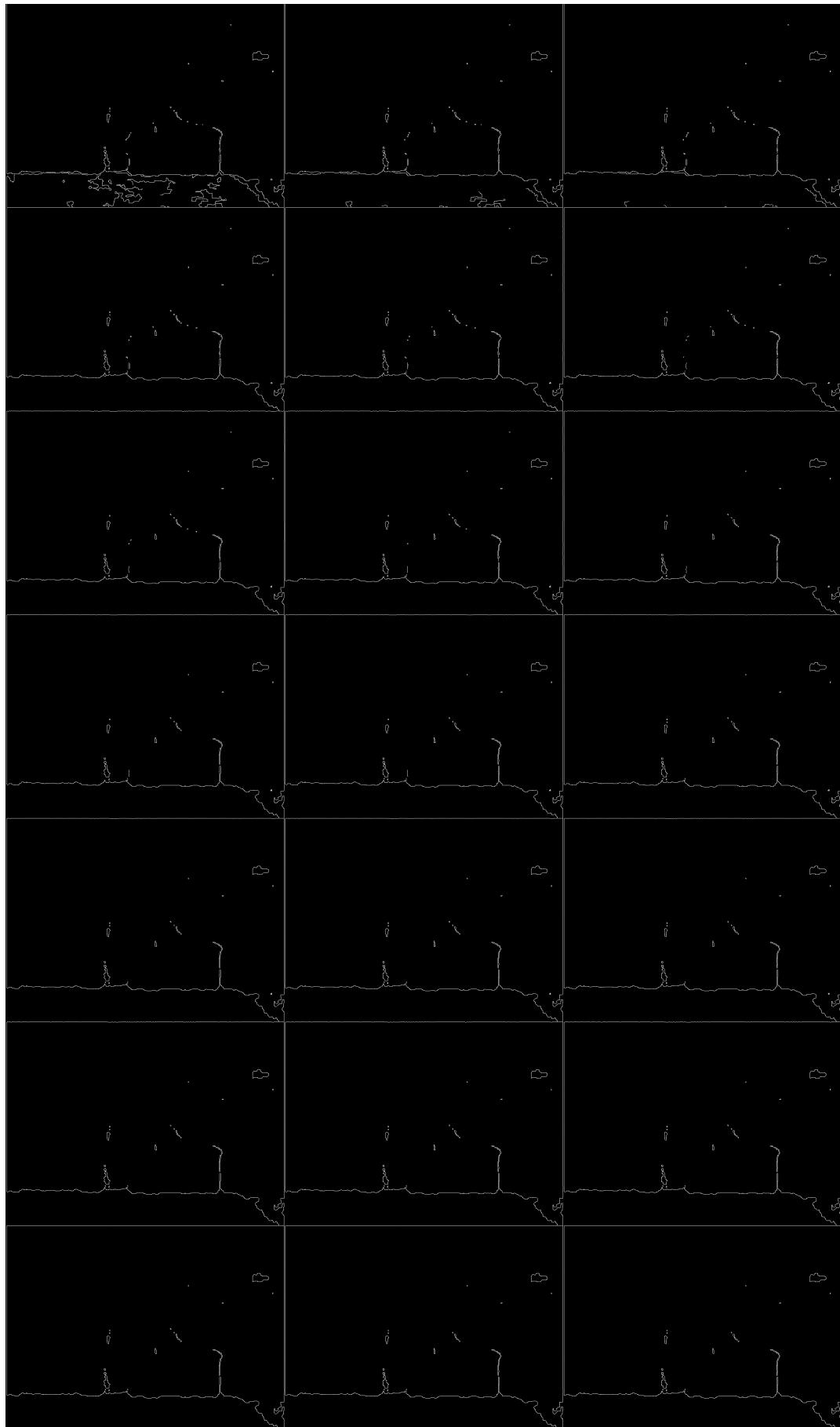


Figure 82: bilateral_09_canny_90_270

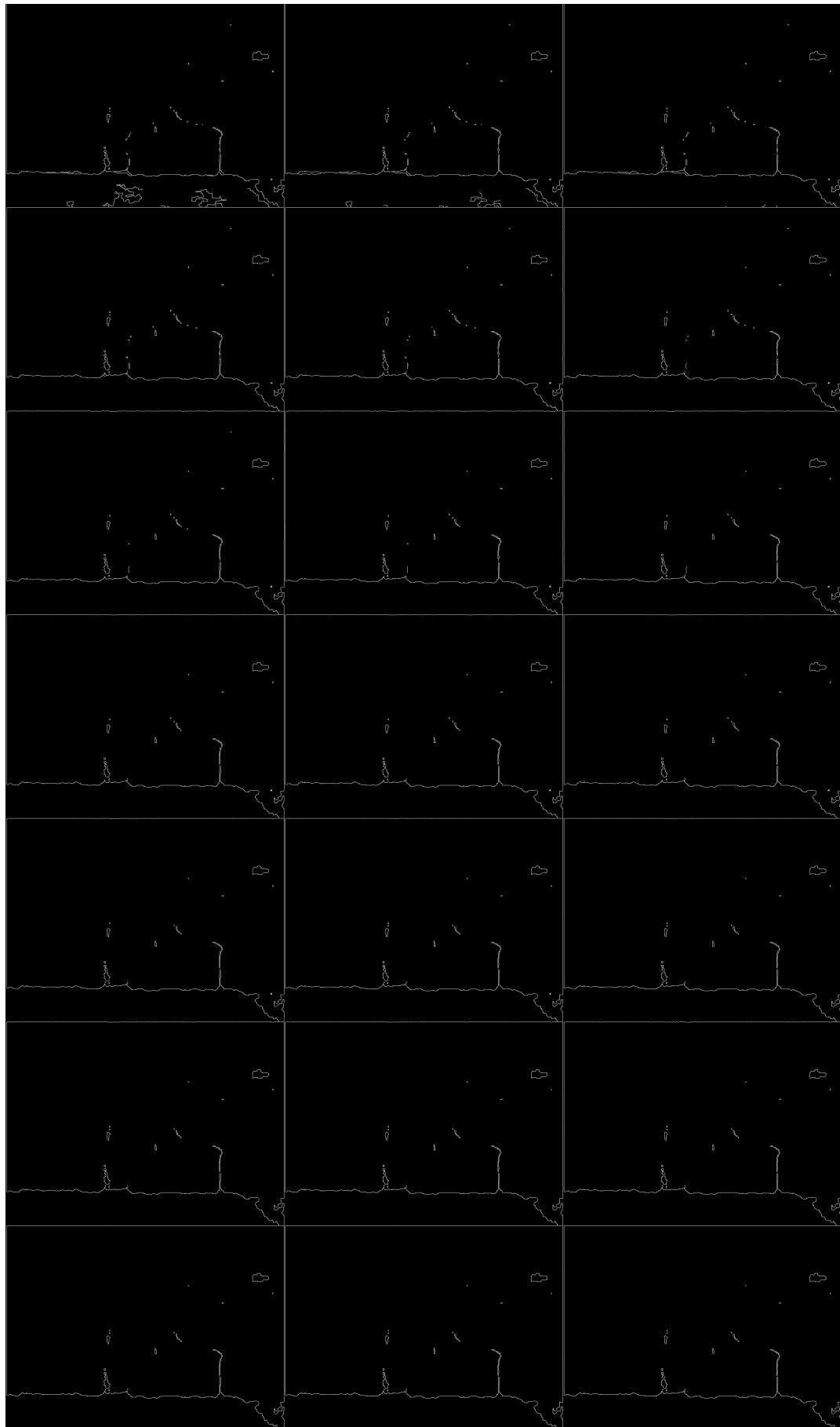


Figure 83: bilateral_10_canny_100_300

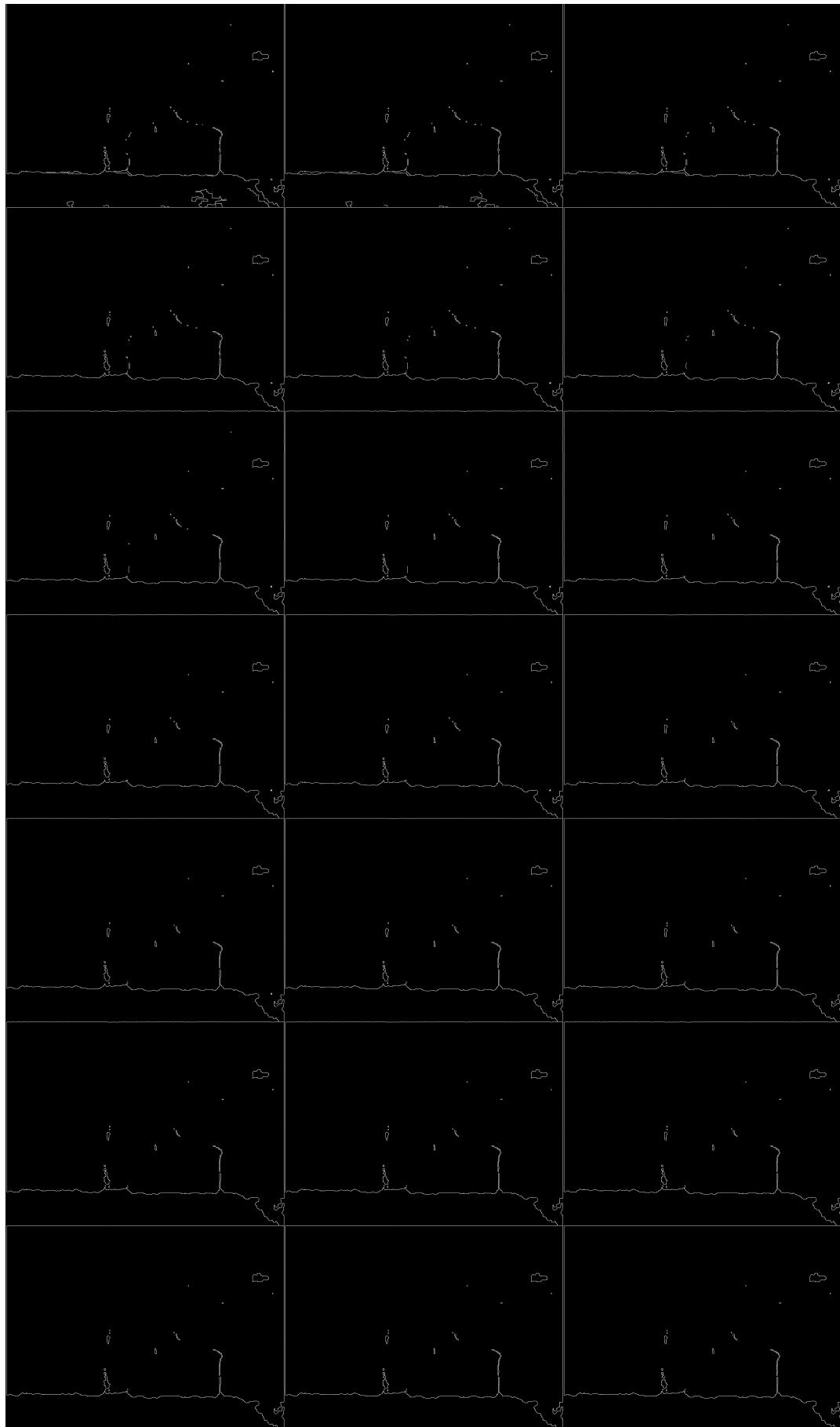


Figure 84: bilateral_11_canny_110_330

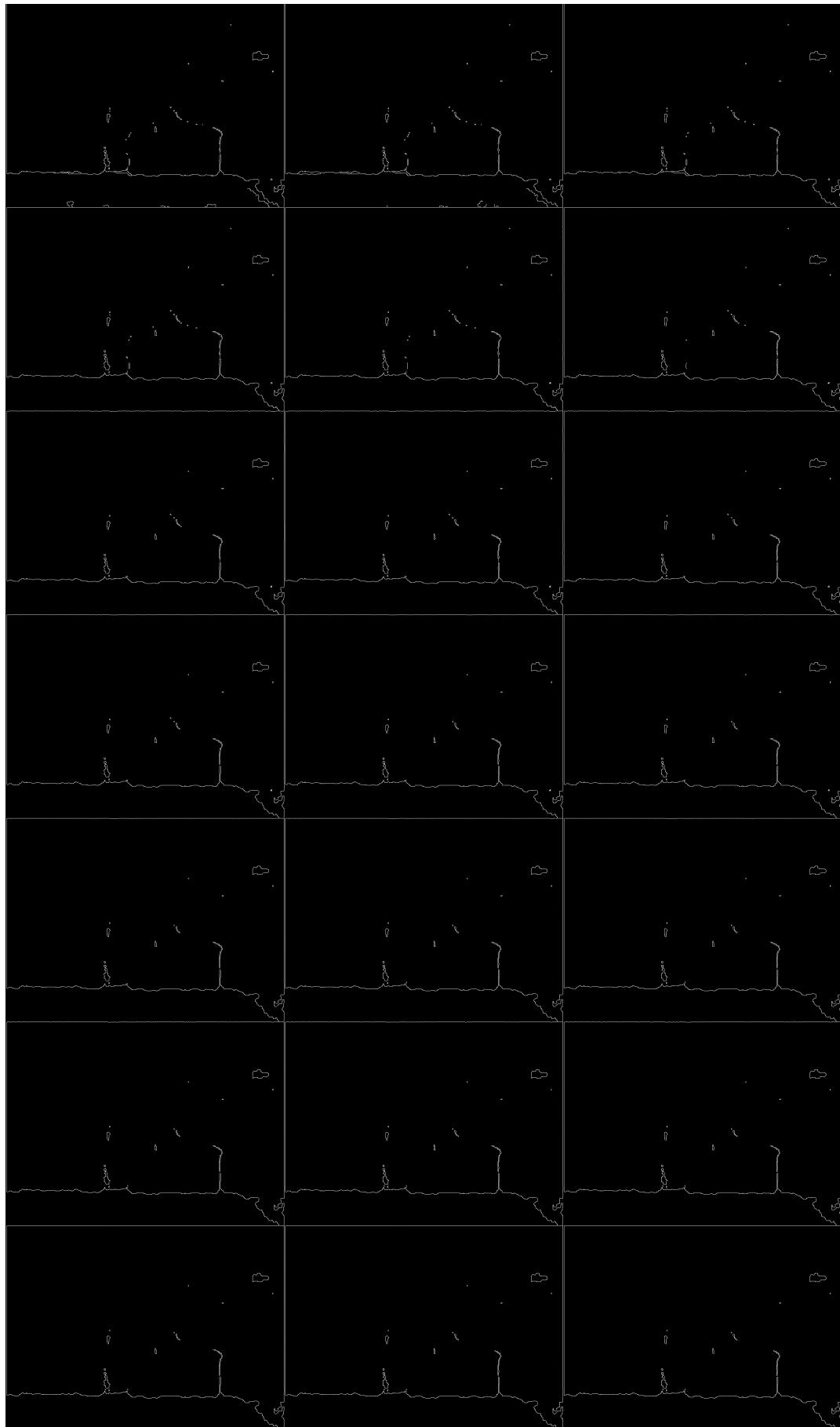


Figure 85: bilateral_12_canny_120_360

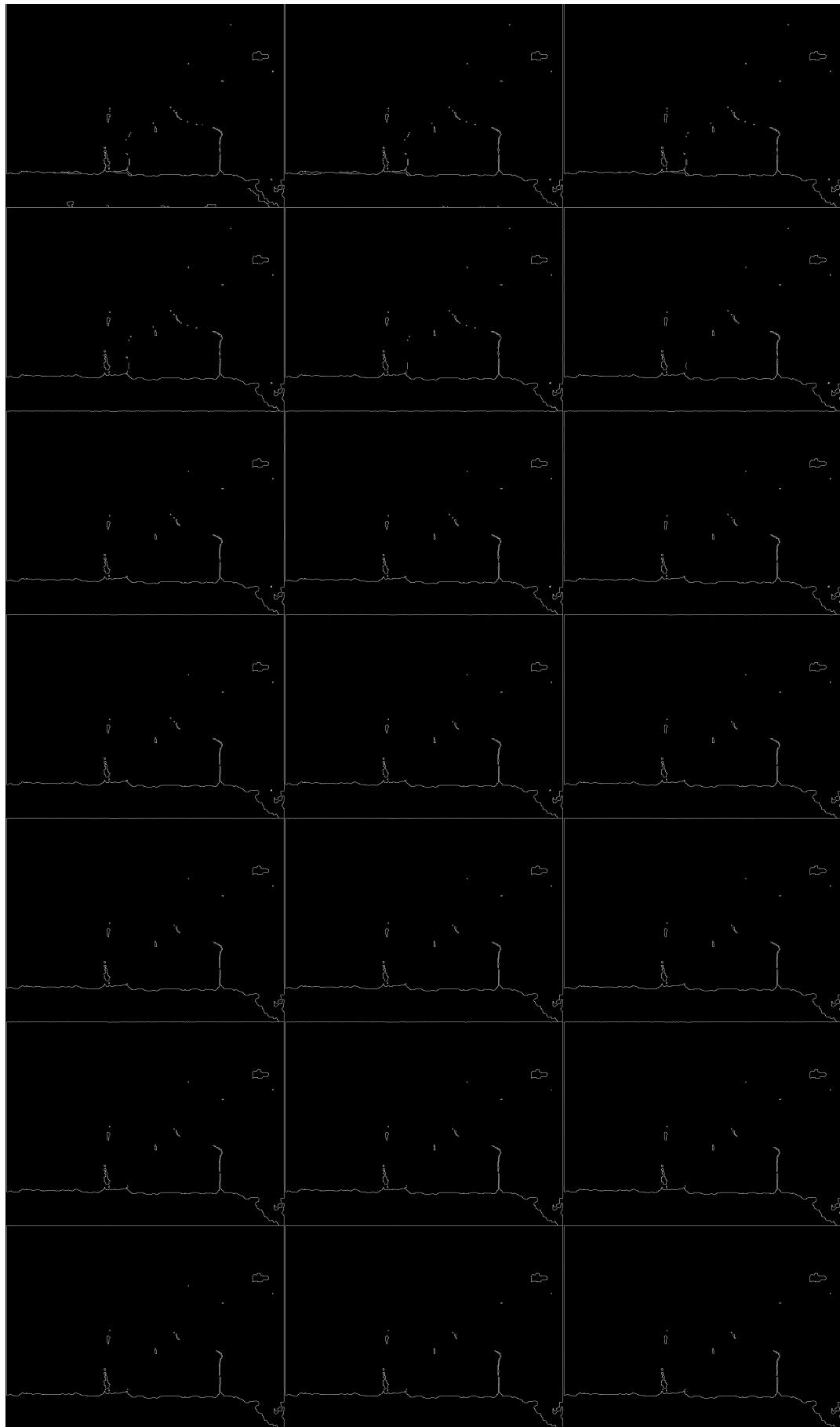


Figure 86: bilateral_13_canny_130_390

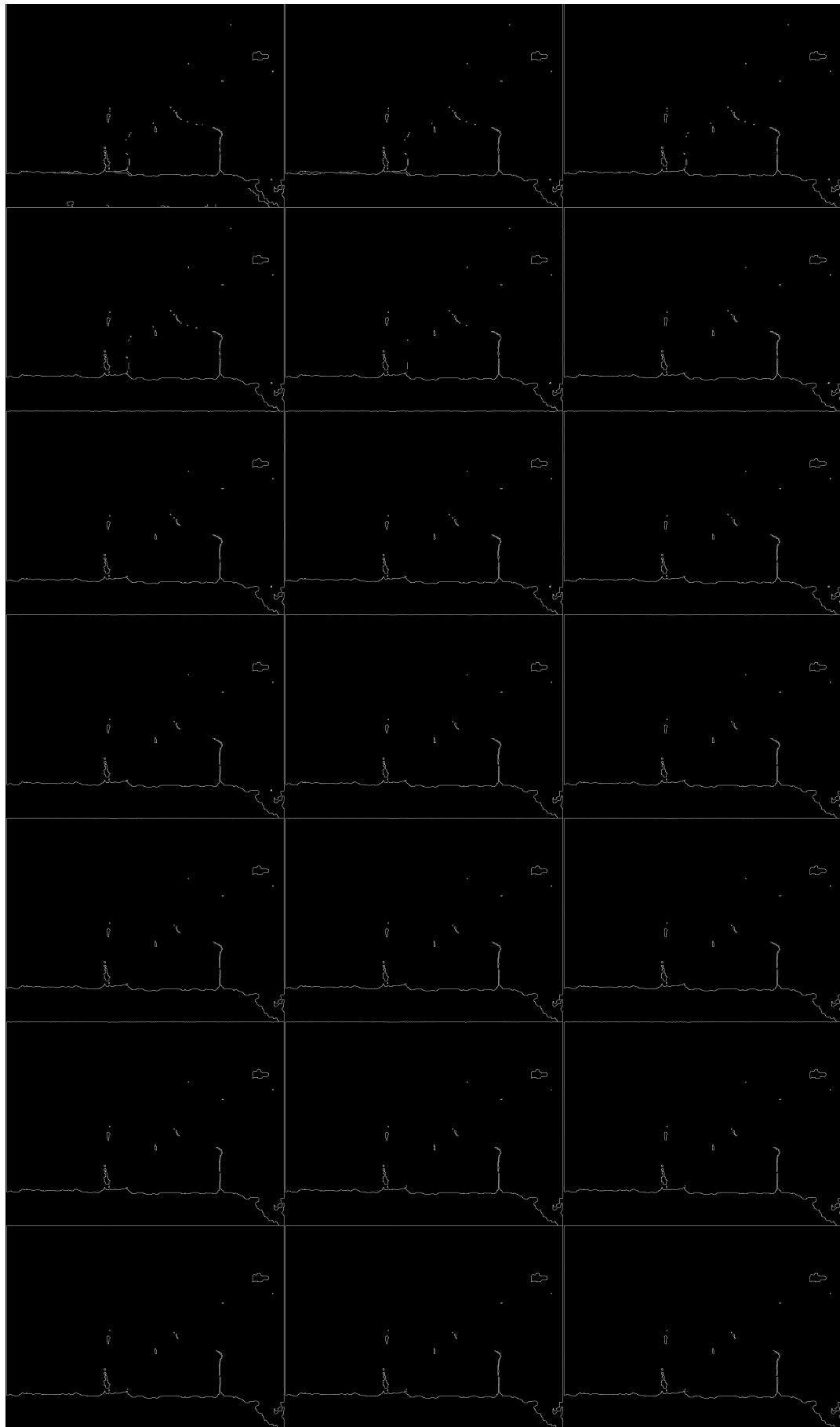


Figure 87: bilateral_14_canny_140_420

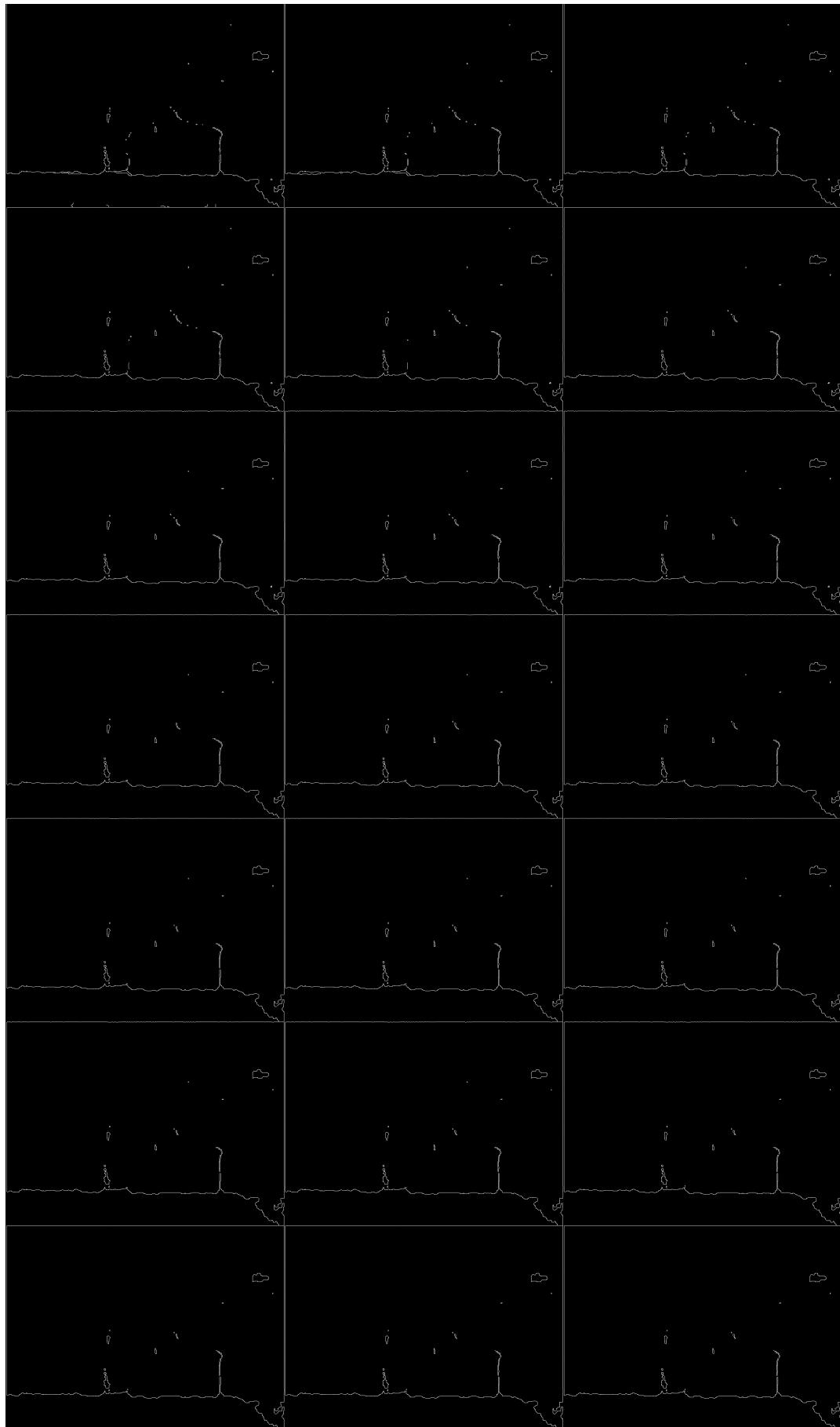


Figure 88: bilateral_15_canny_150_450

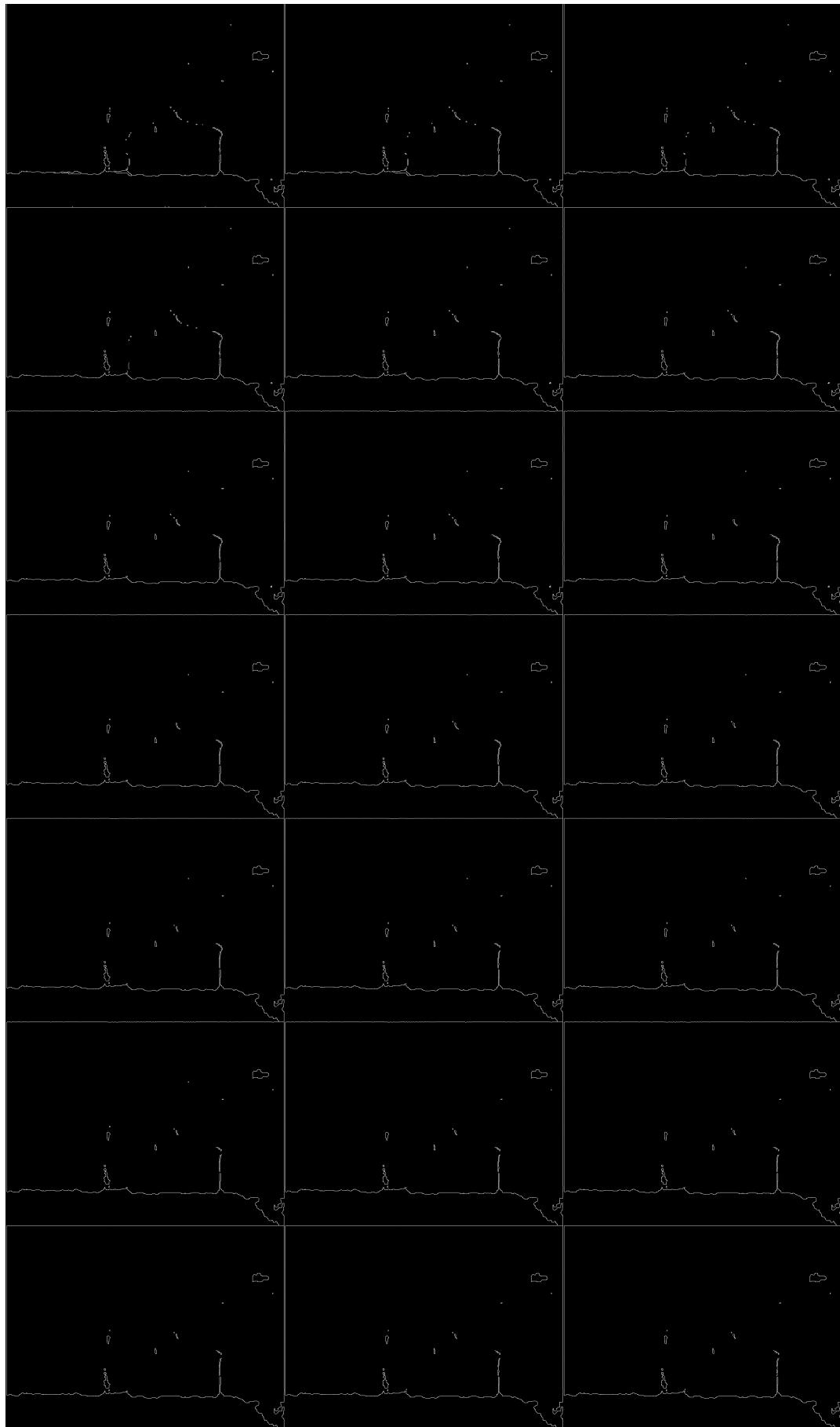


Figure 89: bilateral_16_canny_160_480

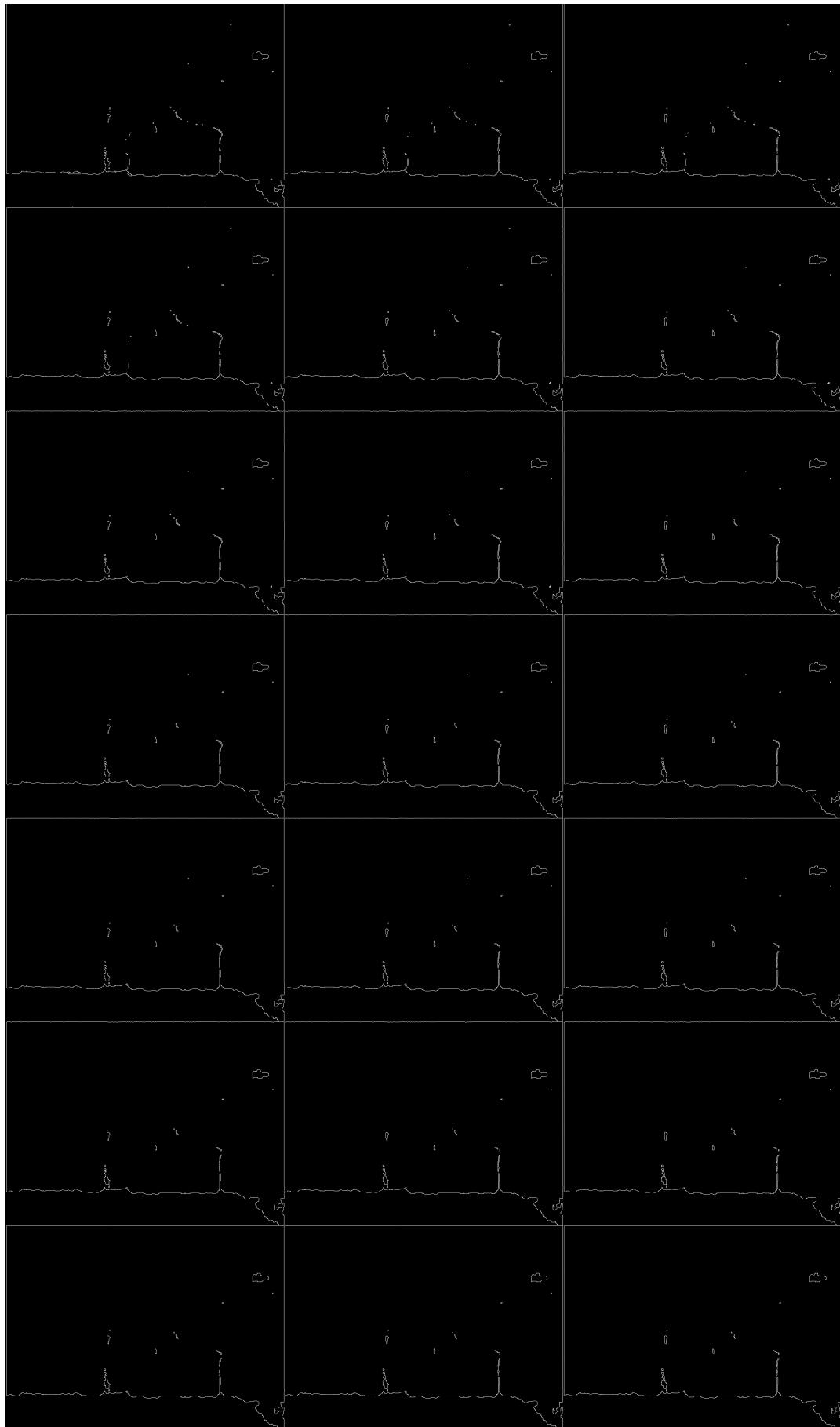


Figure 90: bilateral_17_canny_170_510

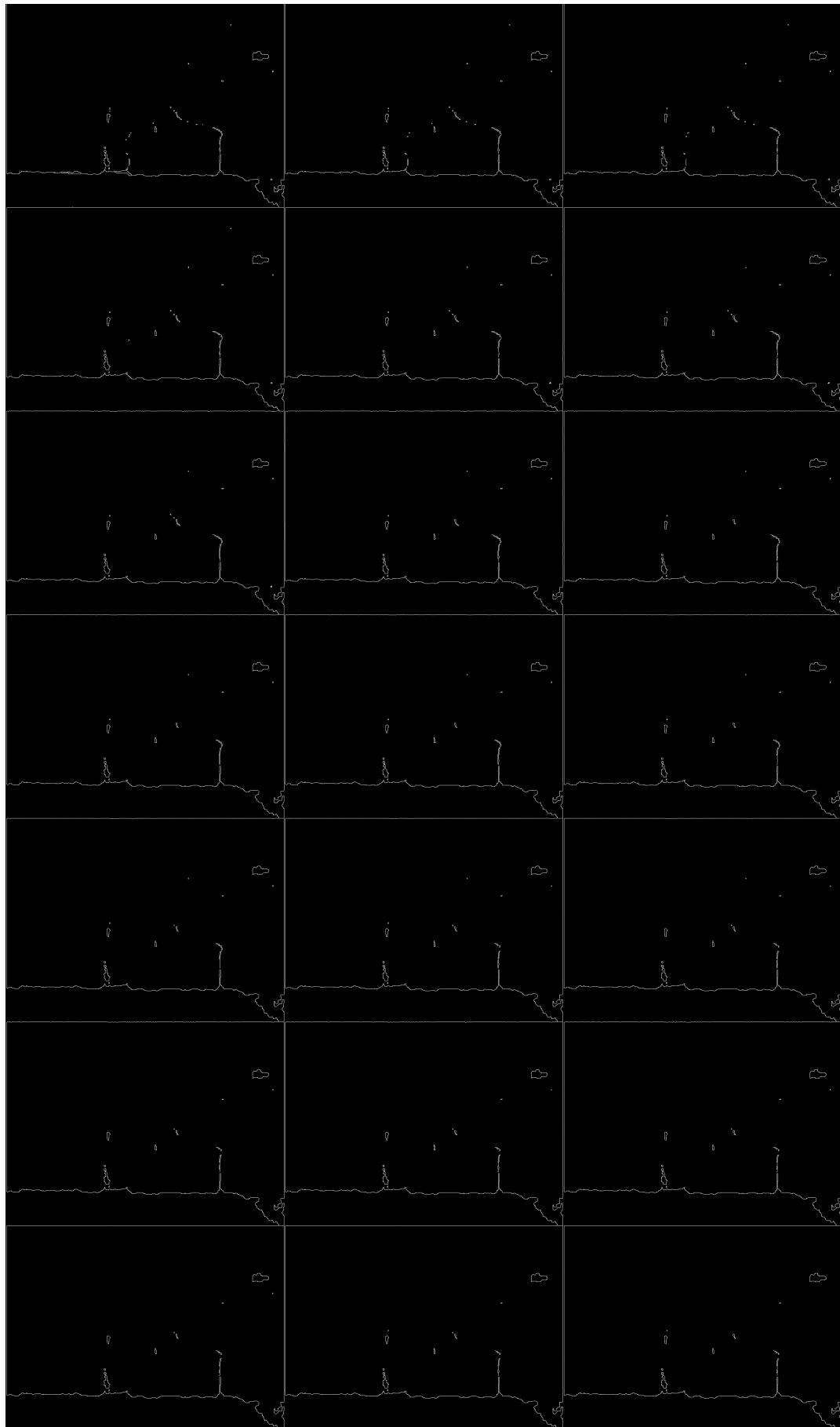


Figure 91: bilateral_18_canny_180_540

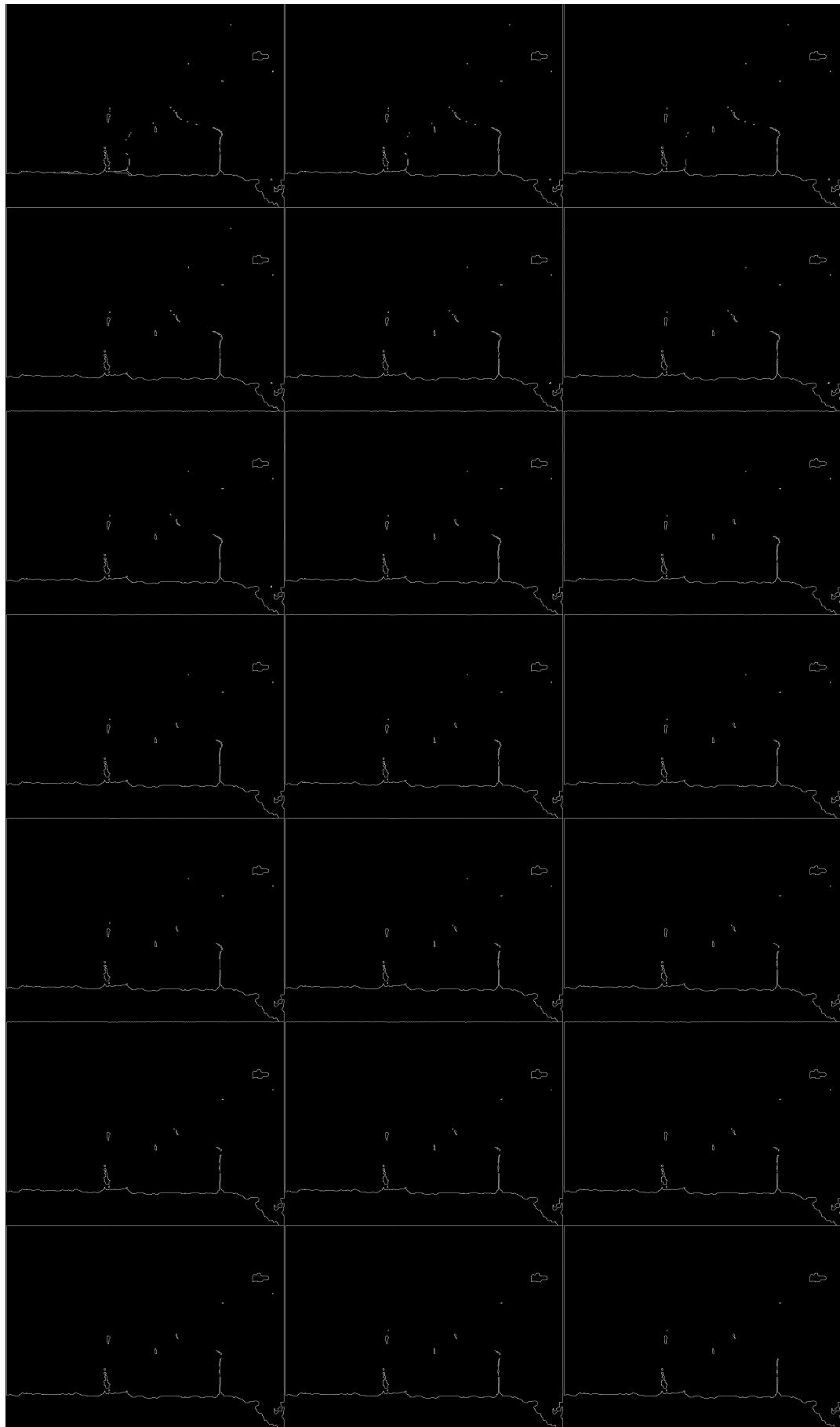


Figure 92: bilateral_19_canny_190_570

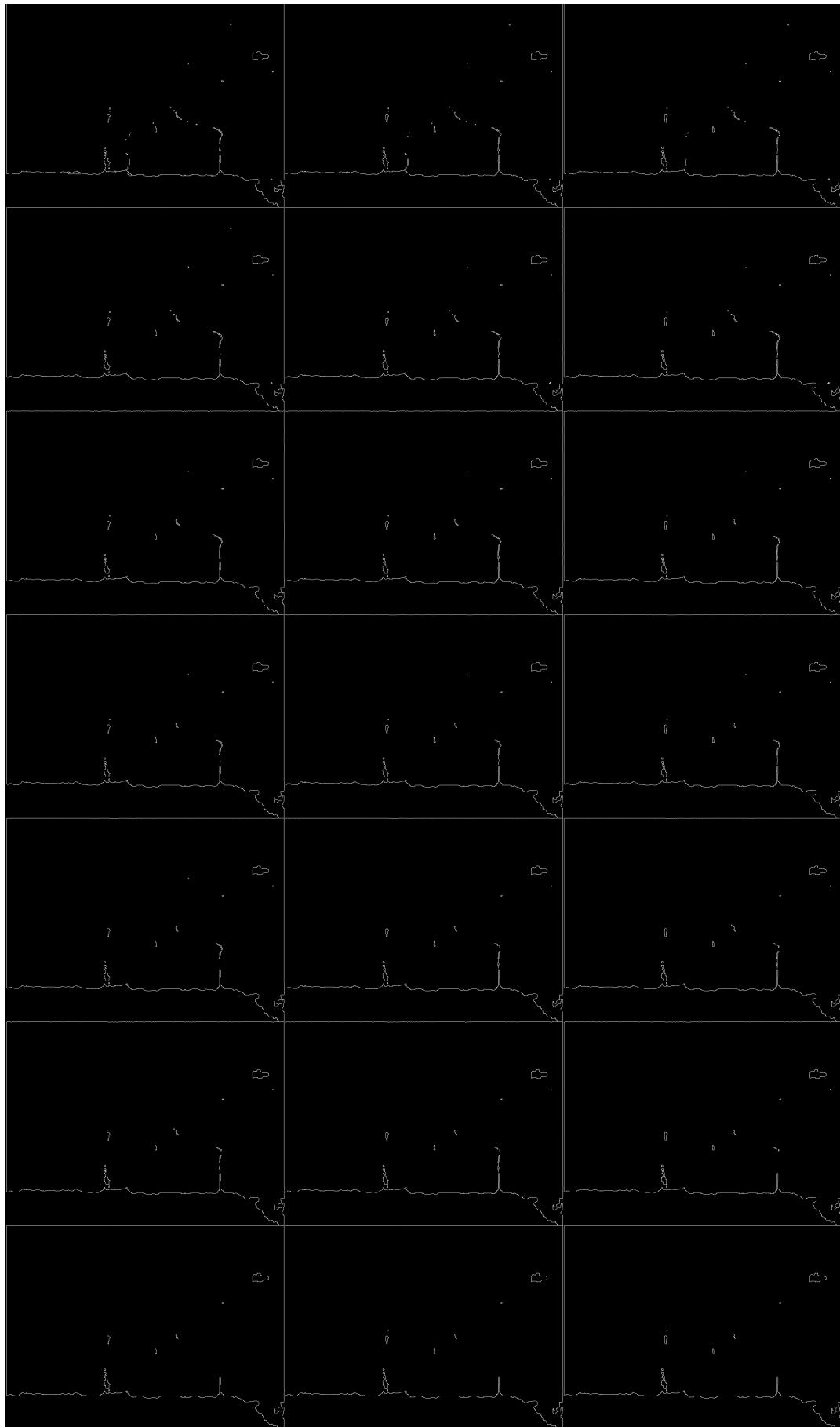


Figure 93: bilateral_20_canny_200_600

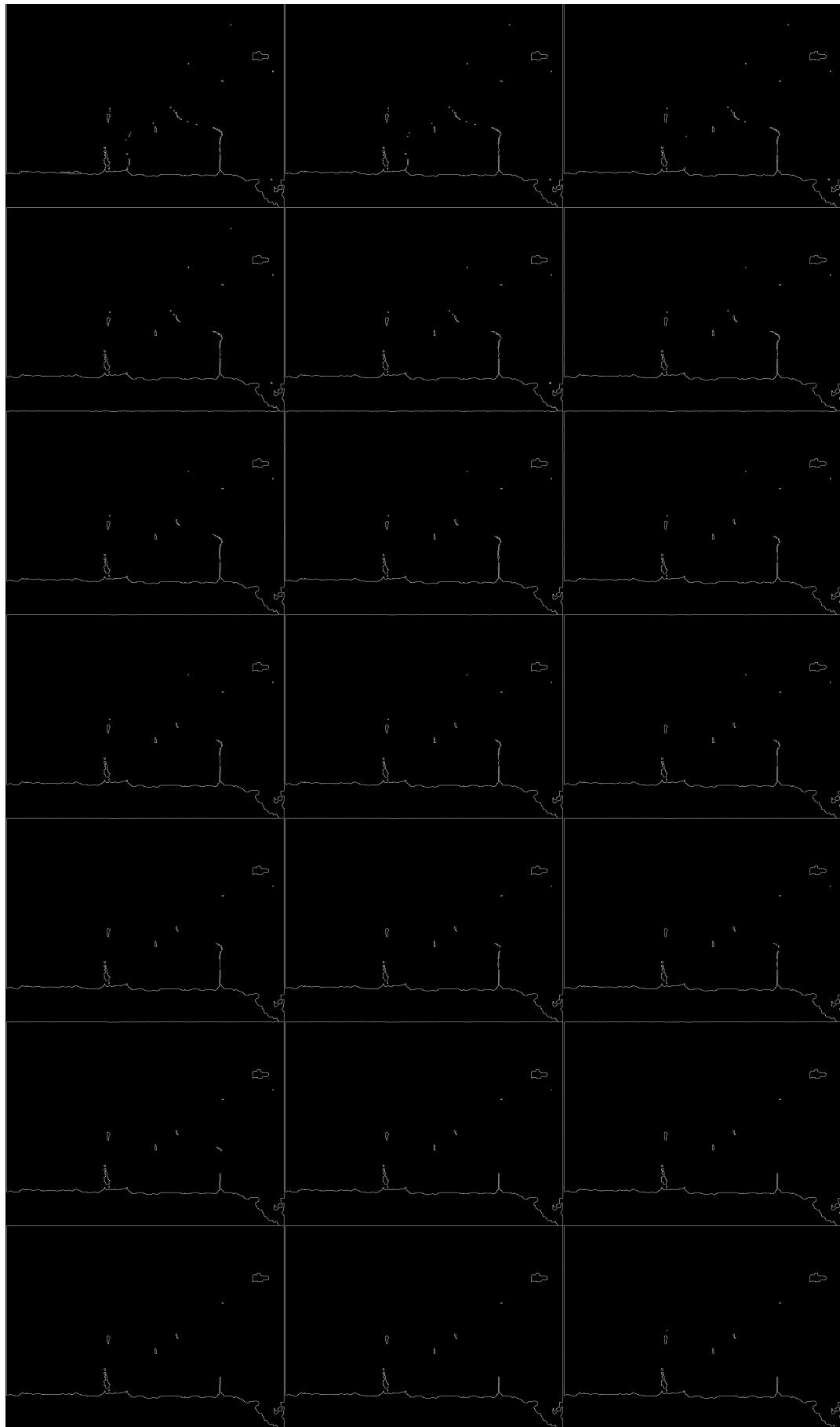


Figure 94: bilateral_21_canny_210_630