Assignment 3 of EIE4512

12204097J.

Problem 1.

We know that Erosion is the set intersection, so the intersection of two convex set is also convex. Since the data set are black parts, in (b), the result of dilation is not convex because of there's no center point in the black set.

Problem 2.

We need to: Identify the Origin & Border of the Structuring Element.; complementary Structuring Element.; do erosion operations and Intersection

Step: ① Erode the image with the foreground part of the structuring Element.

② Erode the complement of the image with the background Part of the Structuring Element.

③ Intersection the positions marked in the previous 2 steps.

Problem 3.

The key difference between Lake & Bay + line is: former forms one closed.

We process one each step (multiple can seperate) using the following steps:

Step 1: Apply an end-point detector to detect till converge. If result ≠ ϕ, object is Lake

Step 2: We can simply choose a line which connect to 2 end of the graph. If the connection between 2 end has no elements. It's a Bay, otherwise, It's a line segment

Problem 4.

(a) In the context described in the problem, we focus on the shape of noise spikes. To remove these spikes, use a cylindrical structuring element with a radius greater than **Rmax** for the opening operation.

(b) The solution is similar to (a), but now we must consider the overlap described in the problem. The structuring element is similar to the one used in (a), but with a radius slightly larger than **4Rmax**.

Problem 5

(a) Color the image border pixels the same white color as the particles. Apply the connected component algorithm. All connected components that contain elements from the newly created set of border pixels are particles that have merged with the border of the image.

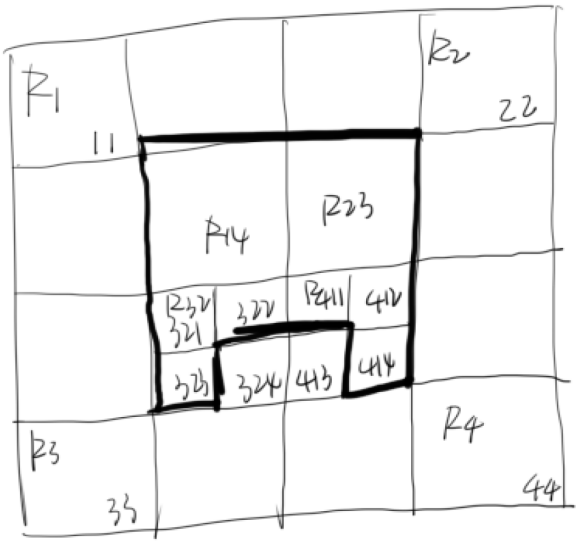
(b) Assume all particles are of the same size and determine the area S of a single particle by counting the number of pixels. Eliminate from the image the particles that have merged with the border of the image. Apply the connected component algorithm to count the number of pixels in each component. A component is designated as a single particle if the number of pixels is less than or equal to S + ε, where ε is a small quantity added to account for variations in size due to noise.

(c) Subtract single particles and particles that have merged with the border from the image, and the remaining particles are overlapping particles.

Problem 6

Because the means are more than 10 standard deviations apart, the valley between the two modes in the image histogram is wide and deep. You can use a simple global thresholding algorithm or Otsu ' s algorithm.

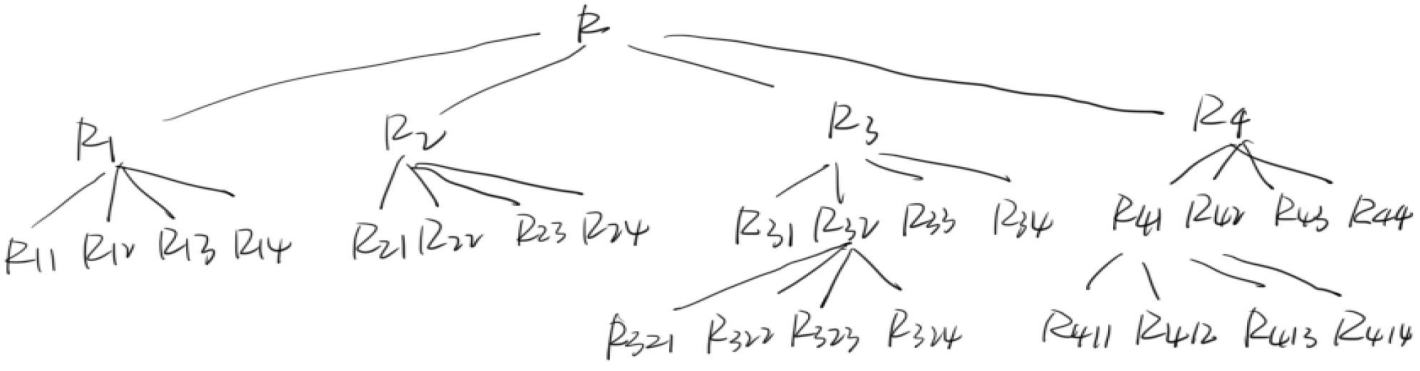
Problem 7



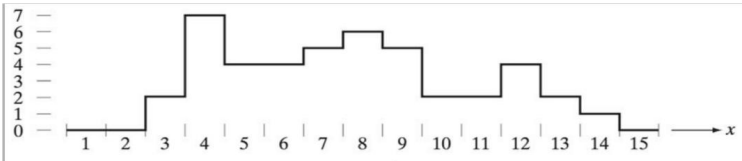
Figures including:

① How to seperate

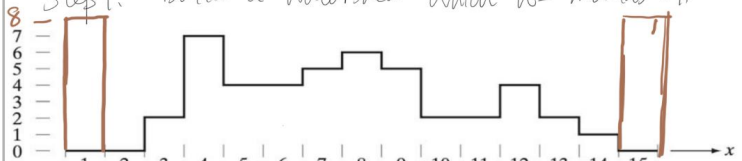
② Quad-tree.



Problem 8

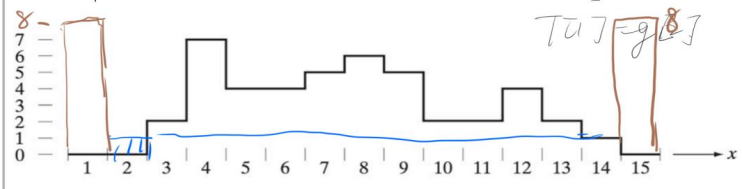


Step 1. Build a watershed which h= max(h) +1.



Step 2. Since box is built, we set C[1] = T[1]

T[1] = g[8]

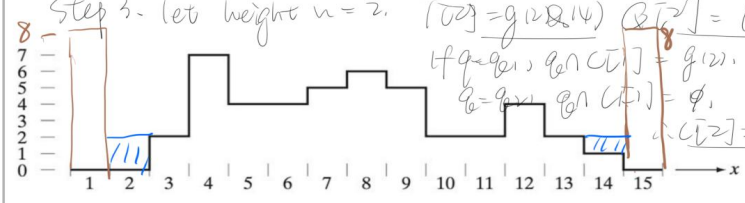


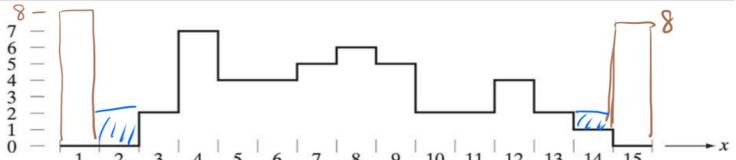
Step 3. let height h = 2. T[2] = g(12,8,14) C[2] = (q4, 02)

If $q = q_{01}, q_{02} \cap C[1] = g(12)$, V

$q = q_{01}, q_{02} \cap C[1] = \phi$, V.

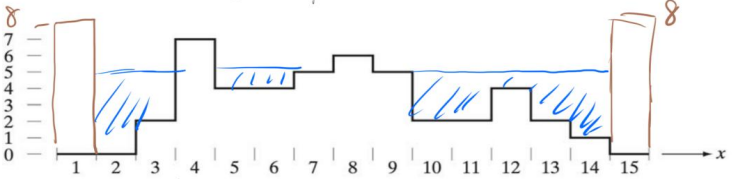
$\therefore C[2] = g(12,8,14)$



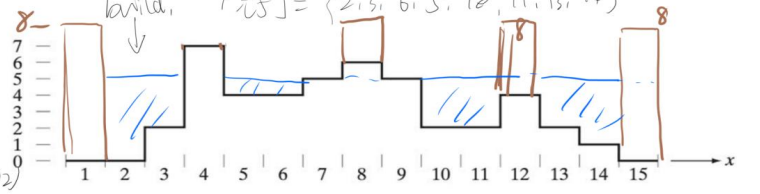


Step 4. h=3, T[3] = {2, 3, 10, 11, 13, 14} Q[3] = {q4, 23}

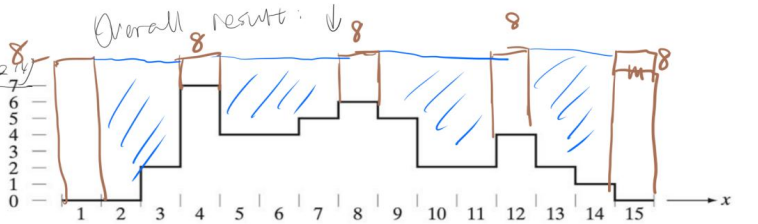
C[4] = C[3] = {2, 3, 10, 11, 13, 14}



Step 5. h= 5. The water connect, This need dell to build. T[5] = {2, 3, 6, 5, 10, 11, 13, 14}



Overall result:



In [21]:
```
# Problem 1
import cv2
import numpy as np
import matplotlib.pyplot as plt
```

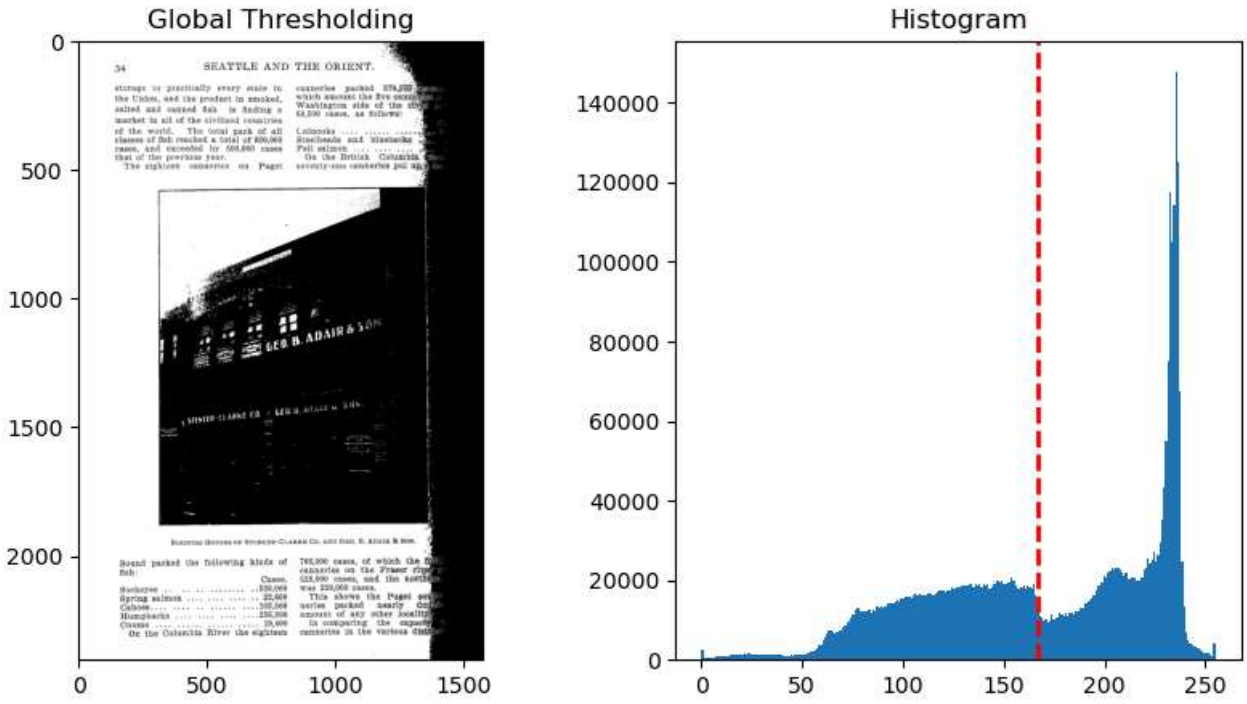
In [22]:
```
# 读取图像
img1 = cv2.imread('./figs/pro1_page1.jpg')
img2 = cv2.imread('./figs/pro1_page2.jpg')
img3 = cv2.imread('./figs/pro1_page3.png')

# 打印读取的图像
plt.figure(figsize=(15, 5))
plt.subplot(1, 3, 1)
plt.title('pro1_page1.jpg')
plt.imshow(cv2.cvtColor(img1, cv2.COLOR_BGR2RGB))

plt.subplot(1, 3, 2)
plt.title('pro1_page2.jpg')
plt.imshow(cv2.cvtColor(img2, cv2.COLOR_BGR2RGB))

plt.subplot(1, 3, 3)
plt.title('pro1_page3.png')
plt.imshow(cv2.cvtColor(img3, cv2.COLOR_BGR2RGB))

plt.show()
```

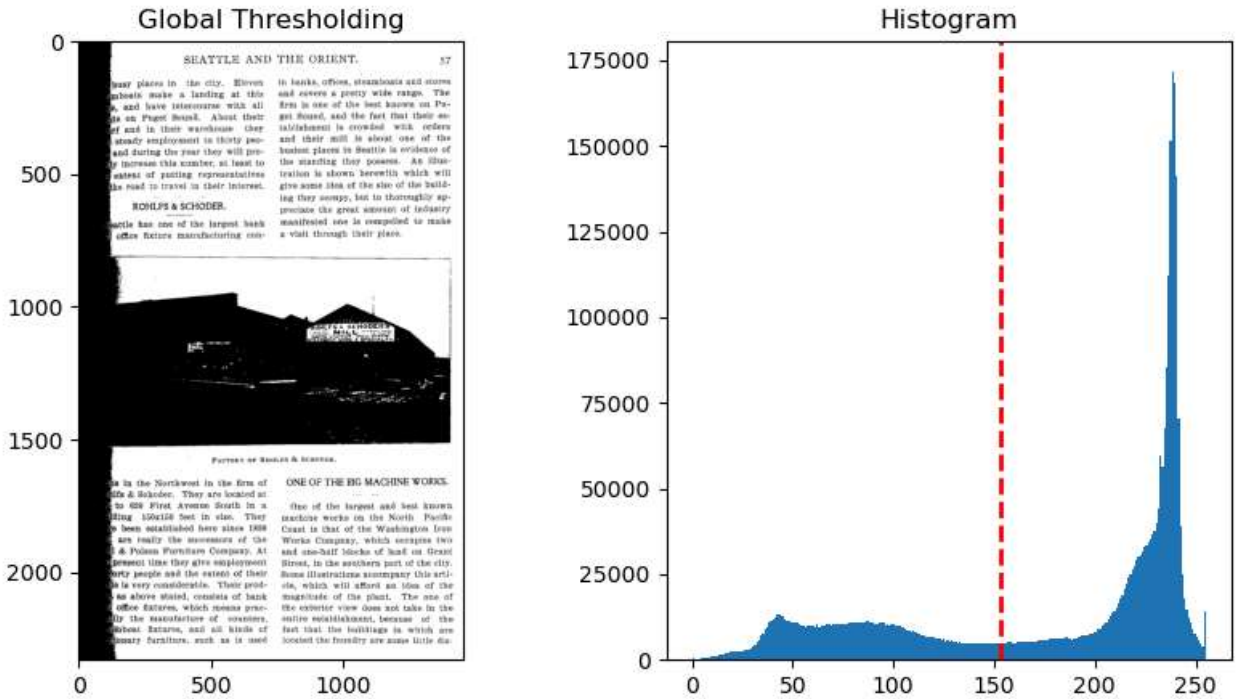
In [23]:
```
def globalThresh(img):
    # 使用 Otsu 方法进行全局阈值化
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    ret, binary_img = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)

    # 显示二值图像
    plt.figure(figsize=(10, 5))
    plt.subplot(1, 2, 1)
    plt.title('Global Thresholding')
    plt.imshow(binary_img, cmap='gray')

    # 显示直方图和阈值
    plt.subplot(1, 2, 2)
    plt.hist(gray.ravel(), bins=256)
    plt.axvline(x=ret, color='r', linestyle='dashed', linewidth=2)
    plt.title('Histogram')
    plt.show()

    return binary_img
```

In [24]:
```
def localAdaptThresh(img):
    # 使用局部自适应阈值化
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    binary_img = cv2.adaptiveThreshold(gray, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
                                       cv2.THRESH_BINARY, 11, 2)

    # 显示二值图像
    plt.figure(figsize=(5, 5))
    plt.title('Local Adaptive Thresholding')
    plt.imshow(binary_img, cmap='gray')
    plt.show()

    return binary_img
```

In [25]:
```
# 全局阈值化处理并显示结果
print("Global Thresholding for pro1_page1.jpg and pro1_page2.jpg")
globalThresh(img1)
globalThresh(img2)

# 质量也就那样
```

Global Thresholding for pro1_page1.jpg and pro1_page2.jpg

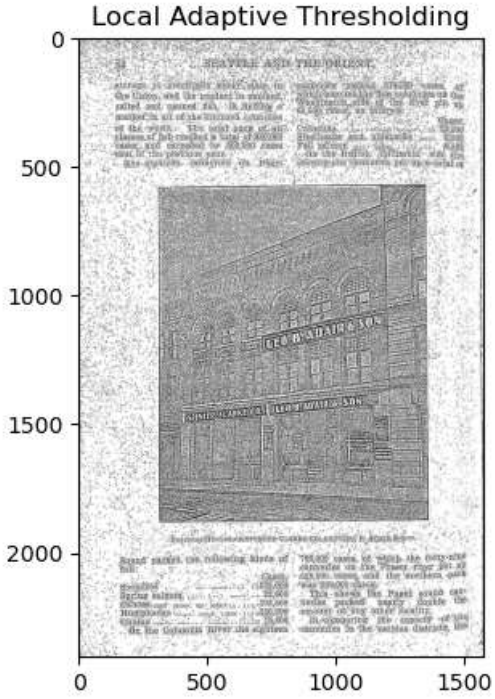


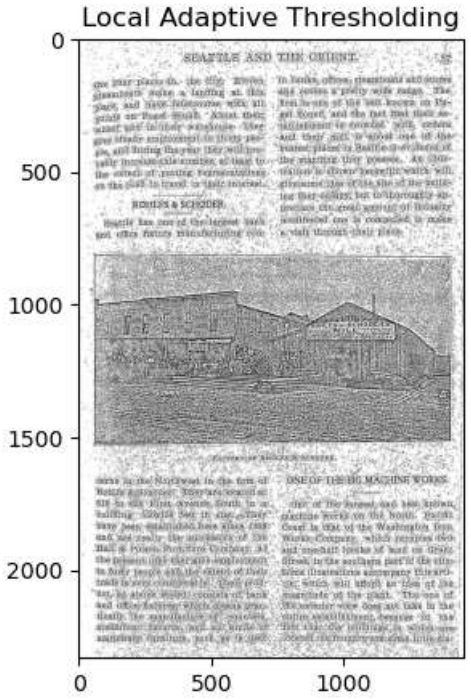


Out[25]:
```
array([[  0,   0,   0, ..., 255, 255, 255],
       [  0,   0,   0, ..., 255, 255, 255],
       [  0,   0,   0, ..., 255, 255, 255],
       ...,
       [  0,   0,   0, ..., 255, 255, 255],
       [  0,   0,   0, ..., 255, 255, 255],
       [  0,   0,   0, ..., 255, 255, 255]], dtype=uint8)
```

In [26]:
```
# 局部自适应阈值化处理并显示结果
print("Local Adaptive Thresholding for pro1_page1.jpg and pro1_page2.jpg")
localAdaptThresh(img1)
localAdaptThresh(img2)

# 质量不太行吧。。
```

Local Adaptive Thresholding for pro1_page1.jpg and pro1_page2.jpg

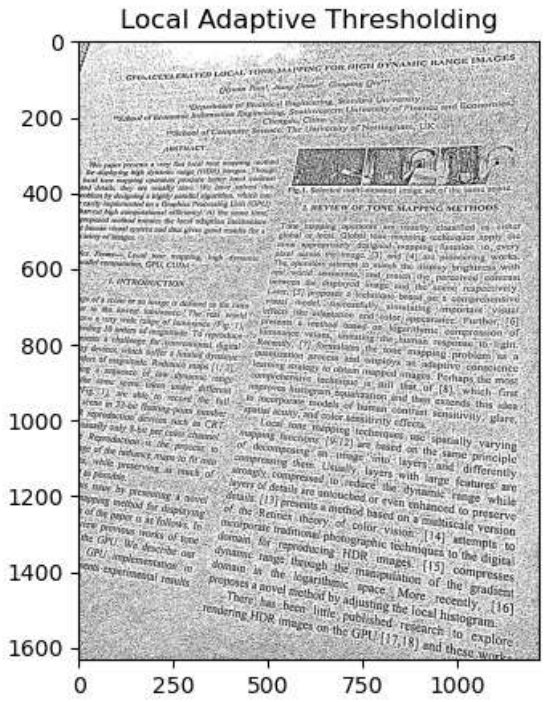




Out[26]:
```
array([[255, 255, 255, ..., 255, 255, 255],
       [255, 255, 255, ..., 255, 255, 255],
       [255, 255, 255, ..., 255, 255, 255],
       ...,
       [255, 255, 255, ..., 255, 255, 255],
       [255, 255, 255, ..., 255, 255, 255],
       [255, 255, 255, ..., 255, 255, 255]], dtype=uint8)
```

In [27]:
```
# 局部自适应阈值化并显示结果
print("Local Adaptive Thresholding for pro1_page3.png")
localAdaptThresh(img3)
```

```
Local Adaptive Thresholding for pro1_page3.png
```

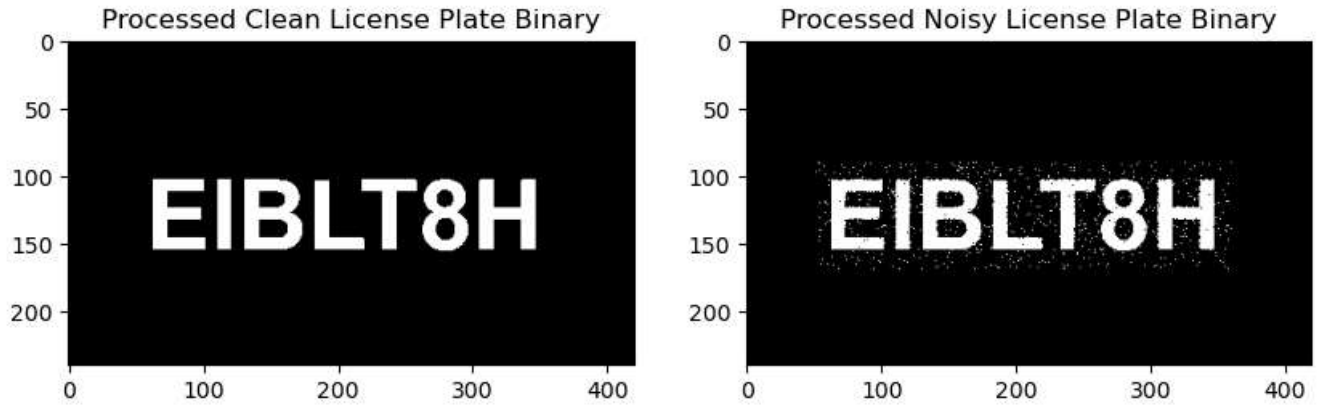


Out[27]:
```
array([[255, 255, 255, ...,   0,   0, 255],
       [255, 255, 255, ...,   0,   0,   0],
       [255, 255,   0, ..., 255,   0,   0],
       ...,
       [255, 255, 255, ..., 255, 255,   0],
       [255, 255,   0, ..., 255, 255, 255],
       [255, 255,   0, ...,   0, 255, 255]], dtype=uint8)
```

In [75]:
```
# Problem 2
import cv2
import numpy as np
import matplotlib.pyplot as plt
import os
```

In [76]:
```
def image_binarization(img):
    # 应用Otsu方法选择阈值
    ret, img_binary = cv2.threshold(img, 0, 255, cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU)

    # 创建一个全黑的图像
    mask = np.zeros_like(img_binary)

    # 只保留指定区域 (y轴90到170, x轴50到360)
    mask[90:170, 50:360] = img_binary[90:170, 50:360]

    return mask
```

In [77]:
```
# 读取干净的车牌图像
clean_plate = cv2.imread('./figs/pro2_license_plate_clean.png', 0)

# 读取噪声车牌图像
noisy_plate = cv2.imread('./figs/pro2_license_plate_noisy.png', 0)

# 检查图像是否正确读取
if clean_plate is None:
    raise ValueError("C!!")
if noisy_plate is None:
    raise ValueError("N!!")

# 处理图像
binary_clean = image_binarization(clean_plate)
binary_noisy = image_binarization(noisy_plate)

# 显示结果
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.title('Processed Clean License Plate Binary')
plt.imshow(binary_clean, cmap='gray')
plt.subplot(1, 2, 2)
plt.title('Processed Noisy License Plate Binary')
plt.imshow(binary_noisy, cmap='gray')
plt.show()

# 保存结果图像
cv2.imwrite('./figs/binary_clean.png', binary_clean)
cv2.imwrite('./figs/binary_noisy.png', binary_noisy)
```



Out[77]:
```
True
```

In [78]:
```
def read_templates(template_dir):
    templates = []
    for filename in os.listdir(template_dir):
        if filename.endswith(".png"):
            template_path = os.path.join(template_dir, filename)
            template_img = cv2.imread(template_path, 0)
            templates.append((filename.split('.')[0], template_img))
    return templates

# 模板图像文件夹路径
template_dir = './figs/Templates'

# 读取模板图像
```

```
templates = read_templates(template_dir)

# 检查模板图像是否正确读取
if not templates:
    raise ValueError("No template images found.")
```

In [79]:
```
# 读取干净的车牌图像
binary_clean = cv2.imread('./figs/binary_clean.png', 0)
# 读取噪声车牌图像
binary_noisy = cv2.imread('./figs/binary_noisy.png', 0)

# 检查图像是否正确读取
if binary_clean is None:
    raise ValueError("Binary clean plate image not found.")
if binary_noisy is None:
    raise ValueError("Binary noisy plate image not found.")
```

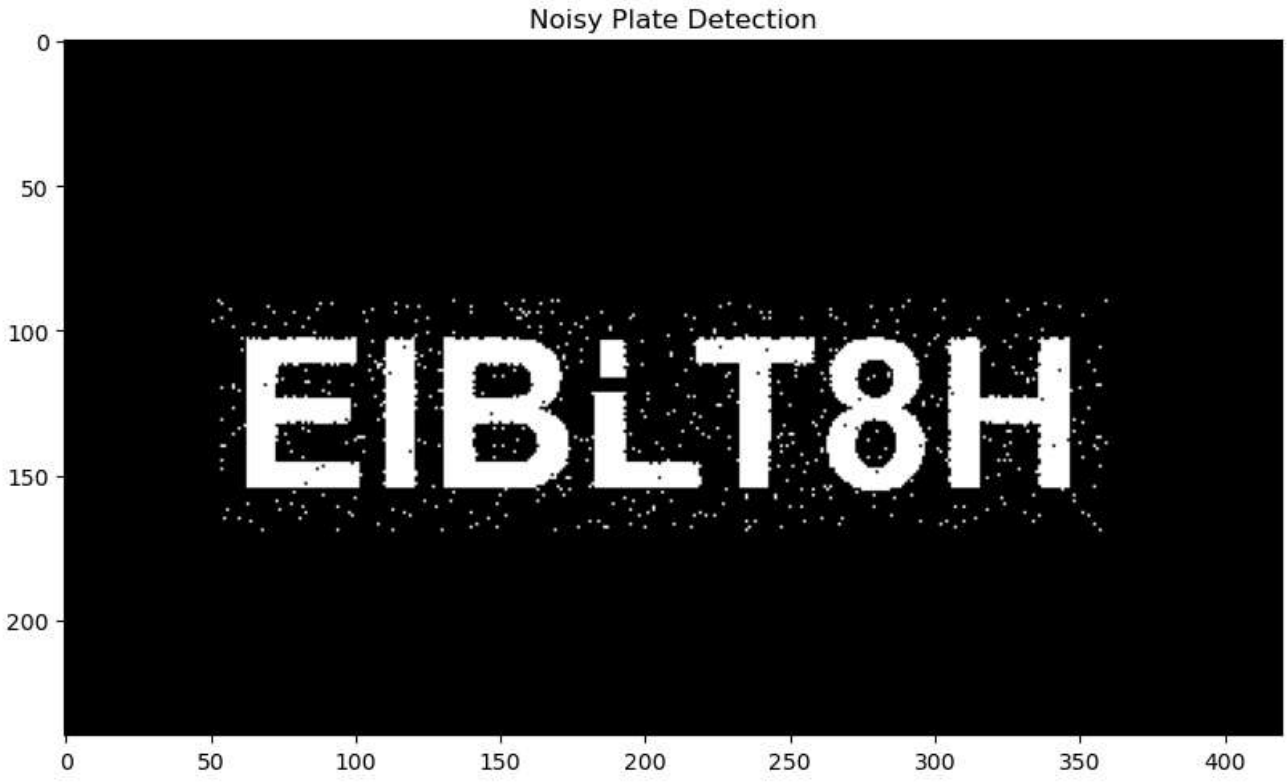
In [80]:
```
def erode_templates(templates):
    eroded_templates = []
    for char_name, template in templates:
        eroded_template = cv2.erode(template, np.ones((3, 3), np.uint8))
        eroded_templates.append((char_name, eroded_template))
    return eroded_templates

# 侵蚀
eroded_templates = erode_templates(templates)
```

In [86]:
```
def character_detection(binary_image, templates):
    detected_characters = []
    for char_name, template in templates:
        # 使用模板匹配进行检测
        result = cv2.matchTemplate(binary_image, template, cv2.TM_CCOEFF_NORMED)
        threshold = 0.5  # 设置匹配阈值
        locations = np.where(result >= threshold)  # 获取匹配结果大于阈值的位置
        if len(locations[0]) > 0:
            detected_characters.append((char_name, locations))
    return detected_characters
```

In [87]:
```
# 检测干净车牌上的字符
detected_chars_clean = character_detection(binary_clean, eroded_templates)
# 检测噪声车牌上的字符
detected_chars_noisy = character_detection(binary_noisy, eroded_templates)
print(detected_chars_clean, detected_chars_noisy)
```

```
['E', (array([88, 88, 89, 89], dtype=int64), array([182, 183, 182, 183], dtype=int64))] ['E', (array([88, 89], dtype=int64), array([183, 183], dtype=int64))]
```

In [88]:
```
# 可视化干净车牌检测结果
plt.figure(figsize=(15, 10))
for i, (char_name, loc) in enumerate(detected_chars_clean):
    for pt in zip(*loc[::-1]):
        cv2.rectangle(binary_clean, pt, (pt[0] + 20, pt[1] + 30), (0, 0, 255), 2)
    plt.subplot(2, 1, 1)
    plt.imshow(binary_clean, cmap='gray')
    plt.title('Clean Plate Detection')

# 可视化噪声车牌检测结果
for i, (char_name, loc) in enumerate(detected_chars_noisy):
    for pt in zip(*loc[::-1]):
        cv2.rectangle(binary_noisy, pt, (pt[0] + 20, pt[1] + 30), (0, 0, 255), 2)
    plt.subplot(2, 1, 2)
    plt.imshow(binary_noisy, cmap='gray')
    plt.title('Noisy Plate Detection')

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

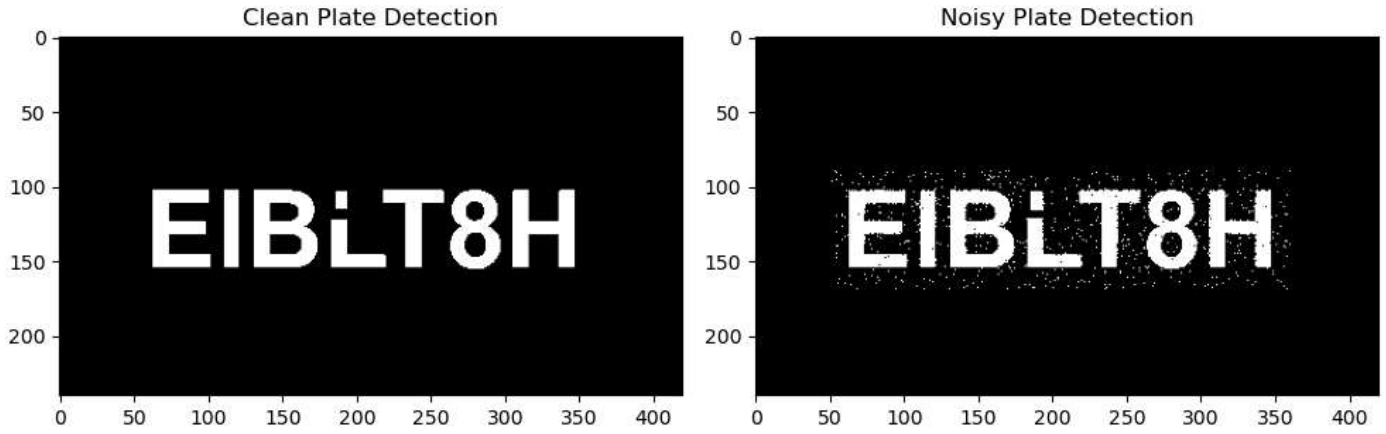


In [90]:
```python
def hit_miss_detection(binary_image, templates):
    detected_characters = []
    foreground_se = np.ones((3, 3), np.uint8)
    background_se = np.zeros((5, 5), np.uint8)
    background_se[1:4, 1:4] = 1

    for char_name, template in templates:
        # 生成击中-击不中结构元素
        fg_se = cv2.erode(template, foreground_se)
        bg_se = cv2.dilate(template, background_se)

        # 击中-击不中滤波器
        hitmiss = cv2.morphologyEx(binary_image, cv2.MORPH_HITMISS, fg_se - bg_se)

    return detected_characters
```

In [92]:
```python
# 使用击中-击不中滤波器检测干净车牌上的字符
detected_chars_clean = hit_miss_detection(binary_clean, eroded_templates)
# 使用击中-击不中滤波器检测噪声车牌上的字符
detected_chars_noisy = hit_miss_detection(binary_noisy, eroded_templates)
print(detected_chars_clean, detected_chars_noisy)
```

[] []

In [93]:
```
# 在图像上绘制检测到的字符边界框
def draw_detections(image, detections):
    for char_name, hitmiss in detections:
        locations = np.where(hitmiss > 0)
        for pt in zip(*locations[::-1]):
            cv2.rectangle(image, pt, (pt[0] + template.shape[1], pt[1] + template.shape[0]), (0, 0, 255), 2)
    return image

# 可视化干净车牌检测结果
binary_clean_detected = draw_detections(binary_clean.copy(), detected_chars_clean)
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.imshow(binary_clean_detected, cmap='gray')
plt.title('Clean Plate Detection')

# 可视化噪声车牌检测结果
binary_noisy_detected = draw_detections(binary_noisy.copy(), detected_chars_noisy)
plt.subplot(1, 2, 2)
plt.imshow(binary_noisy_detected, cmap='gray')
plt.title('Noisy Plate Detection')

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



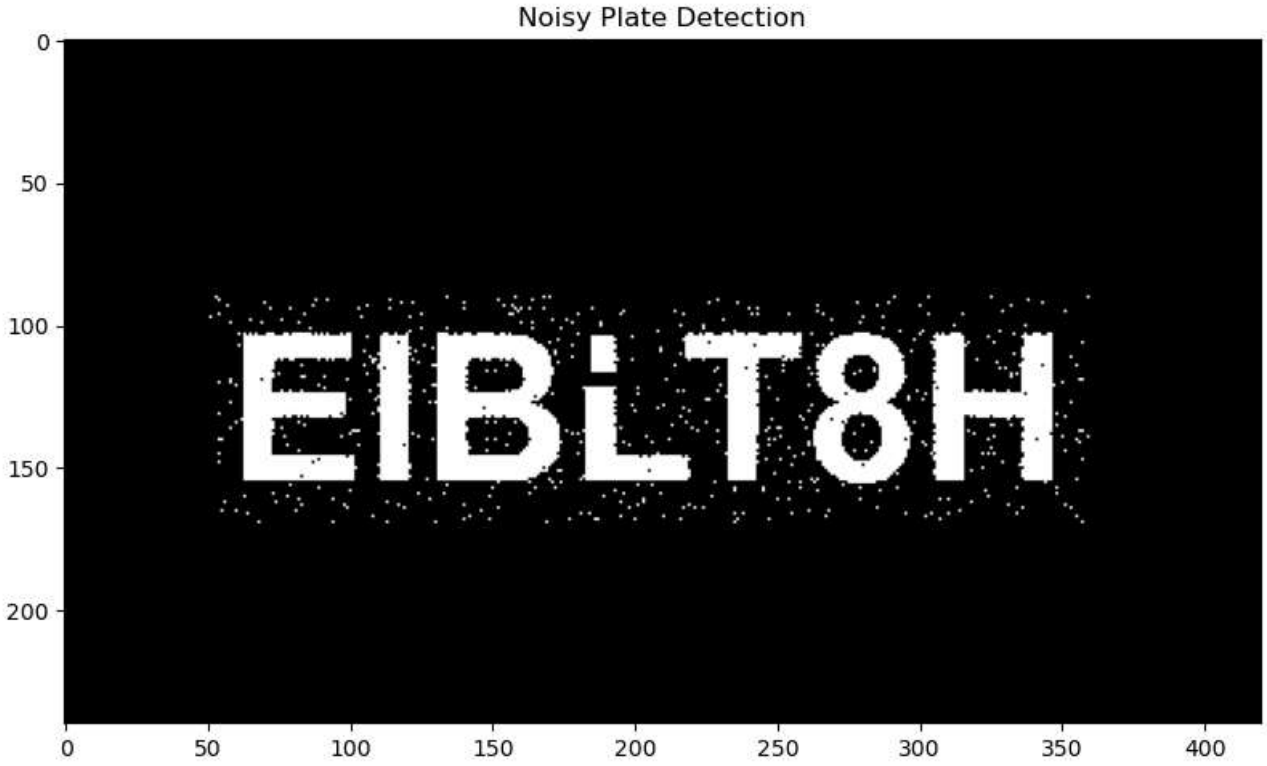
In [94]:
```
def rank_filter(image, rank, se):
    return cv2.erode(image, se) if rank == 1 else cv2.dilate(image, se)

def min_rank_filters(binary_image, templates, p1, p2):
    detected_characters = []
    se1 = np.ones((3, 3), np.uint8)  # 前景结构元素
    se2 = np.ones((5, 5), np.uint8)  # 背景结构元素

    for char_name, template in templates:
        rank_filter1 = rank_filter(binary_image, p1, se1)
        rank_filter2 = rank_filter(binary_image, p2, se2)
        min_filter = np.minimum(rank_filter1, rank_filter2)

        match = cv2.matchTemplate(min_filter, template, cv2.TM_CCOEFF_NORMED)
        threshold = 0.8
        locations = np.where(match >= threshold)

        if len(locations[0]) > 0:
            detected_characters.append((char_name, locations))

        if len(detected_characters) >= 7:
            break

    return detected_characters
```

In [97]:
```
# 使用最小秩滤波器检测噪声车牌上的字符
detected_chars_noisy = min_rank_filters(binary_noisy, eroded_templates, p1=1, p2=1)
print(detected_chars_noisy)
```

[]

In [98]:
```
# 在图像上绘制检测到的字符边界框
def draw_detections(image, detections):
    for char_name, locations in detections:
        for pt in zip(*locations[::-1]):
            cv2.rectangle(image, pt, (pt[0] + template.shape[1], pt[1] + template.shape[0]), (0, 0, 255), 2)
    return image
```

```
# 可视化噪声车牌检测结果
binary_noisy_detected = draw_detections(binary_noisy.copy(), detected_chars_noisy)
plt.figure(figsize=(10, 5))
plt.imshow(binary_noisy_detected, cmap='gray')
plt.title('Noisy Plate Detection')
plt.tight_layout()
plt.show()
```



In [67]:
```
import cv2
import numpy as np
import matplotlib.pyplot as plt

# 读取输入图像
img = cv2.imread('./figs/pro3_road_sign_school_blurry.jpg', cv2.IMREAD_GRAYSCALE)
```

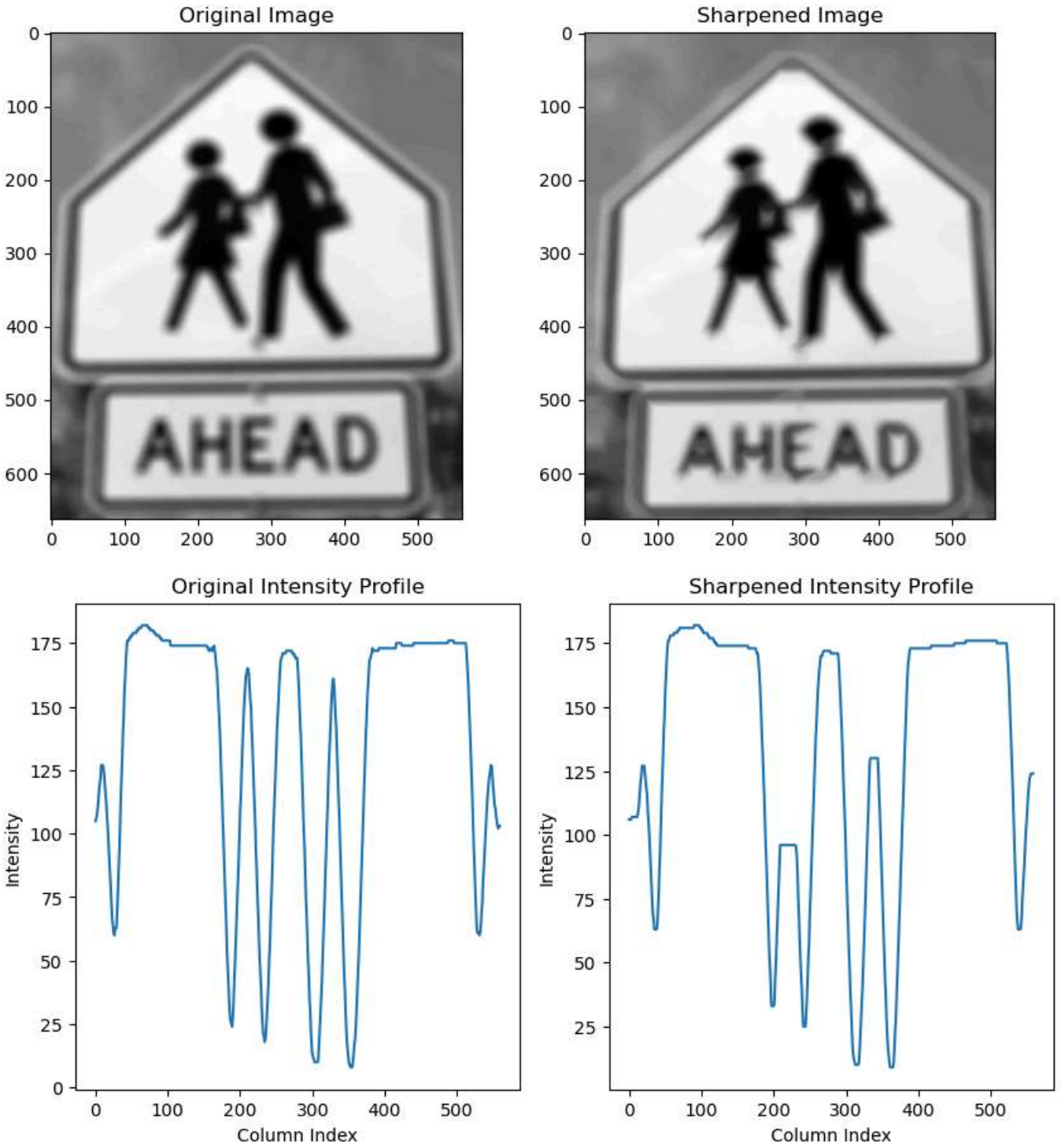
In [84]:
```
# 设计结构元素，这里使用一个3x3的矩形结构元素
structuring_element1 = cv2.getStructuringElement(cv2.MORPH_RECT, (4, 4))
structuring_element2 = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (4, 4))

print(structuring_element1, structuring_element2)
```

```
[[1 1 1 1]
 [1 1 1 1]
 [1 1 1 1]
 [1 1 1 1]] [[0 0 1 0]
 [1 1 1 1]
 [1 1 1 1]
 [1 1 1 1]]
```

In [85]:
```
# 定义迭代算法进行图像锐化
def sharpen_image(img, iterations, structuring_element):
    for _ in range(iterations):
        # 膨胀图像
        img = cv2.dilate(img, structuring_element1)
        # 腐蚀图像
        img = cv2.erode(img, structuring_element2)
        #img = cv2.addWeighted(img, 1.5, dilated, -0.5, 0)
    return img
```

In [86]:
```
# 应用算法进行10次迭代
sharpened_img = sharpen_image(img, 10, structuring_element)
```

In [87]:
```
# 显示结果图像
plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)
plt.title('Original Image')
plt.imshow(img, cmap='gray')

plt.subplot(1, 2, 2)
plt.title('Sharpened Image')
plt.imshow(sharpened_img, cmap='gray')

plt.show()

# 获取第338行的强度轮廓
row = 338
original_profile = img[row, :]
sharpened_profile = sharpened_img[row, :]

# 绘制强度轮廓图
plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)
plt.title('Original Intensity Profile')
plt.plot(original_profile)
plt.xlabel('Column Index')
plt.ylabel('Intensity')

plt.subplot(1, 2, 2)
plt.title('Sharpened Intensity Profile')
plt.plot(sharpened_profile)
plt.xlabel('Column Index')
plt.ylabel('Intensity')

plt.show()
```



In [ ]:
```
# It shows a sharper result.
```