Depth Image Processing Journal

Experiments with Python and OpenCV

Siddharth Patel
HTW Berlin

August 25, 2025

Contents

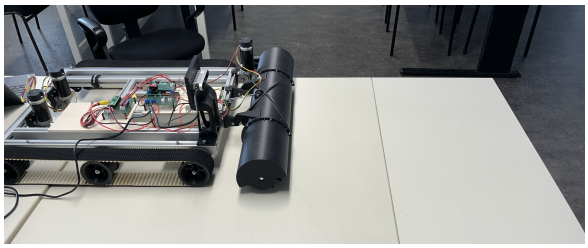
1 Introduction

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:

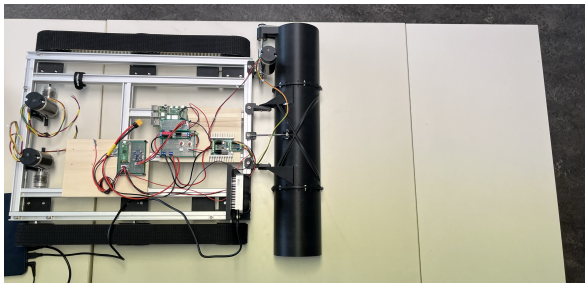
- The Python code used
- Sample output frame(s) for comparison
- Observations about filter performance

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

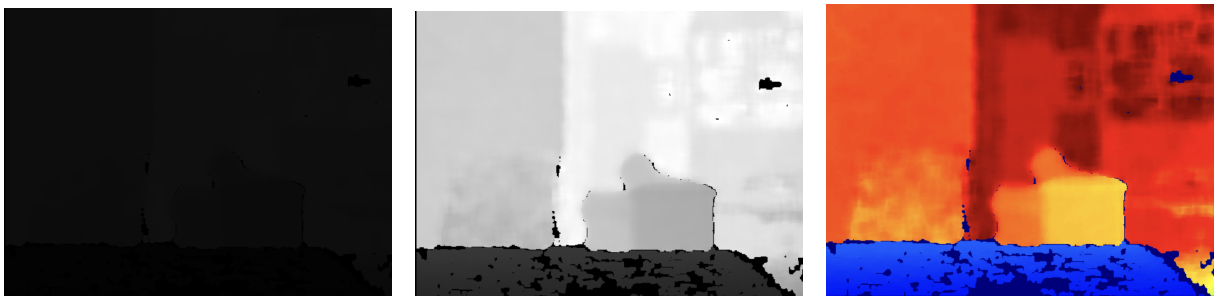
cv_bridge tutorial (Python)

3.1 Code

```python
#!/usr/bin/env python3

import rospy
from sensor_msgs.msg import Image
from cv_bridge import CvBridge
import cv2

class FrameGrabber:
    def __init__(self):
        self.bridge = CvBridge()
        self.depth_image_original = None
```

```
        rospy.Subscriber("/ascamera/depth0/image_raw", Image, self.depth_callback)


    def depth_callback(self, msg):
        self.depth_image_original = self.bridge.imgmsg_to_cv2(msg, desired_encoding="passthrough")

    def show_loop(self):
        rate = rospy.Rate(30)
        while not rospy.is_shutdown():

            if self.depth_image_original is not None:
                depth_gray = cv2.normalize(self.depth_image_original, None, 0, 255, cv2.NORM_MINMAX)
                depth_gray = cv2.convertScaleAbs(depth_gray)
                depth_colored = cv2.applyColorMap(depth_gray, cv2.COLORMAP_JET)
                cv2.imshow("Depth Original", self.depth_image_original)
                cv2.imshow("Depth Gray", depth_gray)
                cv2.imshow("Depth Colored", depth_colored)

            key = cv2.waitKey(1) & 0xFF
            if key == ord('1'):
                print("Key '1' pressed. Exiting...")
                break

            rate.sleep()

        cv2.destroyAllWindows()

if __name__ == "__main__":
    rospy.init_node("rgb_depth_filters_viewer")
    grabber = FrameGrabber()
    grabber.show_loop()
```

3.2 Output Frames



(a) Depth Original (b) Depth Gray (c) Depth Colored

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

```python
import cv2
import numpy as np


depth_gray = cv2.imread("images/depth_gray.png", cv2.IMREAD_GRAYSCALE)

depth_colored = cv2.applyColorMap(depth_gray, cv2.COLORMAP_JET)

# grayscale sequence
g1 = depth_gray
g2 = cv2.blur(depth_gray, (3,3))
g3 = cv2.blur(depth_gray, (5,5))
g4 = cv2.blur(depth_gray, (7,7))
g5 = cv2.blur(depth_gray, (9,9))
g6 = cv2.blur(depth_gray, (11,11))
g7 = cv2.blur(depth_gray, (13,13))
g8 = cv2.blur(depth_gray, (15,15))
g9 = cv2.blur(depth_gray, (17,17))
g10 = cv2.blur(depth_gray, (19,19))
g11 = cv2.blur(depth_gray, (21,21))
g12 = cv2.blur(depth_gray, (23,23))
g13 = cv2.blur(depth_gray, (25,25))
g14 = cv2.blur(depth_gray, (27,27))
g15 = cv2.blur(depth_gray, (29,29))
g16 = cv2.blur(depth_gray, (31,31))
g17 = cv2.blur(depth_gray, (33,33))
g18 = cv2.blur(depth_gray, (35,35))
g19 = cv2.blur(depth_gray, (37,37))
g20 = cv2.blur(depth_gray, (39,39))
g21 = cv2.blur(depth_gray, (41,41))


row1 = np.hstack([g1, g2, g3])
row2 = np.hstack([g4, g5, g6])
row3 = np.hstack([g7, g8, g9])
row4 = np.hstack([g10, g11, g12])
row5 = np.hstack([g13, g14, g15])
row6 = np.hstack([g16, g17, g18])
row7 = np.hstack([g19, g20, g21])

grid_gray = np.vstack([row1, row2, row3, row4, row5, row6, row7])

# colored sequence
c1 = depth_colored
c2 = cv2.blur(depth_colored, (3,3))
c3 = cv2.blur(depth_colored, (5,5))
c4 = cv2.blur(depth_colored, (7,7))
c5 = cv2.blur(depth_colored, (9,9))
c6 = cv2.blur(depth_colored, (11,11))
c7 = cv2.blur(depth_colored, (13,13))
c8 = cv2.blur(depth_colored, (15,15))
c9 = cv2.blur(depth_colored, (17,17))
c10 = cv2.blur(depth_colored, (19,19))
c11 = cv2.blur(depth_colored, (21,21))
```

```
c12 = cv2.blur(depth_colored, (23,23))
c13 = cv2.blur(depth_colored, (25,25))
c14 = cv2.blur(depth_colored, (27,27))
c15 = cv2.blur(depth_colored, (29,29))
c16 = cv2.blur(depth_colored, (31,31))
c17 = cv2.blur(depth_colored, (33,33))
c18 = cv2.blur(depth_colored, (35,35))
c19 = cv2.blur(depth_colored, (37,37))
c20 = cv2.blur(depth_colored, (39,39))
c21 = cv2.blur(depth_colored, (41,41))

row1c = np.hstack([c1, c2, c3])
row2c = np.hstack([c4, c5, c6])
row3c = np.hstack([c7, c8, c9])
row4c = np.hstack([c10, c11, c12])
row5c = np.hstack([c13, c14, c15])
row6c = np.hstack([c16, c17, c18])
row7c = np.hstack([c19, c20, c21])

grid_colored = np.vstack([row1c, row2c, row3c, row4c, row5c, row6c, row7c])


# cv2.imshow("Gray Grid", grid_gray)
# cv2.imshow("Colored Grid", grid_colored)

# while True:
#     key = cv2.waitKey(1) & 0xFF
#     if key in (ord('1'), 27):  # press '1' or ESC
#         break

# cv2.destroyAllWindows()

cv2.imwrite("images/blur/04_average_gray_grid.png", grid_gray)
cv2.imwrite("images/blur/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)

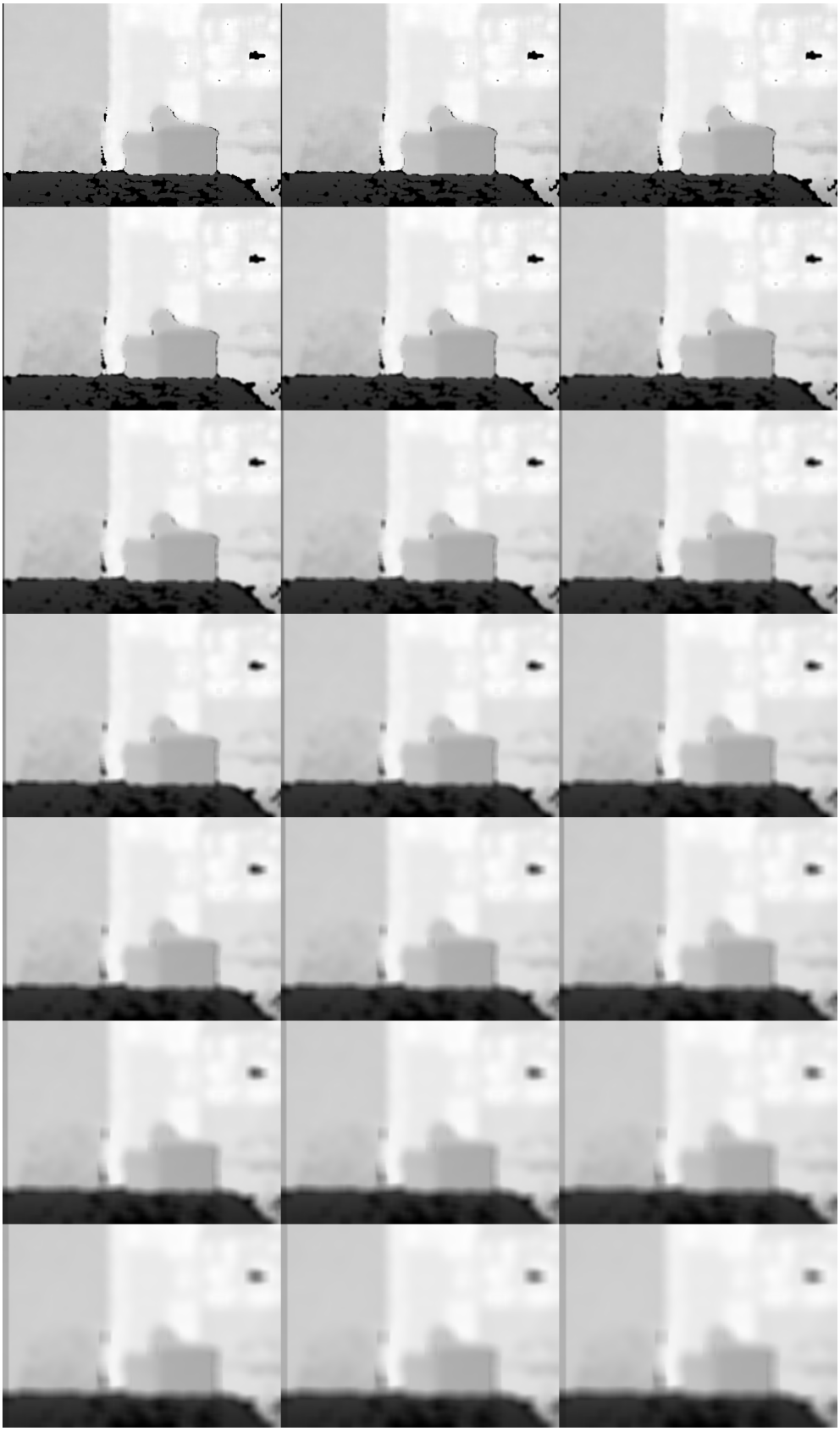


Figure 3: Averaging — Gray grid result

Output (Colored Grid)

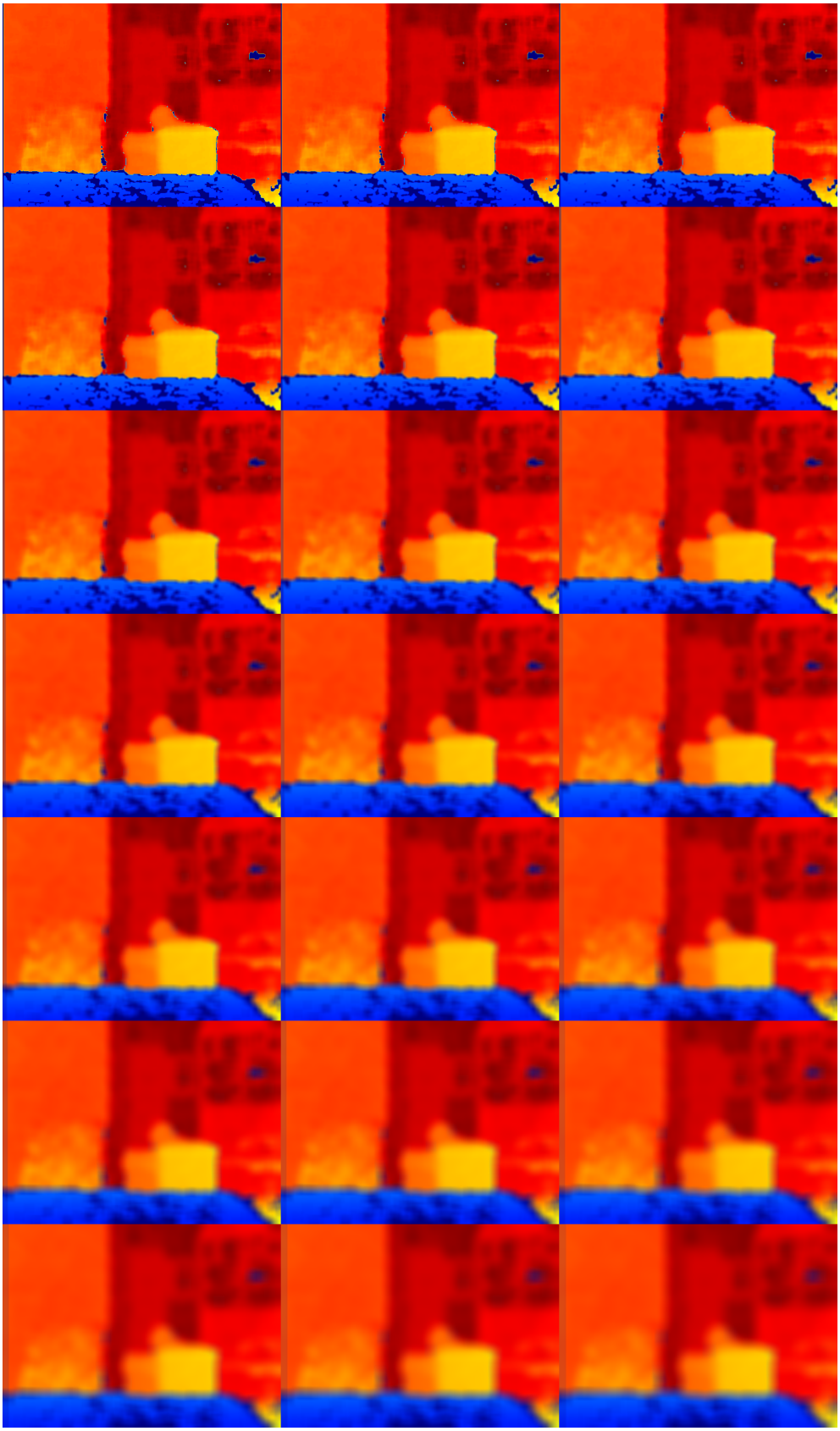


Figure 4: Averaging — Colored grid result

4.2 Gaussian Blurring

Code

```python
import cv2
import numpy as np


depth_gray = cv2.imread("images/depth_gray.png", cv2.IMREAD_GRAYSCALE)

depth_colored = cv2.applyColorMap(depth_gray, cv2.COLORMAP_JET)

# grayscale sequence
g1  = depth_gray
g2  = cv2.GaussianBlur(depth_gray, (3,3), 0)
g3  = cv2.GaussianBlur(depth_gray, (5,5), 0)
g4  = cv2.GaussianBlur(depth_gray, (7,7), 0)
g5  = cv2.GaussianBlur(depth_gray, (9,9), 0)
g6  = cv2.GaussianBlur(depth_gray, (11,11), 0)
g7  = cv2.GaussianBlur(depth_gray, (13,13), 0)
g8  = cv2.GaussianBlur(depth_gray, (15,15), 0)
g9  = cv2.GaussianBlur(depth_gray, (17,17), 0)
g10 = cv2.GaussianBlur(depth_gray, (19,19), 0)
g11 = cv2.GaussianBlur(depth_gray, (21,21), 0)
g12 = cv2.GaussianBlur(depth_gray, (23,23), 0)
g13 = cv2.GaussianBlur(depth_gray, (25,25), 0)
g14 = cv2.GaussianBlur(depth_gray, (27,27), 0)
g15 = cv2.GaussianBlur(depth_gray, (29,29), 0)
g16 = cv2.GaussianBlur(depth_gray, (31,31), 0)
g17 = cv2.GaussianBlur(depth_gray, (33,33), 0)
g18 = cv2.GaussianBlur(depth_gray, (35,35), 0)
g19 = cv2.GaussianBlur(depth_gray, (37,37), 0)
g20 = cv2.GaussianBlur(depth_gray, (39,39), 0)
g21 = cv2.GaussianBlur(depth_gray, (41,41), 0)


# build 7×3 grid
row1 = np.hstack([g1, g2, g3])
row2 = np.hstack([g4, g5, g6])
row3 = np.hstack([g7, g8, g9])
row4 = np.hstack([g10, g11, g12])
row5 = np.hstack([g13, g14, g15])
row6 = np.hstack([g16, g17, g18])
row7 = np.hstack([g19, g20, g21])

grid_gray = np.vstack([row1, row2, row3, row4, row5, row6, row7])

# colored sequence
c1  = depth_colored
c2  = cv2.GaussianBlur(depth_colored, (3,3), 0)
c3  = cv2.GaussianBlur(depth_colored, (5,5), 0)
c4  = cv2.GaussianBlur(depth_colored, (7,7), 0)
c5  = cv2.GaussianBlur(depth_colored, (9,9), 0)
c6  = cv2.GaussianBlur(depth_colored, (11,11), 0)
c7  = cv2.GaussianBlur(depth_colored, (13,13), 0)
c8  = cv2.GaussianBlur(depth_colored, (15,15), 0)
c9  = cv2.GaussianBlur(depth_colored, (17,17), 0)
c10 = cv2.GaussianBlur(depth_colored, (19,19), 0)
c11 = cv2.GaussianBlur(depth_colored, (21,21), 0)
c12 = cv2.GaussianBlur(depth_colored, (23,23), 0)
c13 = cv2.GaussianBlur(depth_colored, (25,25), 0)
c14 = cv2.GaussianBlur(depth_colored, (27,27), 0)
```

```
c15 = cv2.GaussianBlur(depth_colored, (29,29), 0)
c16 = cv2.GaussianBlur(depth_colored, (31,31), 0)
c17 = cv2.GaussianBlur(depth_colored, (33,33), 0)
c18 = cv2.GaussianBlur(depth_colored, (35,35), 0)
c19 = cv2.GaussianBlur(depth_colored, (37,37), 0)
c20 = cv2.GaussianBlur(depth_colored, (39,39), 0)
c21 = cv2.GaussianBlur(depth_colored, (41,41), 0)

row1c = np.hstack([c1, c2, c3])
row2c = np.hstack([c4, c5, c6])
row3c = np.hstack([c7, c8, c9])
row4c = np.hstack([c10, c11, c12])
row5c = np.hstack([c13, c14, c15])
row6c = np.hstack([c16, c17, c18])
row7c = np.hstack([c19, c20, c21])

grid_colored = np.vstack([row1c, row2c, row3c, row4c, row5c, row6c, row7c])

# cv2.imshow("Gray Grid", grid_gray)
# cv2.imshow("Colored Grid", grid_colored)

# while True:
#     key = cv2.waitKey(1) & 0xFF
#     if key in (ord('1'), 27):  # press '1' or ESC
#         break

# cv2.destroyAllWindows()

cv2.imwrite("images/blur/05_gaussian_gray_grid.png", grid_gray)
cv2.imwrite("images/blur/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)

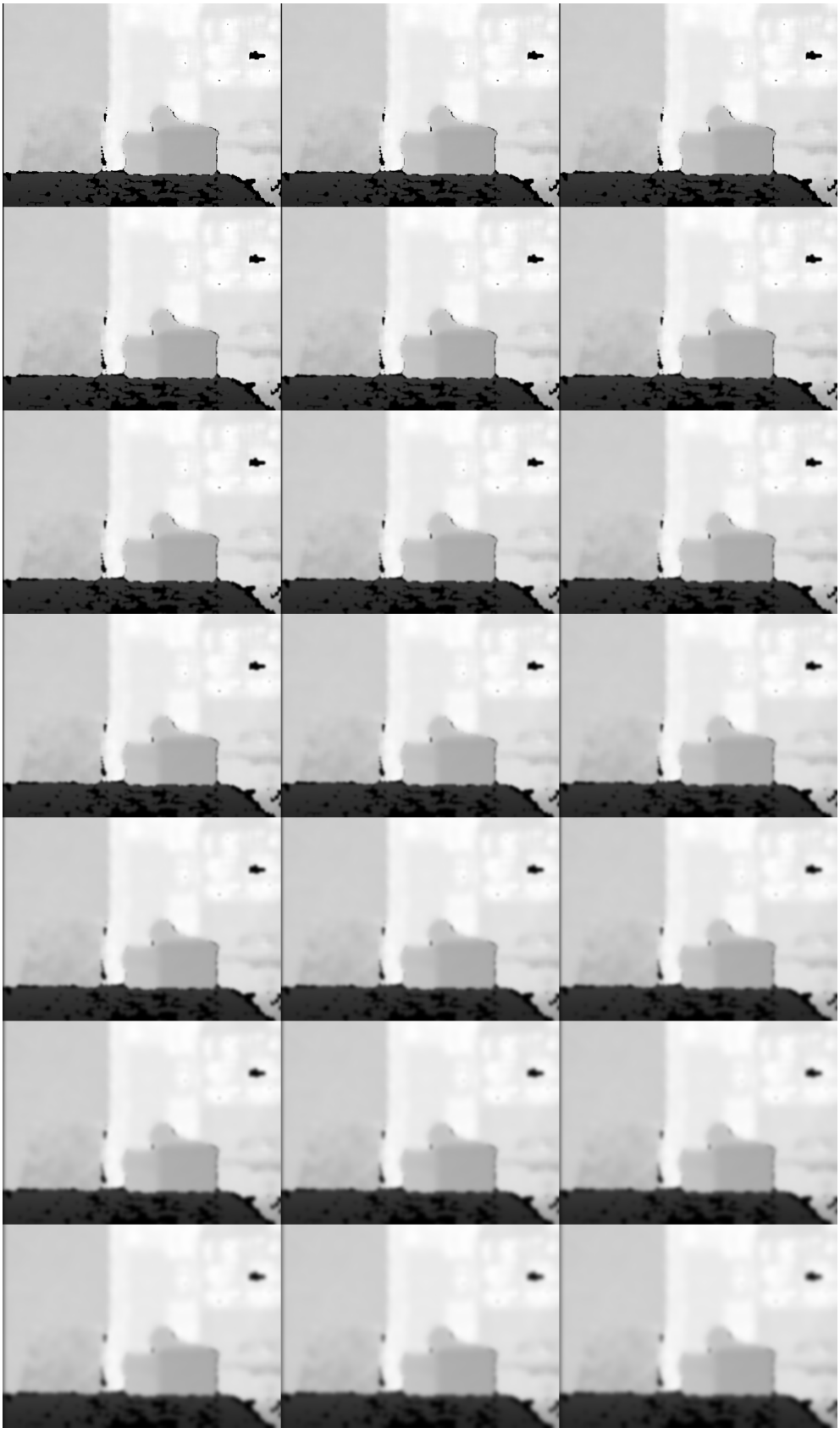


Figure 5: Gaussian blurring — Gray grid result

Output (Colored Grid)

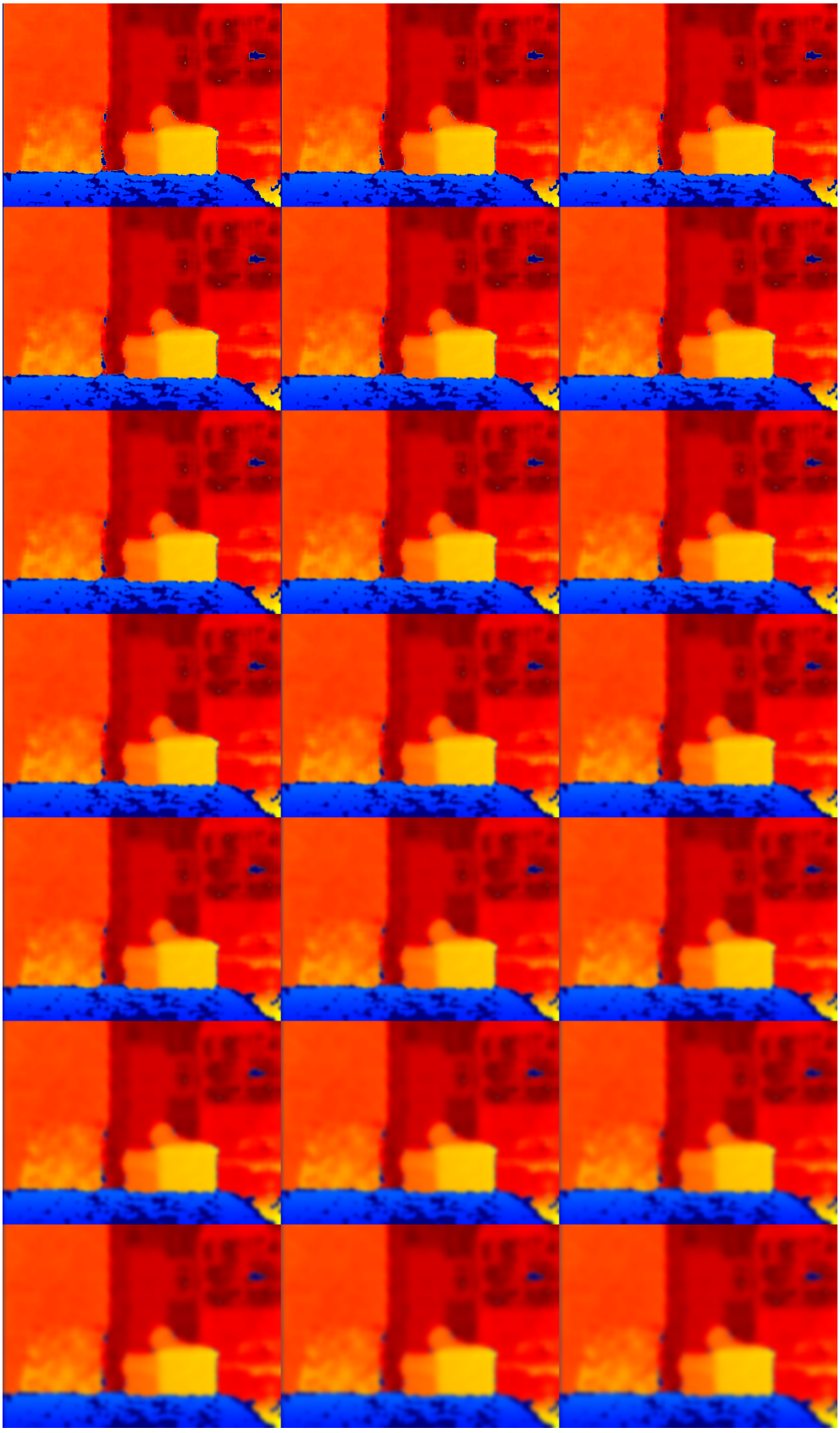


Figure 6: Gaussian blurring — Colored grid result

4.3 Median Blurring

Code

```python
import cv2
import numpy as np


depth_gray = cv2.imread("images/depth_gray.png", cv2.IMREAD_GRAYSCALE)

depth_colored = cv2.applyColorMap(depth_gray, cv2.COLORMAP_JET)

# grayscale sequence
g1  = depth_gray
g2  = cv2.medianBlur(depth_gray, 3)
g3  = cv2.medianBlur(depth_gray, 5)
g4  = cv2.medianBlur(depth_gray, 7)
g5  = cv2.medianBlur(depth_gray, 9)
g6  = cv2.medianBlur(depth_gray, 11)
g7  = cv2.medianBlur(depth_gray, 13)
g8  = cv2.medianBlur(depth_gray, 15)
g9  = cv2.medianBlur(depth_gray, 17)
g10 = cv2.medianBlur(depth_gray, 19)
g11 = cv2.medianBlur(depth_gray, 21)
g12 = cv2.medianBlur(depth_gray, 23)
g13 = cv2.medianBlur(depth_gray, 25)
g14 = cv2.medianBlur(depth_gray, 27)
g15 = cv2.medianBlur(depth_gray, 29)
g16 = cv2.medianBlur(depth_gray, 31)
g17 = cv2.medianBlur(depth_gray, 33)
g18 = cv2.medianBlur(depth_gray, 35)
g19 = cv2.medianBlur(depth_gray, 37)
g20 = cv2.medianBlur(depth_gray, 39)
g21 = cv2.medianBlur(depth_gray, 41)



# build 7×3 grid
row1 = np.hstack([g1, g2, g3])
row2 = np.hstack([g4, g5, g6])
row3 = np.hstack([g7, g8, g9])
row4 = np.hstack([g10, g11, g12])
row5 = np.hstack([g13, g14, g15])
row6 = np.hstack([g16, g17, g18])
row7 = np.hstack([g19, g20, g21])

grid_gray = np.vstack([row1, row2, row3, row4, row5, row6, row7])

# colored sequence
c1  = depth_colored
c2  = cv2.medianBlur(depth_colored, 3)
c3  = cv2.medianBlur(depth_colored, 5)
c4  = cv2.medianBlur(depth_colored, 7)
c5  = cv2.medianBlur(depth_colored, 9)
c6  = cv2.medianBlur(depth_colored, 11)
c7  = cv2.medianBlur(depth_colored, 13)
c8  = cv2.medianBlur(depth_colored, 15)
c9  = cv2.medianBlur(depth_colored, 17)
c10 = cv2.medianBlur(depth_colored, 19)
c11 = cv2.medianBlur(depth_colored, 21)
c12 = cv2.medianBlur(depth_colored, 23)
c13 = cv2.medianBlur(depth_colored, 25)
```

```
c14 = cv2.medianBlur(depth_colored, 27)
c15 = cv2.medianBlur(depth_colored, 29)
c16 = cv2.medianBlur(depth_colored, 31)
c17 = cv2.medianBlur(depth_colored, 33)
c18 = cv2.medianBlur(depth_colored, 35)
c19 = cv2.medianBlur(depth_colored, 37)
c20 = cv2.medianBlur(depth_colored, 39)
c21 = cv2.medianBlur(depth_colored, 41)

row1c = np.hstack([c1, c2, c3])
row2c = np.hstack([c4, c5, c6])
row3c = np.hstack([c7, c8, c9])
row4c = np.hstack([c10, c11, c12])
row5c = np.hstack([c13, c14, c15])
row6c = np.hstack([c16, c17, c18])
row7c = np.hstack([c19, c20, c21])

grid_colored = np.vstack([row1c, row2c, row3c, row4c, row5c, row6c, row7c])

# cv2.imshow("Gray Grid", grid_gray)
# cv2.imshow("Colored Grid", grid_colored)

# while True:
#     key = cv2.waitKey(1) & 0xFF
#     if key in (ord('1'), 27):  # press '1' or ESC
#         break

# cv2.destroyAllWindows()

# save each grid separately
cv2.imwrite("images/blur/06_median_gray_grid.png", grid_gray)
cv2.imwrite("images/blur/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)

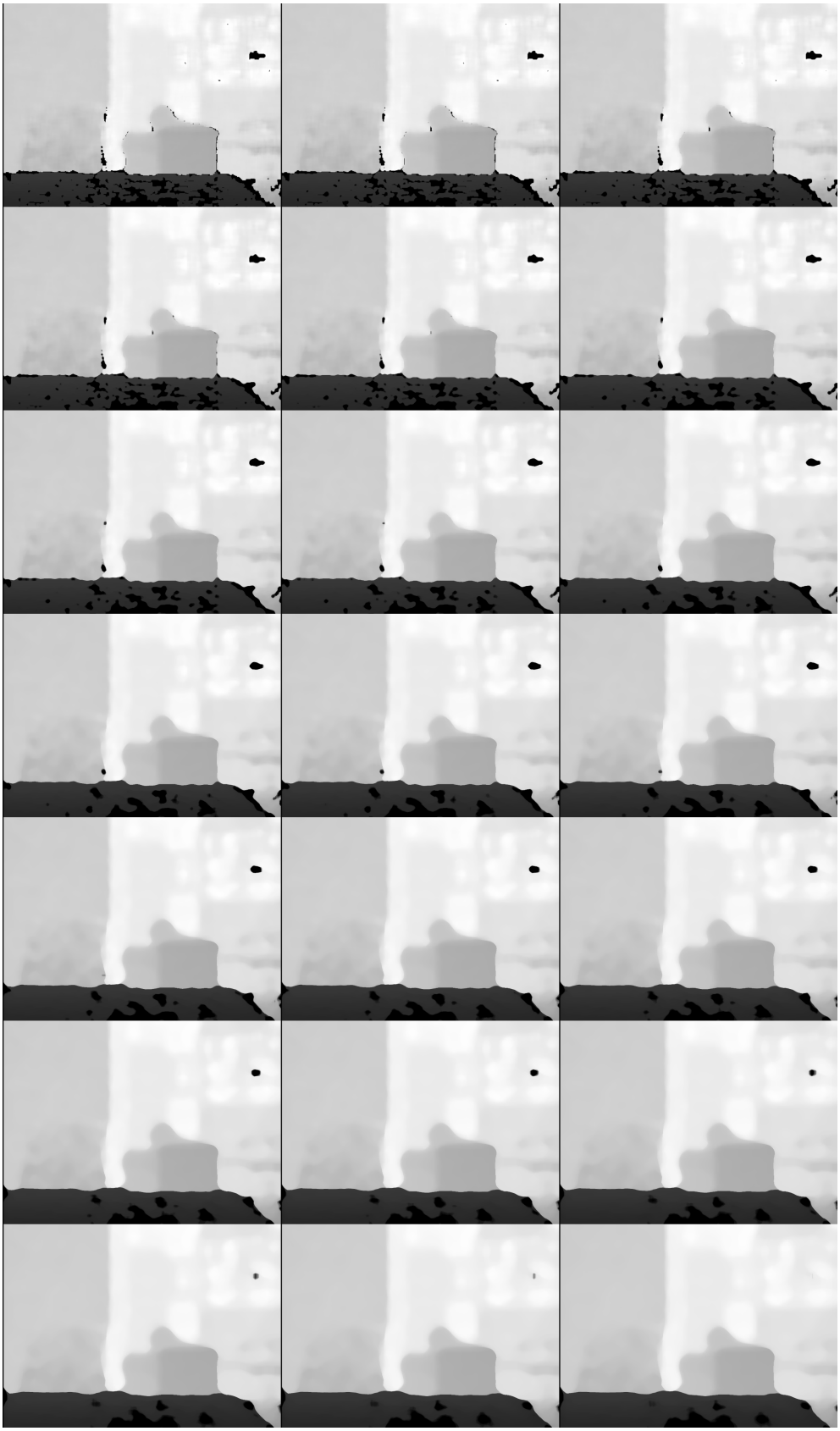


Figure 7: Median blurring — Gray grid result

Output (Colored Grid)

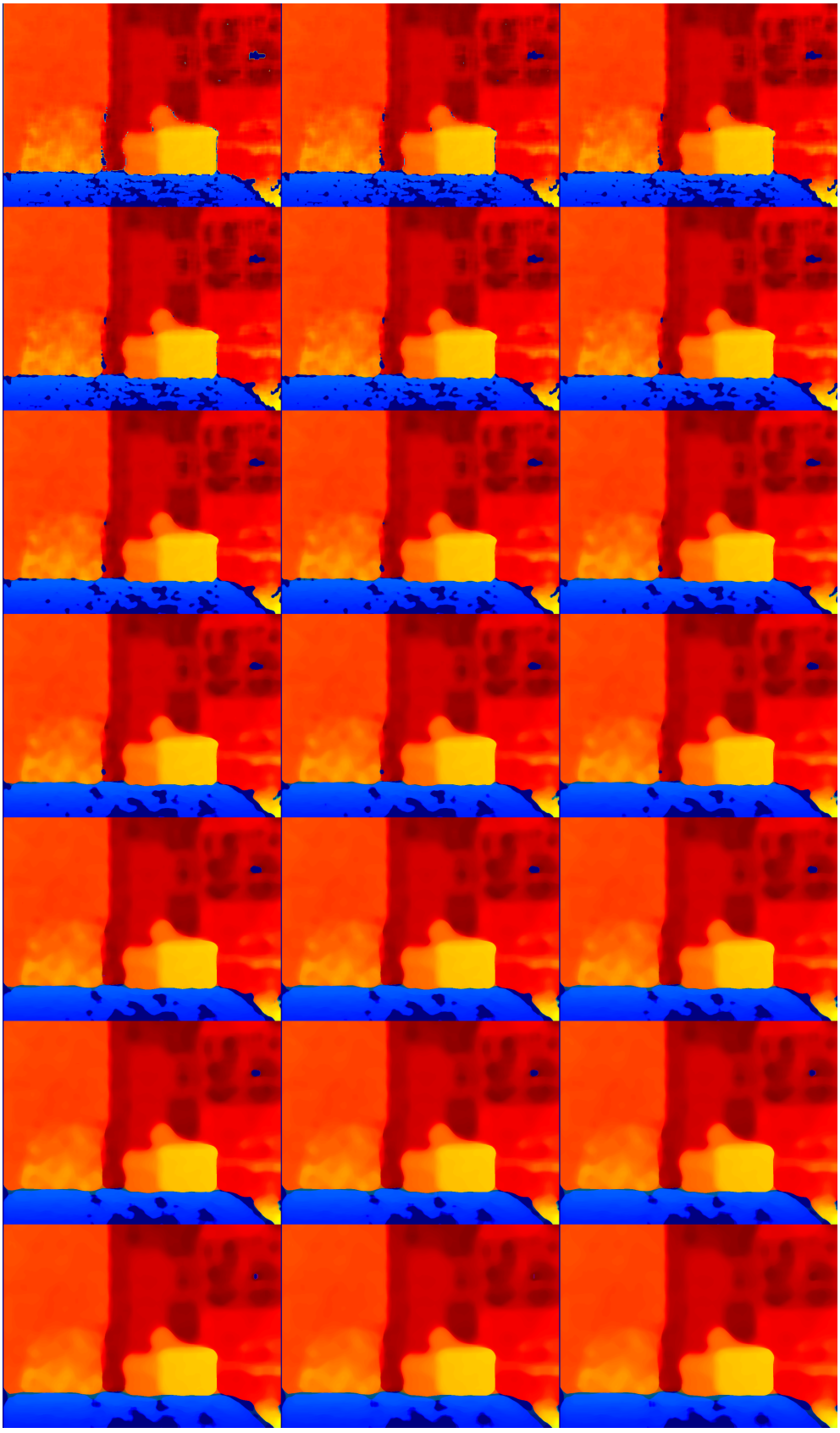


Figure 8: Median blurring — Colored grid result

4.4 Bilateral Filtering

Code

```python
import cv2
import numpy as np


depth_gray = cv2.imread("images/depth_gray.png", cv2.IMREAD_GRAYSCALE)

depth_colored = cv2.applyColorMap(depth_gray, cv2.COLORMAP_JET)

# grayscale sequence
g1  = depth_gray
g2  = cv2.bilateralFilter(depth_gray, 3, 75, 75)
g3  = cv2.bilateralFilter(depth_gray, 5, 75, 75)
g4  = cv2.bilateralFilter(depth_gray, 7, 75, 75)
g5  = cv2.bilateralFilter(depth_gray, 9, 75, 75)
g6  = cv2.bilateralFilter(depth_gray, 11, 75, 75)
g7  = cv2.bilateralFilter(depth_gray, 13, 75, 75)
g8  = cv2.bilateralFilter(depth_gray, 15, 75, 75)
g9  = cv2.bilateralFilter(depth_gray, 17, 75, 75)
g10 = cv2.bilateralFilter(depth_gray, 19, 75, 75)
g11 = cv2.bilateralFilter(depth_gray, 21, 75, 75)
g12 = cv2.bilateralFilter(depth_gray, 23, 75, 75)
g13 = cv2.bilateralFilter(depth_gray, 25, 75, 75)
g14 = cv2.bilateralFilter(depth_gray, 27, 75, 75)
g15 = cv2.bilateralFilter(depth_gray, 29, 75, 75)
g16 = cv2.bilateralFilter(depth_gray, 31, 75, 75)
g17 = cv2.bilateralFilter(depth_gray, 33, 75, 75)
g18 = cv2.bilateralFilter(depth_gray, 35, 75, 75)
g19 = cv2.bilateralFilter(depth_gray, 37, 75, 75)
g20 = cv2.bilateralFilter(depth_gray, 39, 75, 75)
g21 = cv2.bilateralFilter(depth_gray, 41, 75, 75)



# build 7×3 grid
row1 = np.hstack([g1, g2, g3])
row2 = np.hstack([g4, g5, g6])
row3 = np.hstack([g7, g8, g9])
row4 = np.hstack([g10, g11, g12])
row5 = np.hstack([g13, g14, g15])
row6 = np.hstack([g16, g17, g18])
row7 = np.hstack([g19, g20, g21])

grid_gray = np.vstack([row1, row2, row3, row4, row5, row6, row7])

# colored sequence
c1  = depth_colored
c2  = cv2.bilateralFilter(depth_colored, 3, 75, 75)
c3  = cv2.bilateralFilter(depth_colored, 5, 75, 75)
c4  = cv2.bilateralFilter(depth_colored, 7, 75, 75)
c5  = cv2.bilateralFilter(depth_colored, 9, 75, 75)
c6  = cv2.bilateralFilter(depth_colored, 11, 75, 75)
c7  = cv2.bilateralFilter(depth_colored, 13, 75, 75)
c8  = cv2.bilateralFilter(depth_colored, 15, 75, 75)
c9  = cv2.bilateralFilter(depth_colored, 17, 75, 75)
c10 = cv2.bilateralFilter(depth_colored, 19, 75, 75)
c11 = cv2.bilateralFilter(depth_colored, 21, 75, 75)
c12 = cv2.bilateralFilter(depth_colored, 23, 75, 75)
c13 = cv2.bilateralFilter(depth_colored, 25, 75, 75)
```

```
c14 = cv2.bilateralFilter(depth_colored, 27, 75, 75)
c15 = cv2.bilateralFilter(depth_colored, 29, 75, 75)
c16 = cv2.bilateralFilter(depth_colored, 31, 75, 75)
c17 = cv2.bilateralFilter(depth_colored, 33, 75, 75)
c18 = cv2.bilateralFilter(depth_colored, 35, 75, 75)
c19 = cv2.bilateralFilter(depth_colored, 37, 75, 75)
c20 = cv2.bilateralFilter(depth_colored, 39, 75, 75)
c21 = cv2.bilateralFilter(depth_colored, 41, 75, 75)

row1c = np.hstack([c1, c2, c3])
row2c = np.hstack([c4, c5, c6])
row3c = np.hstack([c7, c8, c9])
row4c = np.hstack([c10, c11, c12])
row5c = np.hstack([c13, c14, c15])
row6c = np.hstack([c16, c17, c18])
row7c = np.hstack([c19, c20, c21])

grid_colored = np.vstack([row1c, row2c, row3c, row4c, row5c, row6c, row7c])

# cv2.imshow("Gray Grid", grid_gray)
# cv2.imshow("Colored Grid", grid_colored)

# while True:
#     key = cv2.waitKey(1) & 0xFF
#     if key in (ord('1'), 27):  # press '1' or ESC
#         break

# cv2.destroyAllWindows()

# save each grid separately
cv2.imwrite("images/blur/07_bilateral_gray_grid.png", grid_gray)
cv2.imwrite("images/blur/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.

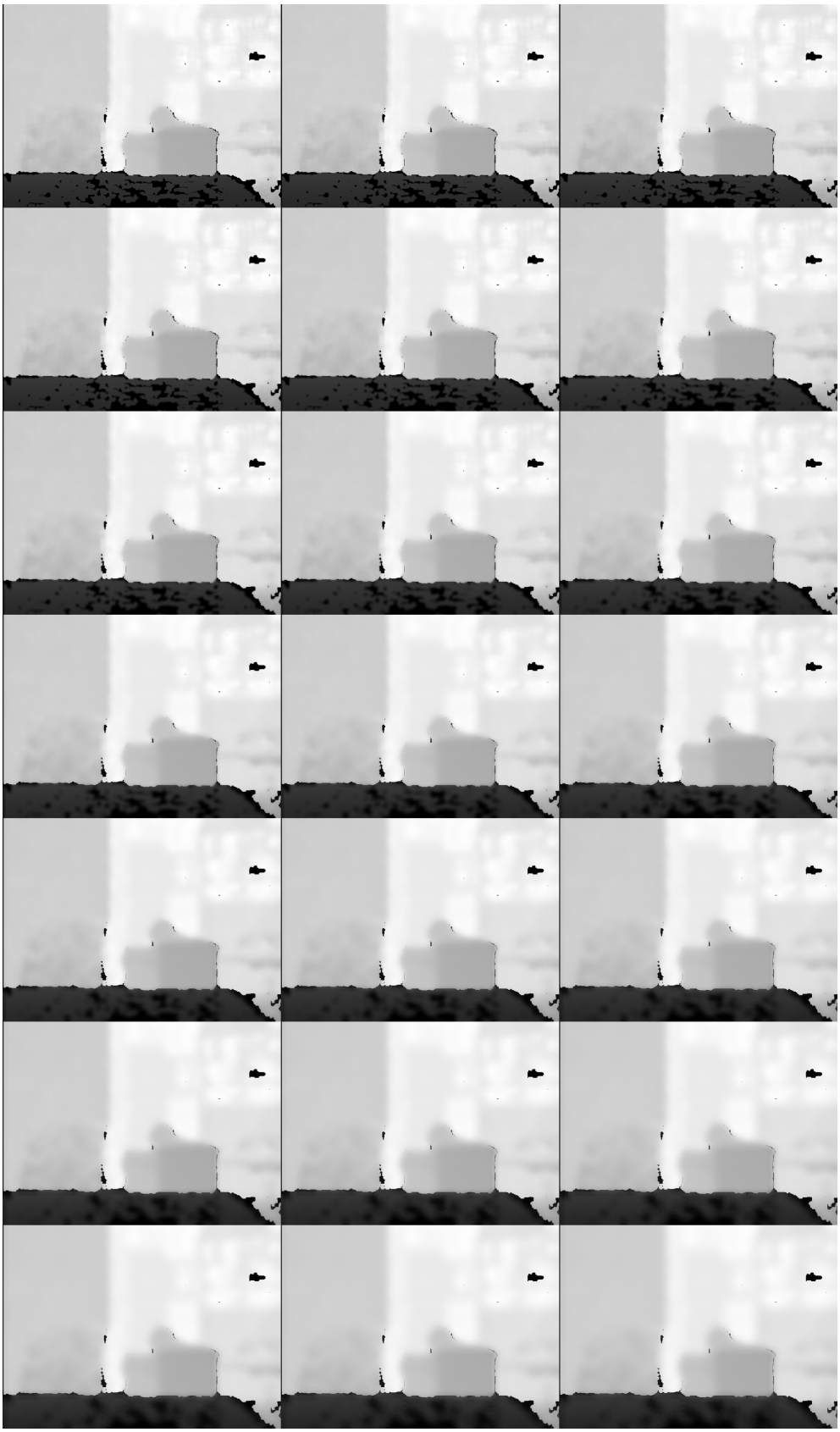


Figure 9: Bilateral filtering — Gray grid result

Output (Colored Grid)

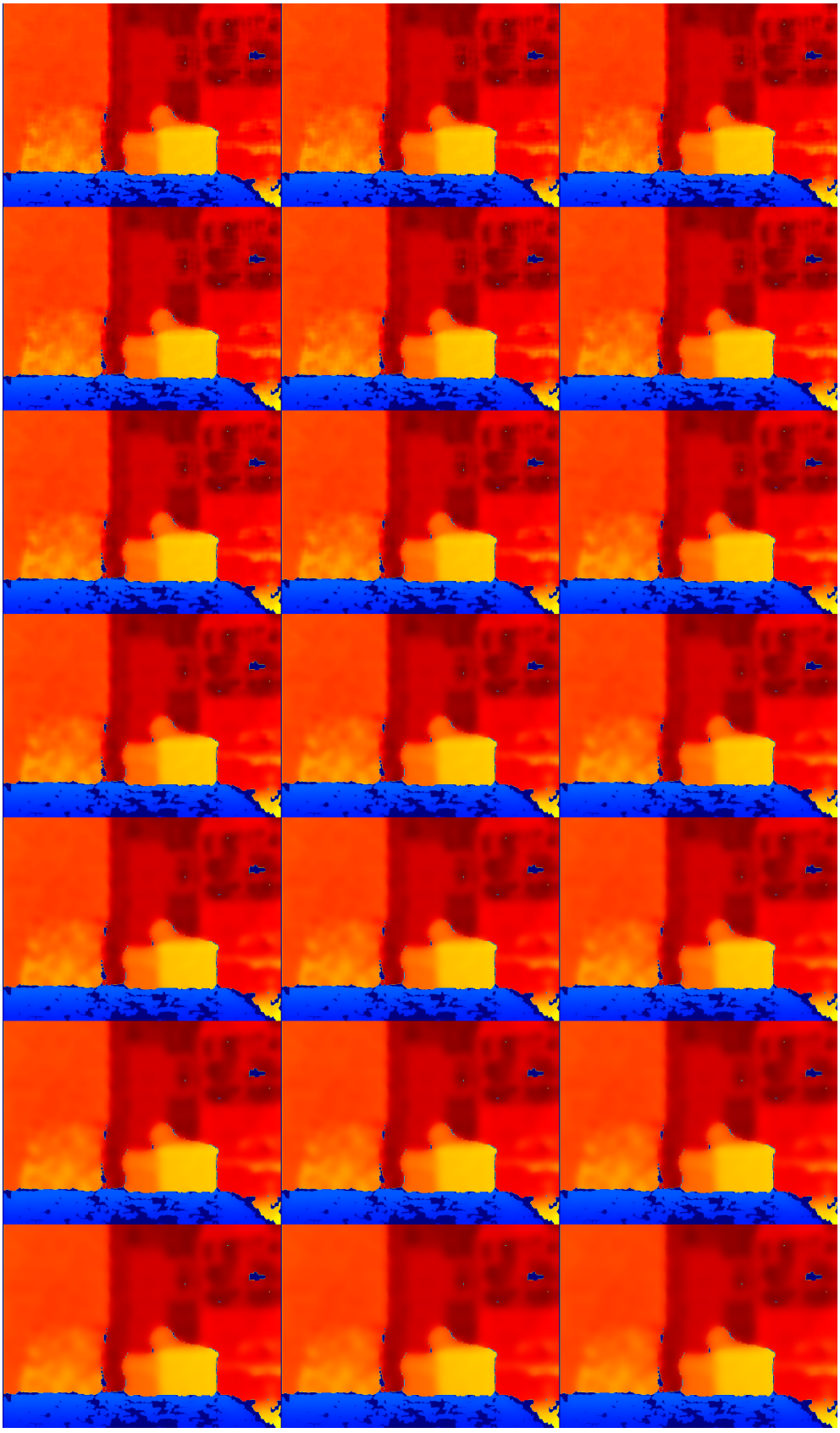


Figure 10: Bilateral filtering — Colored grid result

5 Canny Edge Detection on Blur

Code

```python
import cv2, os

# inputs (gray grids you already saved)
base = "images"
inputs = [
    ("blur/04_average_gray_grid.png",  "avg"),
    ("blur/05_gaussian_gray_grid.png", "gauss"),
    ("blur/06_median_gray_grid.png",   "median"),
    ("blur/07_bilateral_gray_grid.png","bilateral"),
]

# output folder
out_dir = os.path.join(base, "canny_on_gray")
os.makedirs(out_dir, exist_ok=True)

# 21 Canny settings: (low, high) = (t, 3t) for t = 10..210
thresholds = list(range(10, 211, 10))  # 10,20,...,210  → 21 combos

for fname, tag in inputs:
    img = cv2.imread(os.path.join(base, fname), cv2.IMREAD_GRAYSCALE)
    if img is None:
        print(f"Skipped (not found): {fname}")
        continue
    for idx, t in enumerate(thresholds, 1):
        edges = cv2.Canny(img, t, 3*t, apertureSize=3, L2gradient=False)
        out = os.path.join(out_dir, f"{tag}_{idx:02d}_canny_{t}_{3*t}.png")
        cv2.imwrite(out, edges)

```

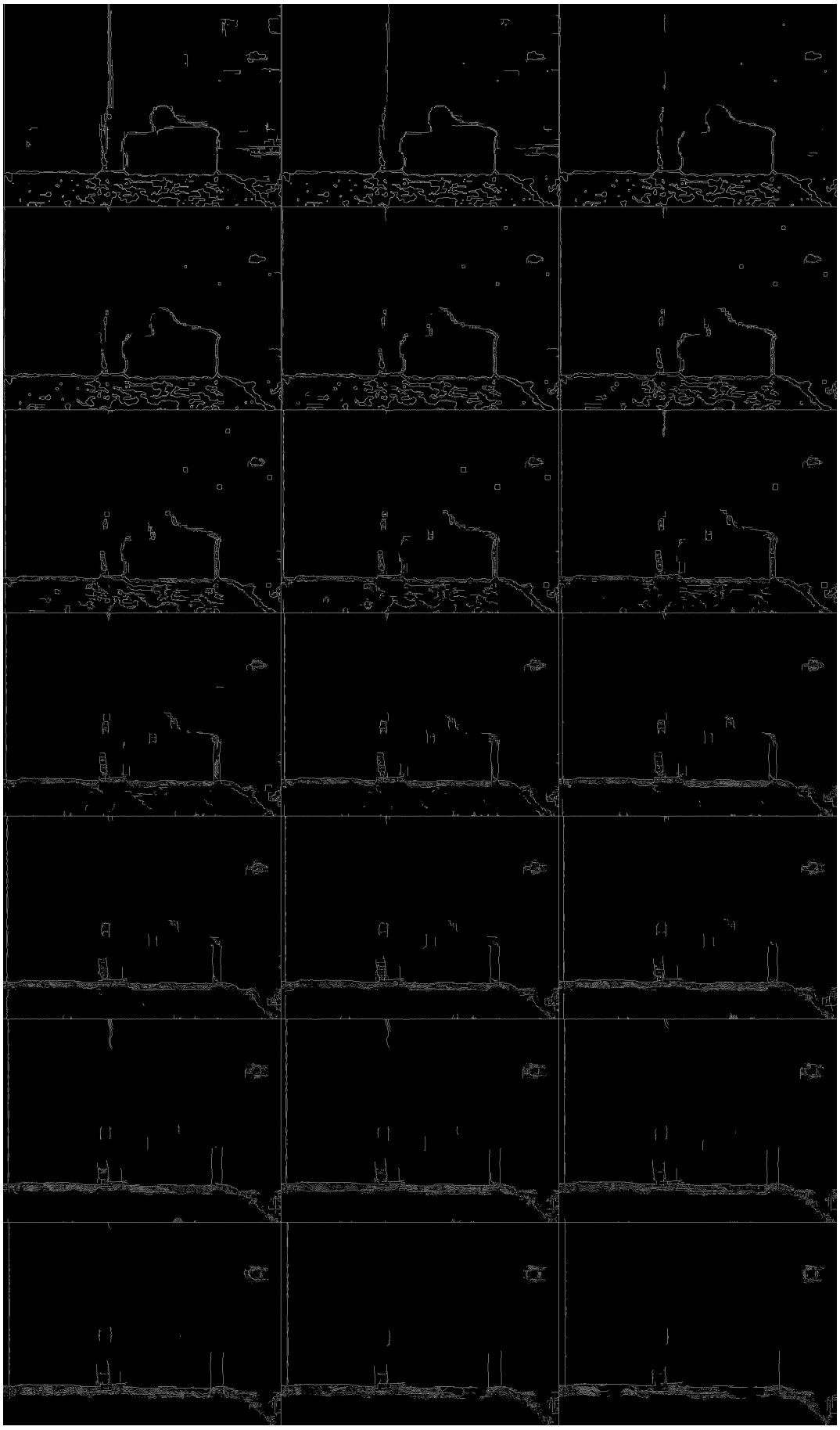


Figure 11: avg_01_canny_10_30

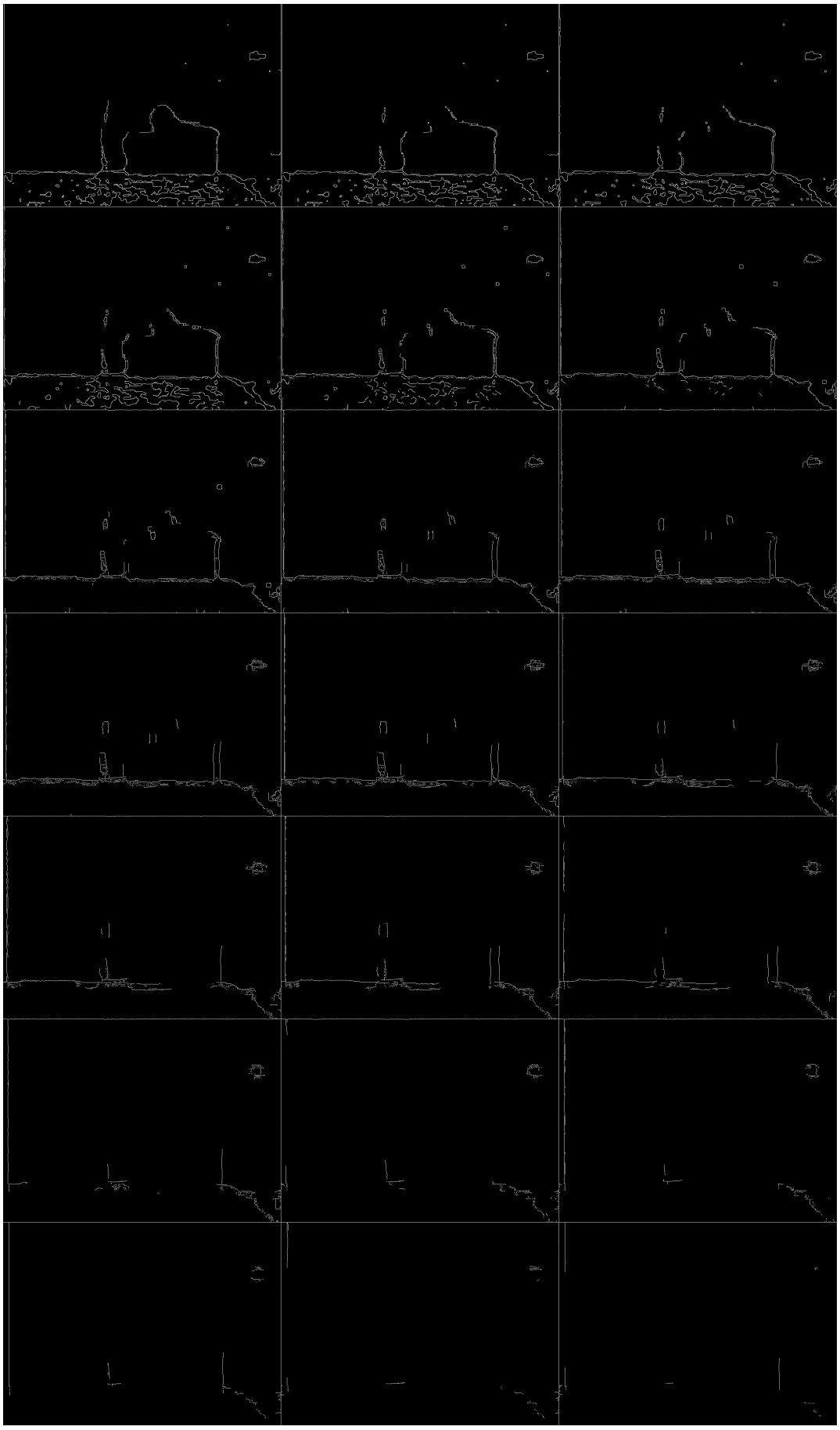


Figure 12: avg_02_canny_20_60

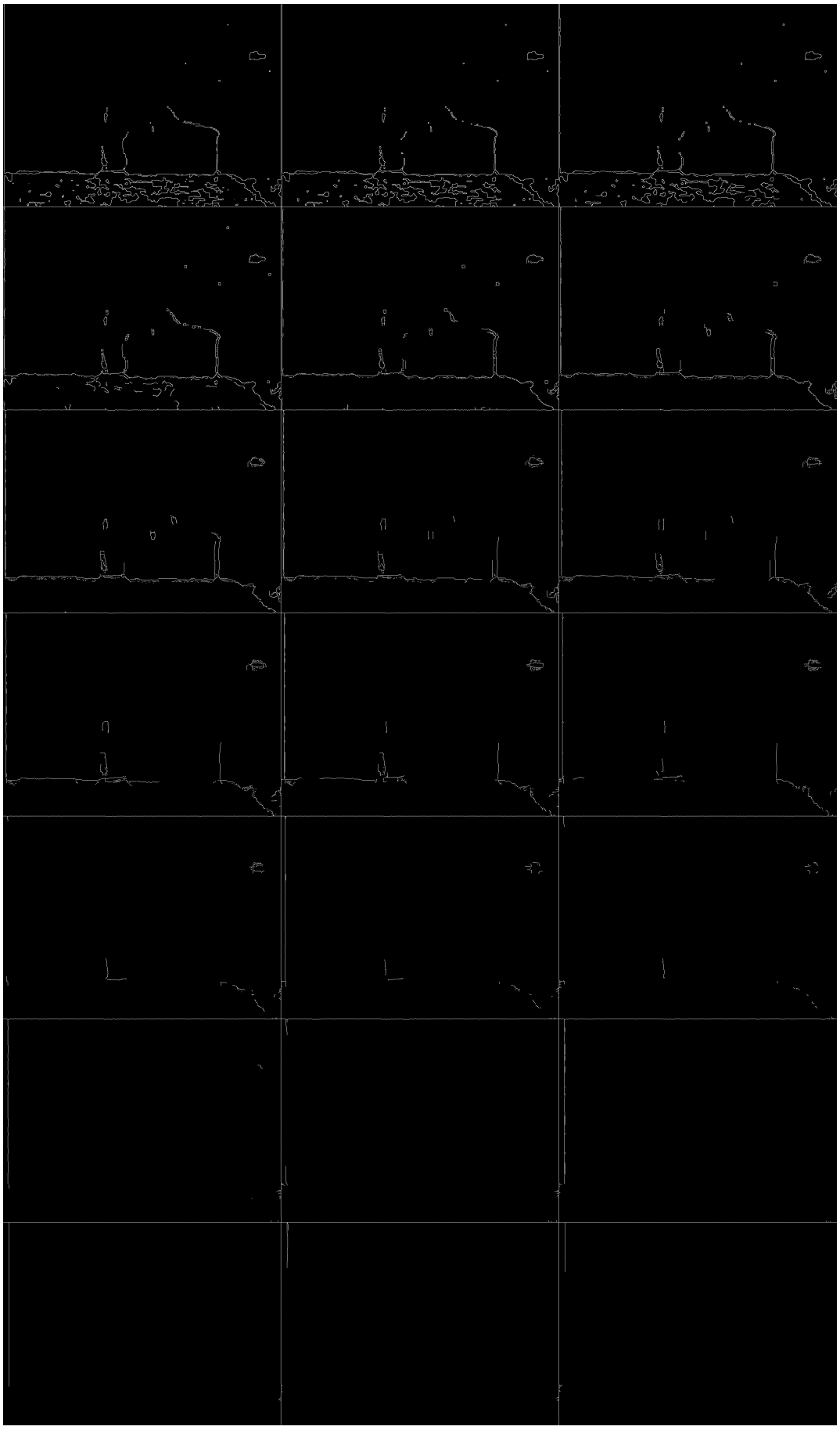


Figure 13: avg_03_canny_30_90

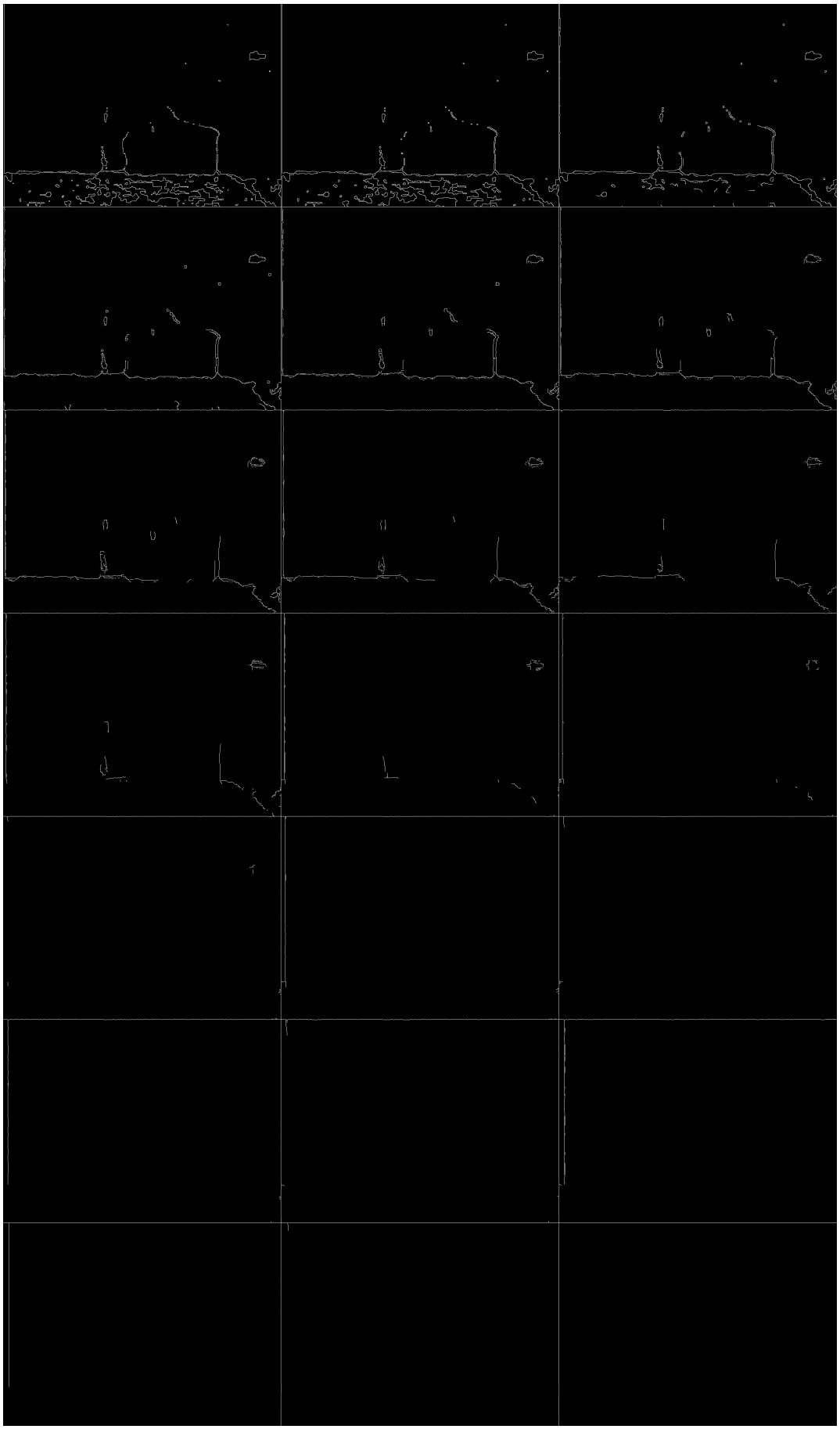


Figure 14: avg_04_canny_40_120

Figure 15: avg_05_canny_50_150

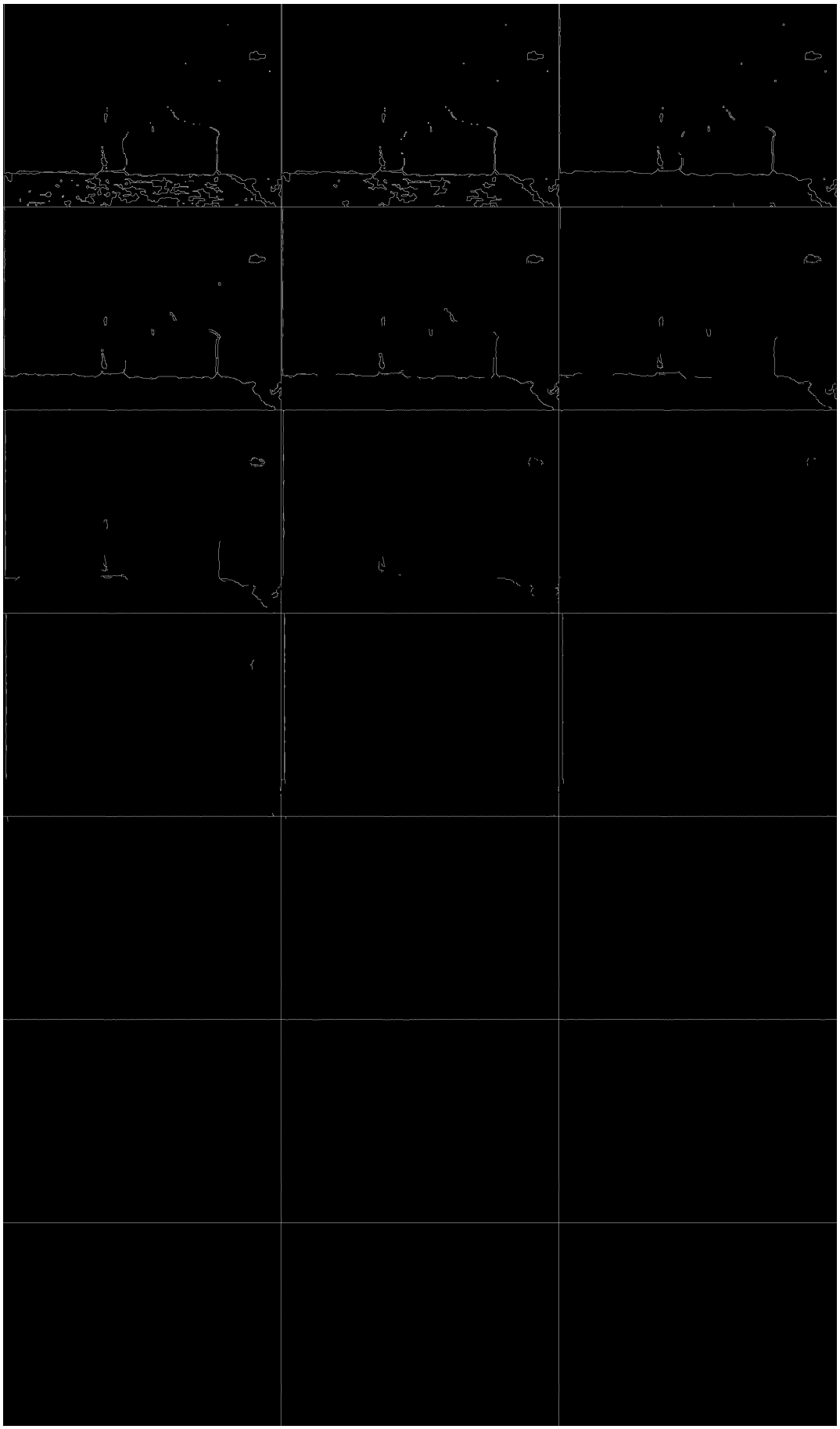


Figure 16: avg_06_canny_60_180

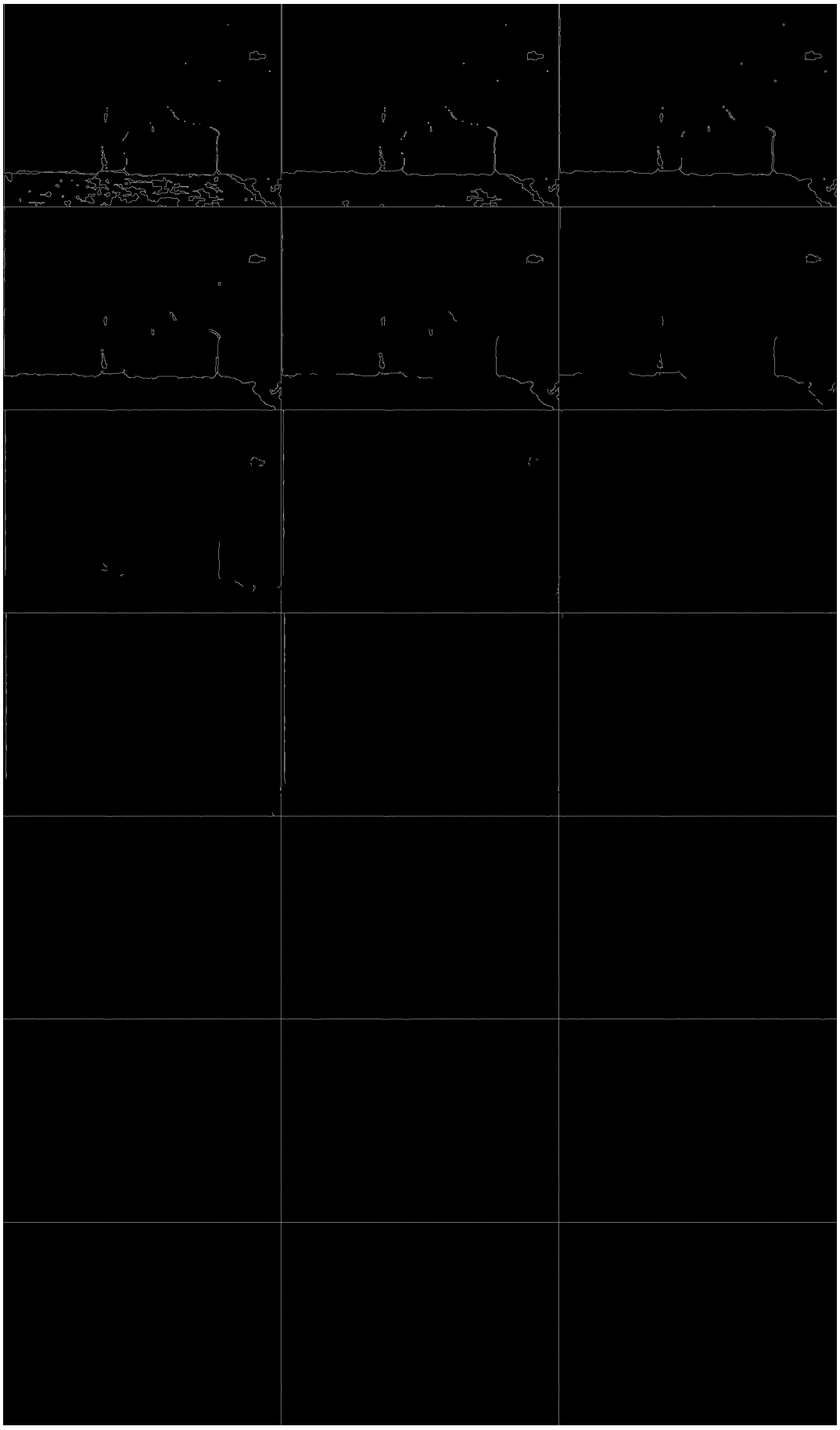


Figure 17: avg_07_canny_70_210

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

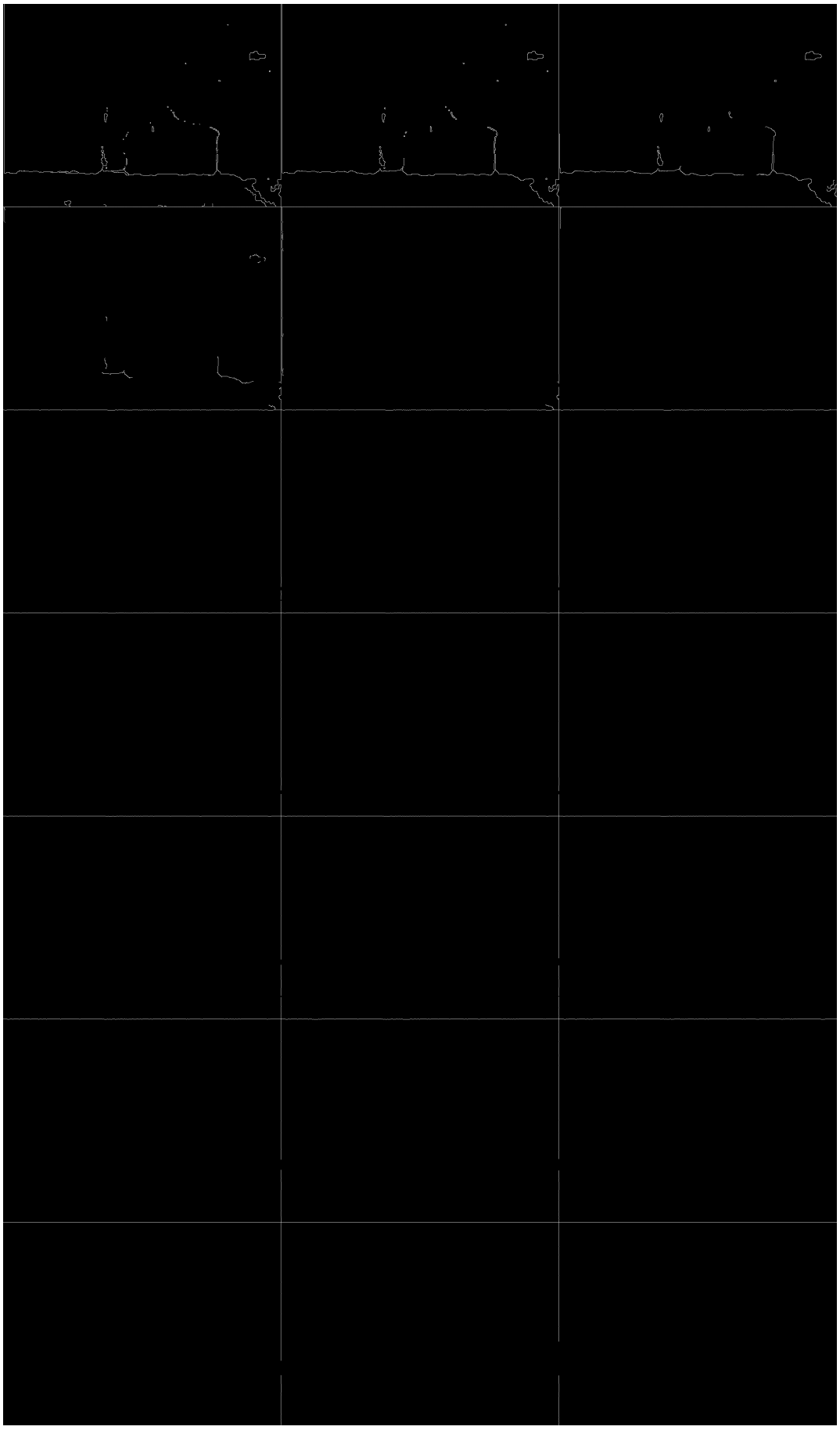


Figure 24: avg_14_canny_140_420

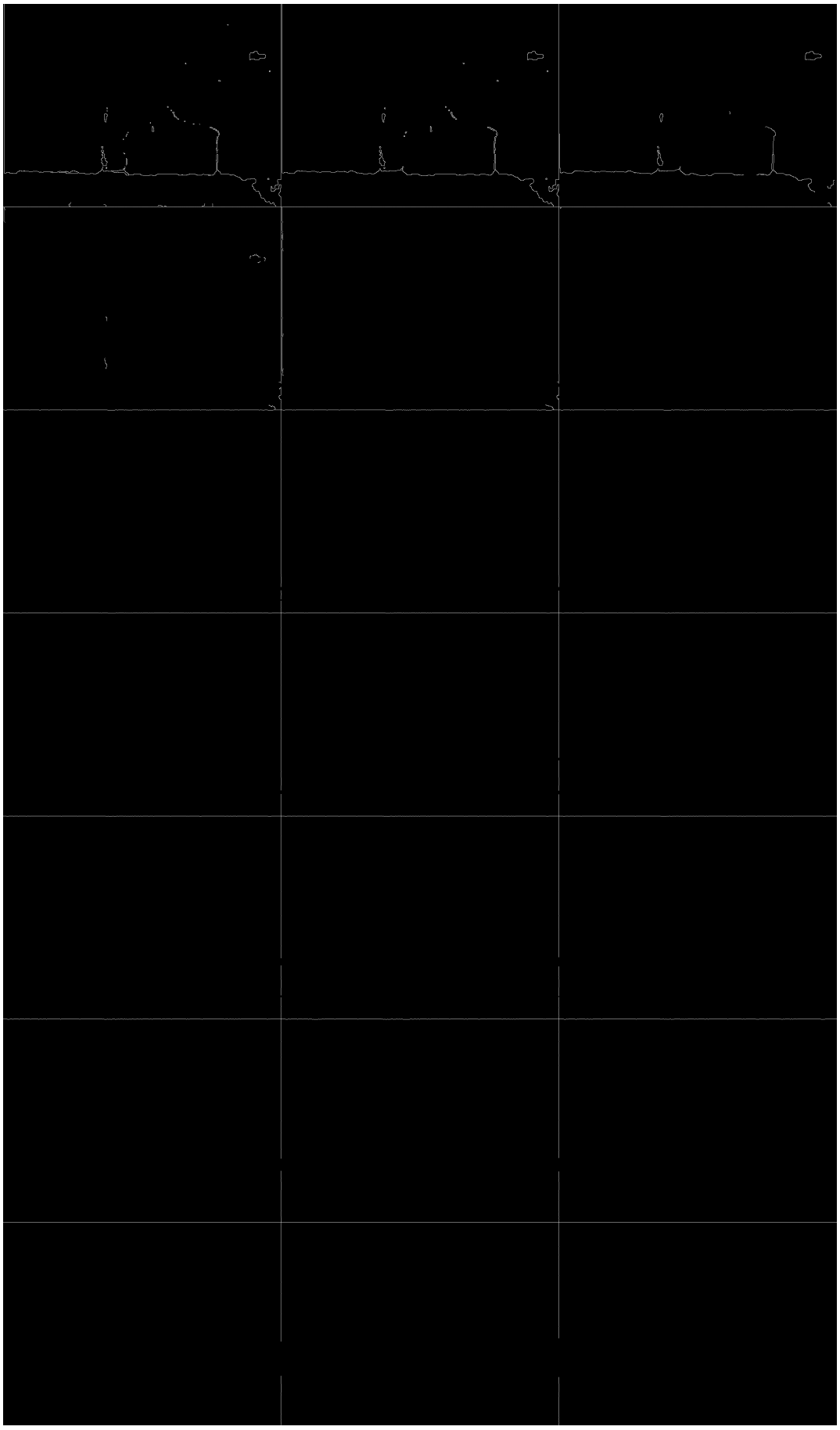


Figure 25: avg_15_canny_150_450

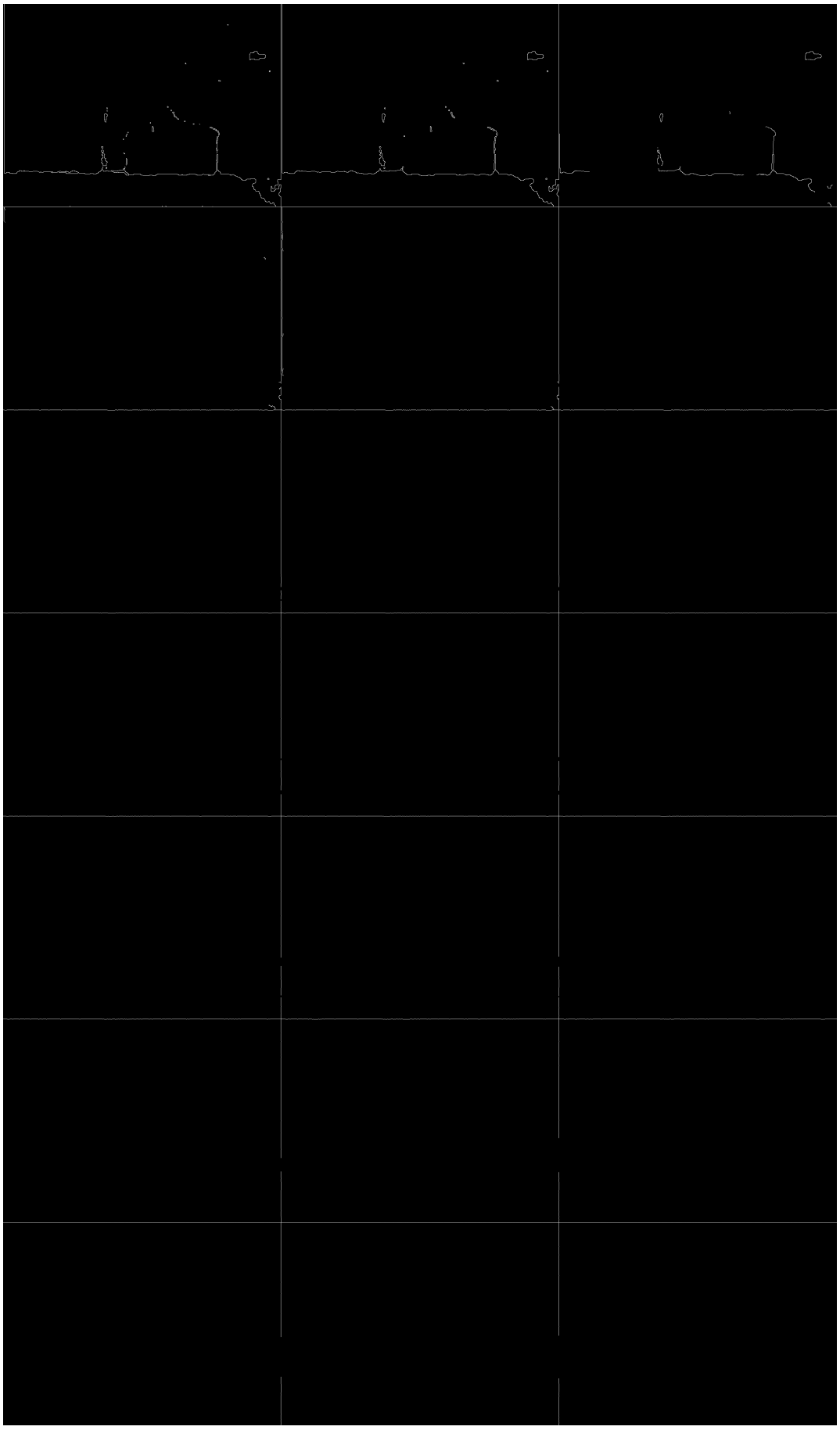


Figure 26: avg_16_canny_160_480

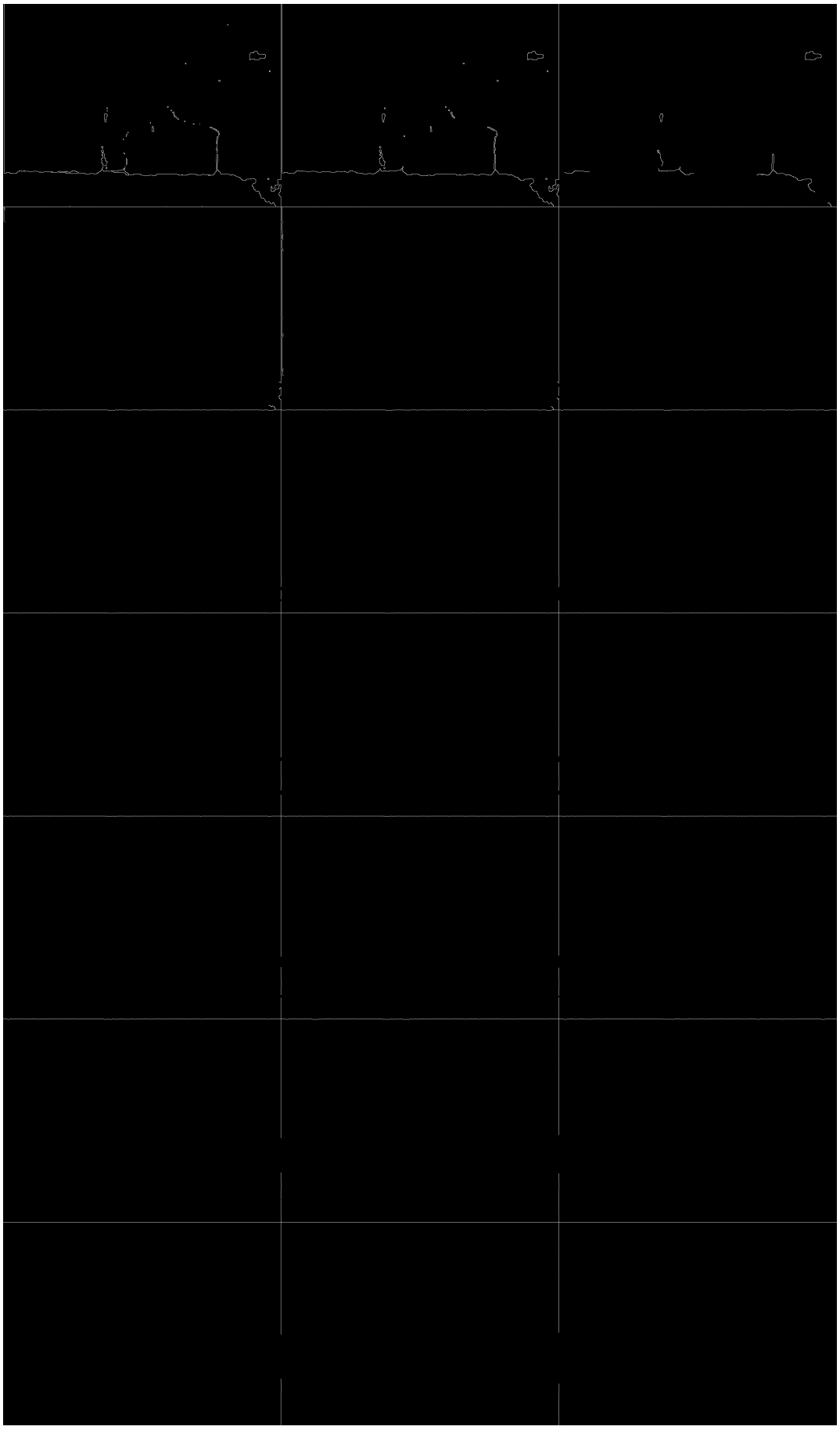


Figure 27: avg_17_canny_170_510

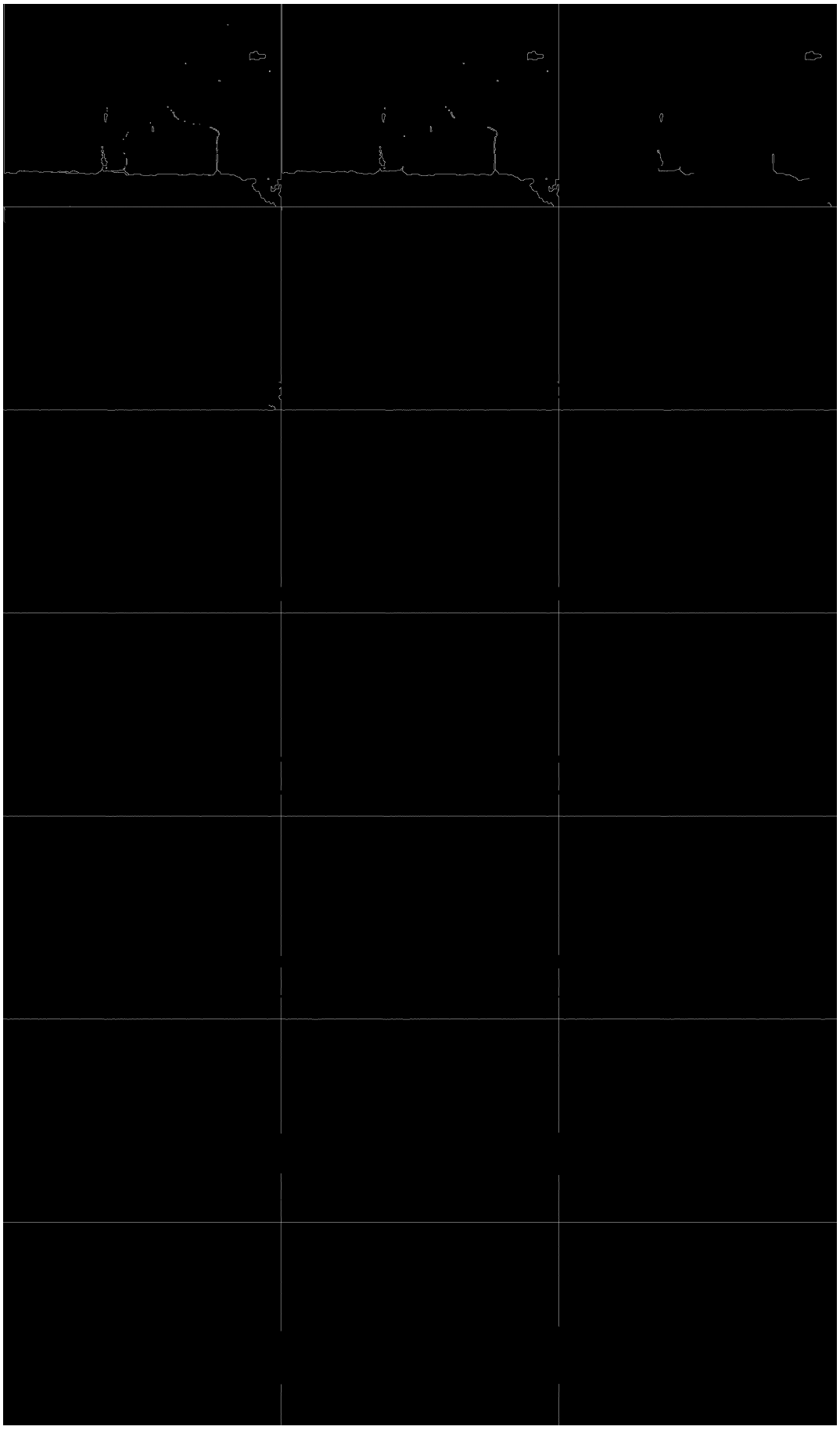


Figure 28: avg_18_canny_180_540

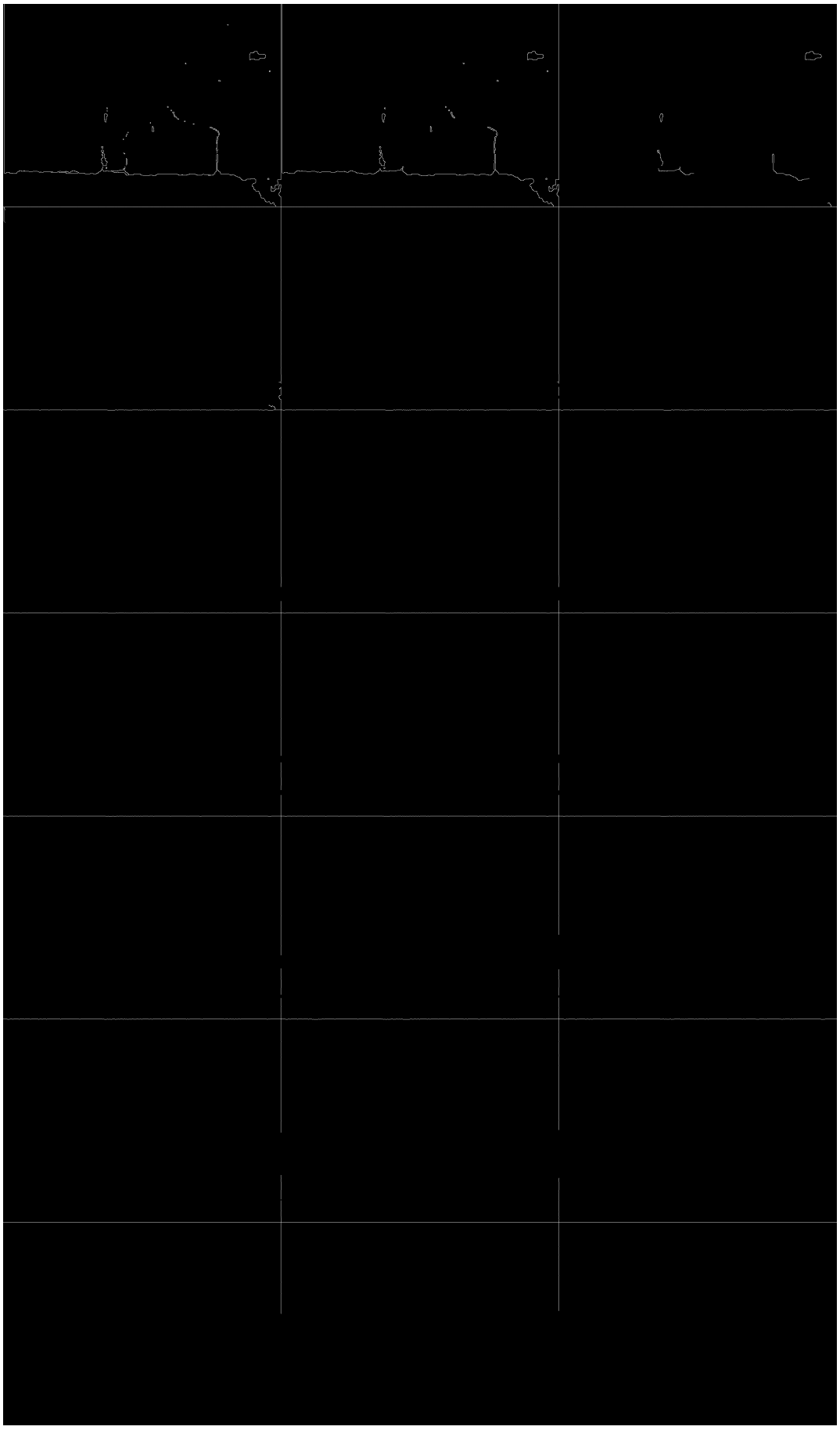


Figure 29: avg_19_canny_190_570

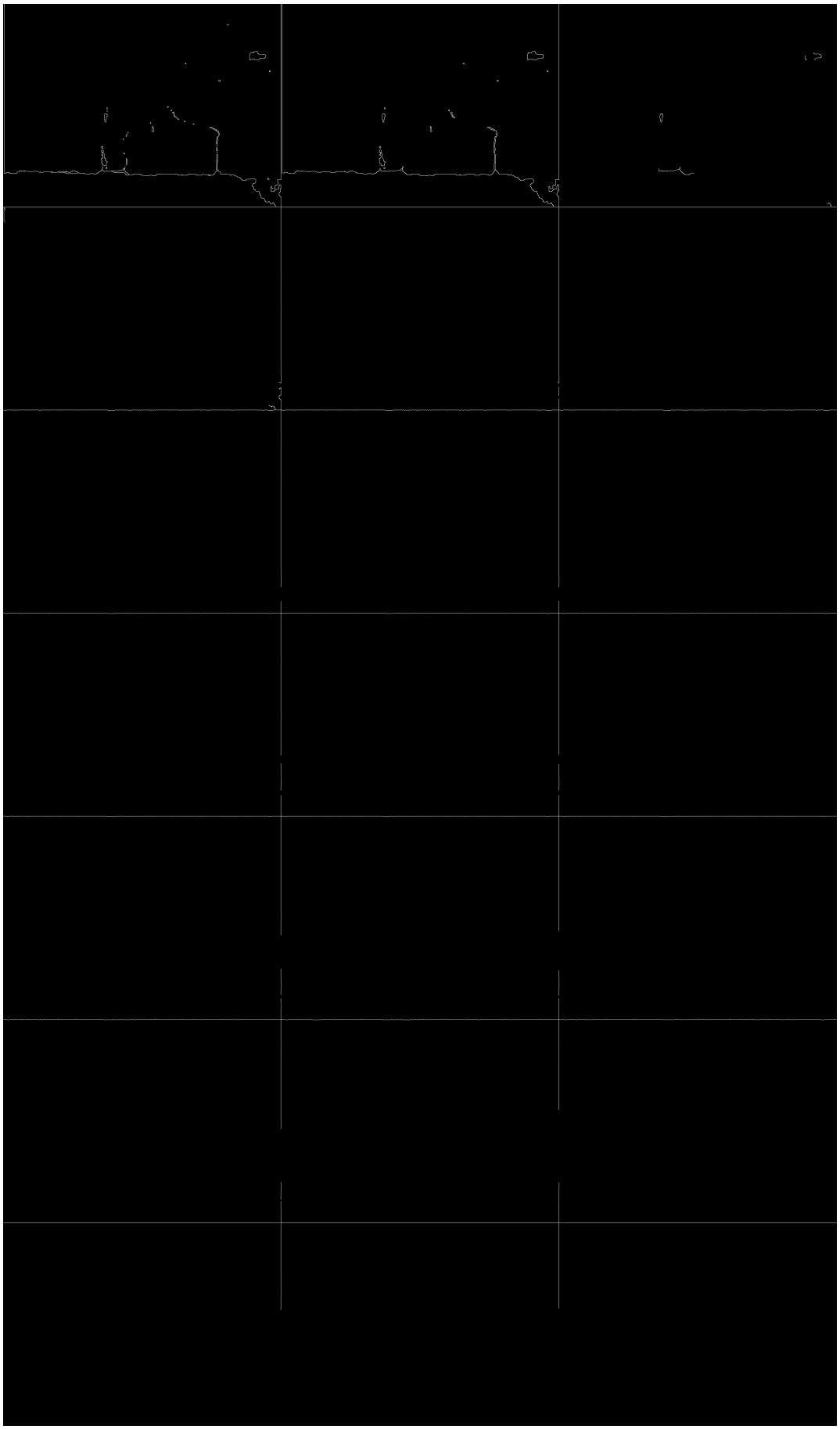


Figure 30: avg_20_canny_200_600

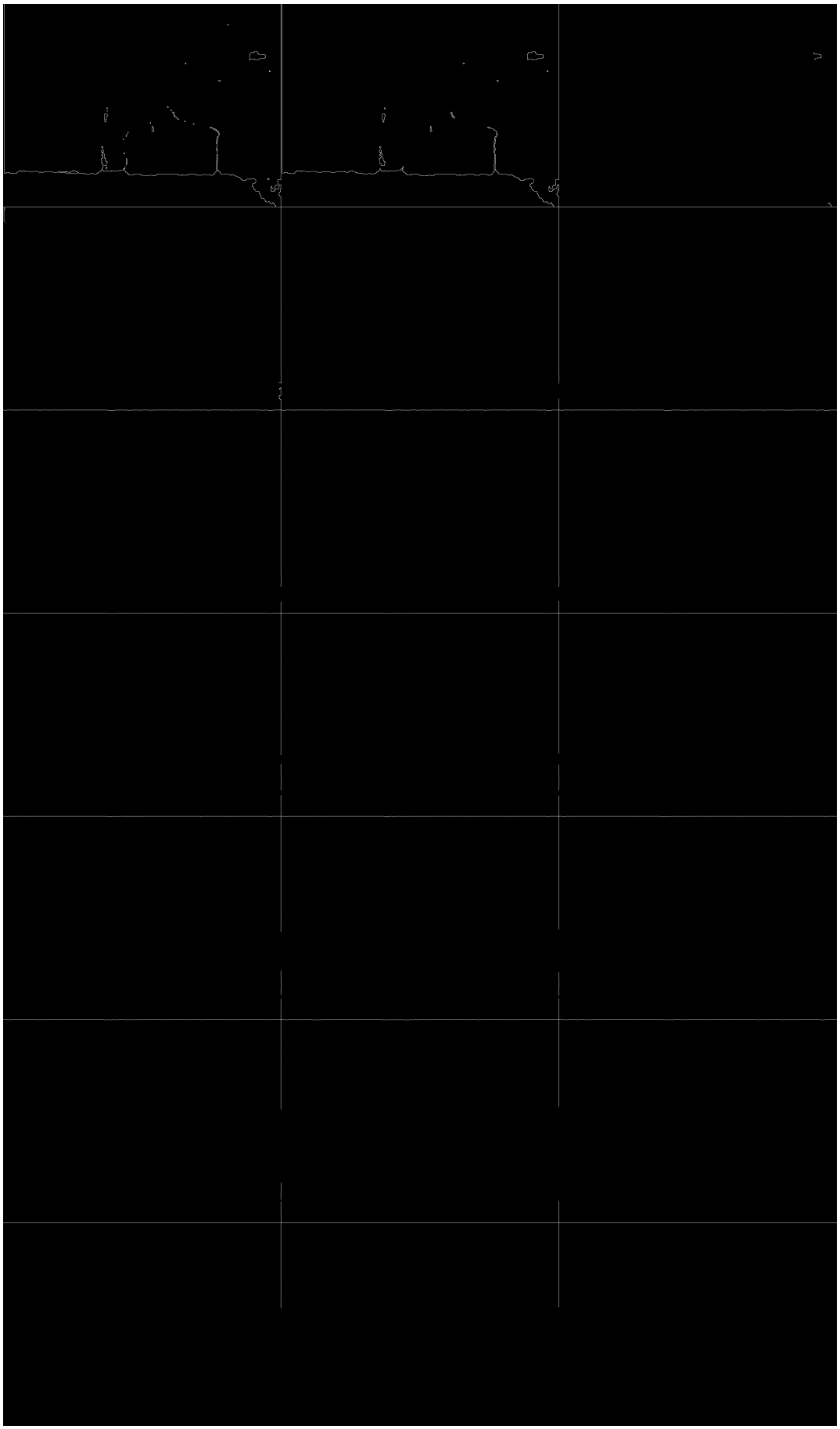


Figure 31: avg_21_canny_210_630

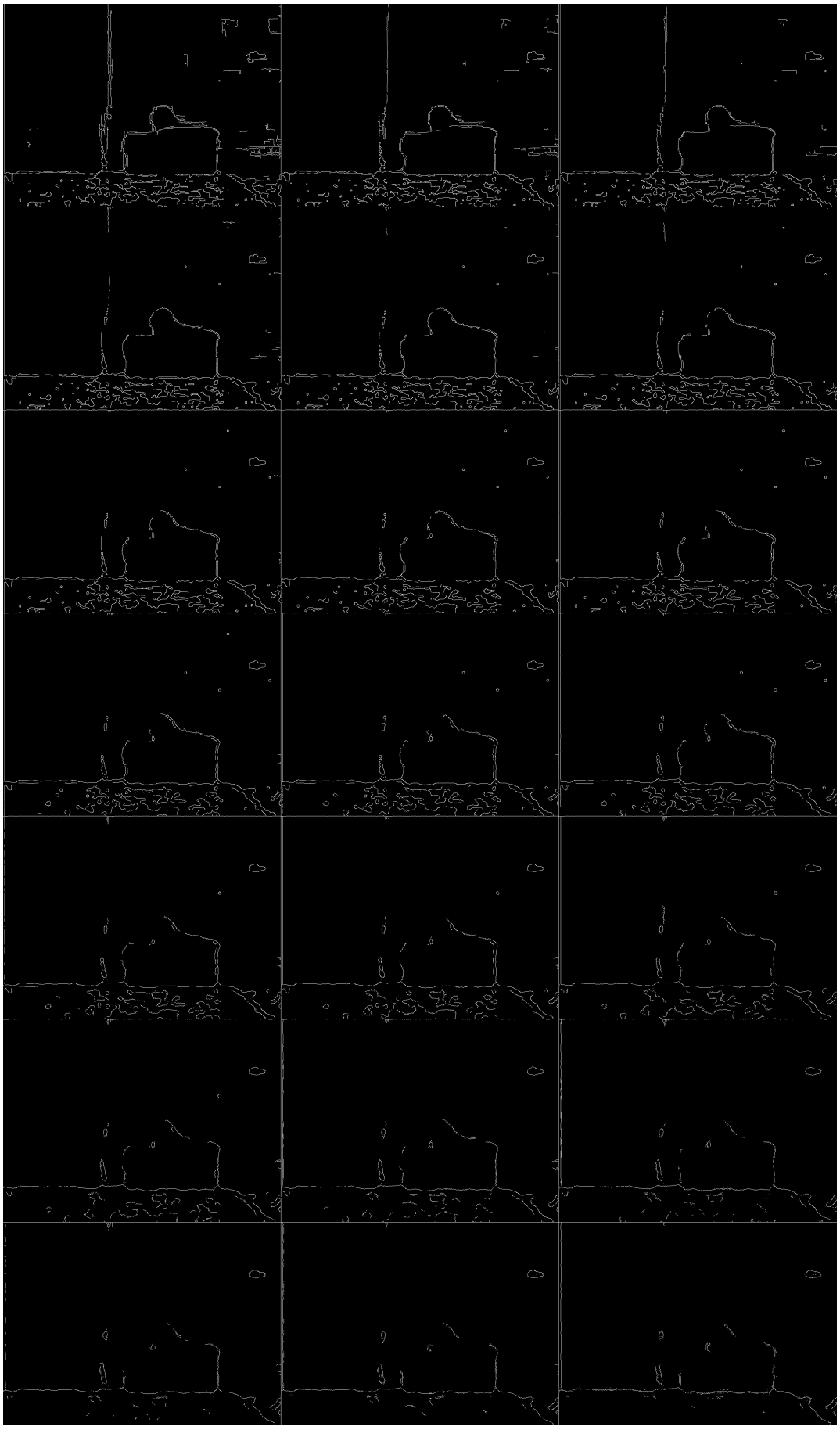


Figure 32: gauss_01_canny_10_30

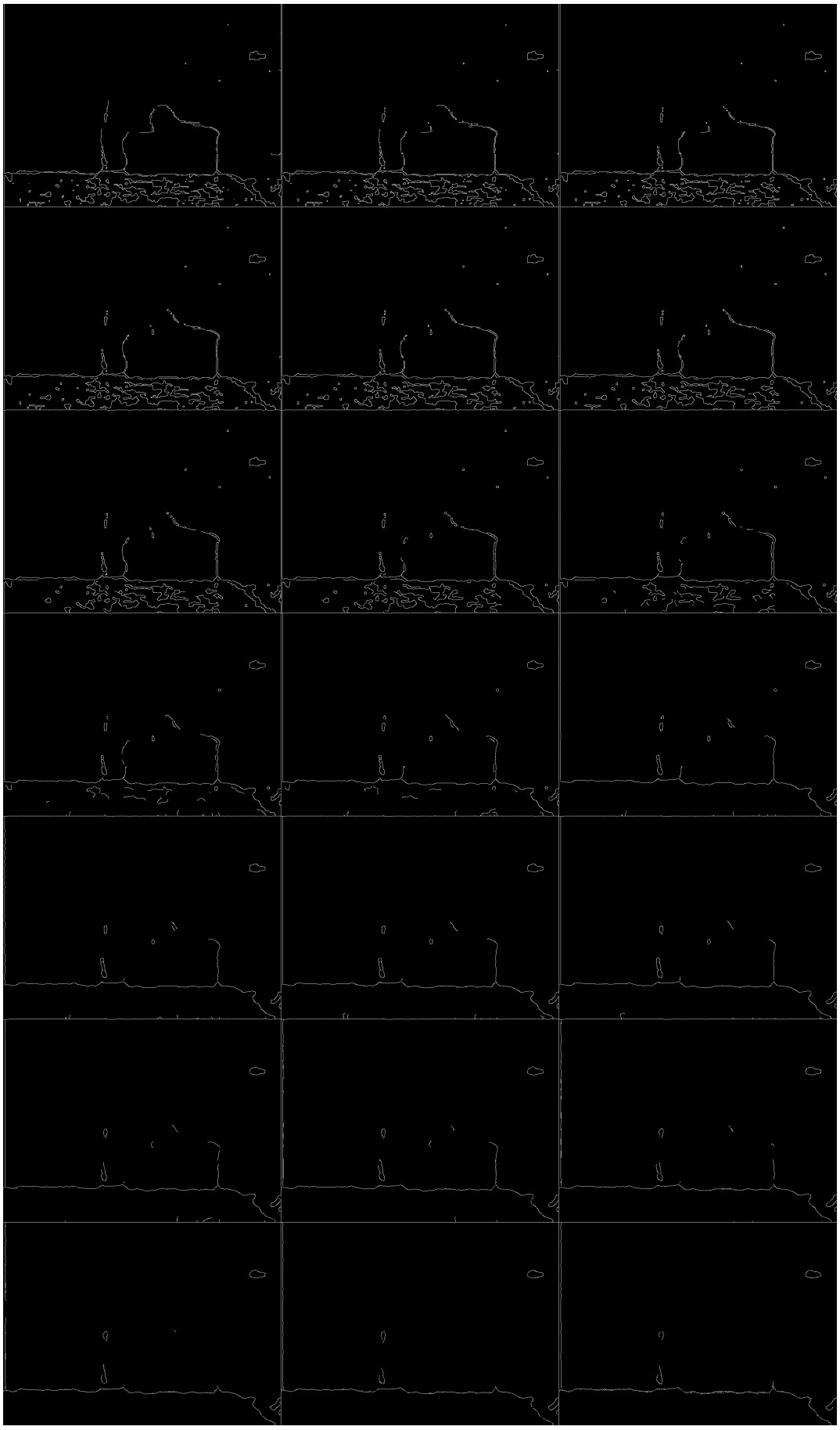


Figure 33: gauss_02_canny_20_60

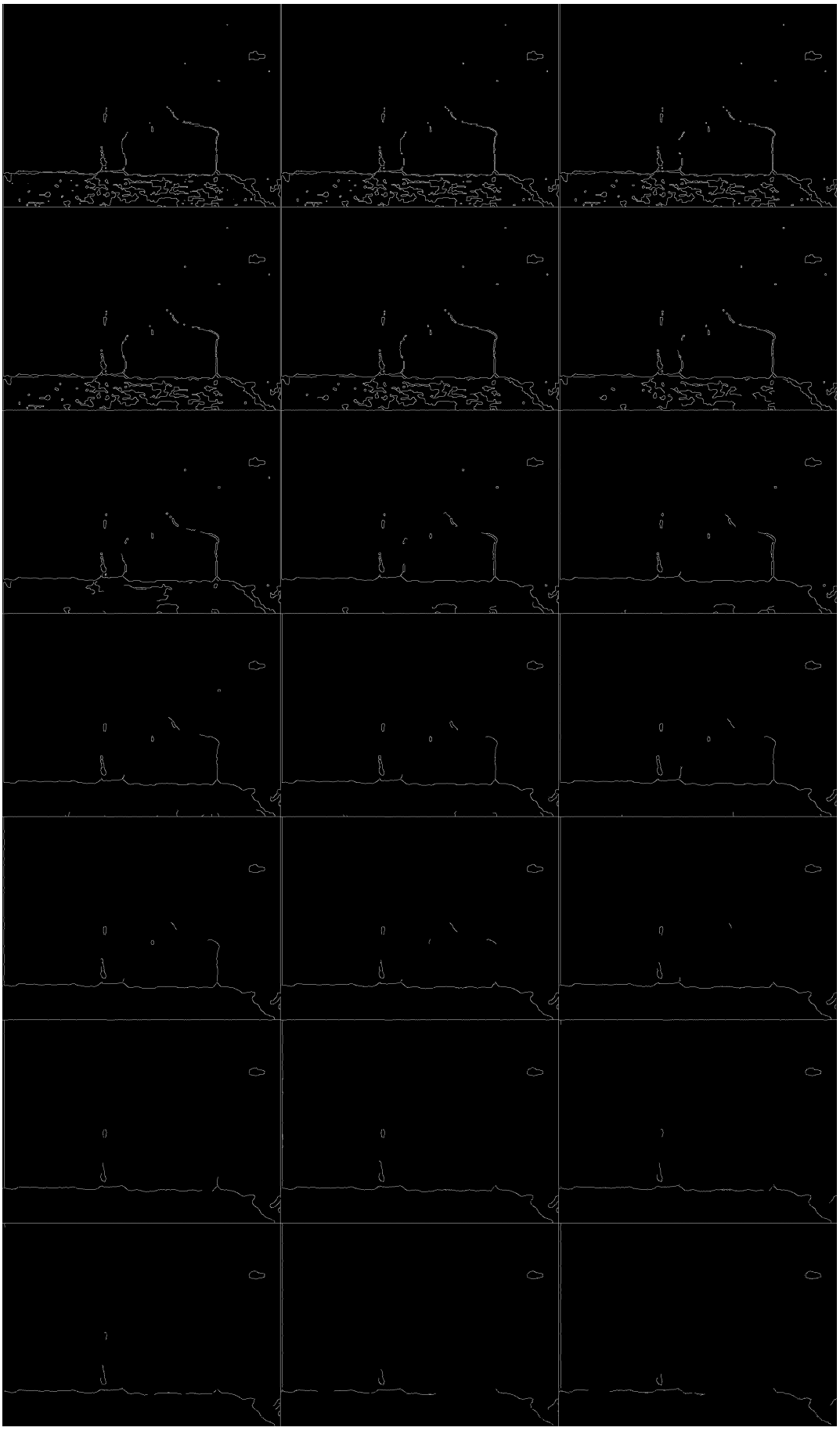


Figure 34: gauss_03_canny_30_90

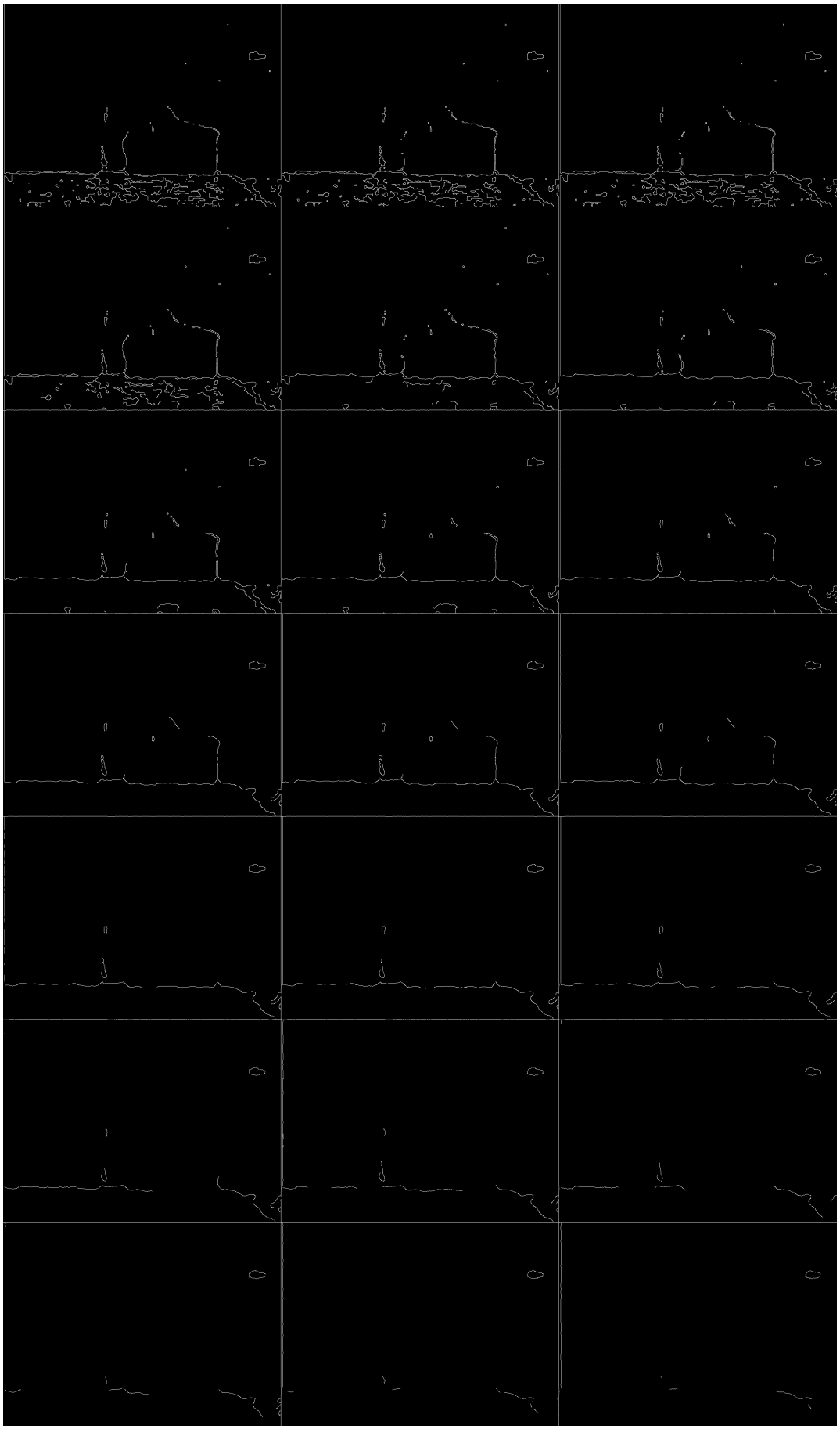


Figure 35: gauss_04_canny_40_120

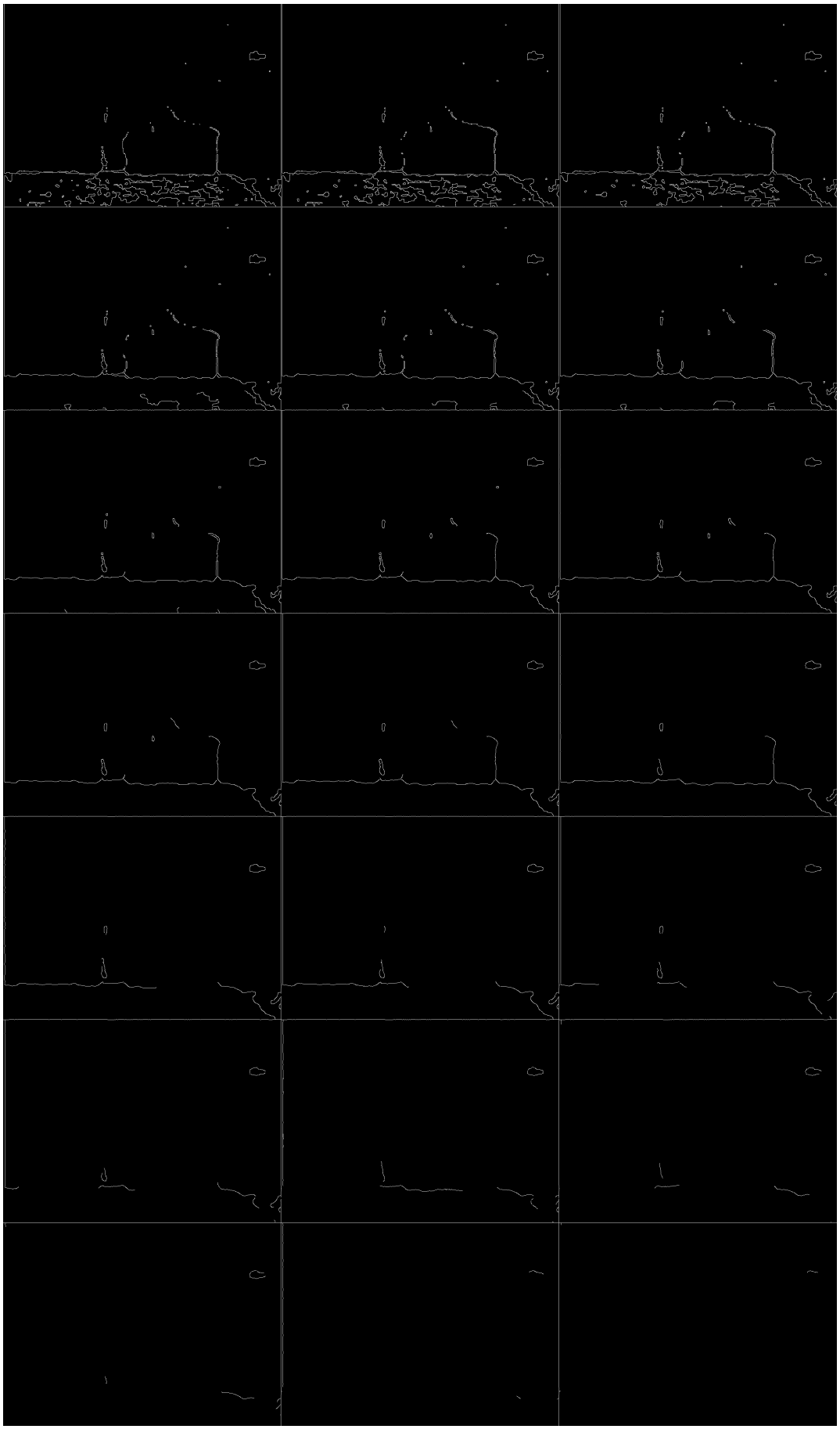


Figure 36: gauss_05_canny_50_150

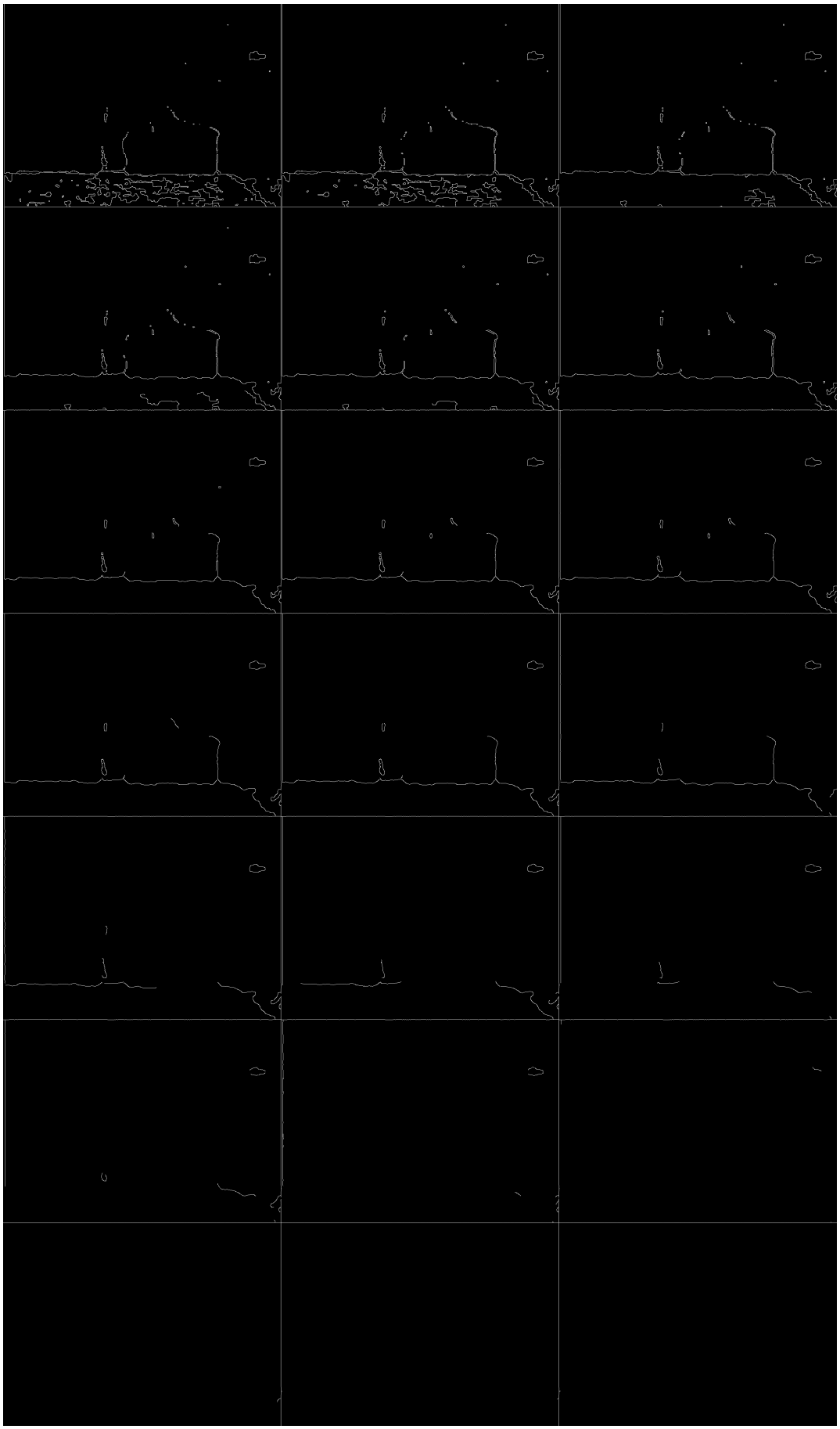


Figure 37: gauss_06_canny_60_180

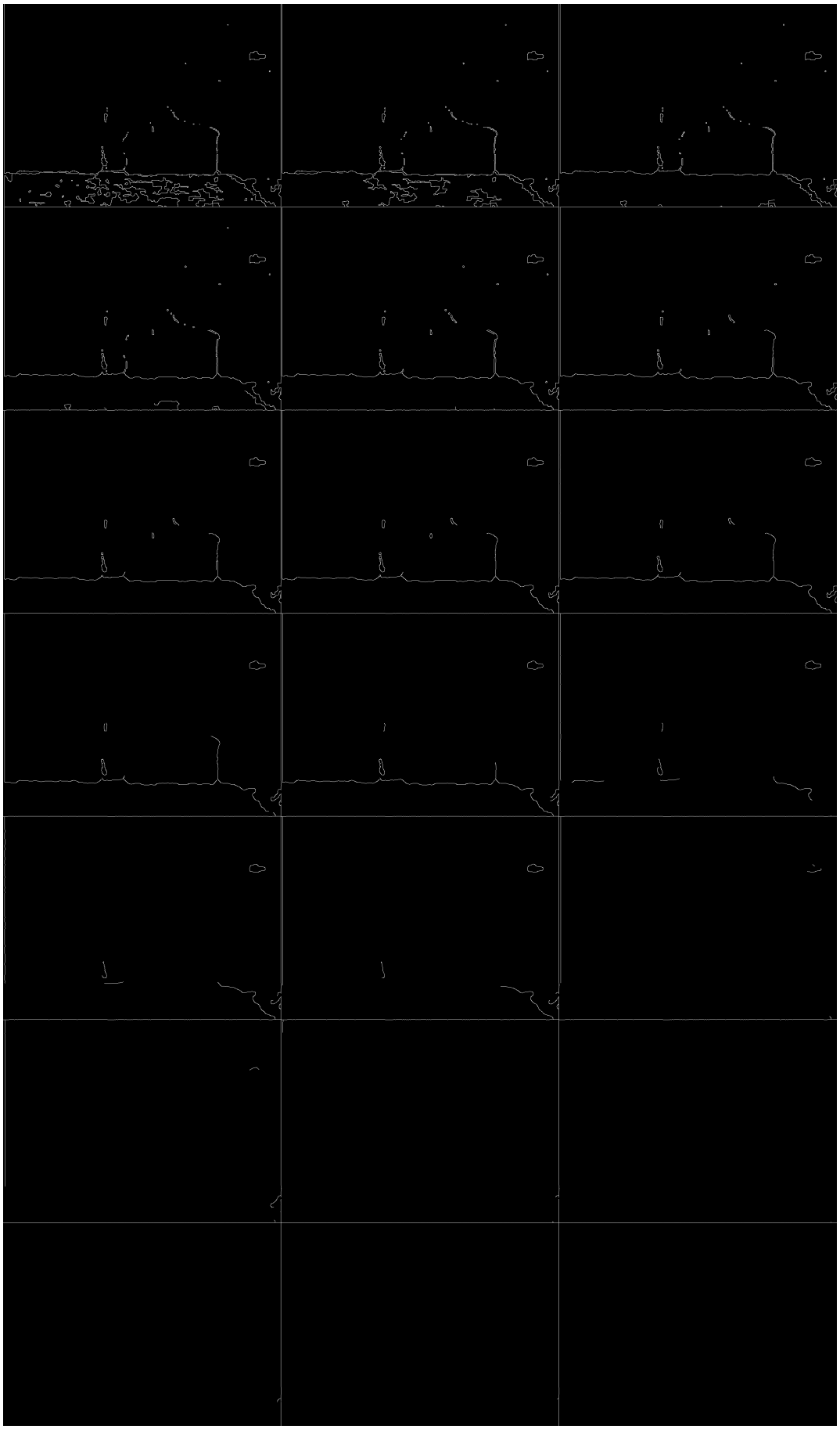


Figure 38: gauss_07_canny_70_210

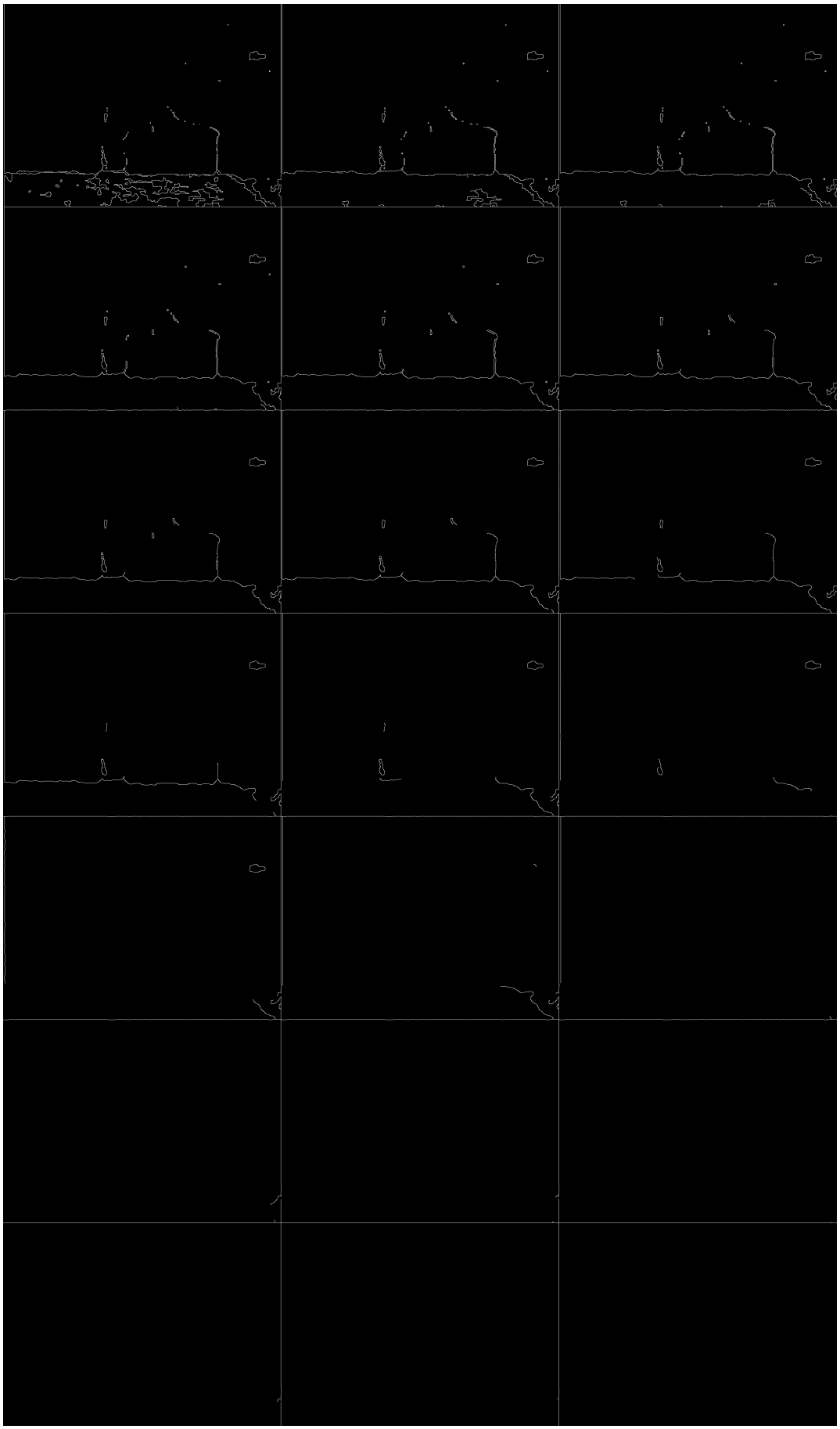


Figure 39: gauss_08_canny_80_240

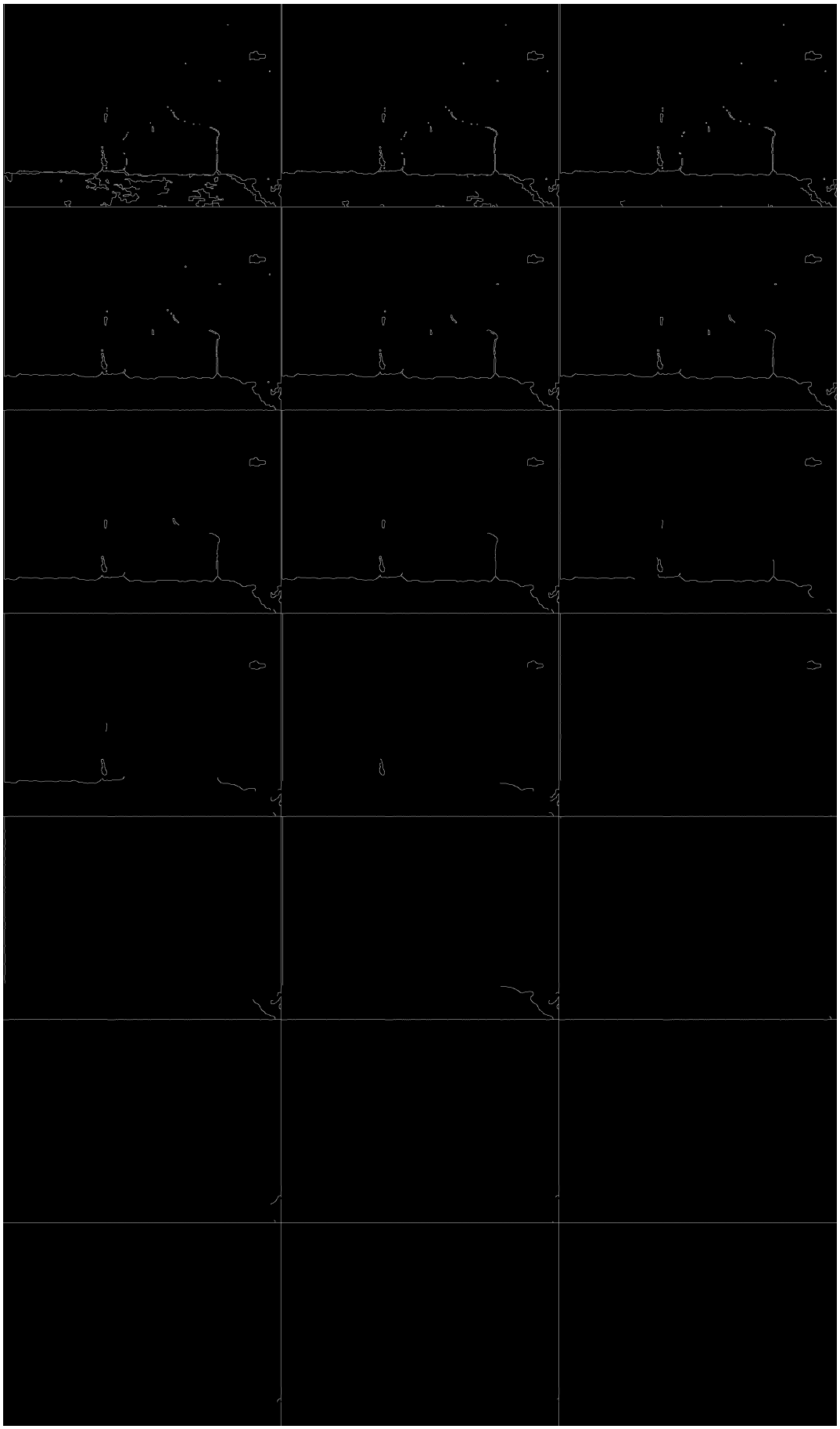


Figure 40: gauss_09_canny_90_270

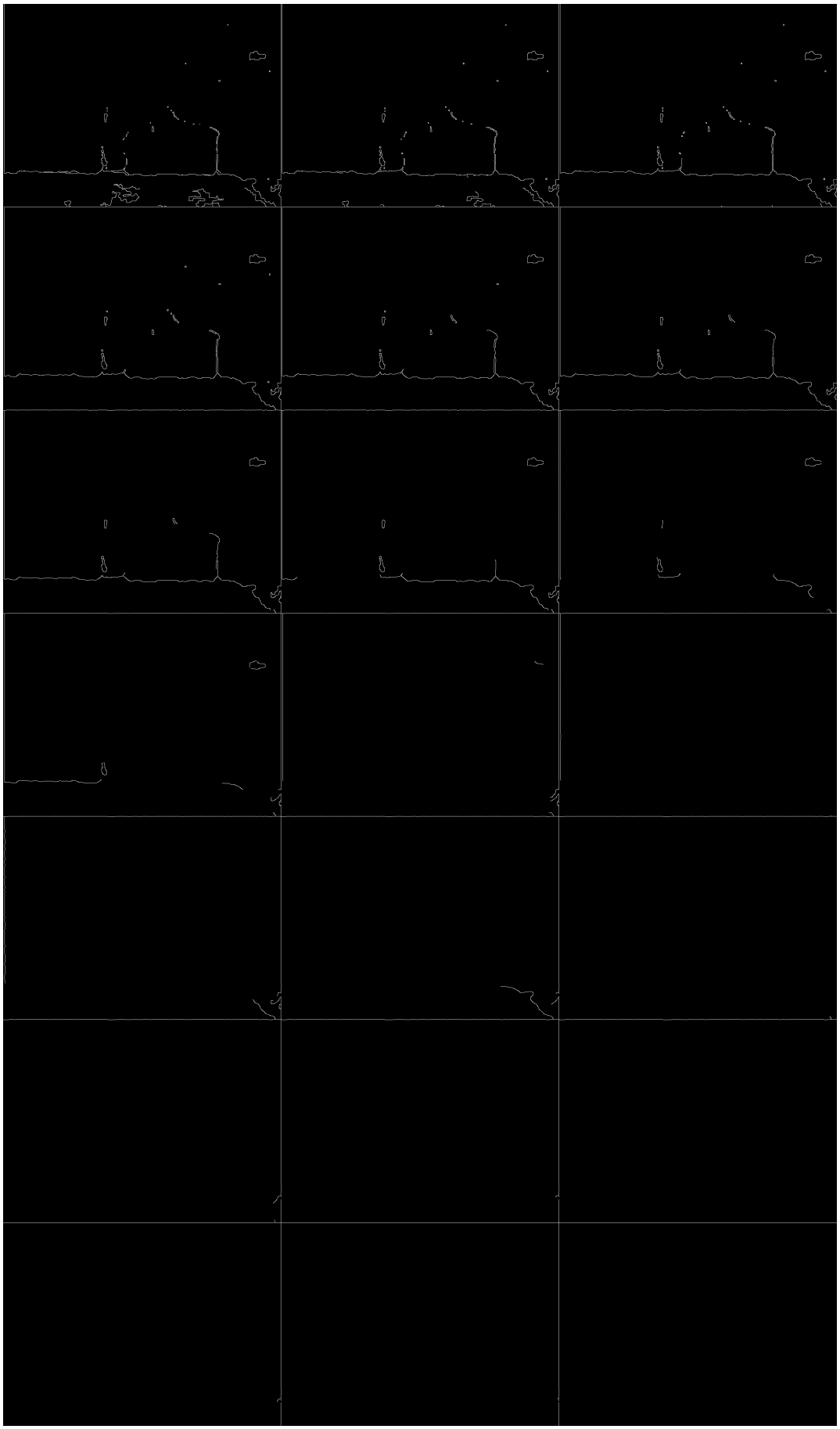


Figure 41: gauss_10_canny_100_300

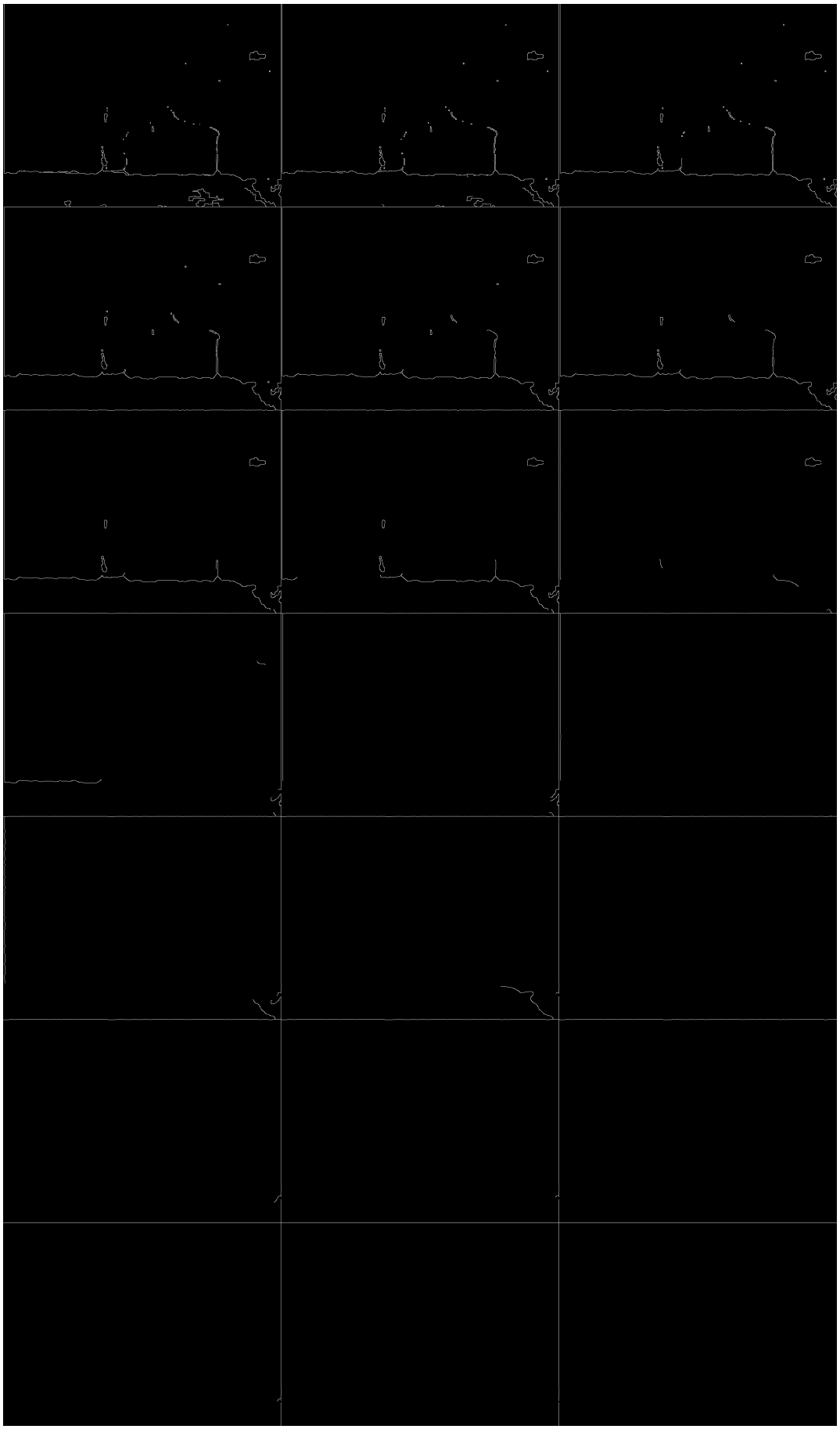


Figure 42: gauss_11_canny_110_330

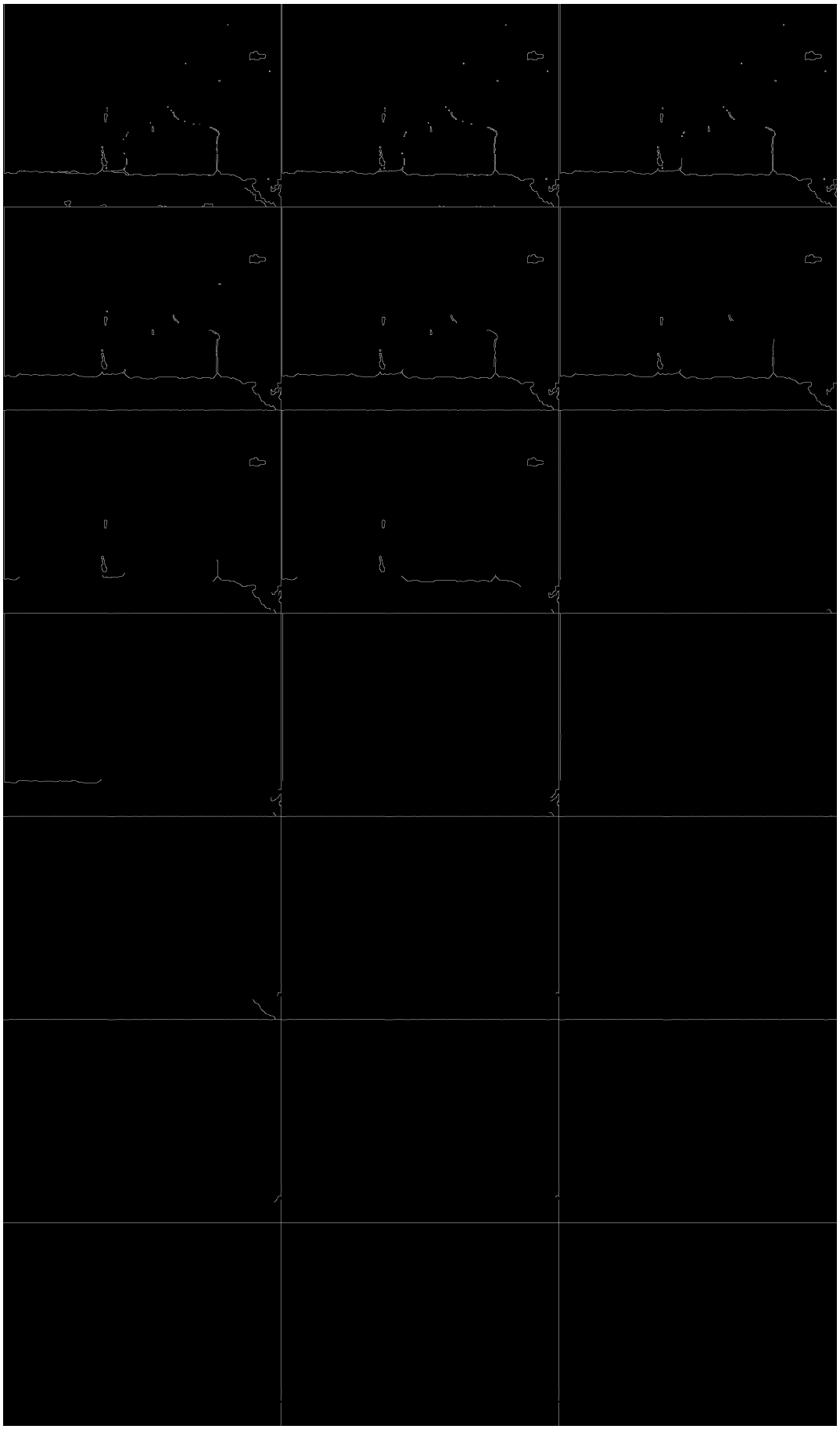


Figure 43: gauss_12_canny_120_360

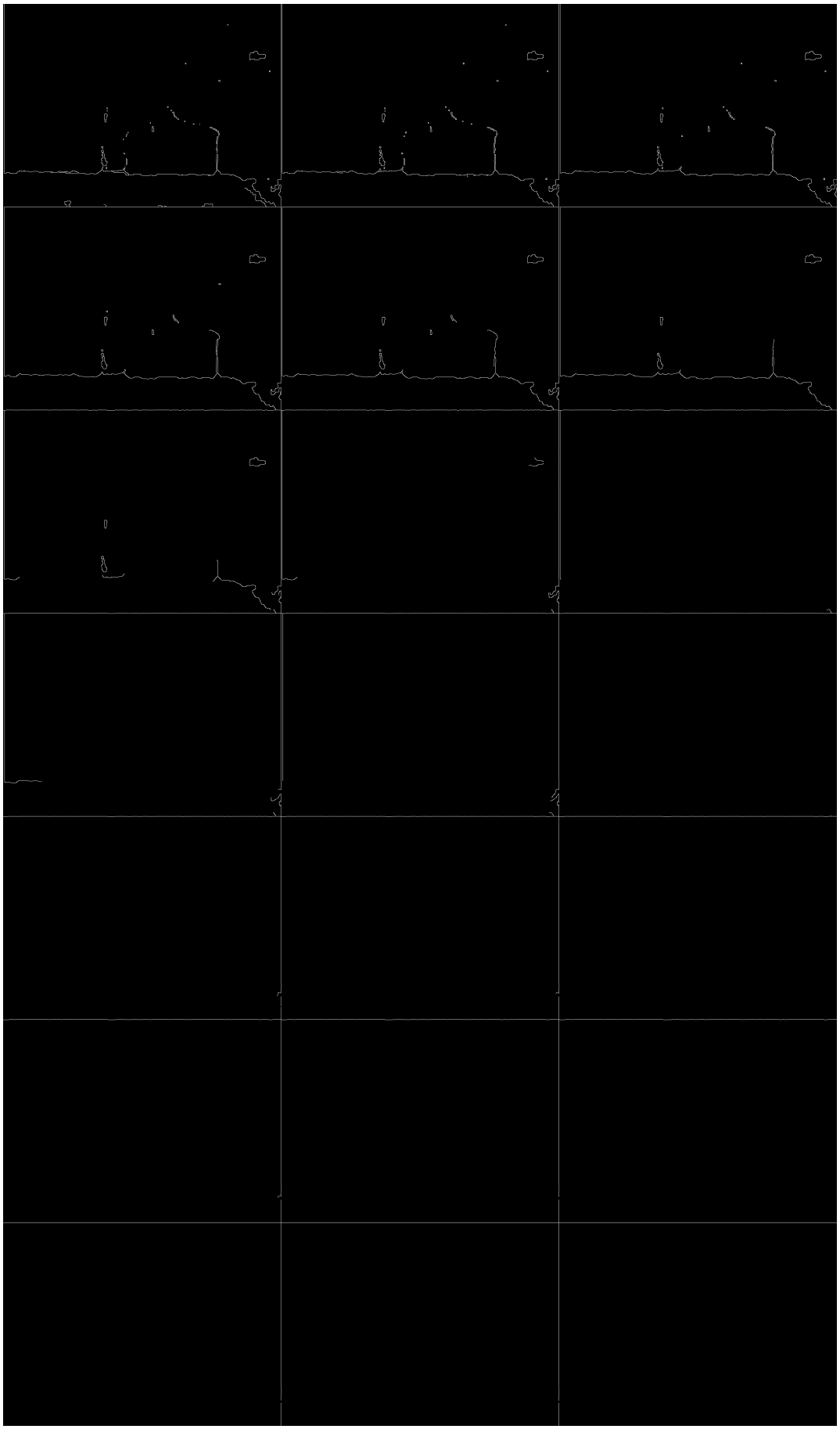


Figure 44: gauss_13_canny_130_390

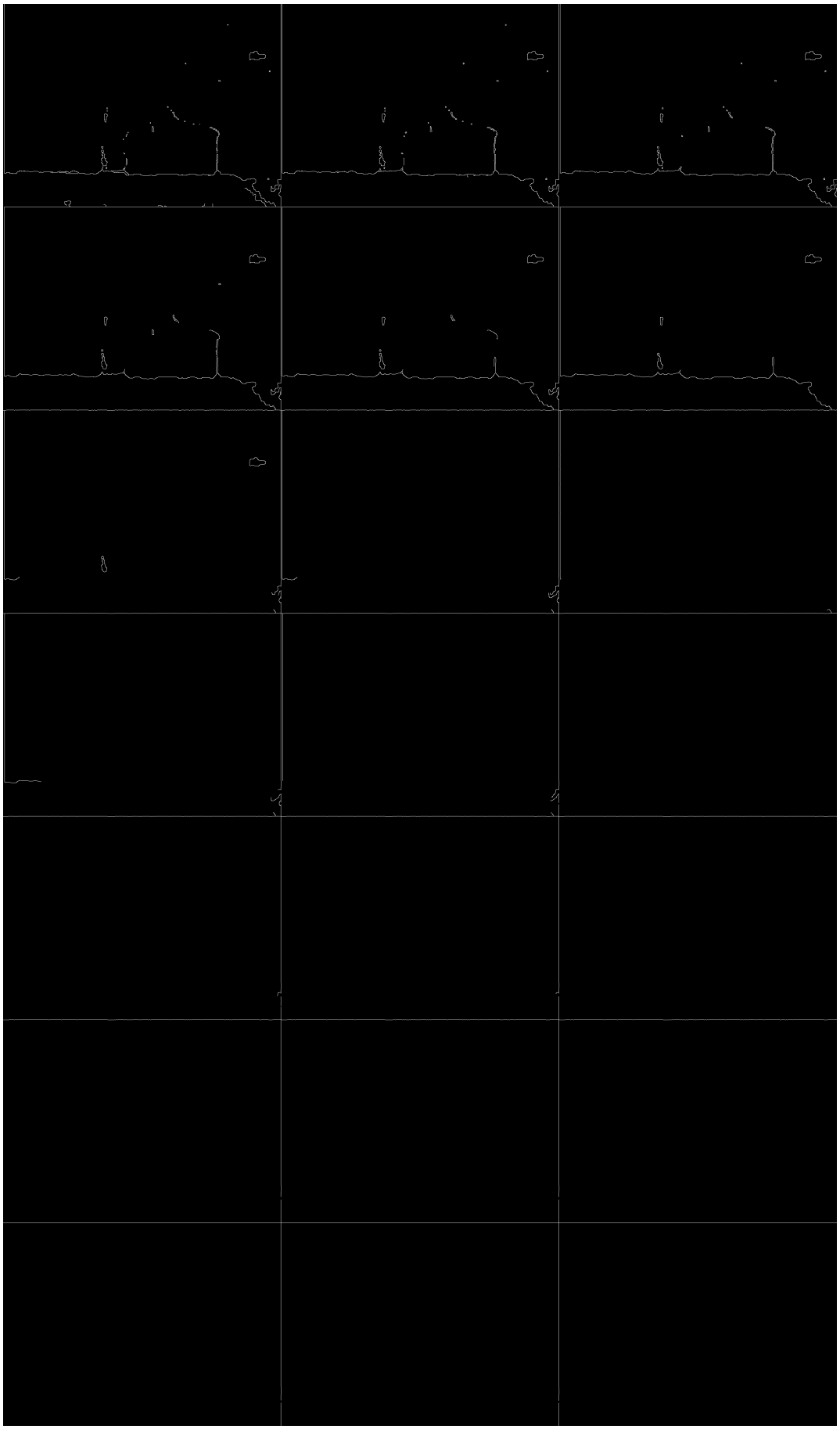


Figure 45: gauss_14_canny_140_420

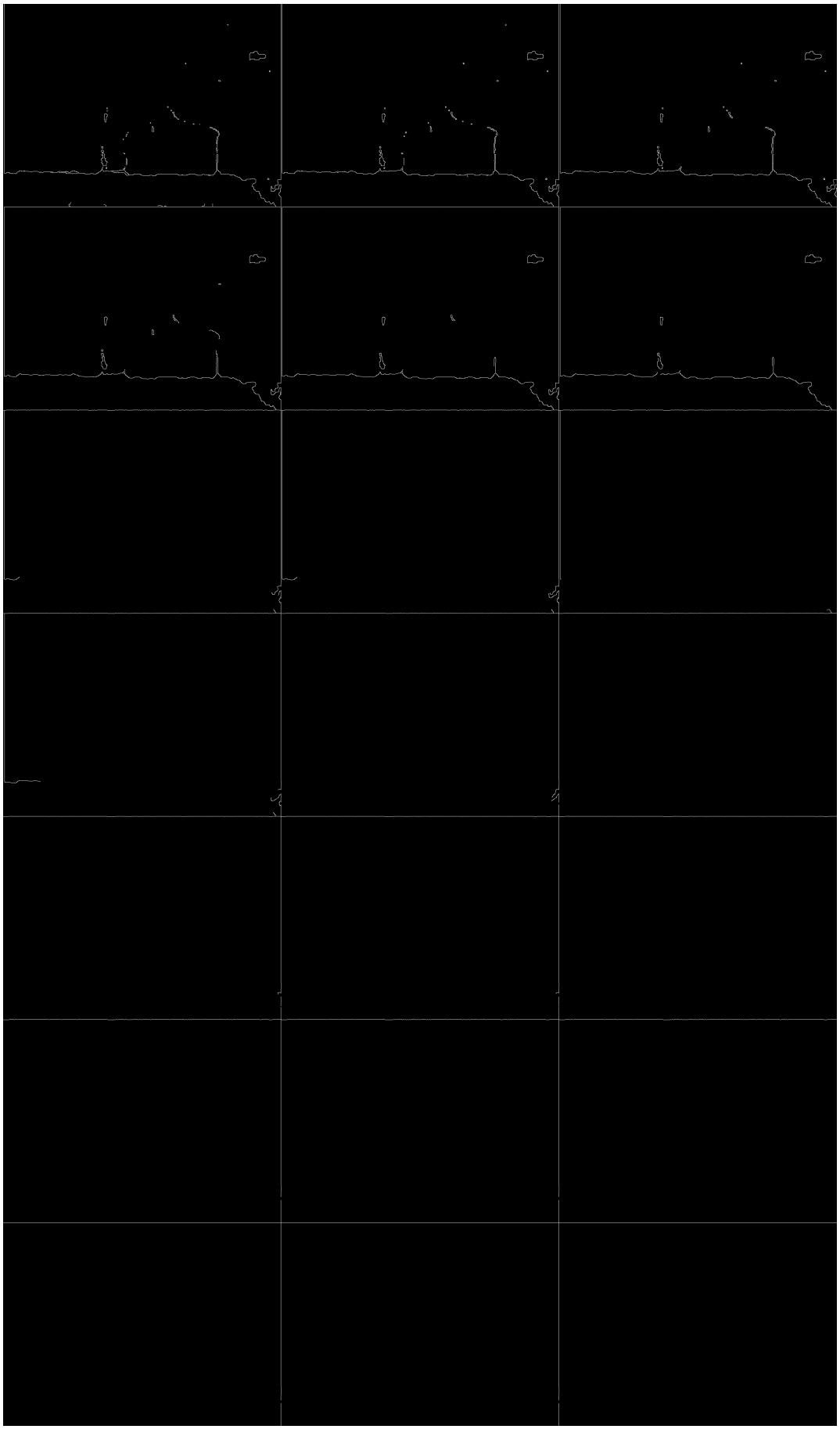


Figure 46: gauss_15_canny_150_450

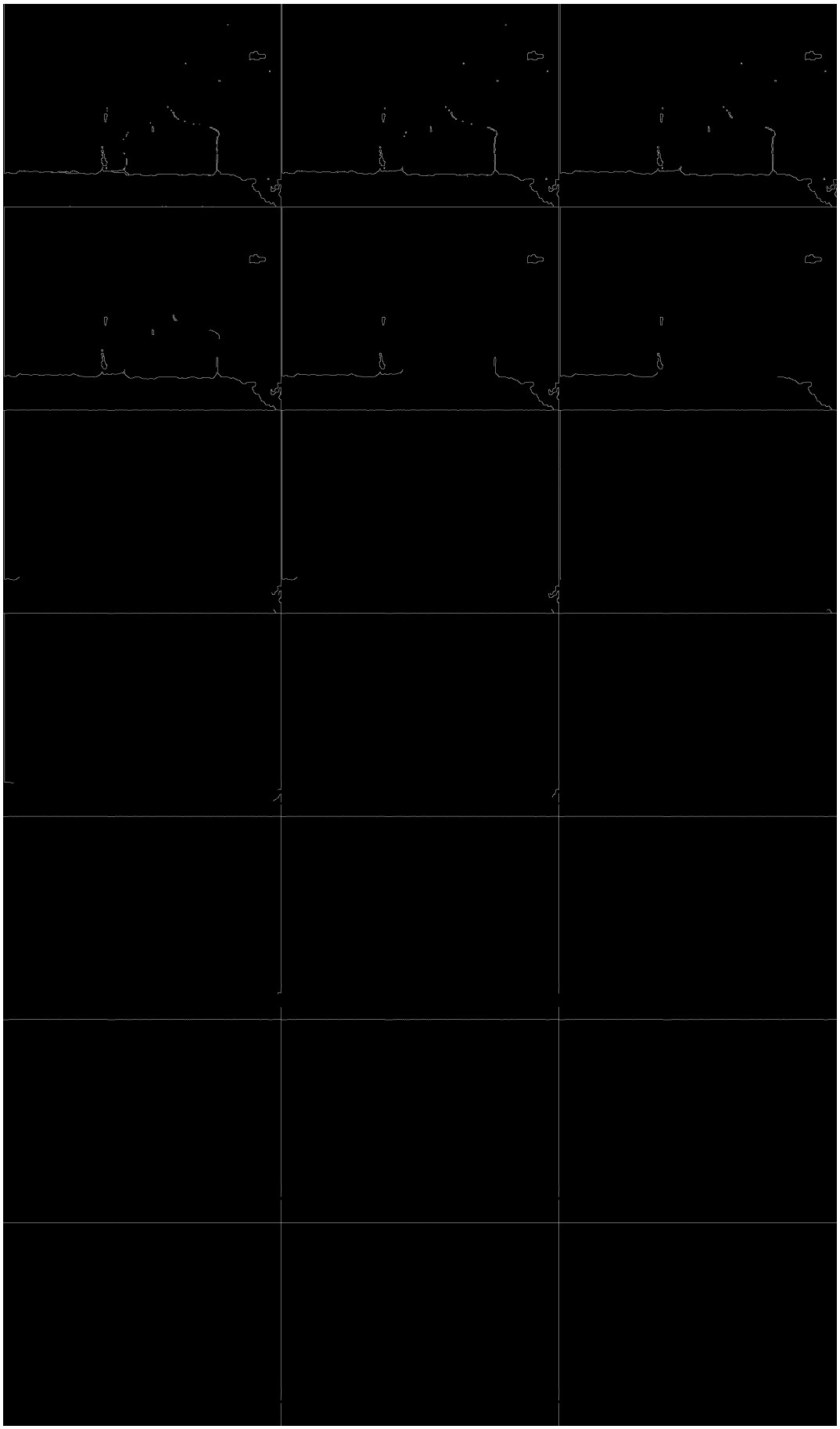


Figure 47: gauss_16_canny_160_480

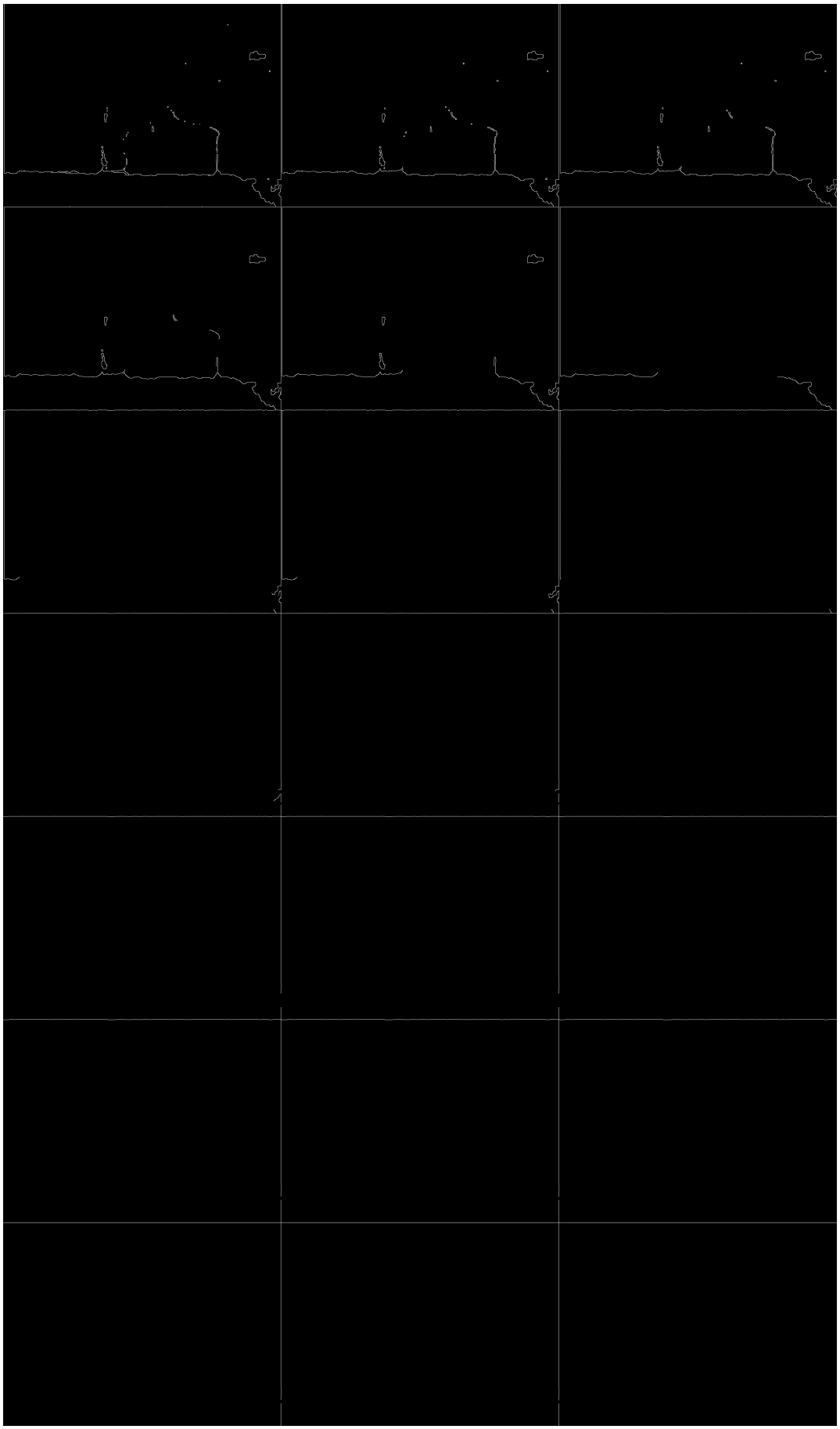


Figure 48: gauss_17_canny_170_510

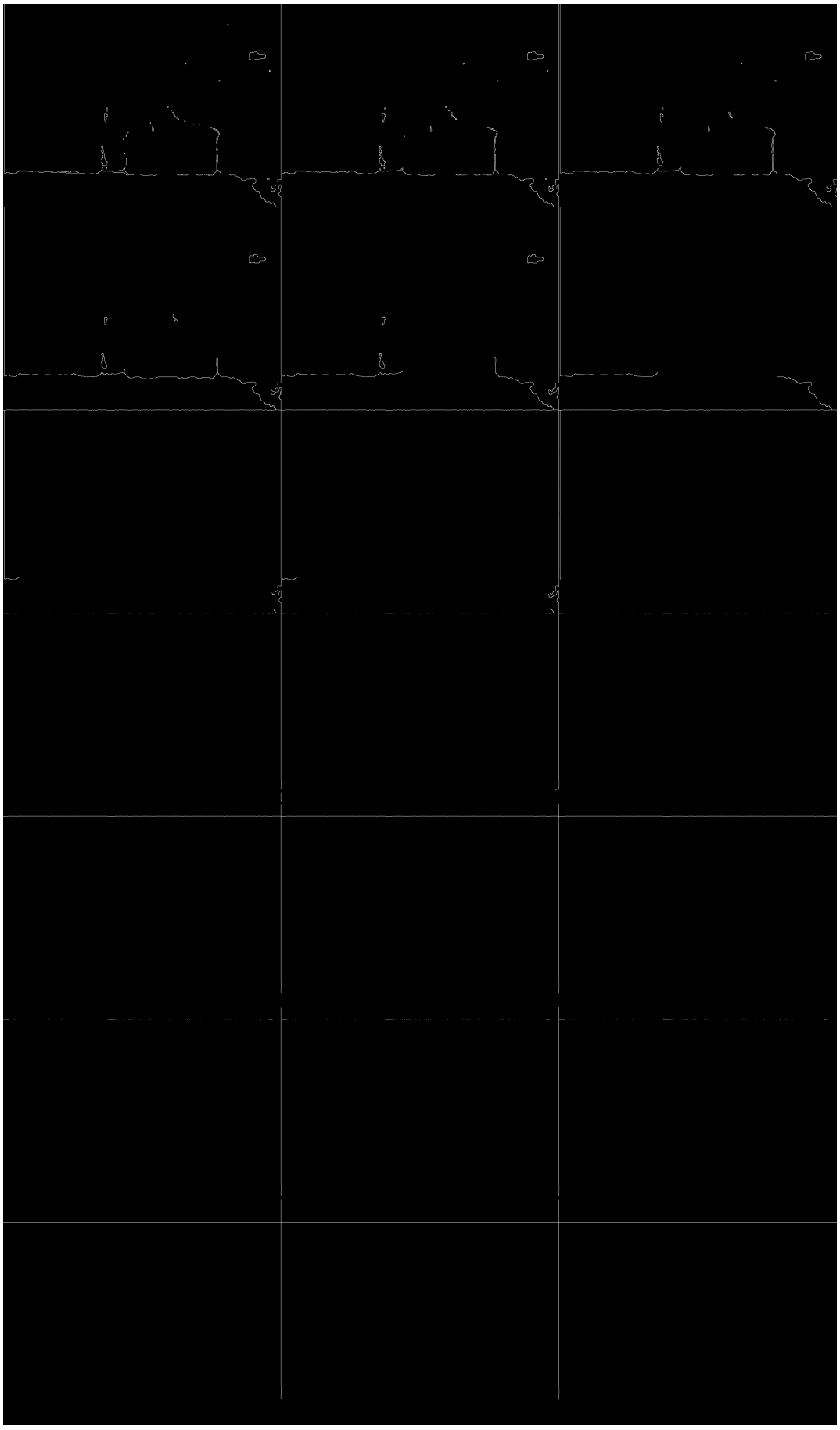


Figure 49: gauss_18_canny_180_540

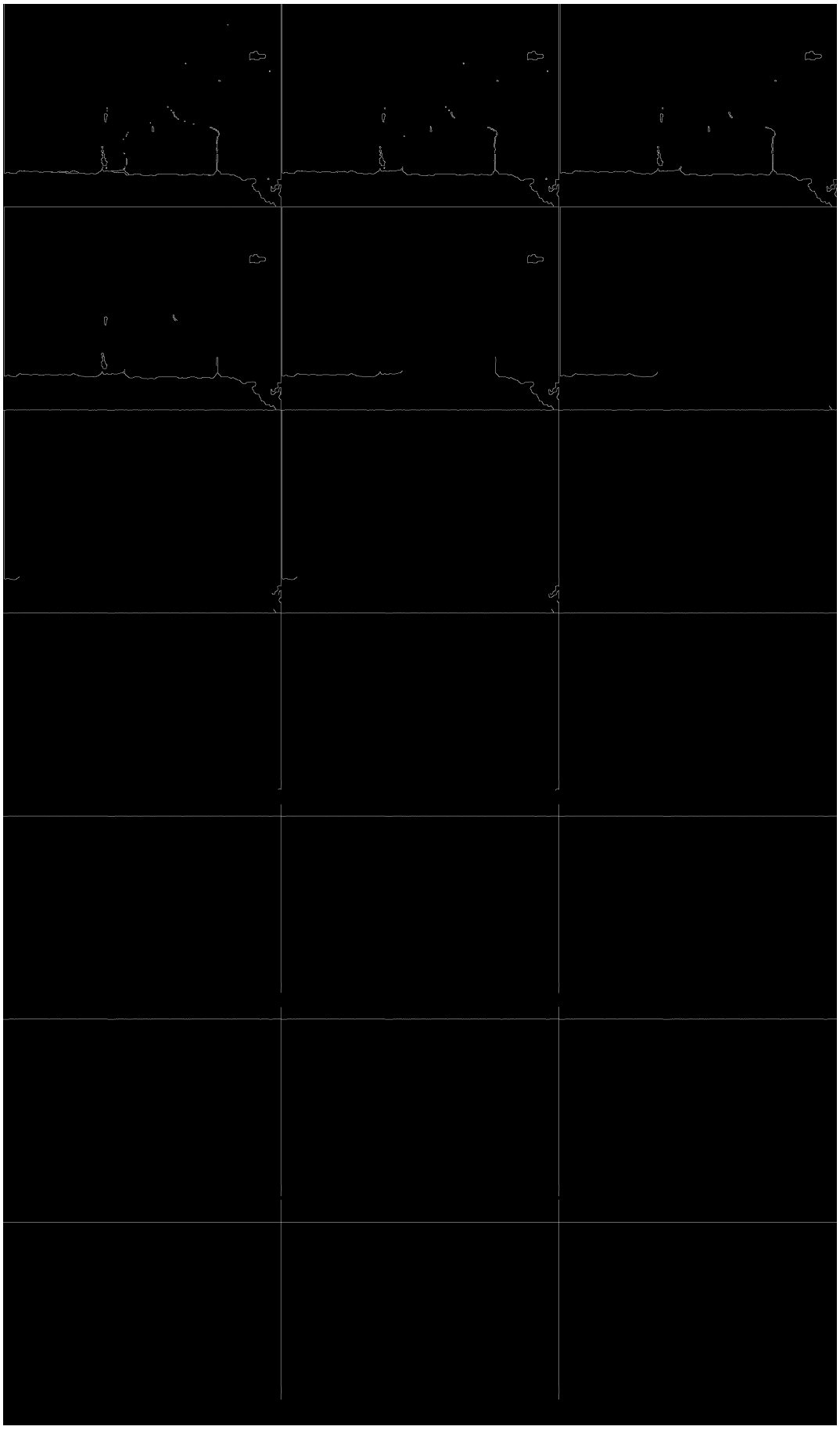


Figure 50: gauss_19_canny_190_570

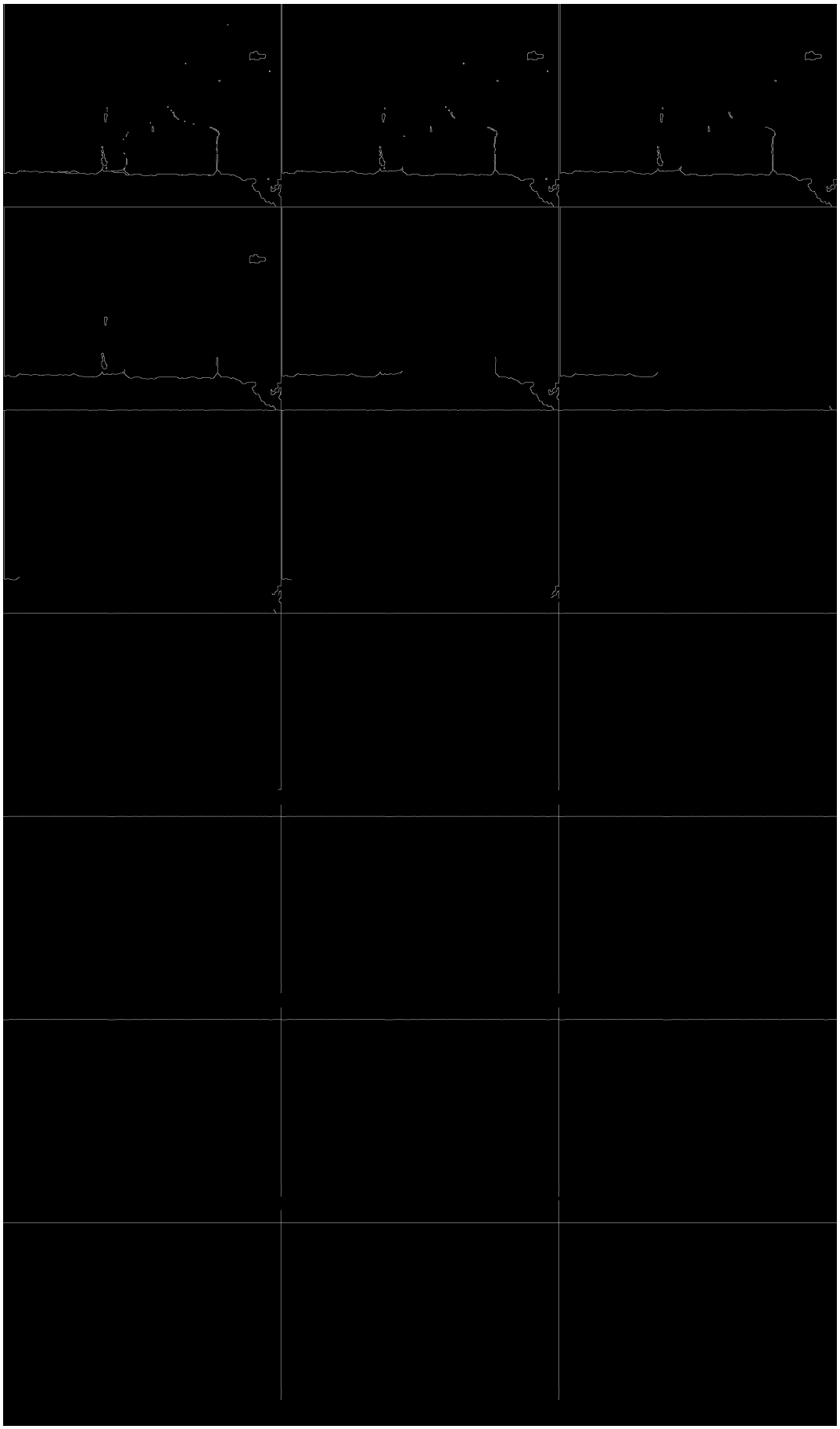


Figure 51: gauss_20_canny_200_600

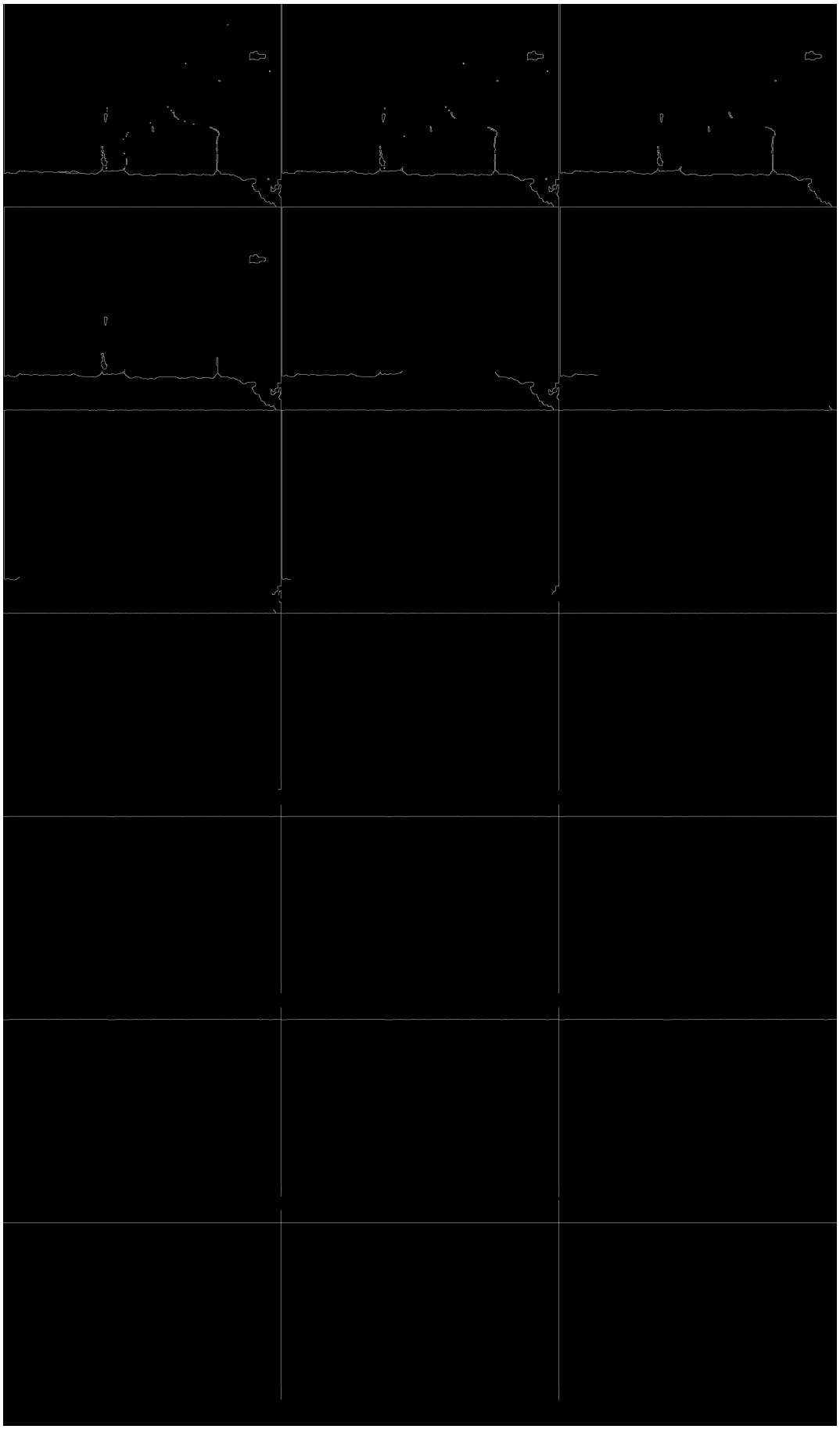


Figure 52: gauss_21_canny_210_630

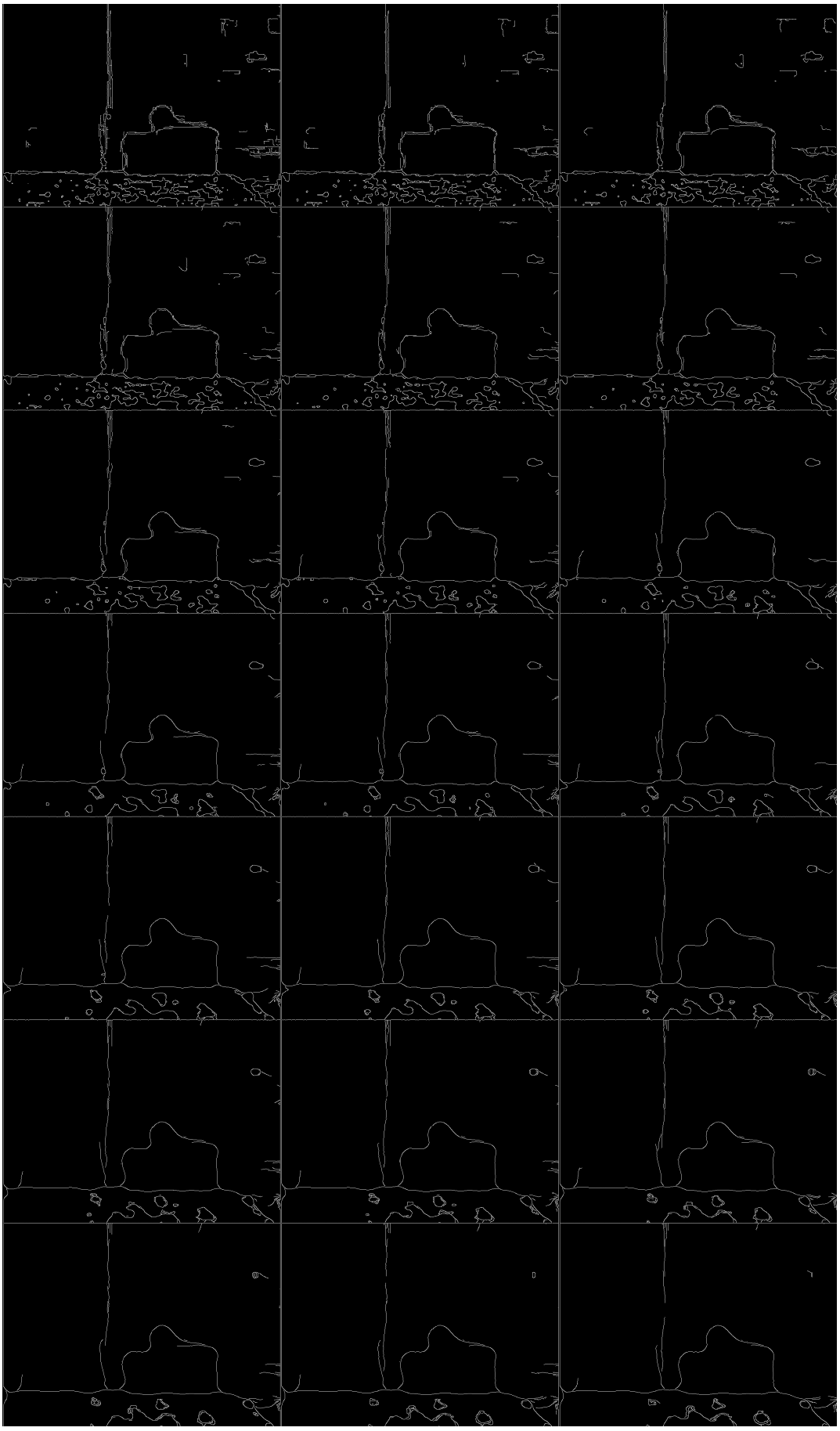


Figure 53: median_01_canny_10_30

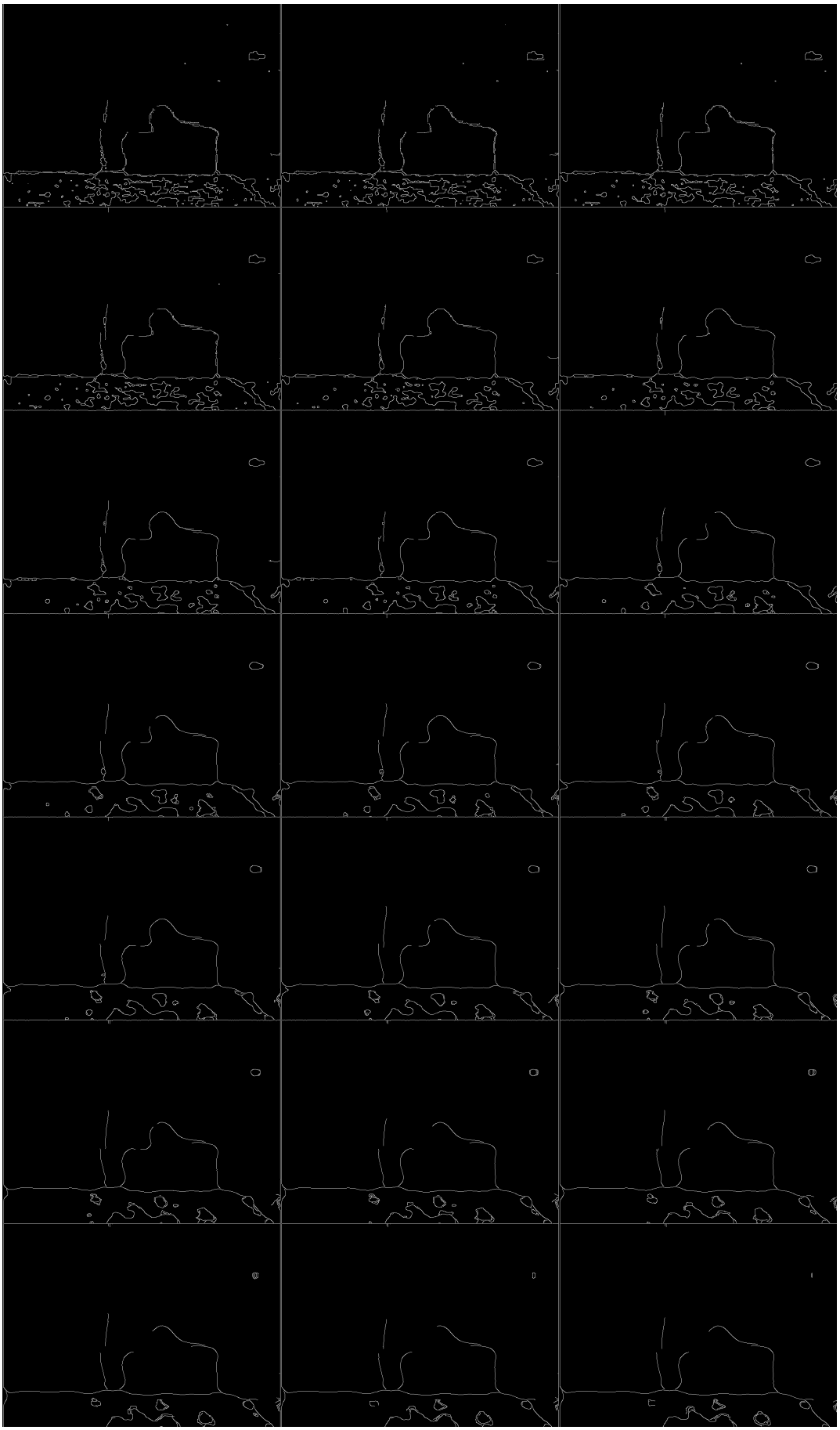


Figure 54: median_02_canny_20_60

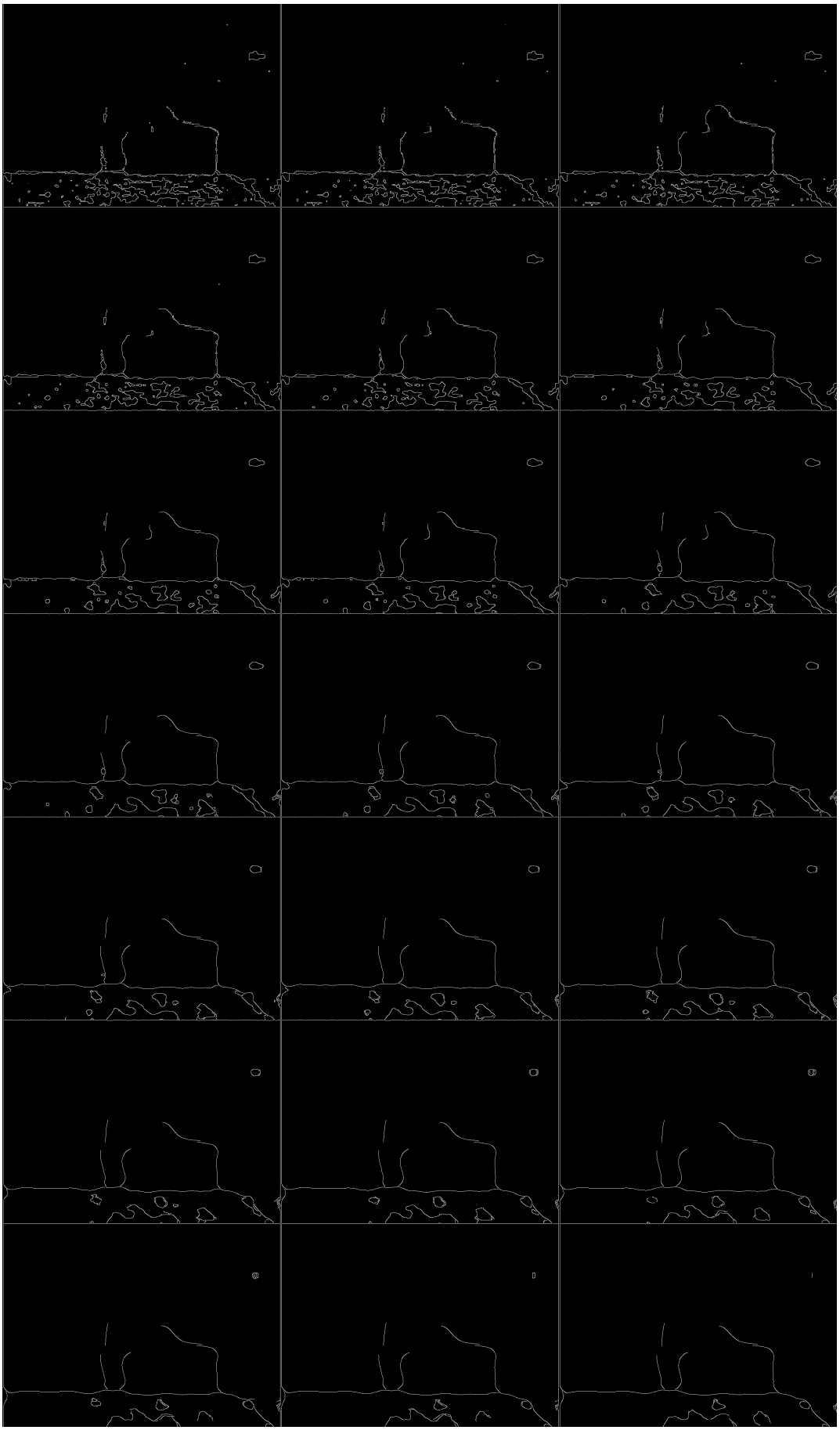


Figure 55: median_03_canny_30_90

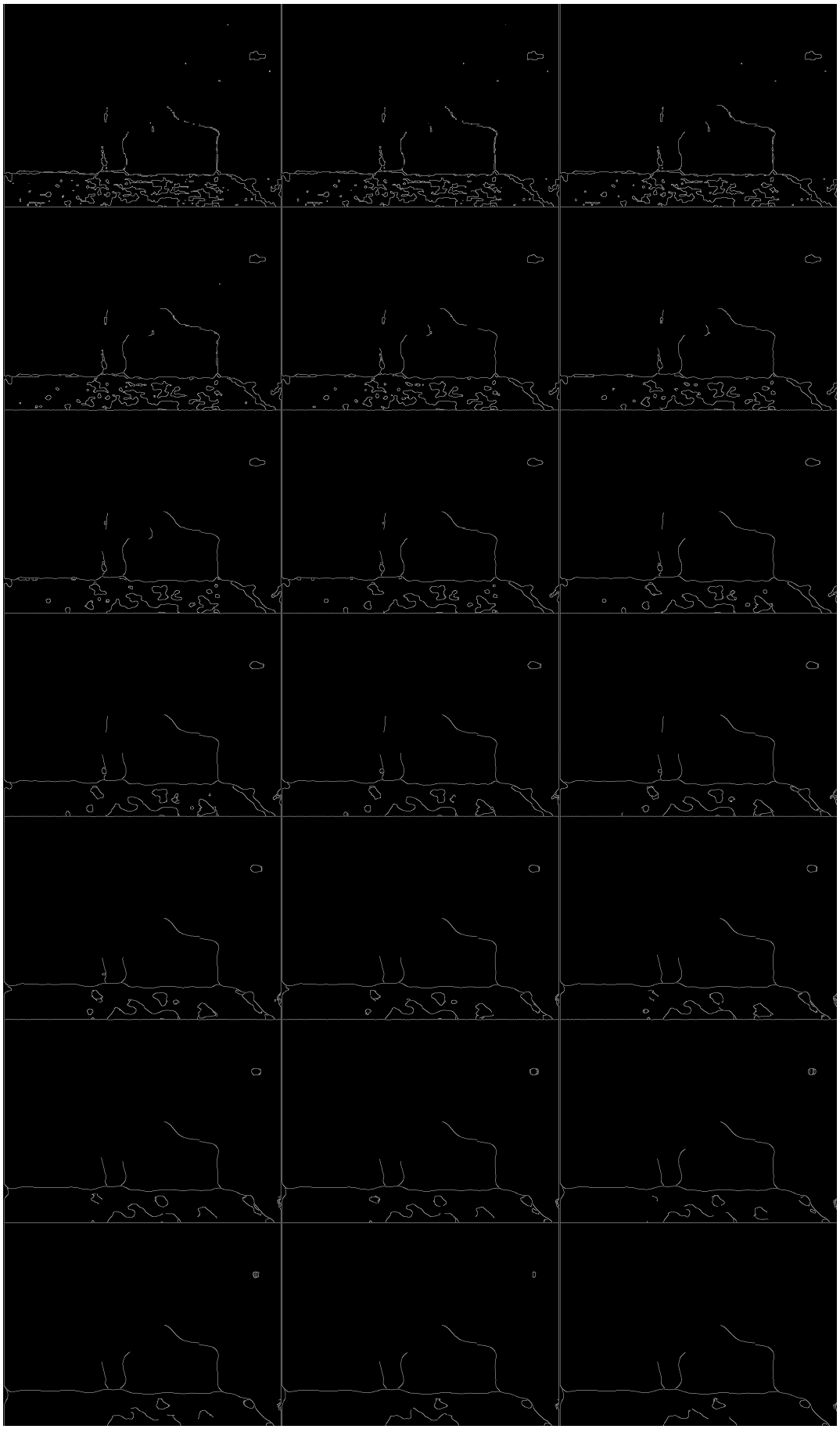


Figure 56: median_04_canny_40_120

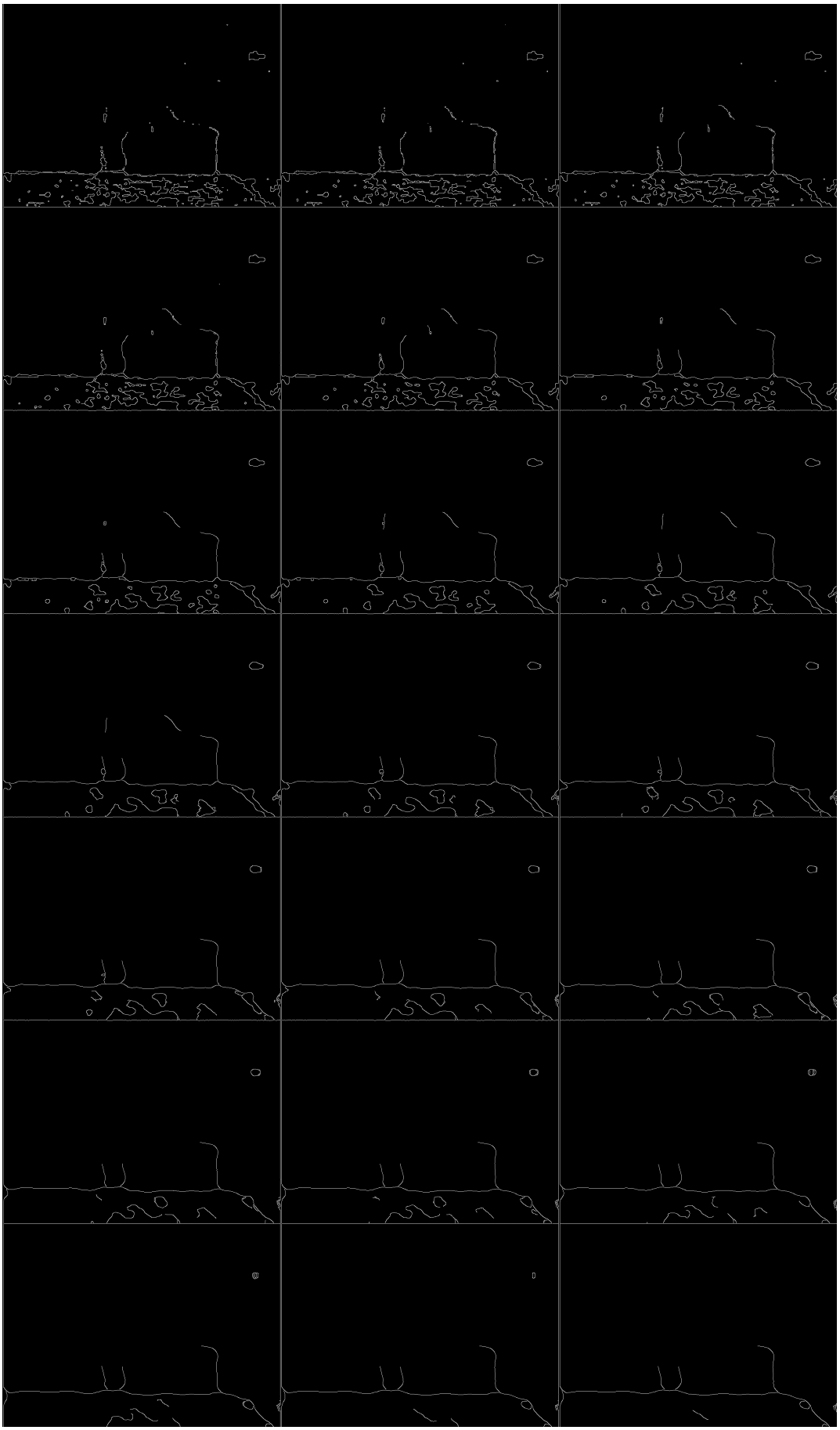


Figure 57: median_05_canny_50_150

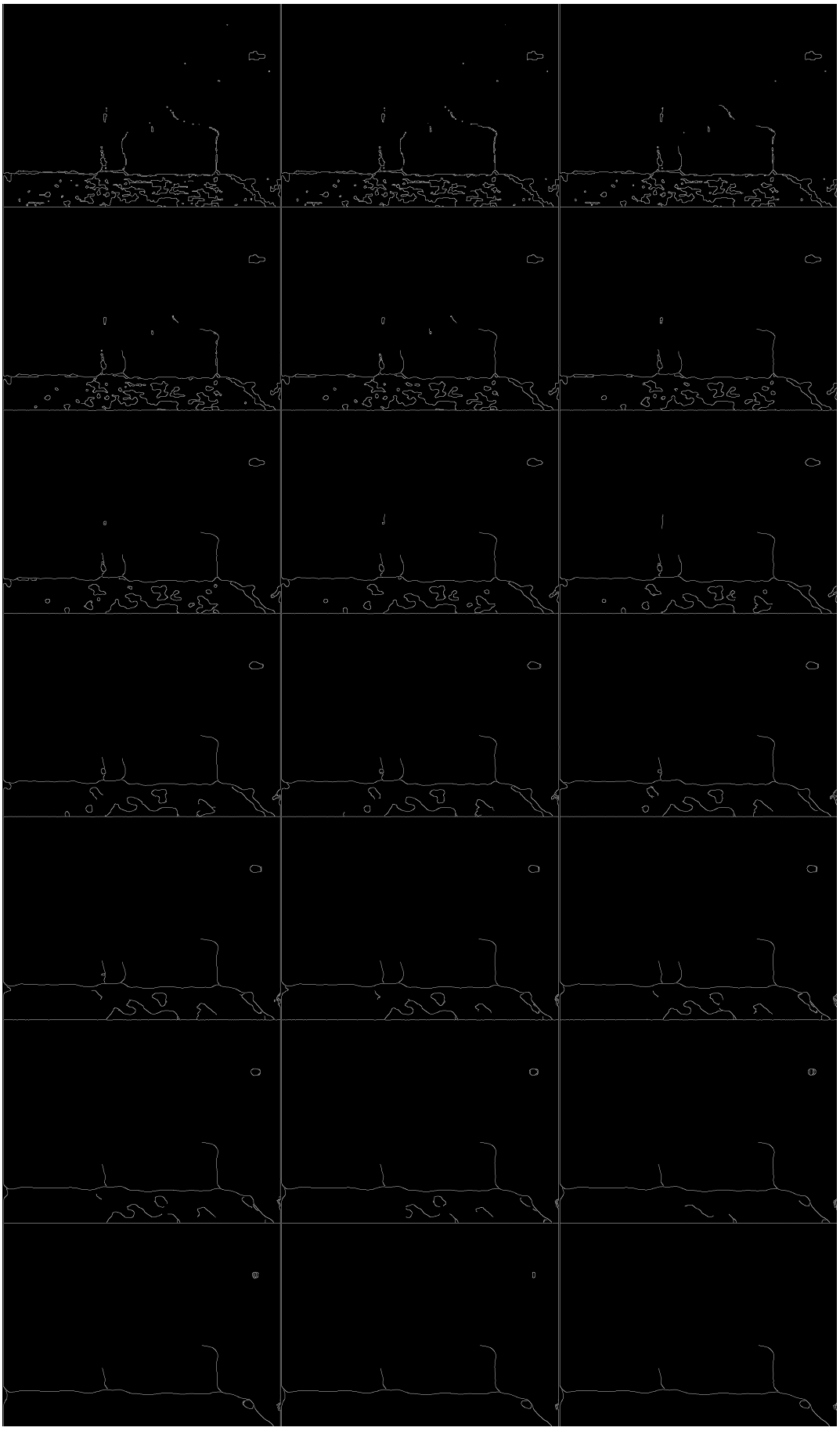


Figure 58: median_06_canny_60_180

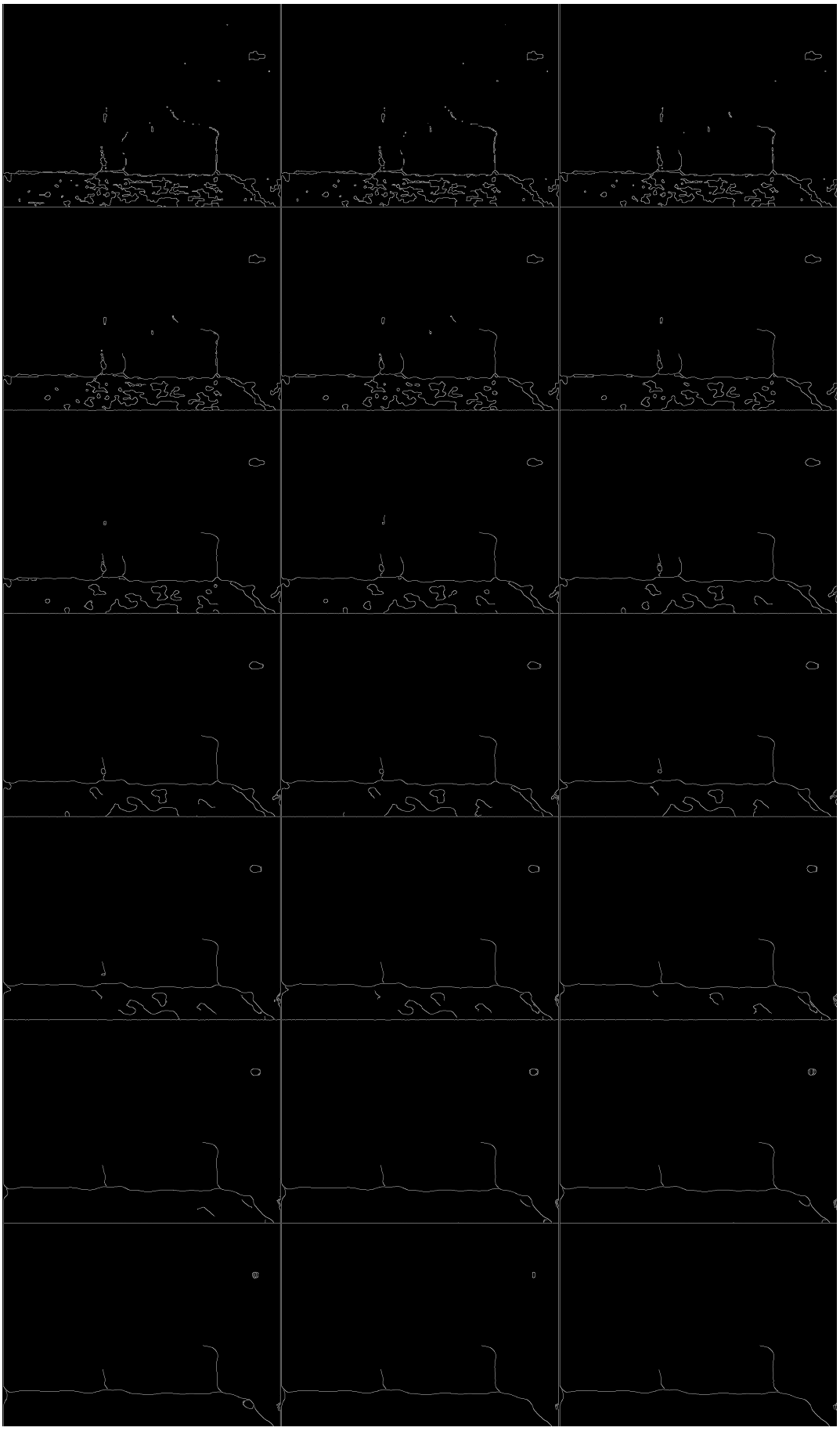


Figure 59: median_07_canny_70_210

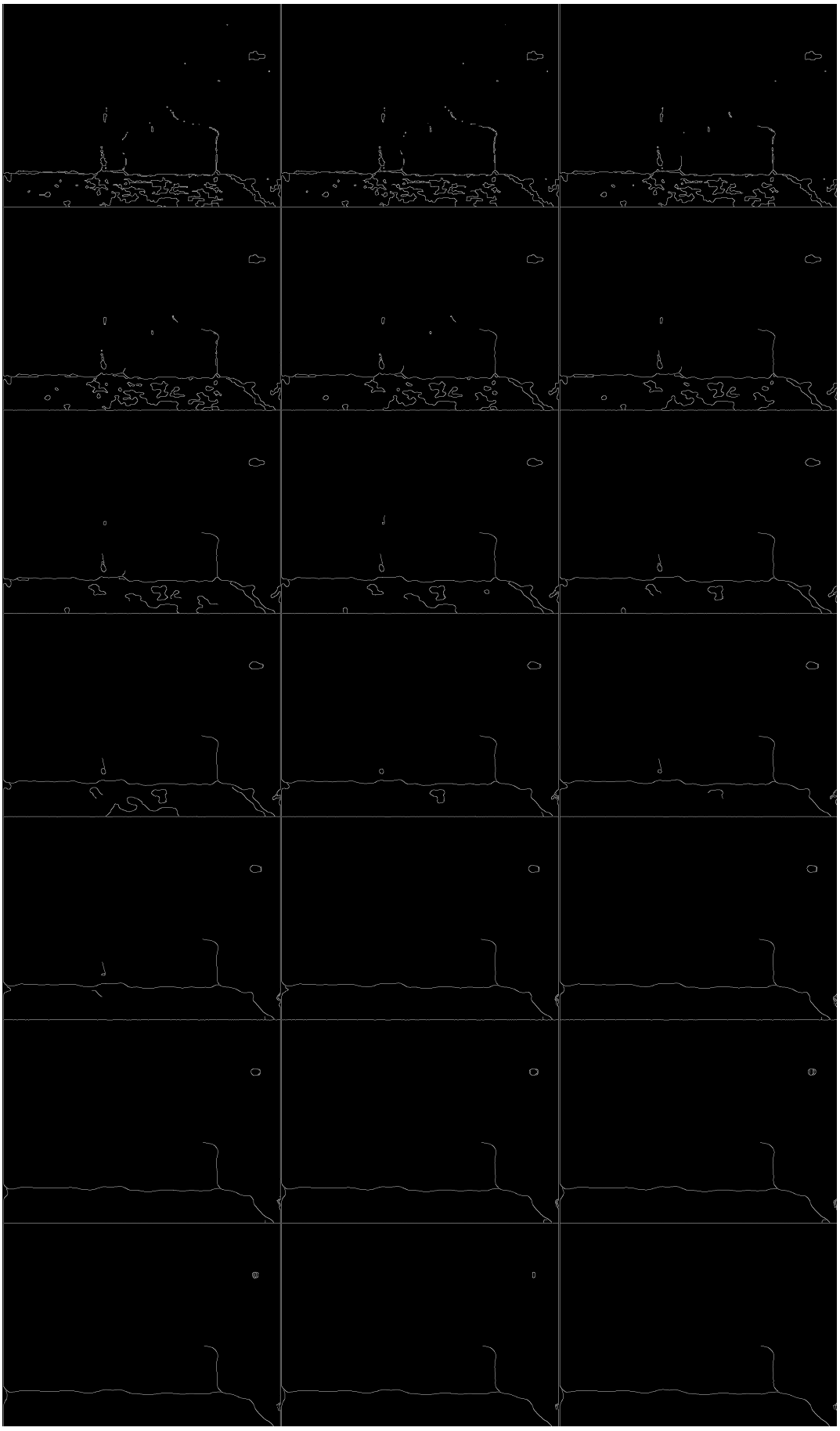


Figure 60: median_08_canny_80_240

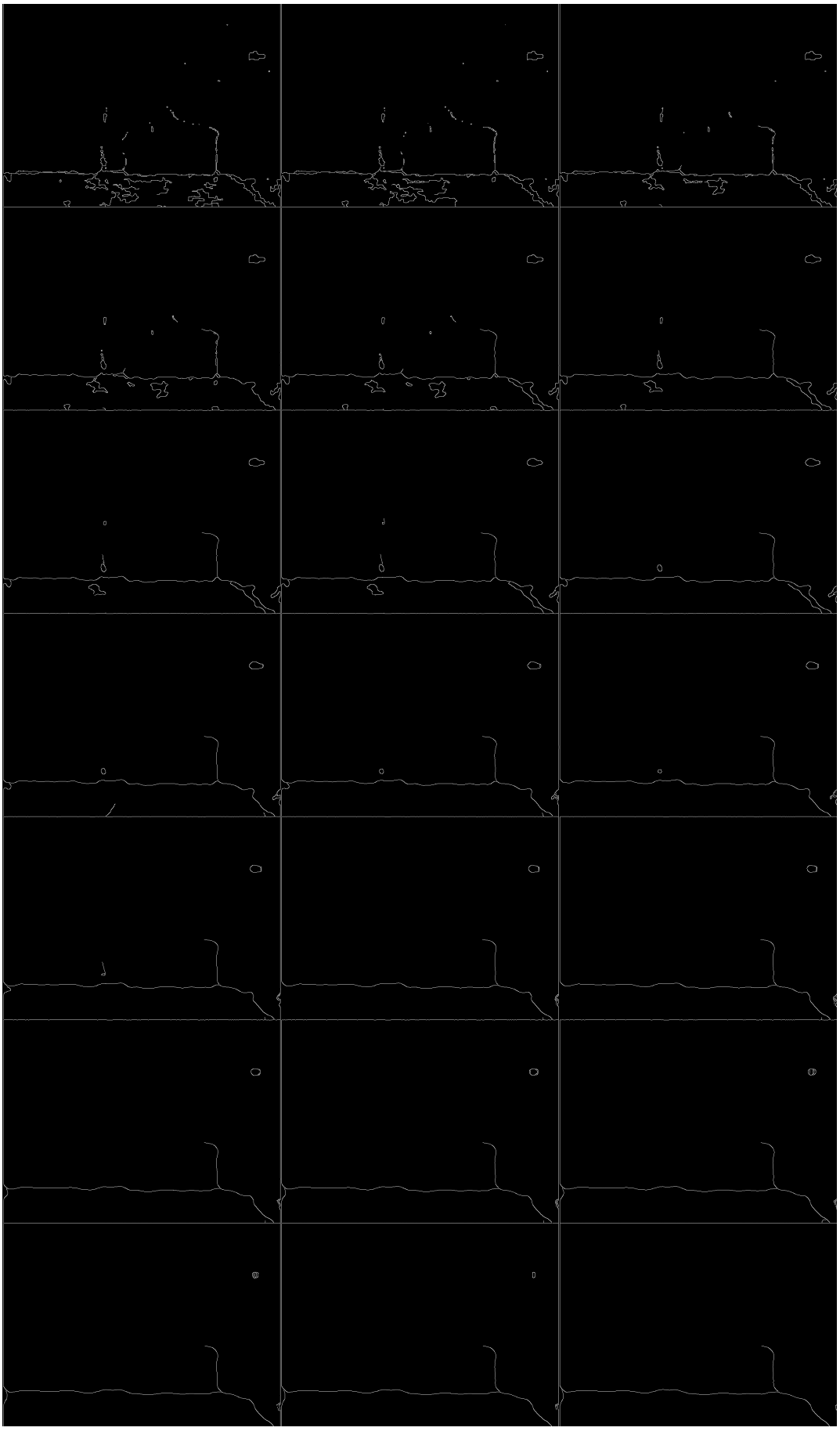


Figure 61: `median_09_canny_90_270`

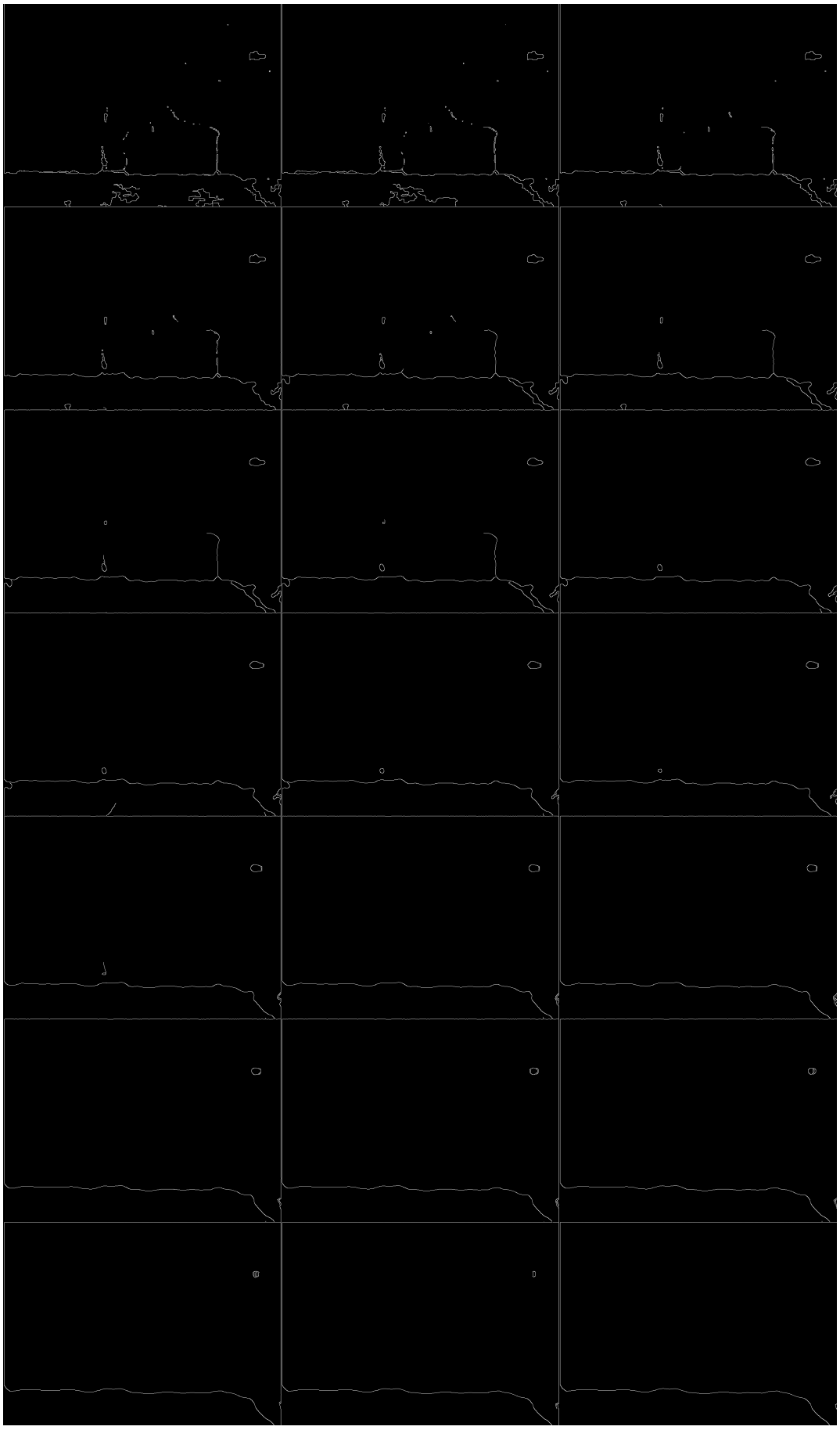


Figure 62: median_10_canny_100_300

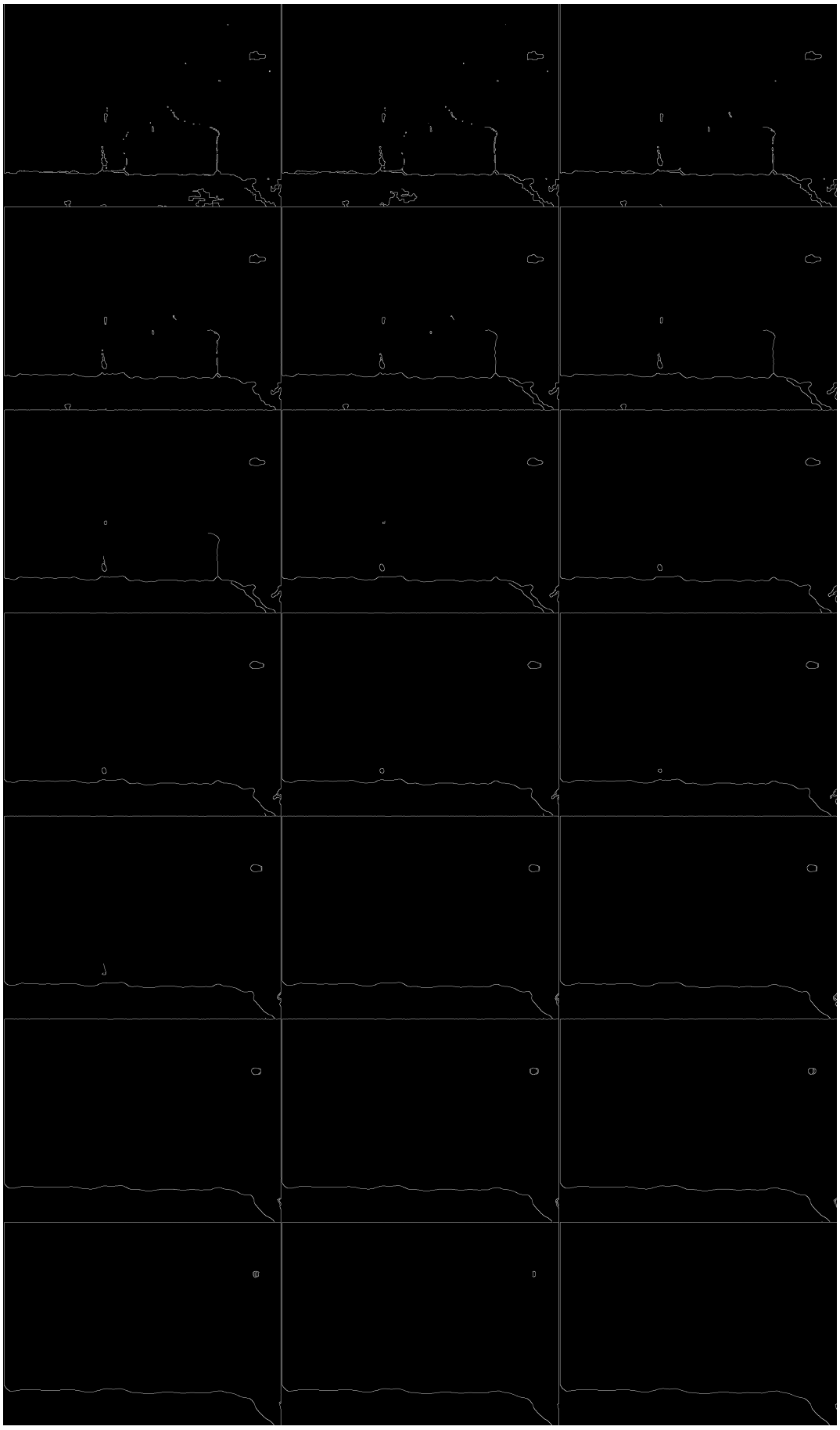


Figure 63: median_11_canny_110_330

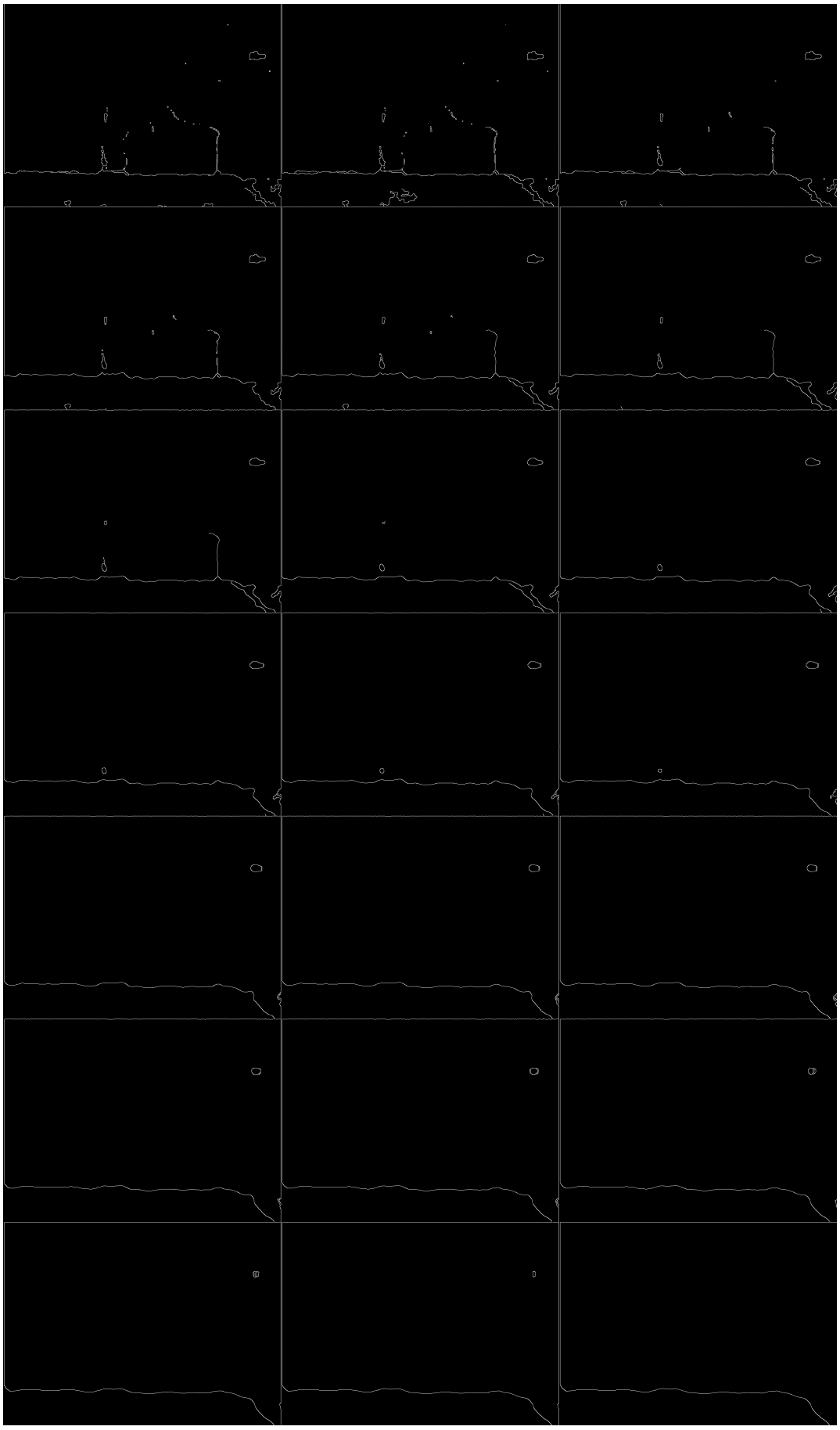


Figure 64: median_12_canny_120_360

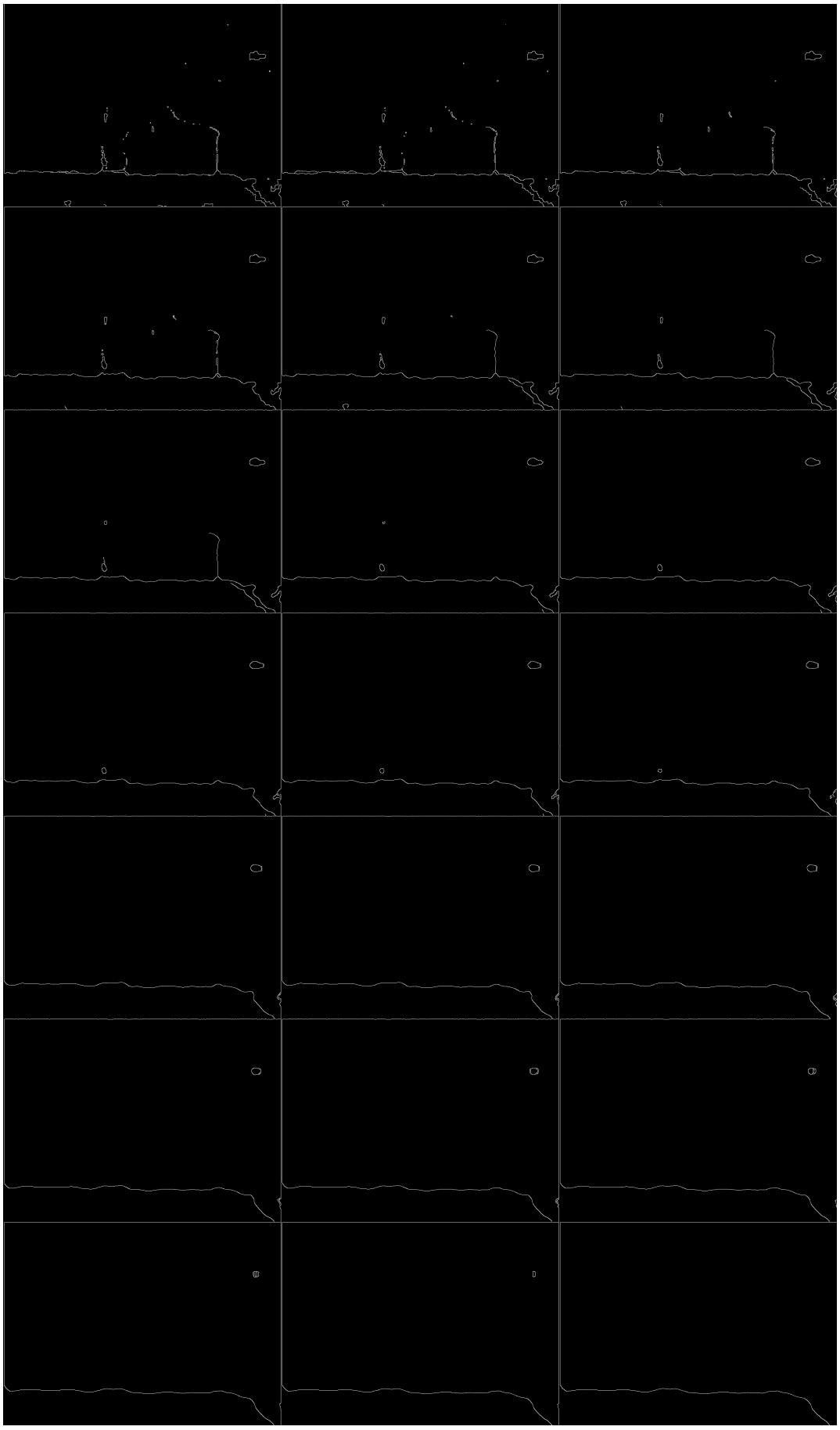


Figure 65: median_13_canny_130_390

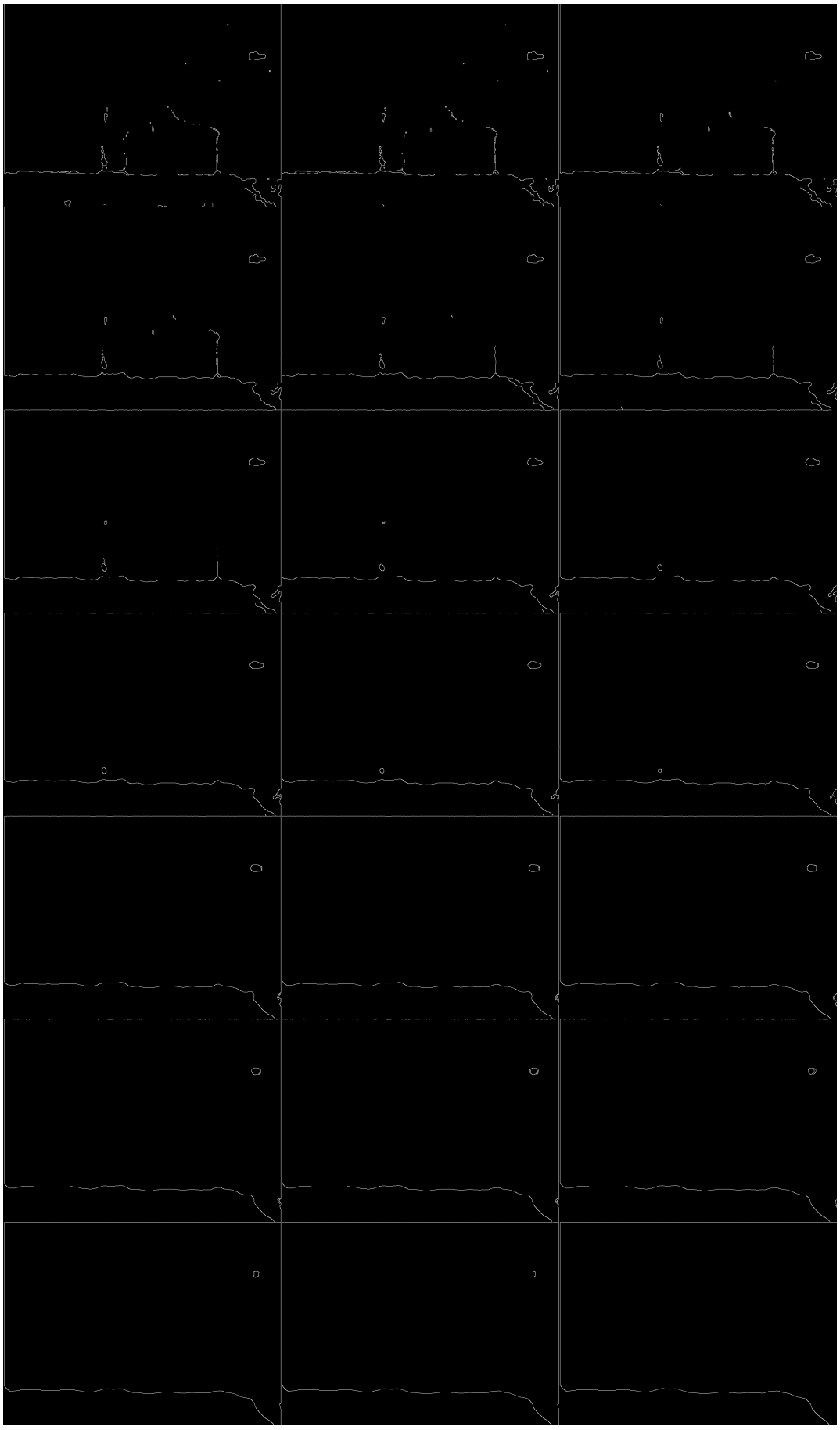


Figure 66: median_14_canny_140_420

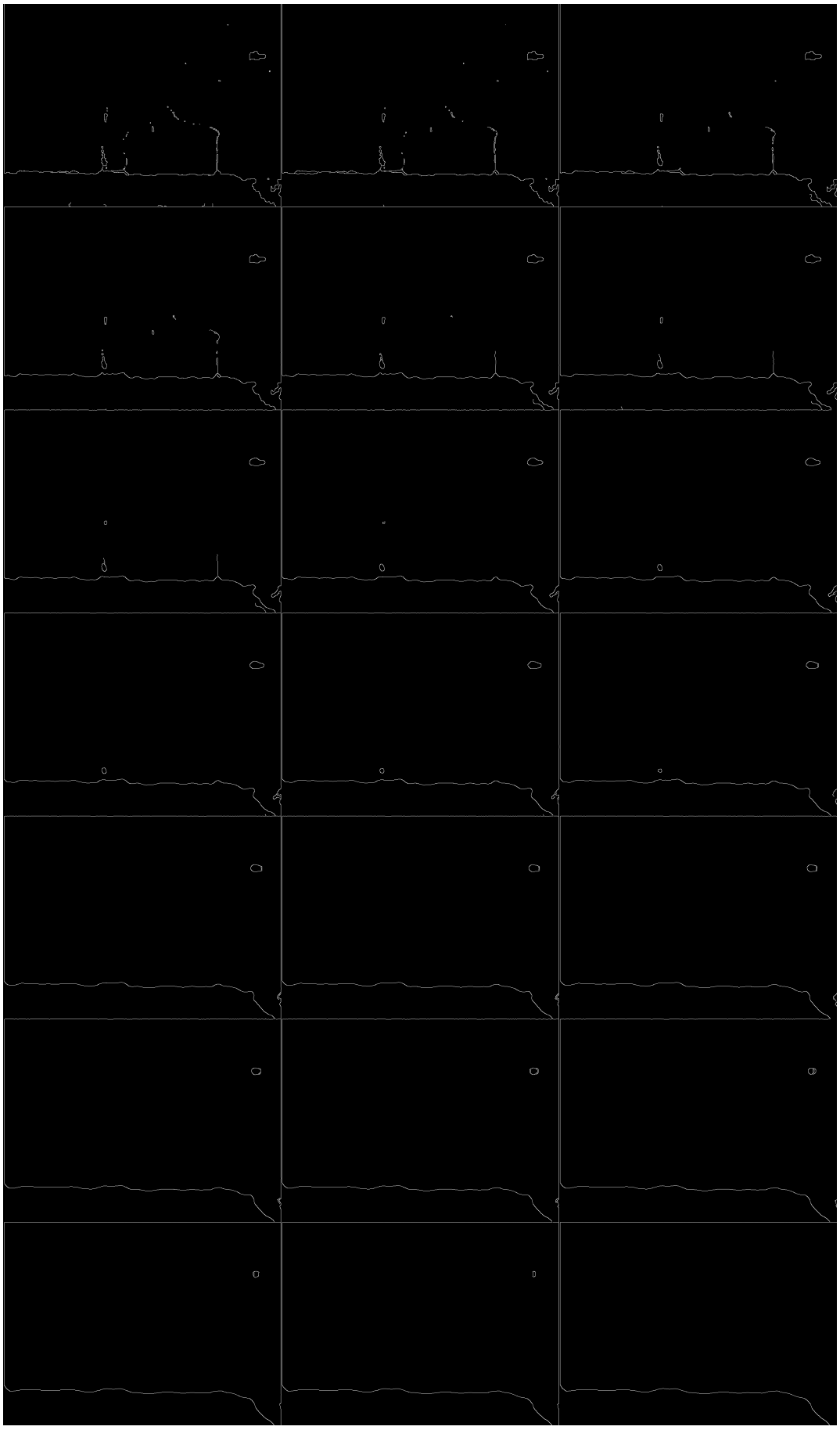


Figure 67: median_15_canny_150_450

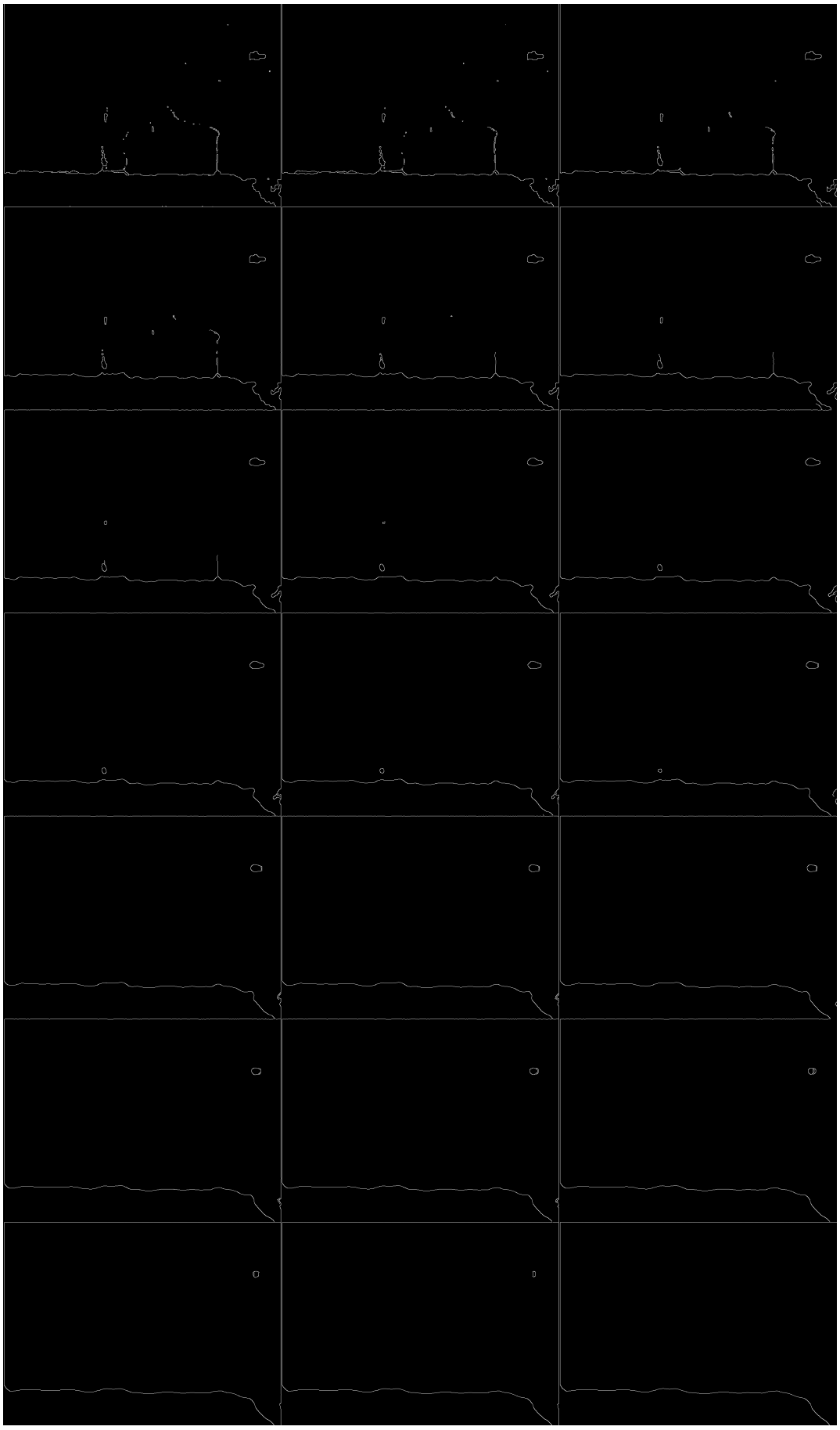


Figure 68: median_16_canny_160_480

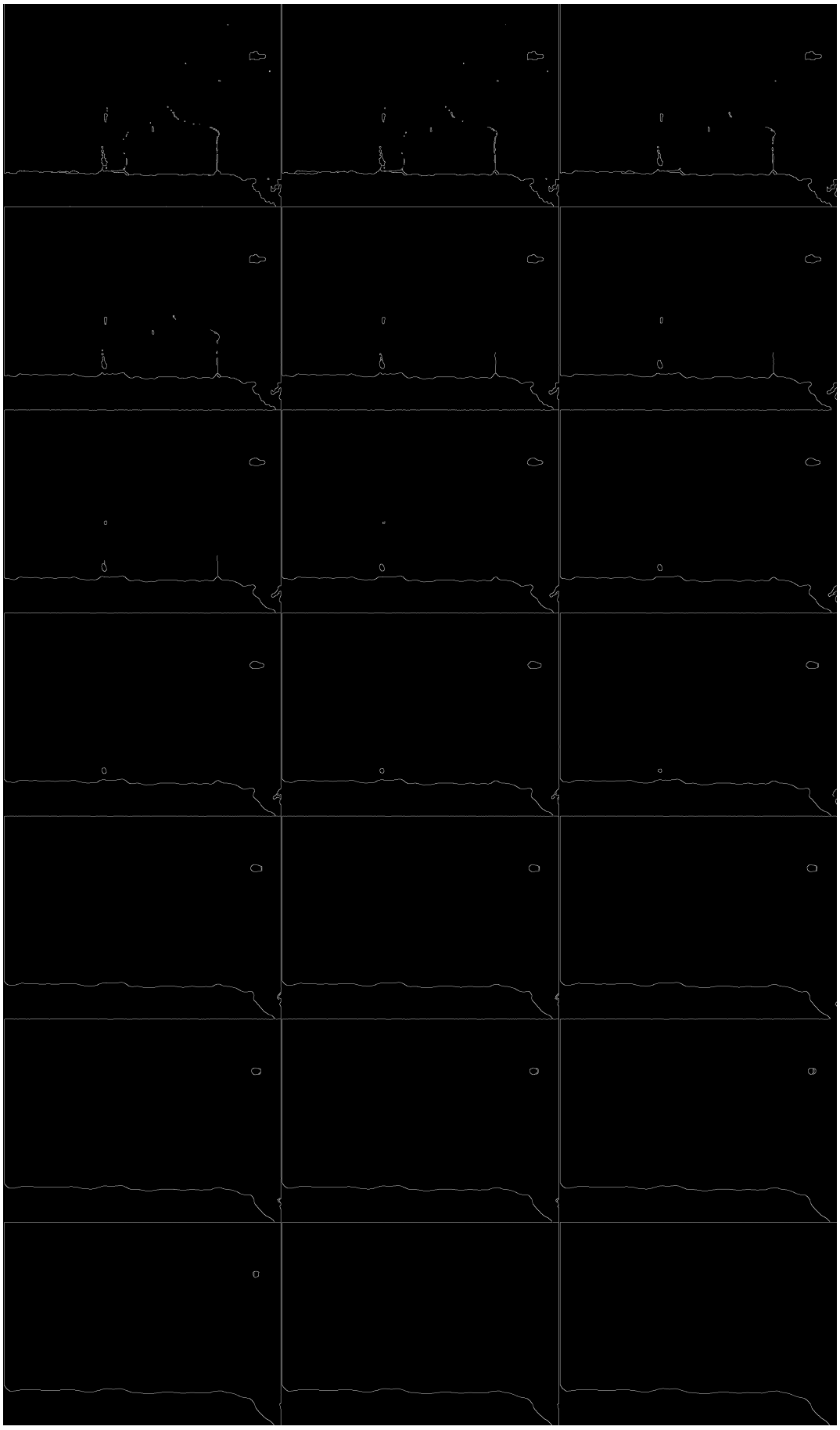


Figure 69: median_17_canny_170_510

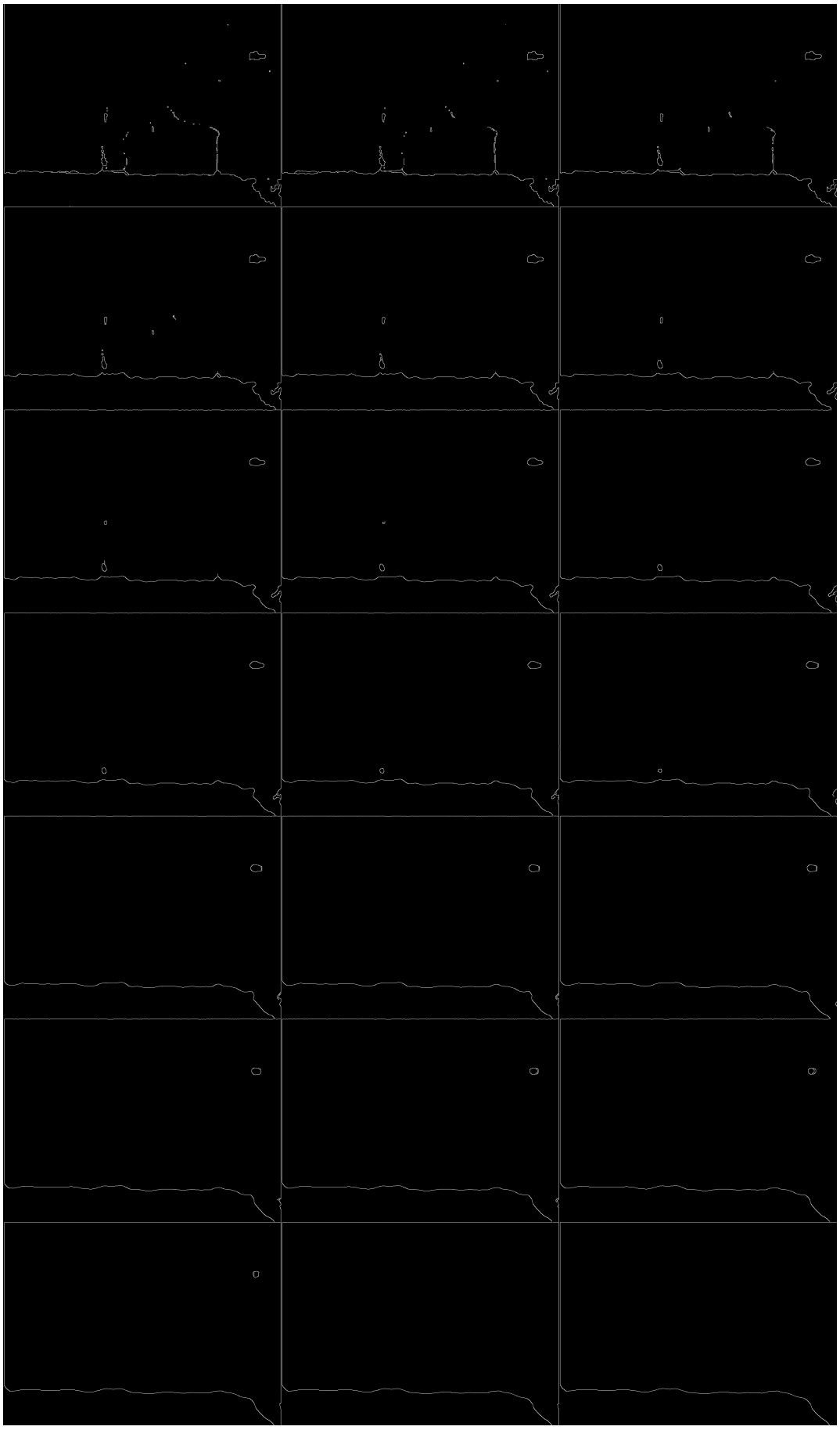


Figure 70: median_18_canny_180_540

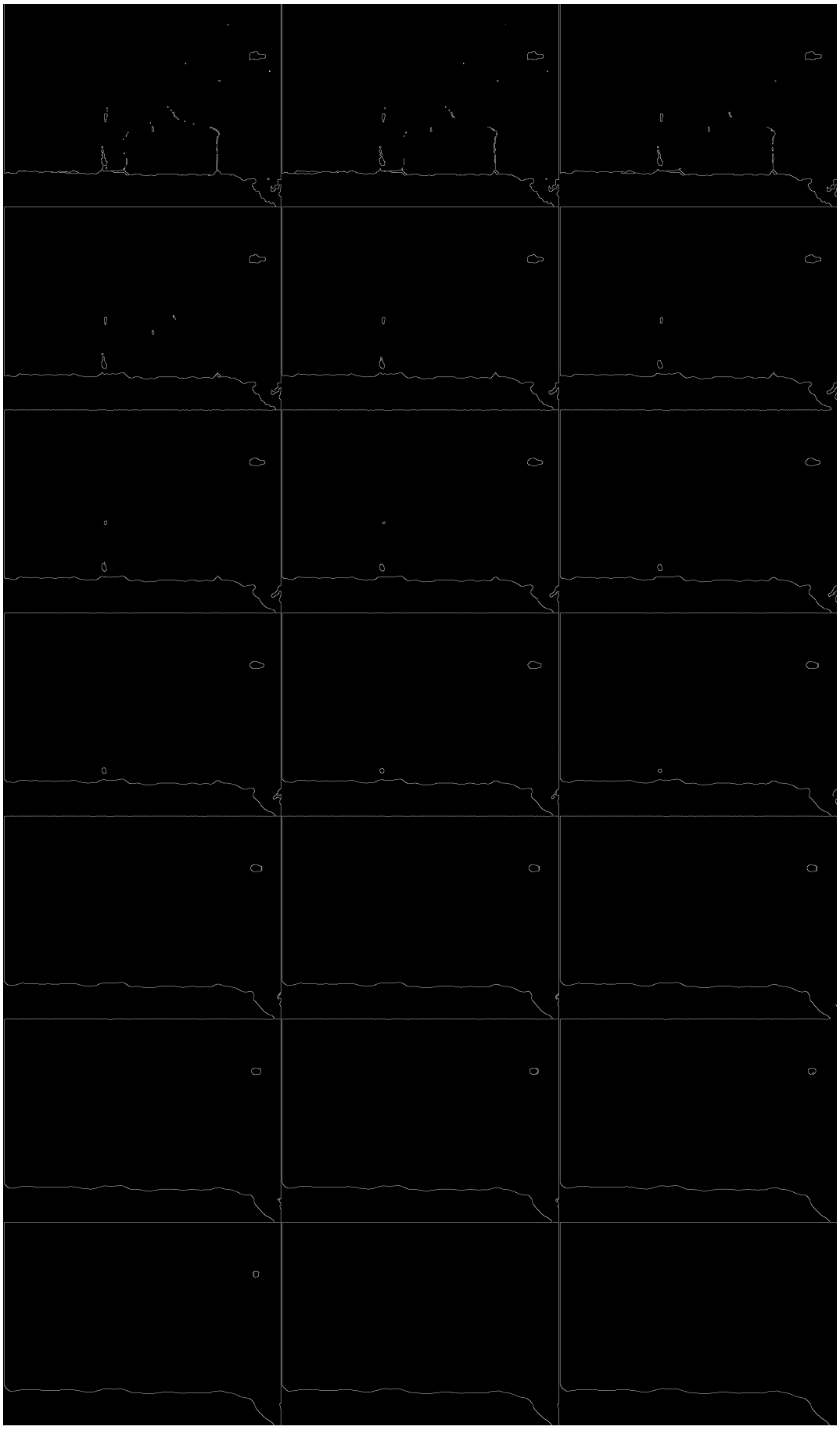


Figure 71: median_19_canny_190_570

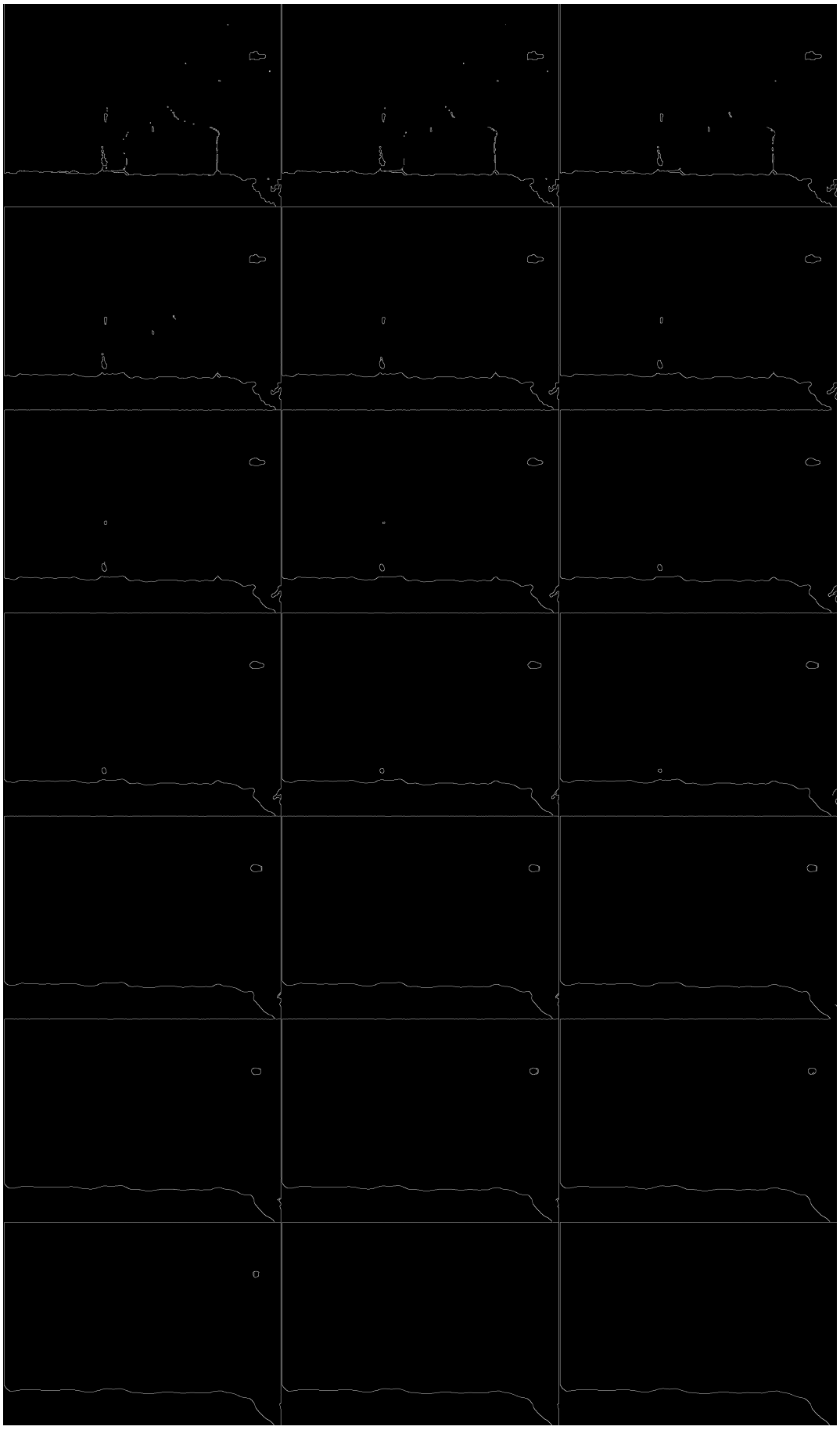


Figure 72: median_20_canny_200_600

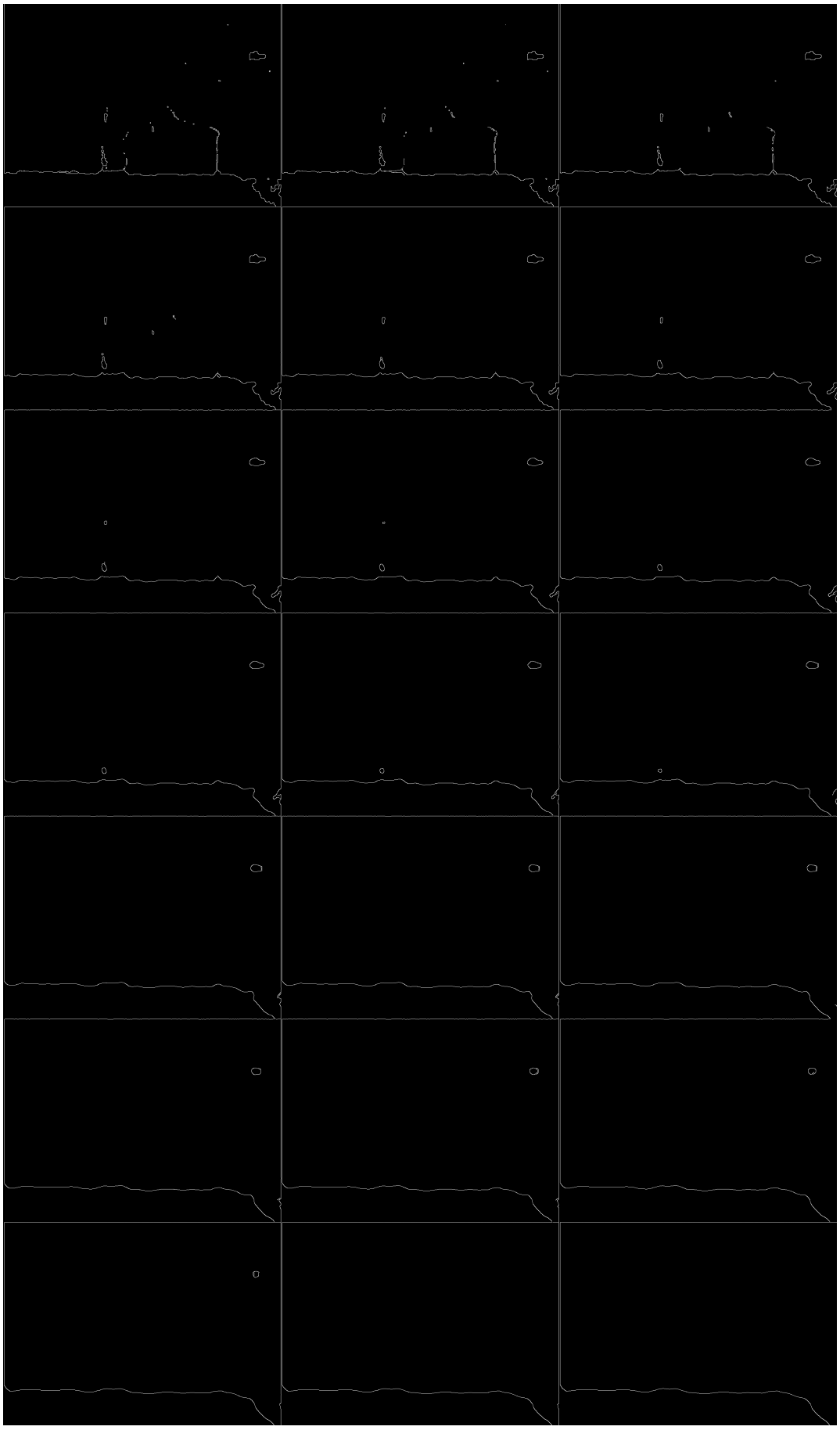


Figure 73: median_21_canny_210_630

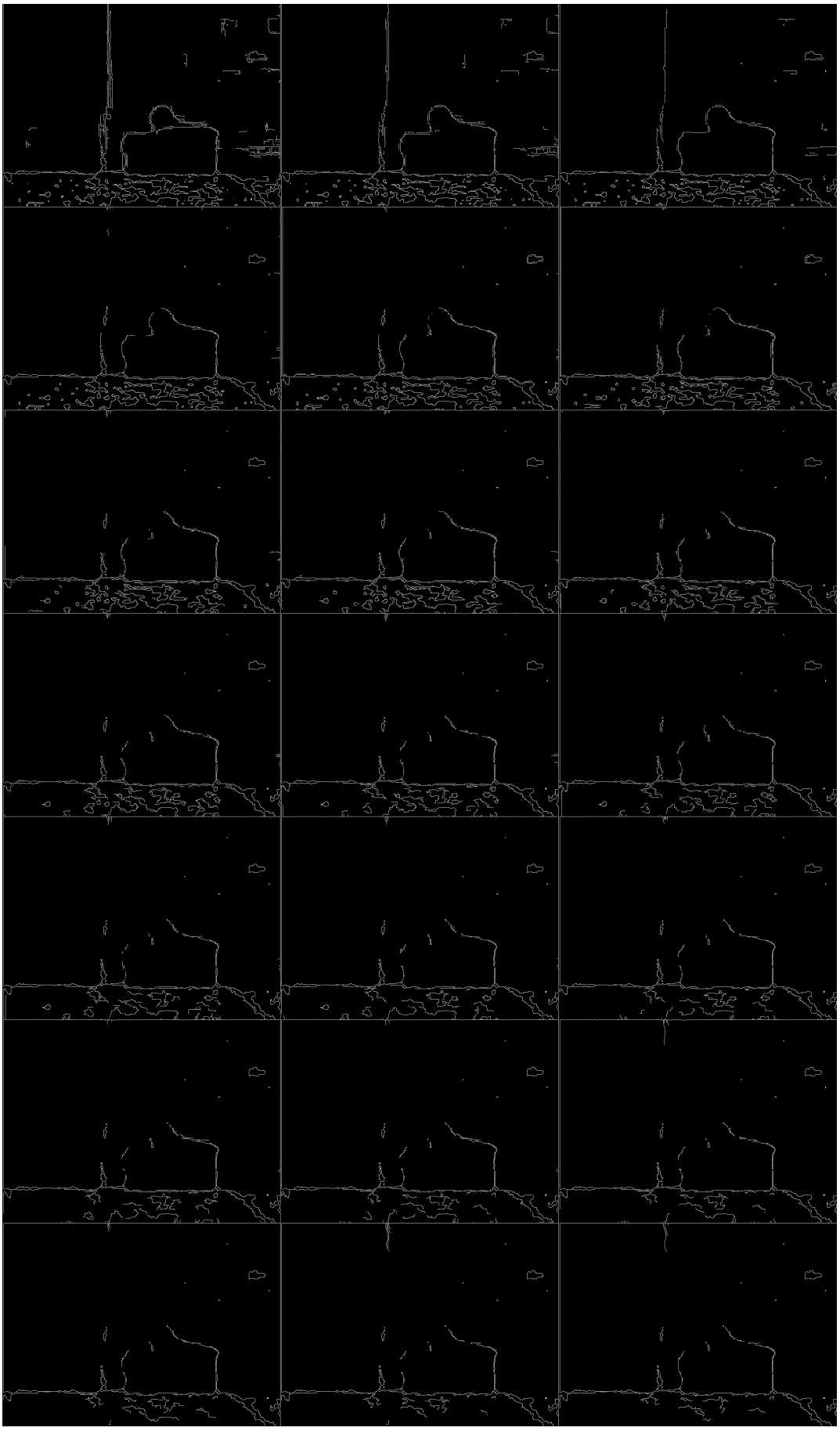


Figure 74: `bilateral_01_canny_10_30`

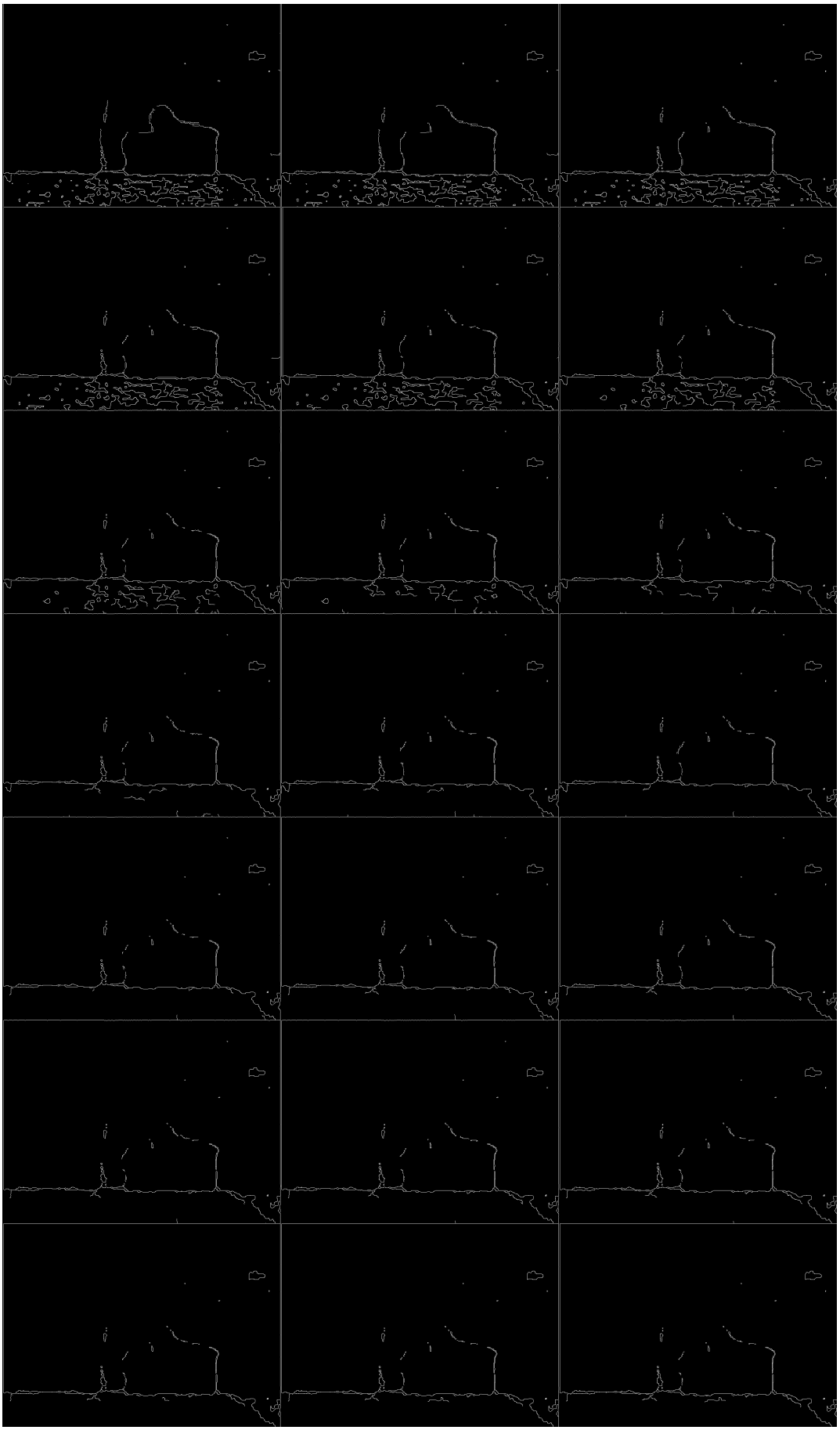


Figure 75: `bilateral_02_canny_20_60`

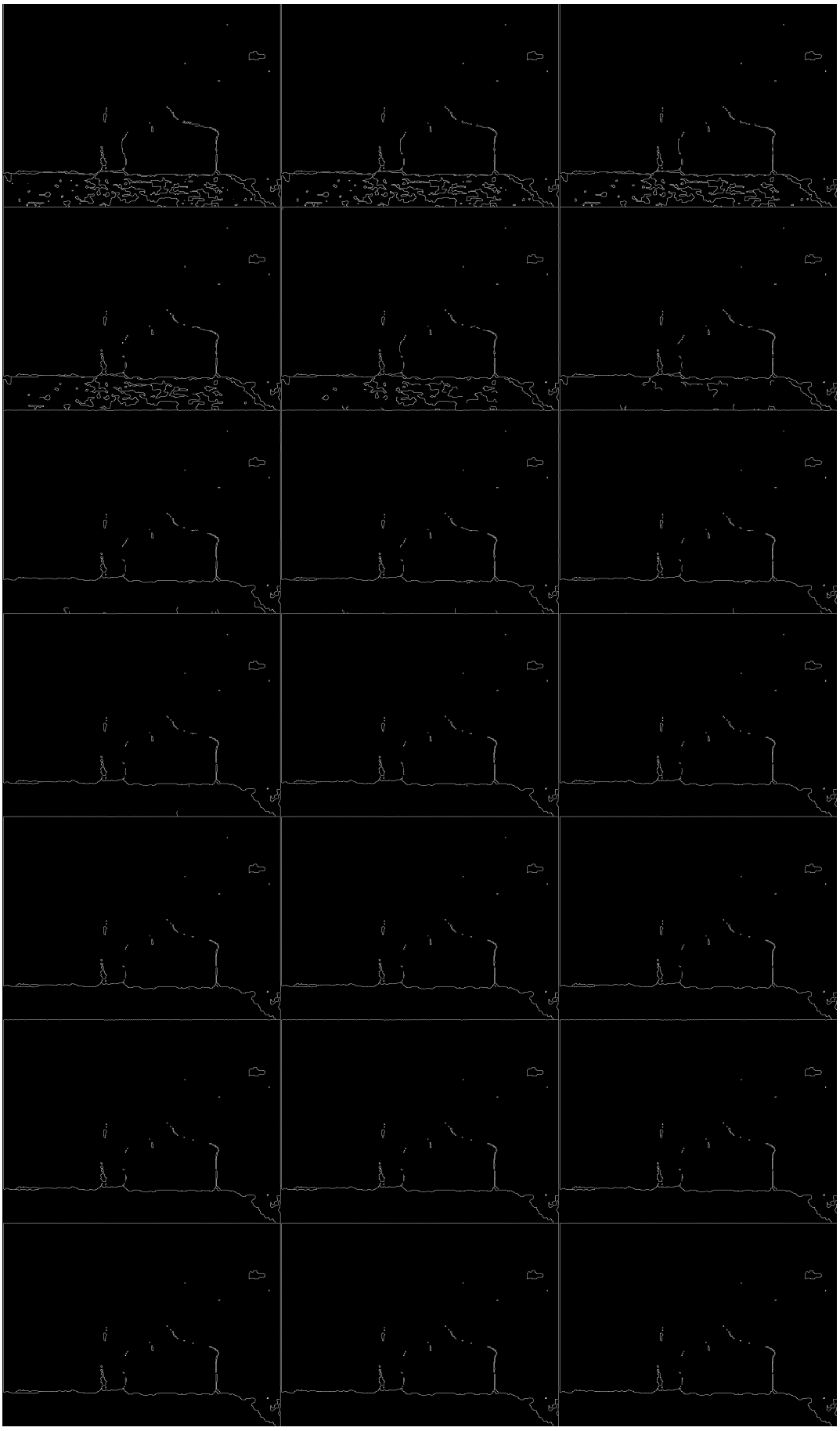


Figure 76: `bilateral_03_canny_30_90`

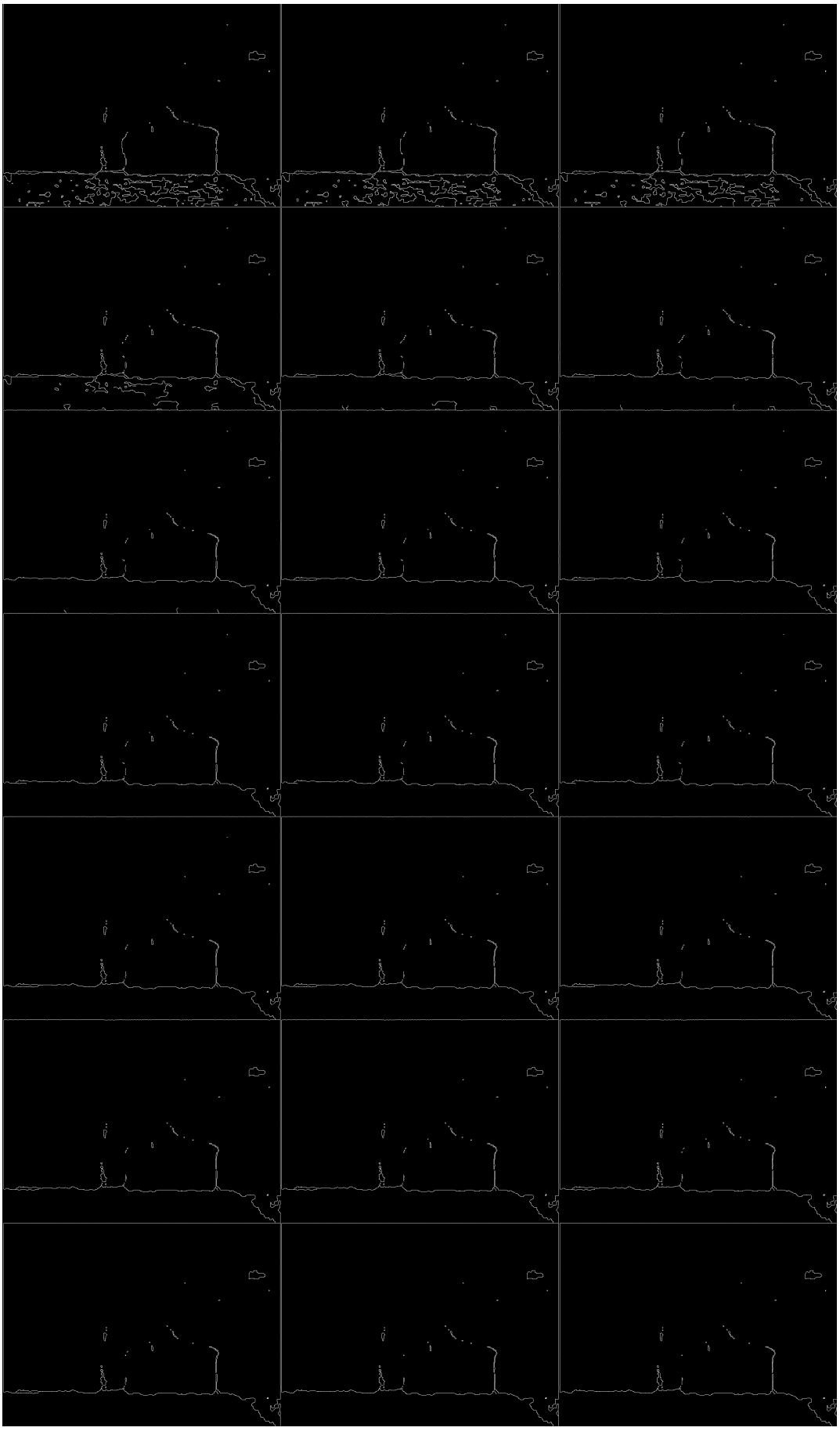


Figure 77: `bilateral_04_canny_40_120`

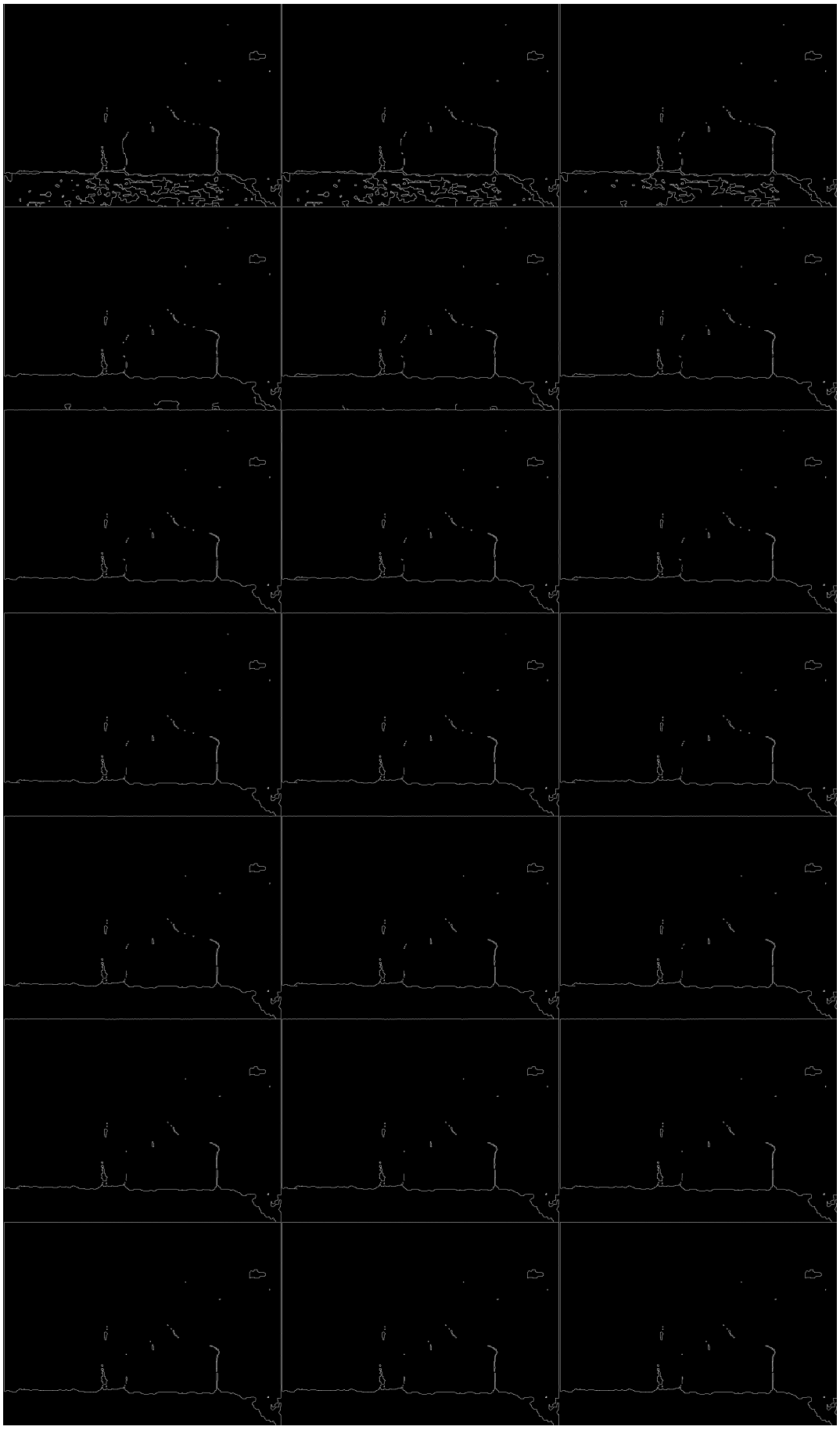


Figure 78: `bilateral_05_canny_50_150`

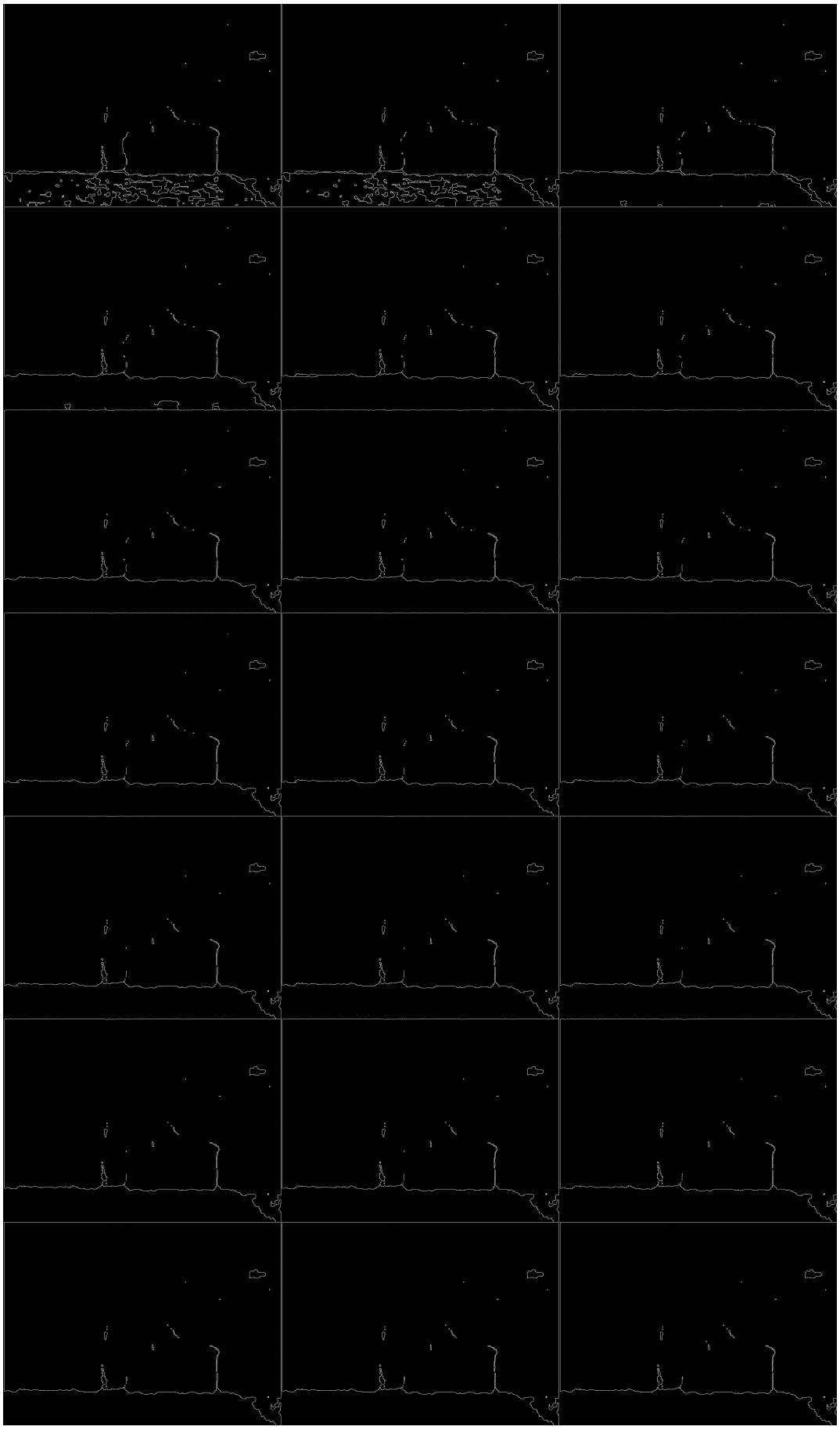


Figure 79: `bilateral_06_canny_60_180`

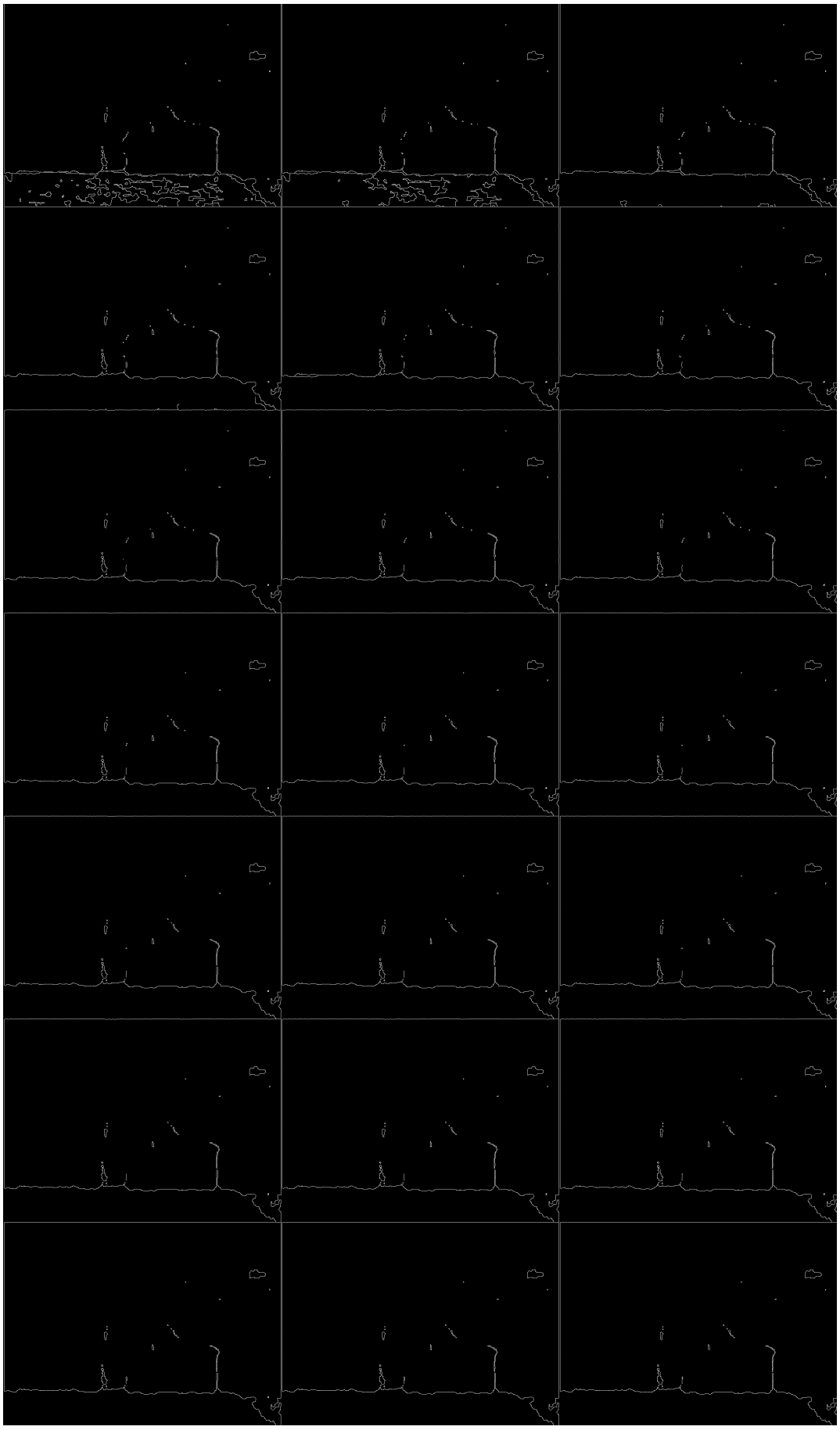


Figure 80: `bilateral_07_canny_70_210`

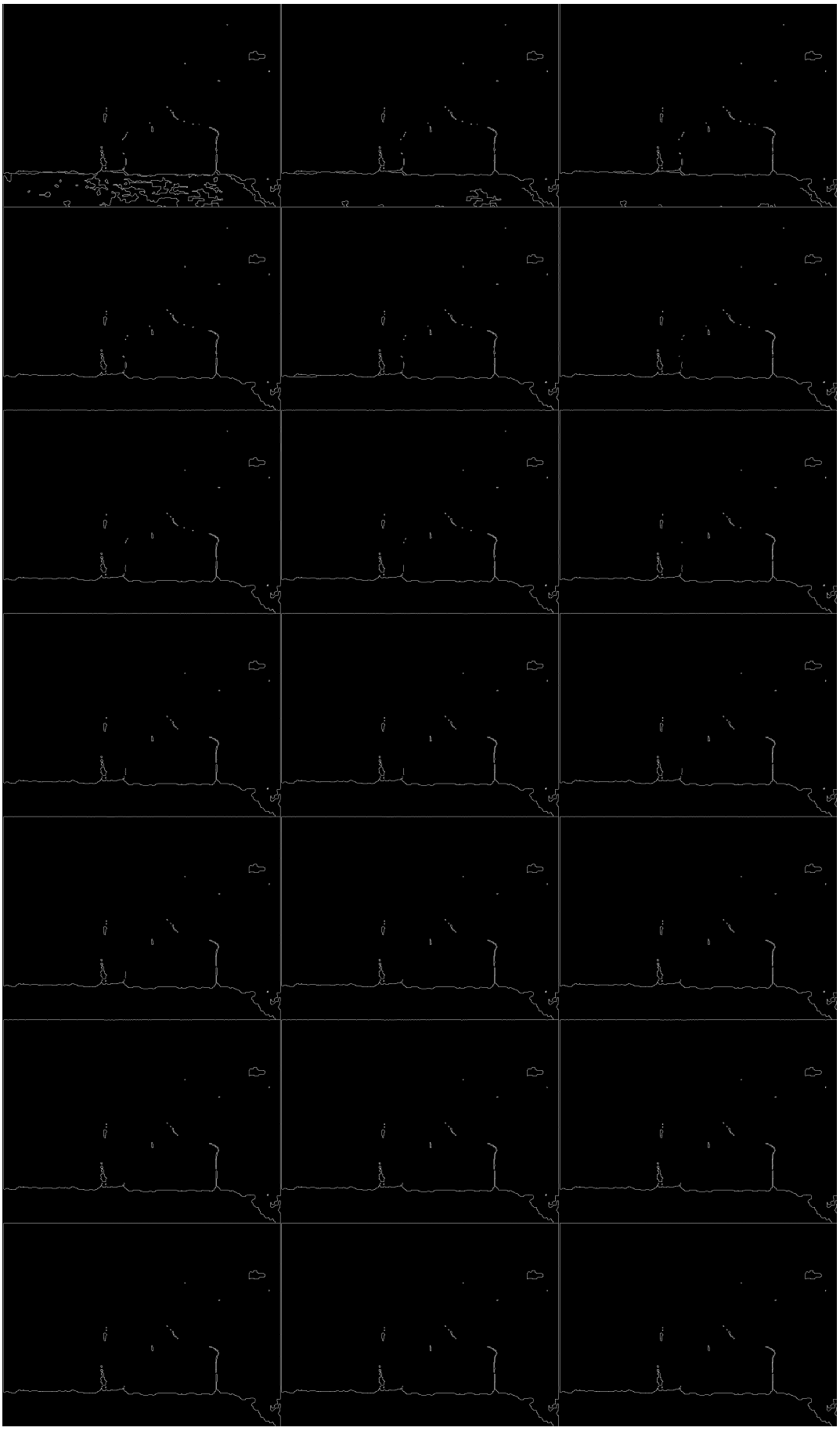


Figure 81: `bilateral_08_canny_80_240`

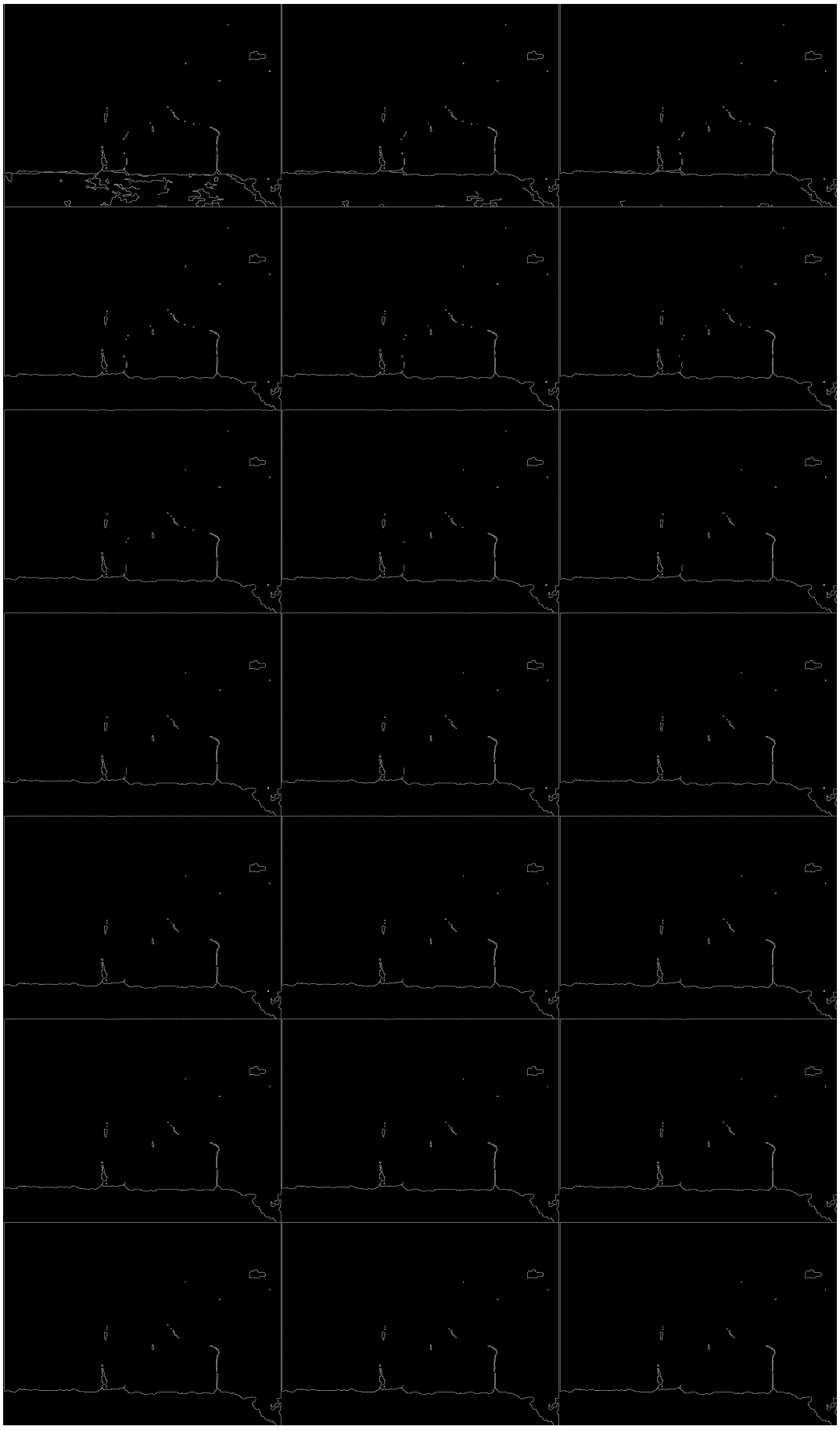


Figure 82: `bilateral_09_canny_90_270`

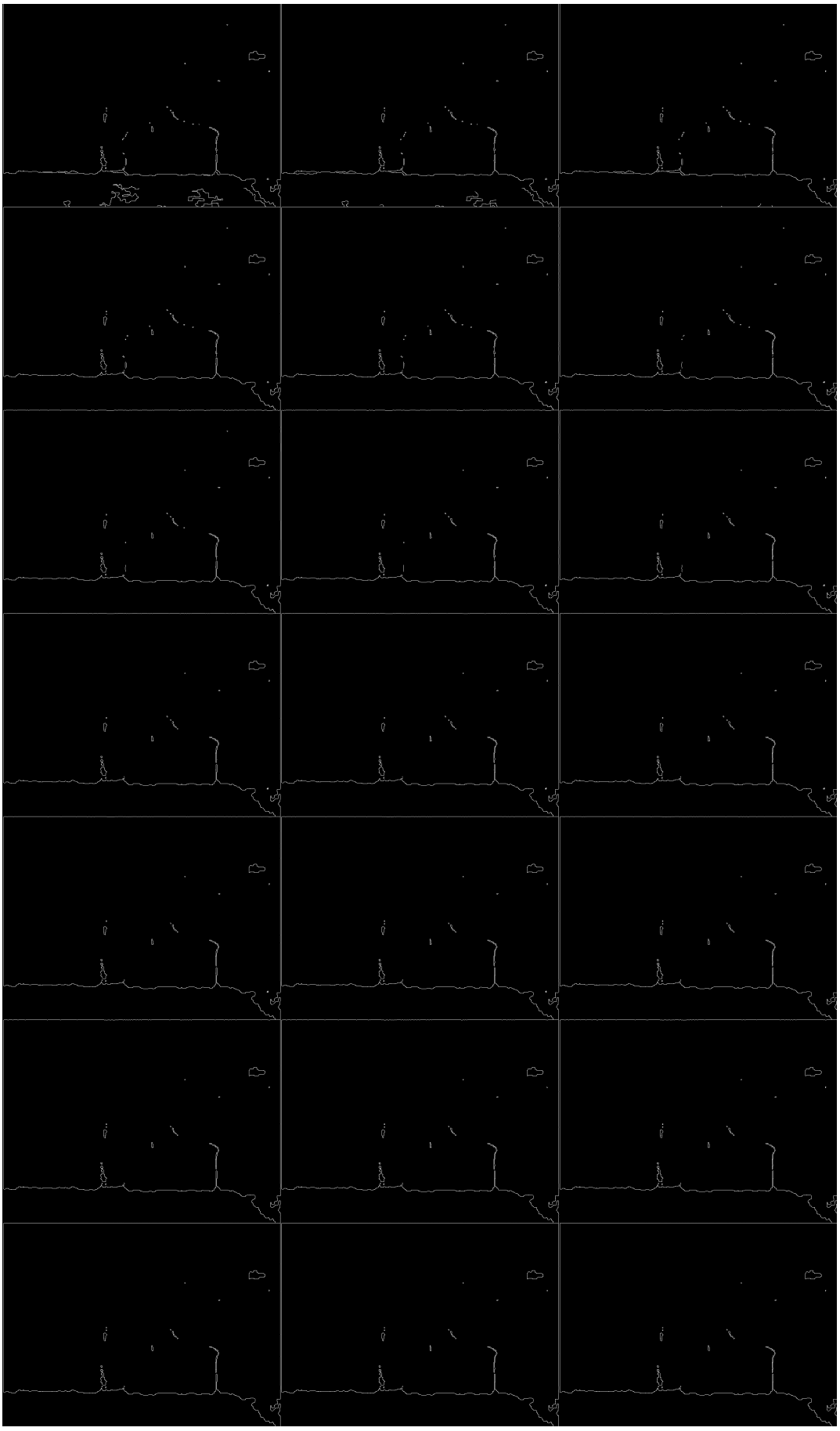


Figure 83: `bilateral_10_canny_100_300`

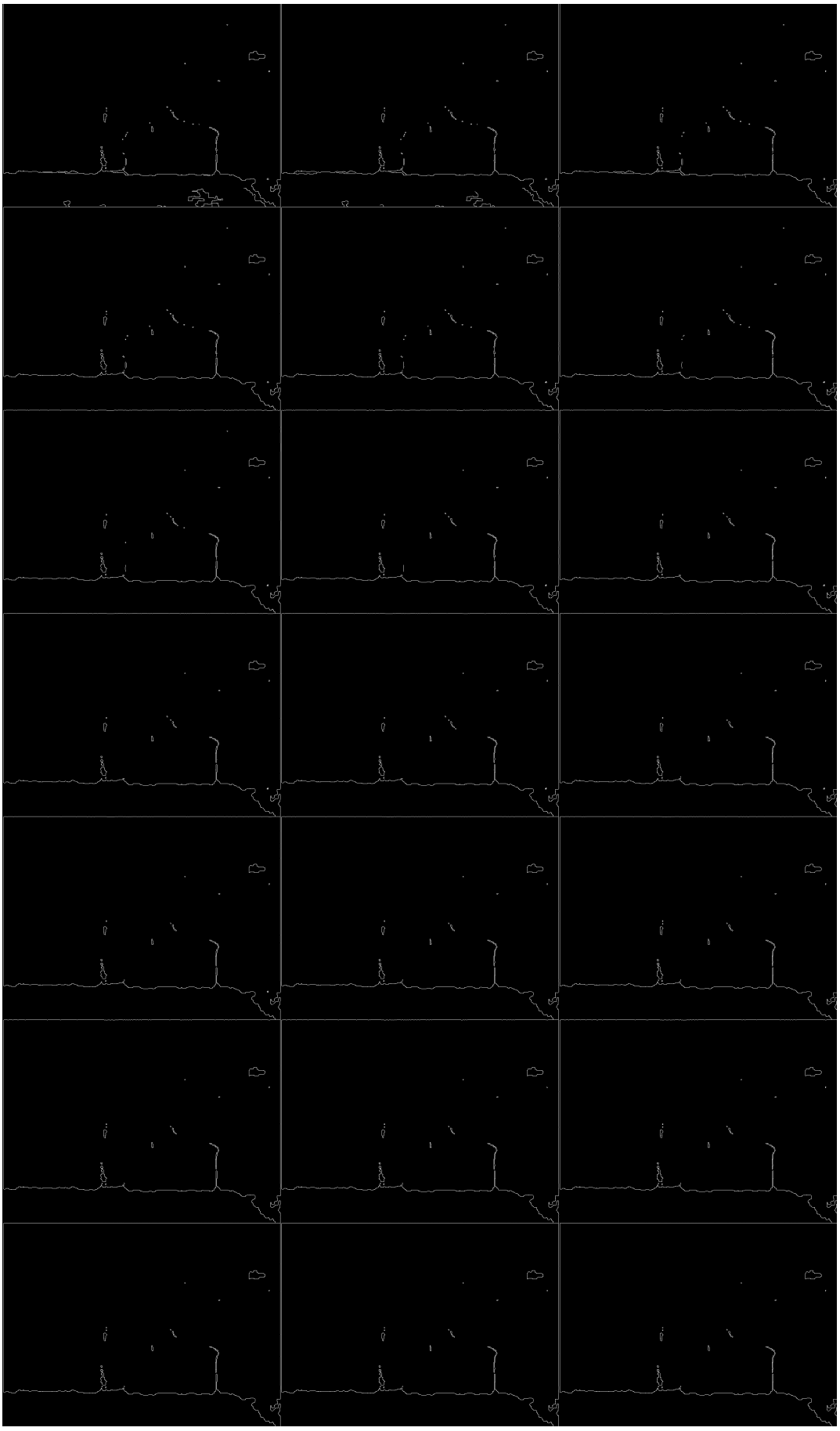


Figure 84: `bilateral_11_canny_110_330`

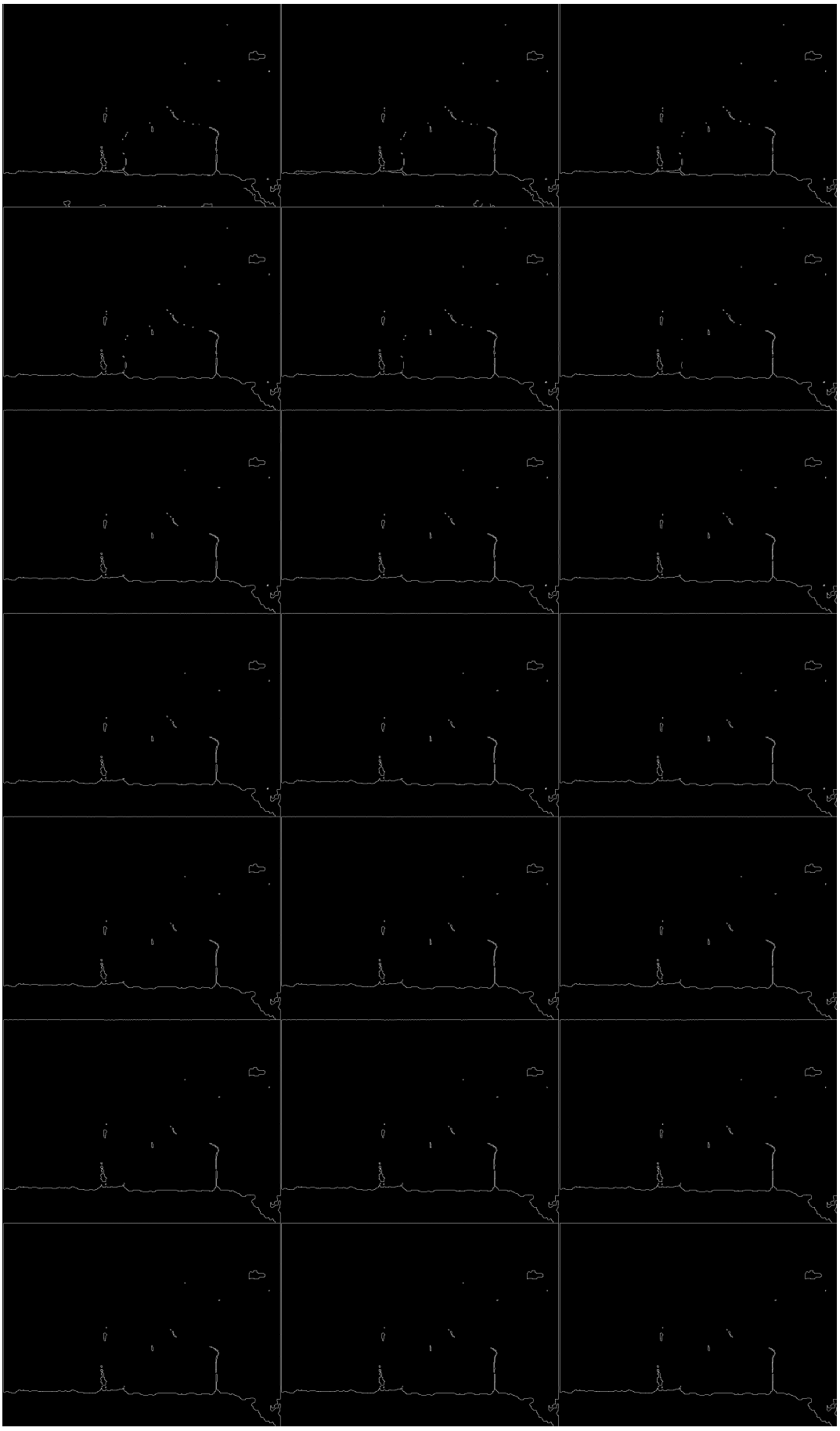


Figure 85: `bilateral_12_canny_120_360`

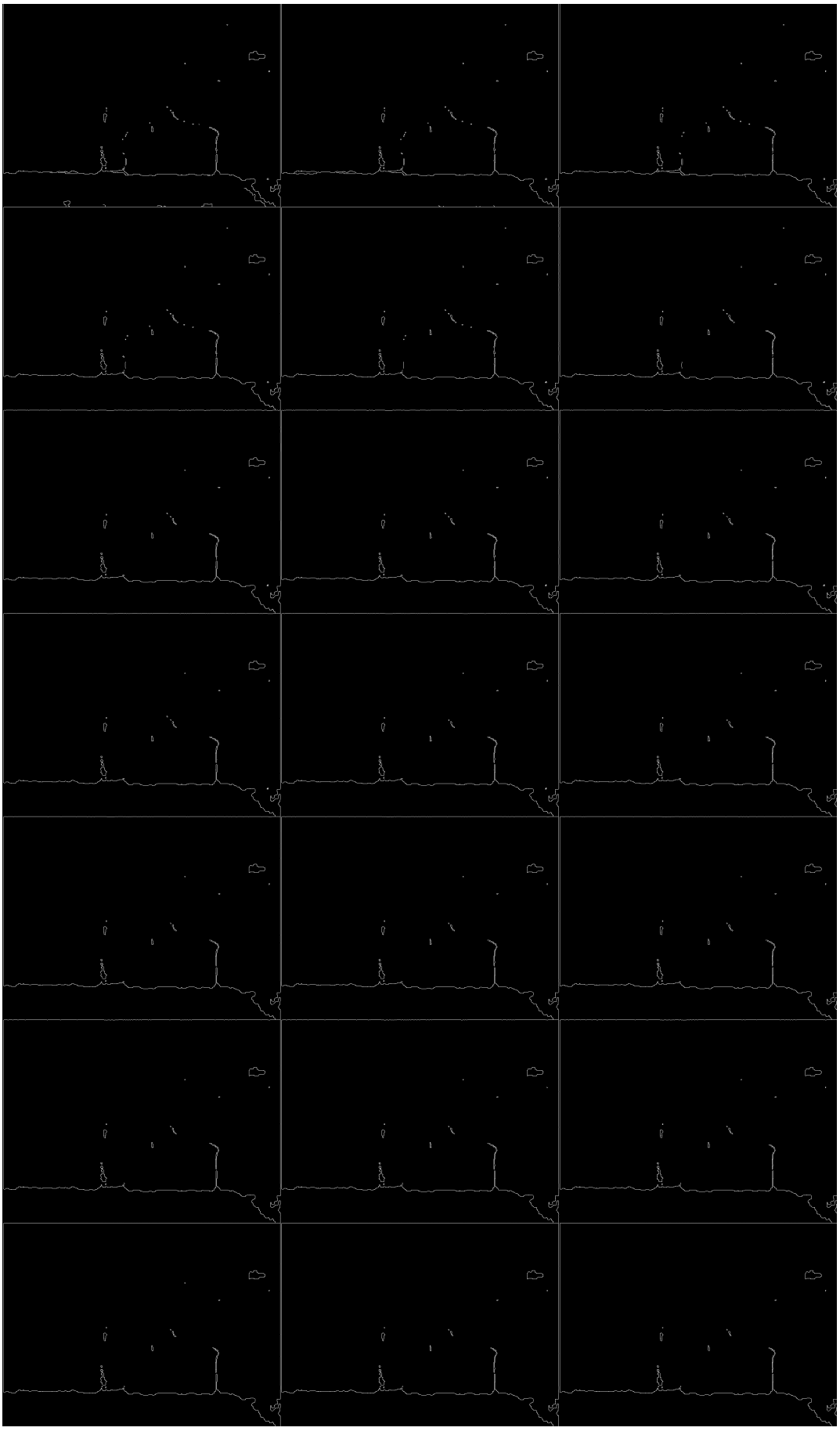


Figure 86: `bilateral_13_canny_130_390`

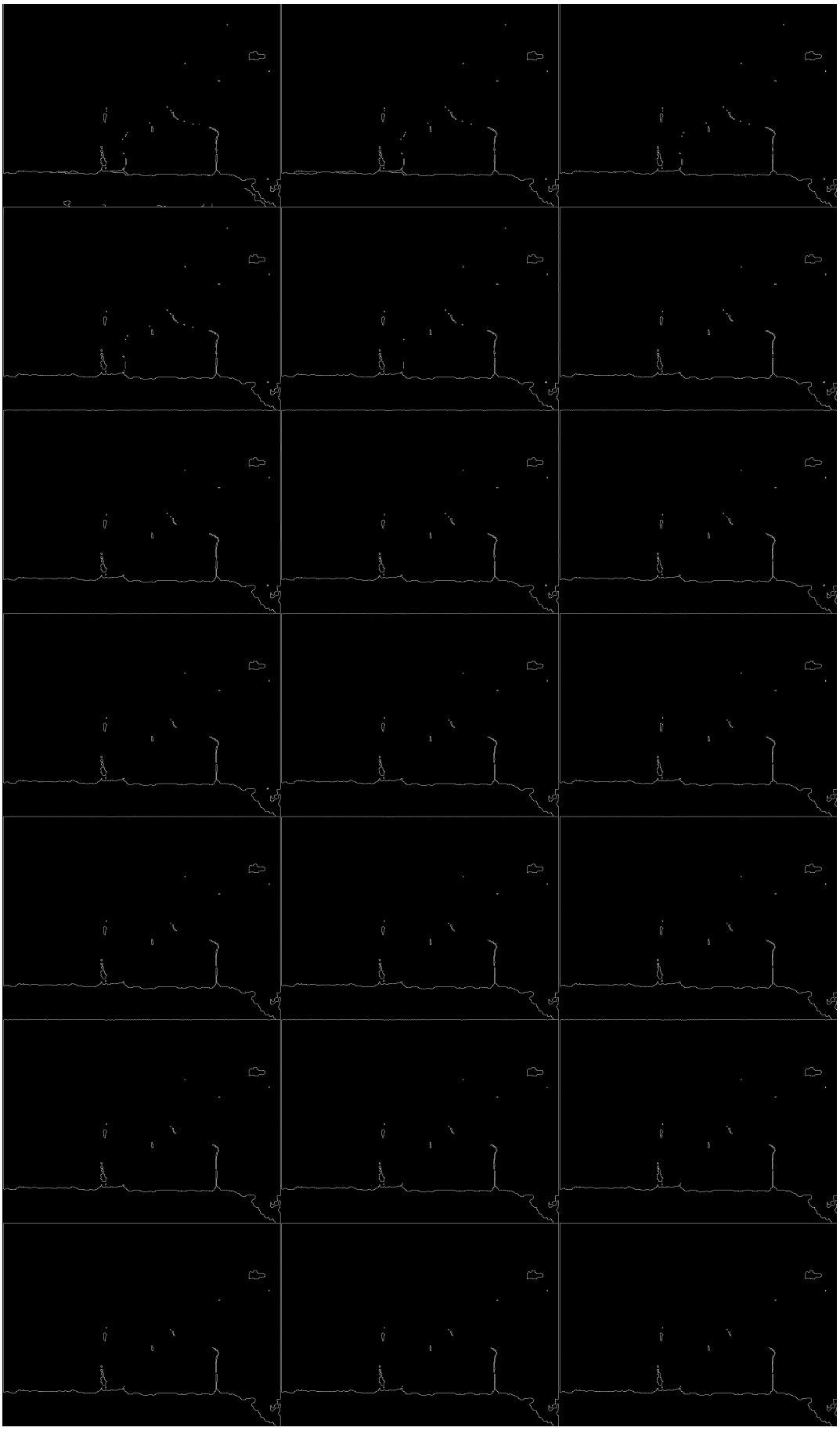


Figure 87: `bilateral_14_canny_140_420`

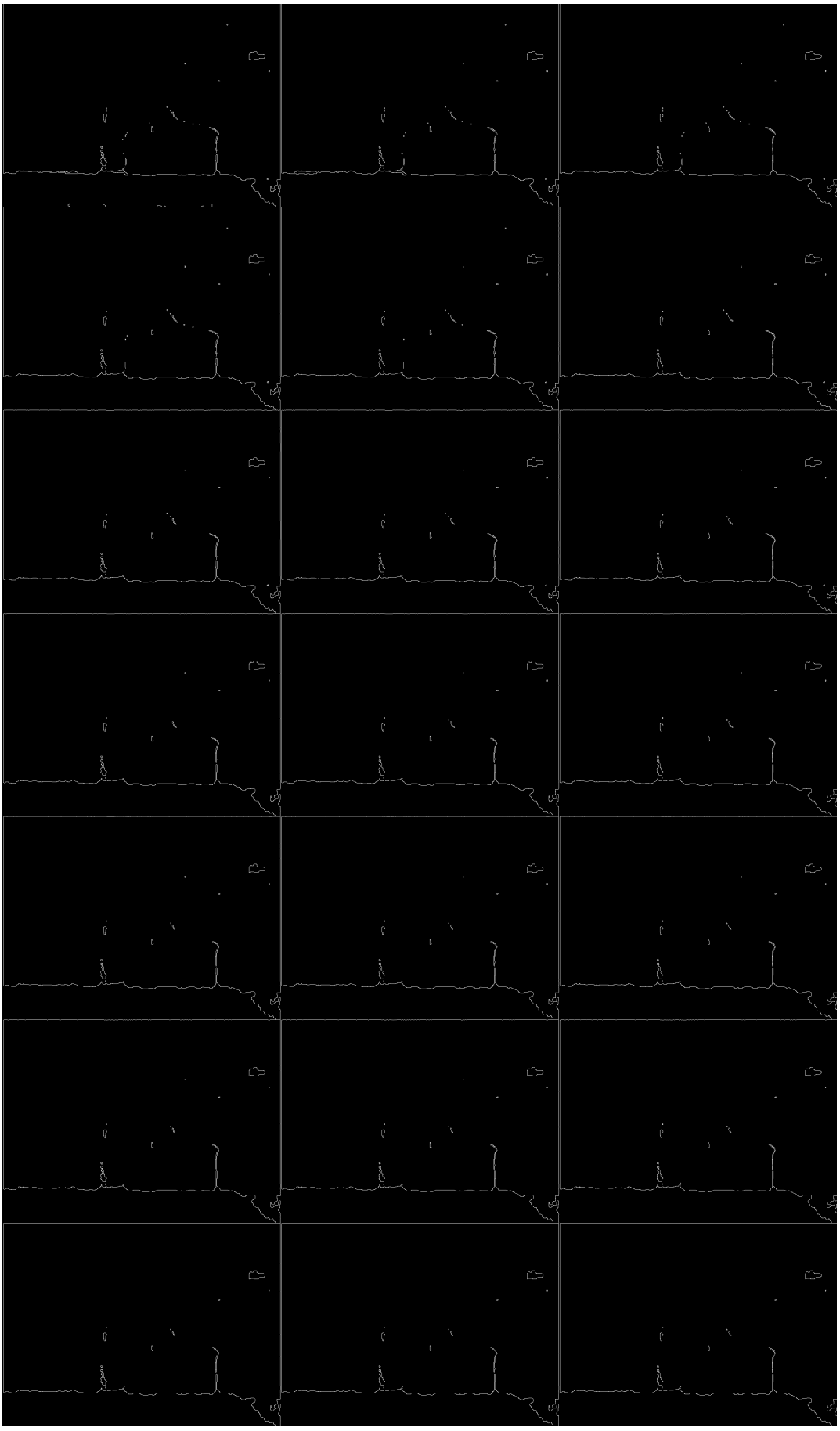


Figure 88: `bilateral_15_canny_150_450`

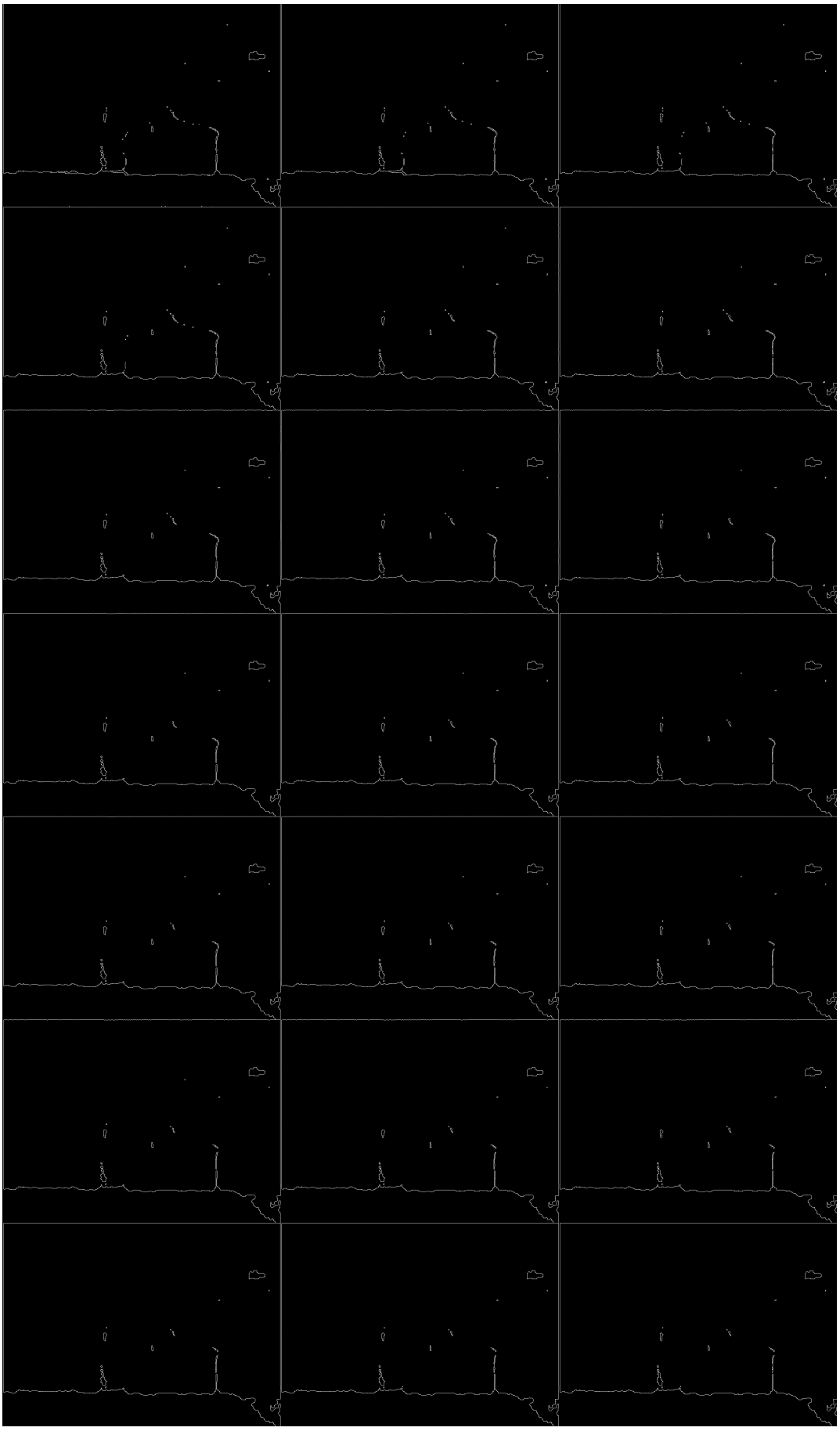


Figure 89: `bilateral_16_canny_160_480`

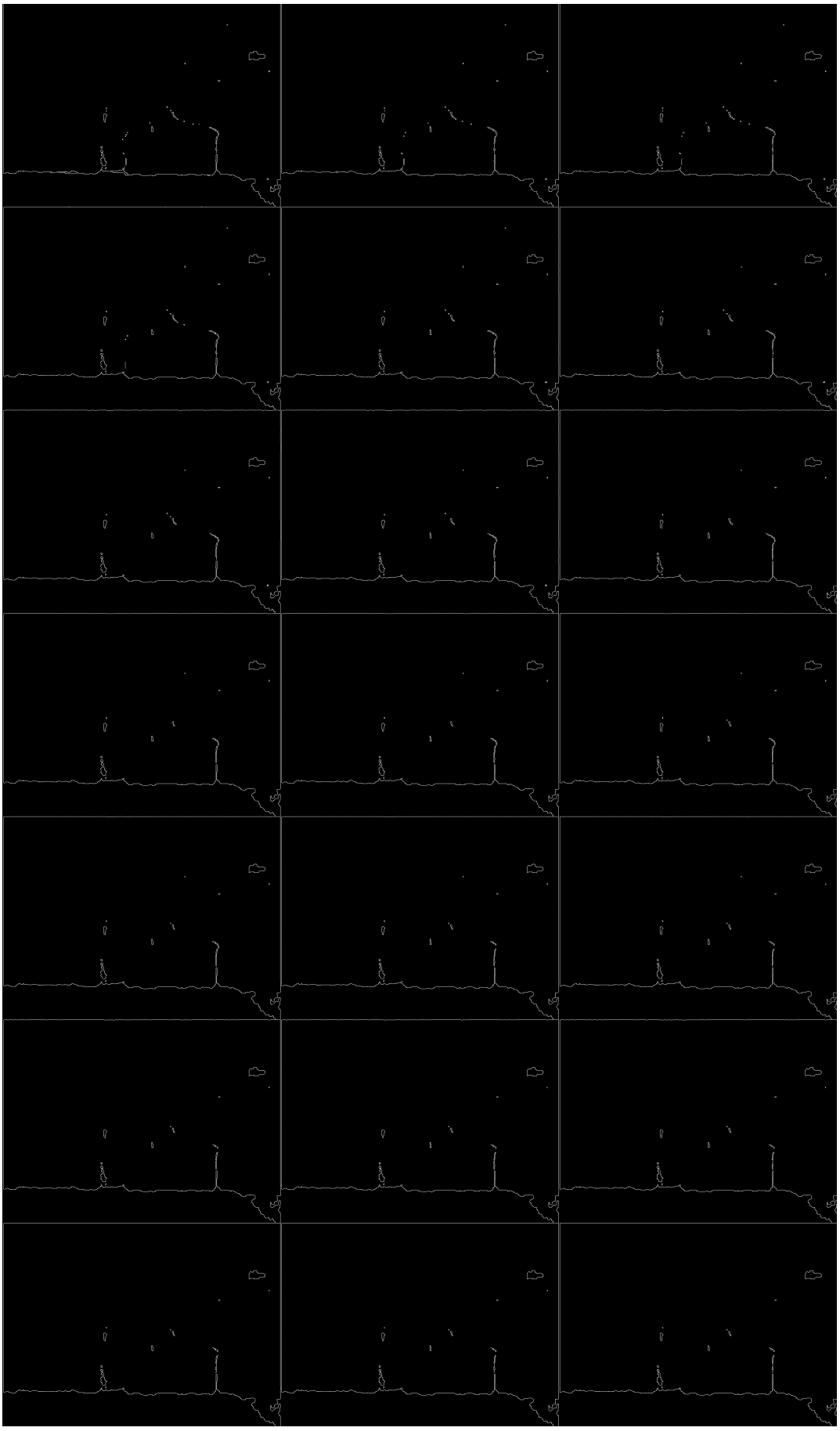


Figure 90: `bilateral_17_canny_170_510`

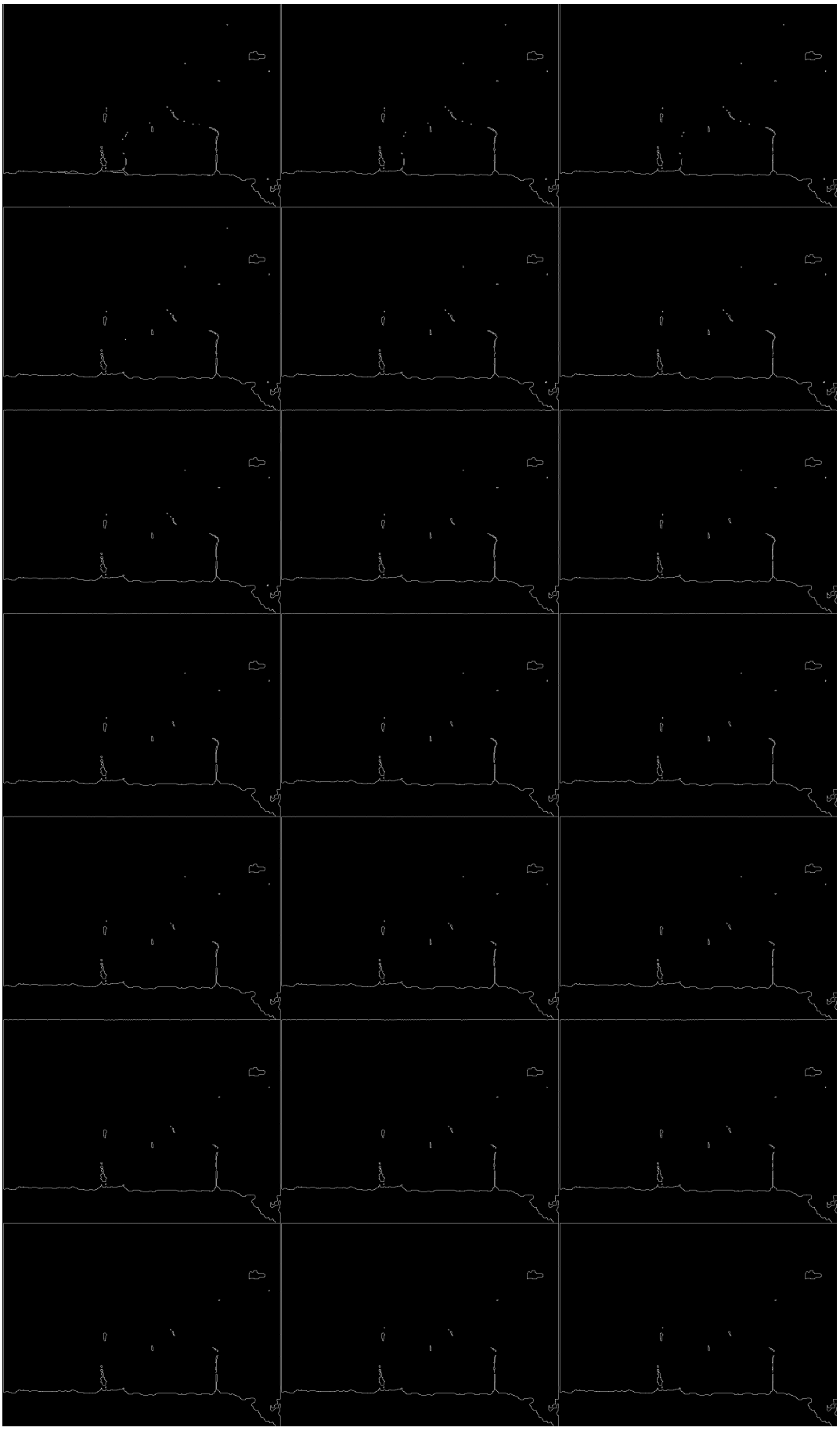


Figure 91: `bilateral_18_canny_180_540`

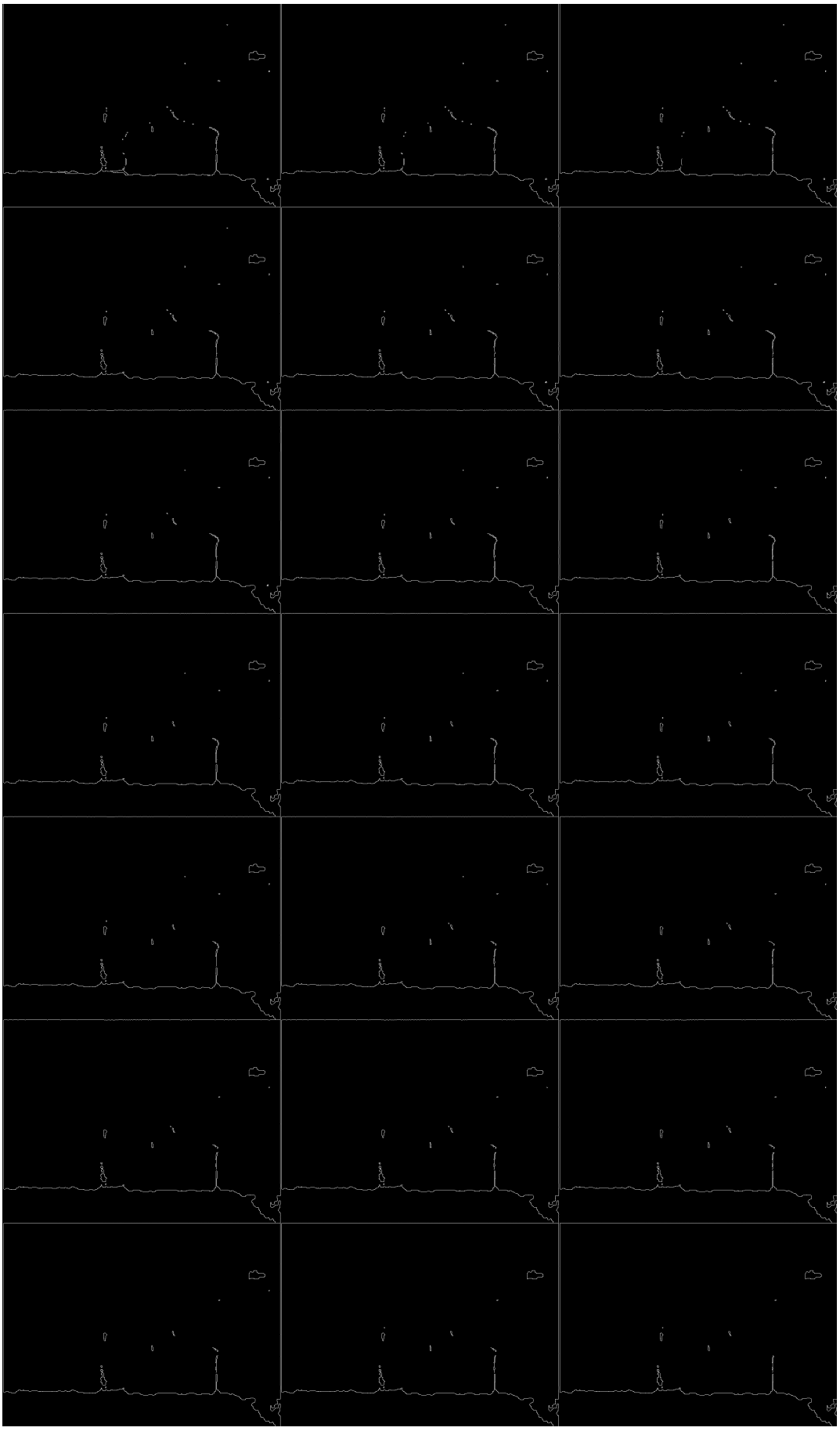


Figure 92: `bilateral_19_canny_190_570`

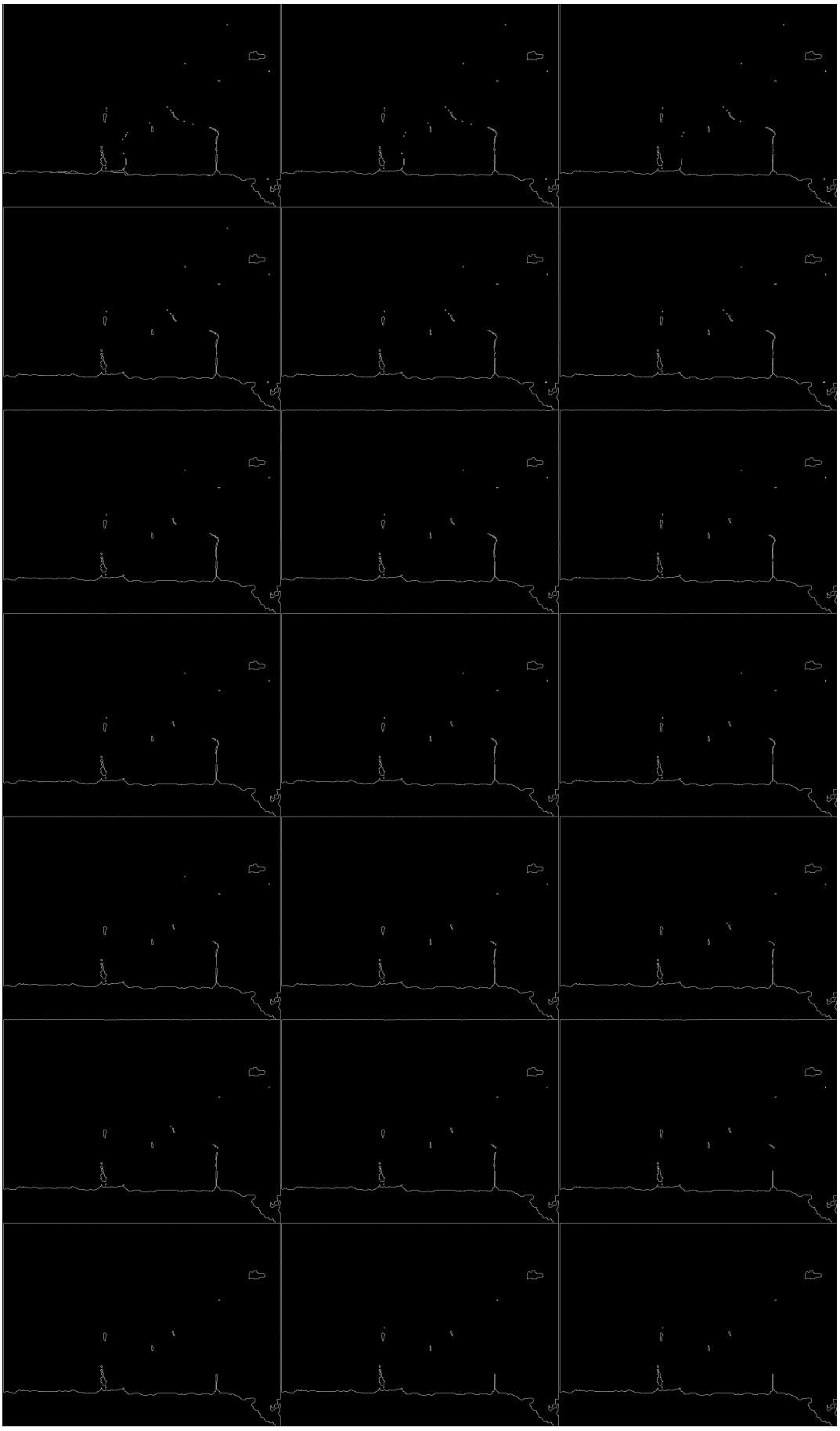


Figure 93: `bilateral_20_canny_200_600`

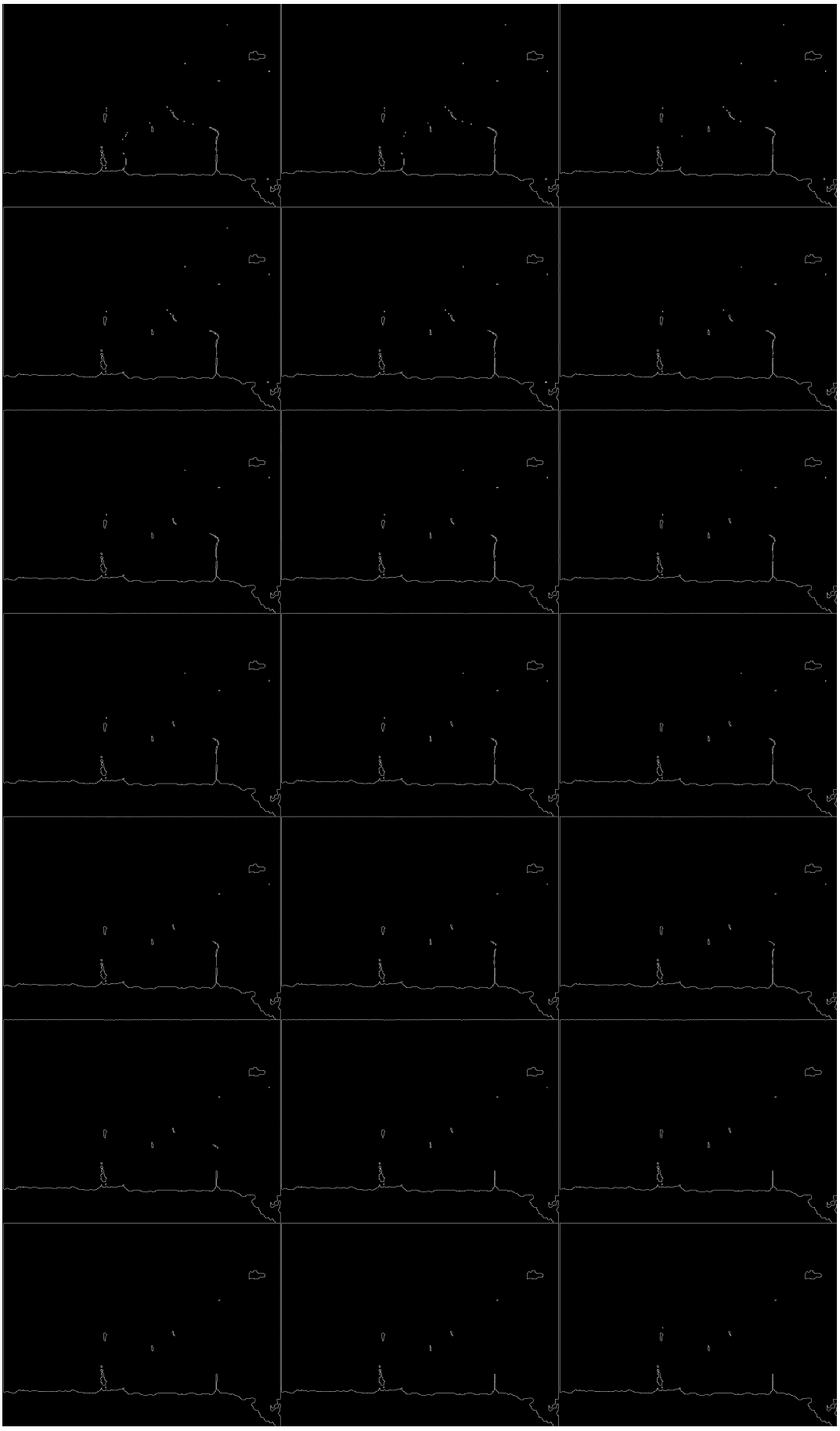


Figure 94: `bilateral_21_canny_210_630`

6 morphological transforms

Code

```python
import cv2
import numpy as np
from matplotlib import pyplot as plt
from pathlib import Path

# -------- paths (repo-root aware) --------
ROOT = Path(__file__).resolve().parents[2]          # .../repo/
IMG_PATH = ROOT / "images" / "test-2.jpeg"          # input image
OUT_DIR  = ROOT / "images" / "morph_transform"      # output dir
OUT_DIR.mkdir(parents=True, exist_ok=True)

# -------- load --------
img = cv2.imread(str(IMG_PATH), cv2.IMREAD_GRAYSCALE)
if img is None:
    raise FileNotFoundError(f"Could not load image: {IMG_PATH}")

# -------- thresholding --------
# fixed/global threshold (your original)
_, mask = cv2.threshold(img, 70, 255, cv2.THRESH_BINARY)

# adaptive thresholds (tweak blockSize & C as needed; blockSize must be odd >= 3)
blockSize = 3
C = 35


mask_mean = cv2.adaptiveThreshold(
    img, 255, cv2.ADAPTIVE_THRESH_MEAN_C, cv2.THRESH_BINARY, blockSize, C
)

mask_gauss = cv2.adaptiveThreshold(
    img, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY, blockSize, C
)

# -------- morphology on the (fixed) mask (unchanged) --------
kernel = np.ones((5, 5), np.uint8)
dilation = cv2.dilate(mask, kernel, iterations=1)
erosion  = cv2.erode(mask, kernel, iterations=3)
opening  = cv2.morphologyEx(mask, cv2.MORPH_OPEN, kernel)
closing  = cv2.morphologyEx(mask, cv2.MORPH_CLOSE, kernel)
mg       = cv2.morphologyEx(mask, cv2.MORPH_GRADIENT, kernel)
th       = cv2.morphologyEx(mask, cv2.MORPH_TOPHAT, kernel)

# -------- save outputs --------
to_save = {
    "adaptive_mean.png":   mask_mean,
    "adaptive_gaussian.png": mask_gauss,
    "dilation.png": dilation,
    "erosion.png":  erosion,
    "opening.png":  opening,
    "closing.png":  closing,
    "gradient.png": mg,
    "tophat.png":   th,
    "image.png": img,
    "global_mask.png": mask
}
for name, arr in to_save.items():
    cv2.imwrite(str(OUT_DIR / name), arr)
```

```

# -------- show (2 x 5 grid) --------
titles = [
    'image', 'global mask', 'adaptive mean', 'adaptive gaussian',
    'dilation', 'erosion', 'opening', 'closing', 'gradient', 'tophat'
]
images = [img, mask, mask_mean, mask_gauss, dilation, erosion, opening, closing, mg, th]

plt.figure(figsize=(12, 6))
for i in range(len(images)):
    plt.subplot(2, 5, i + 1)
    plt.imshow(images[i], 'gray')
    plt.title(titles[i], fontsize=9)
    plt.xticks([]); plt.yticks([])

plt.tight_layout()
plt.show()
```

Figure 95: adaptive_mean.png

Figure 96: adaptive_gaussian.png

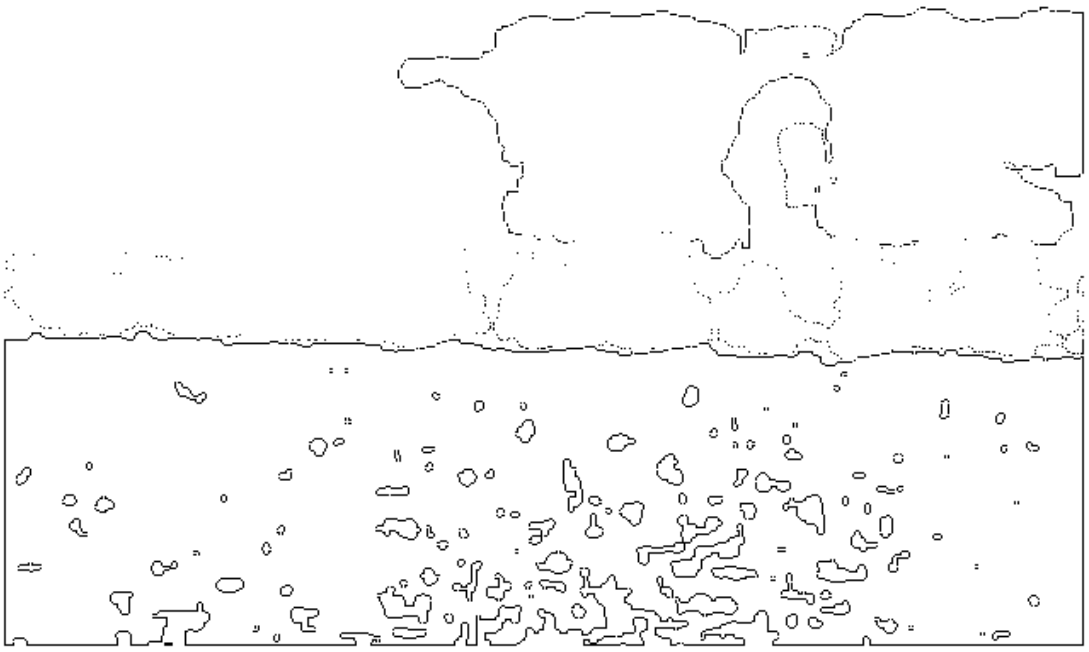


Figure 97: adaptive_mean.png

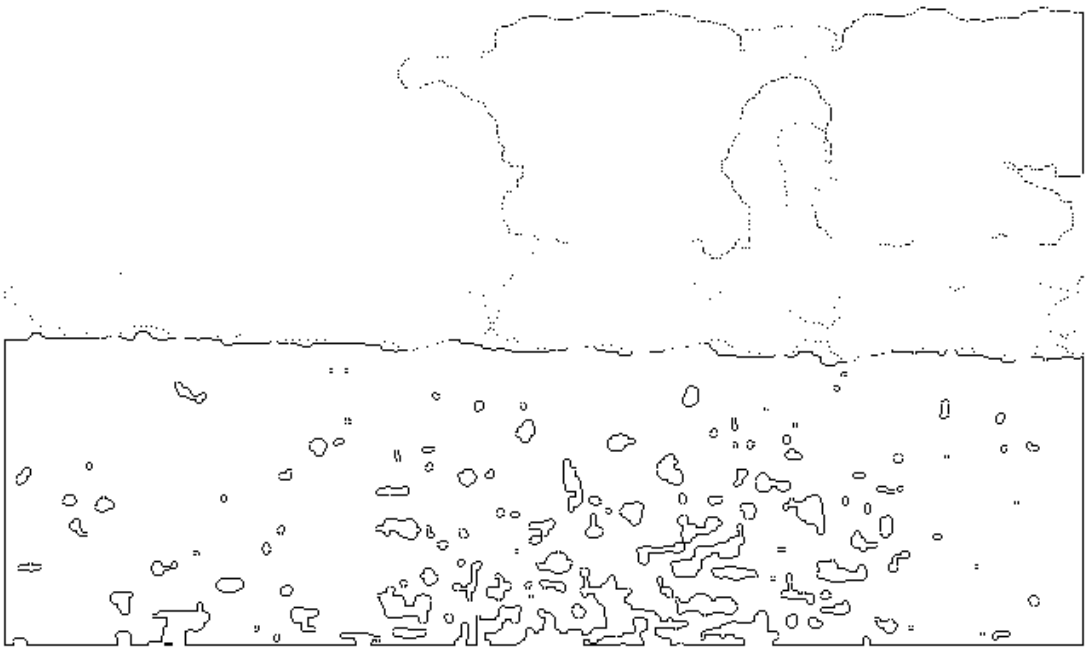


Figure 98: adaptive_gaussian.png

Figure 99: dilation.png

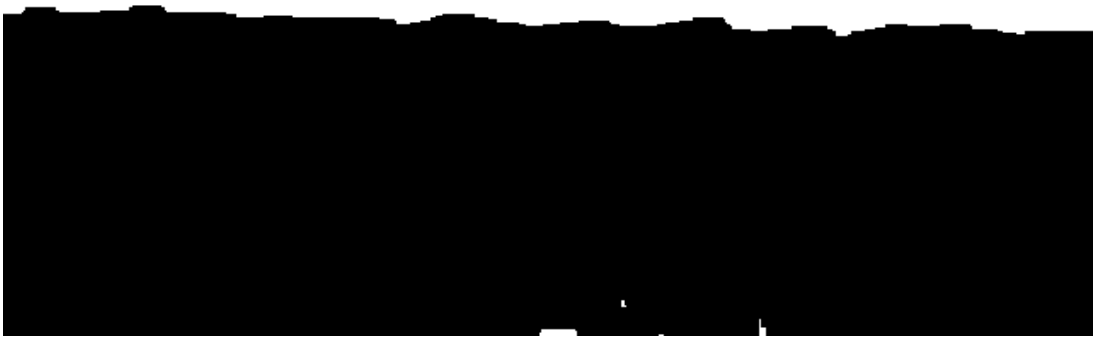


Figure 100: erosion.png

Figure 101: opening.png

Figure 102: closing.png

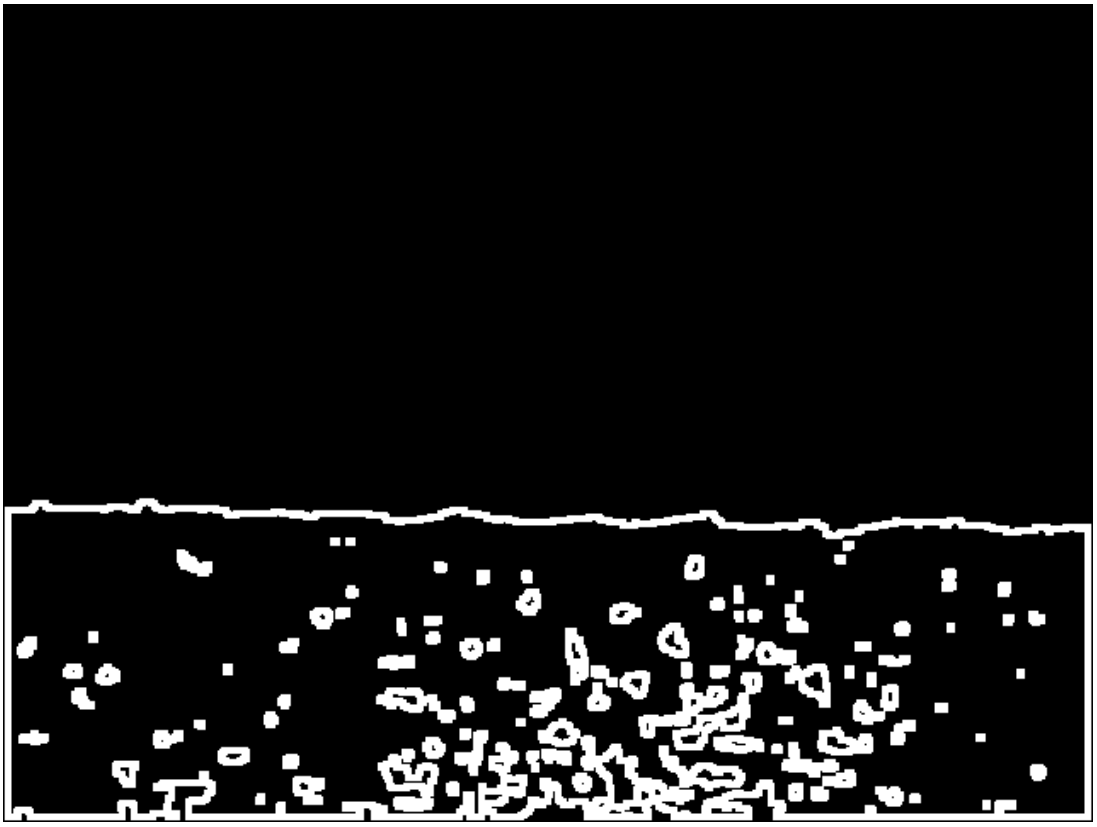


Figure 103: gradient.png

Figure 104: tophat.png