甲状腺前景提取报告

肖蔚尔 520030910314

1. 分水岭

(1) 算法原理

分水岭算法是一种基于图像梯度的图像分割算法，它通过模拟水流从高处向低处流动的过程来实现分割。算法的基本原理是将图像梯度视为地形的高度，将图像的亮度视为地形的海拔，然后在图像中建立起水流分割的边界，形成不同的区域。主要步骤如下：

1. 对原始图像进行预处理，如平滑滤波、梯度计算等。
2. 根据预处理结果确定分水岭的种子点，种子点表示不同物体或背景。
3. 构建分水岭标记图，其中种子点的标记为不同的整数。
4. 通过分水岭算法计算图像中的水流路径，将图像分割为不同的区域。
5. 对分割结果进行后处理，如去除小区域、平滑边界等。

(2) 实现方案

以下是使用分水岭算法进行甲状腺前景提取的具体实现方案:

1. 图像预处理:对原始图像进行中值滤波，去除噪声。

```
def preprocess(self, img):
        blur = cv2.medianBlur(img, MEDIAN_KERNEL_SIZE)
        return blur
```

2. 确定种子点:通过用户交互方式确定甲状腺前景和背景的种子点。

```
def draw_seed(self, event, x, y, flags, param):
        if event == cv2.EVENT_LBUTTONDOWN:
            self.is_drawing = True
            self.seeds_fg.append((x, y))
        elif event == cv2.EVENT_RBUTTONDOWN:
            self.is_drawing = True
            self.seeds_bg.append((x, y))
        elif event == cv2.EVENT_LBUTTONUP or event == cv2.EVENT_RBUTTONUP:
            self.is_drawing = False
```

3. 构建分水岭标记图:将种子点标记为不同的整数。
4. 执行分水岭算法:使用 OpenCV,提供的 cv2.watershed()函数执行分水岭算法，计算出分割结果。

```
def segment(self):
        self.marks = np.zeros(self.blur.shape[:2], dtype=np.int32)
        for seed in self.seeds_fg:
            cv2.drawMarker(self.marks, tuple(seed), 1, cv2.MARKER_TILTED_CROSS)
        for seed in self.seeds_bg:
            cv2.drawMarker(self.marks, tuple(seed), 2, cv2.MARKER_TILTED_CROSS)
        cv2.watershed(self.blur, self.marks)
```

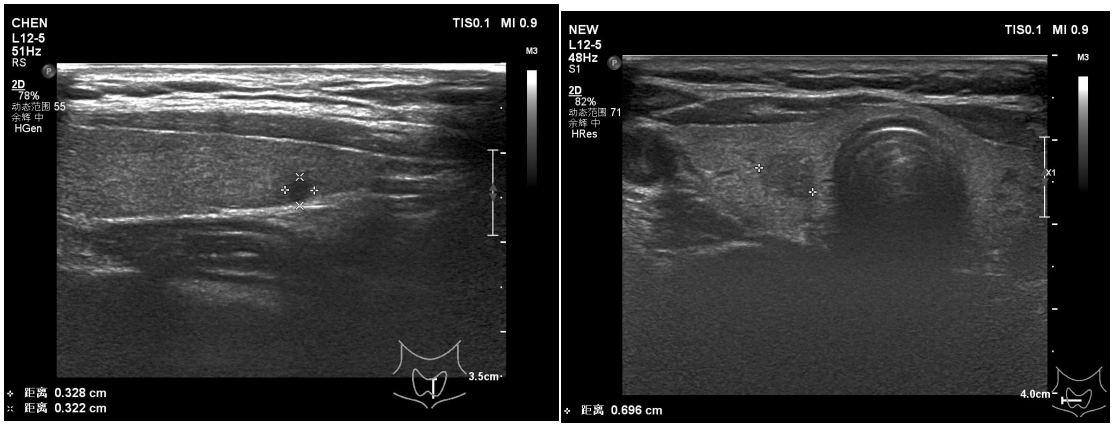
5. 后处理:去除小区域，并平滑分割边界。

```python
@staticmethod
    def postprocess(img):
        kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, MORPH_KERNEL_SIZE)
        img = cv2.morphologyEx(img, cv2.MORPH_CLOSE, kernel)
        img = cv2.blur(img, BLUR_KERNEL_SIZE, borderType=cv2.BORDER_REPLICATE)
        img = cv2.threshold(img, POST_THRESHOLD, 255, cv2.THRESH_BINARY)[1]
        return img
```

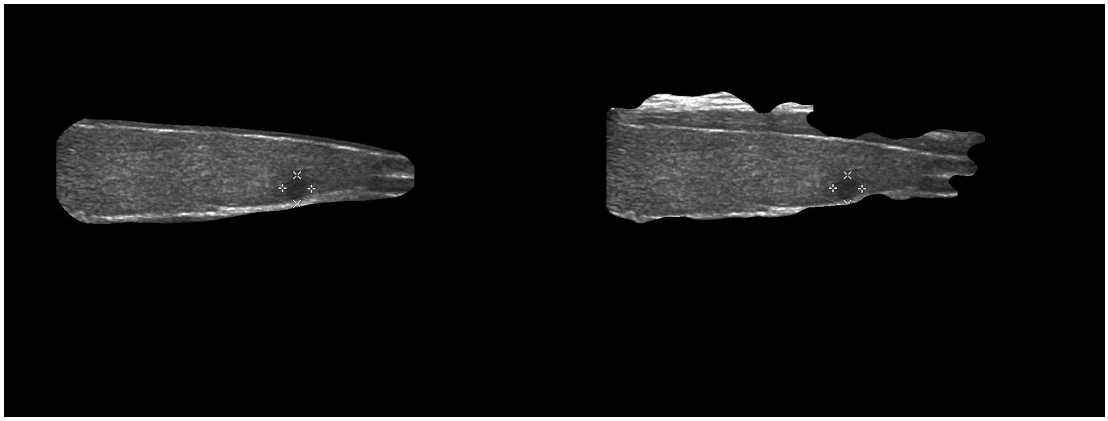
6. 输出结果:将分割结果保存为图像文件。

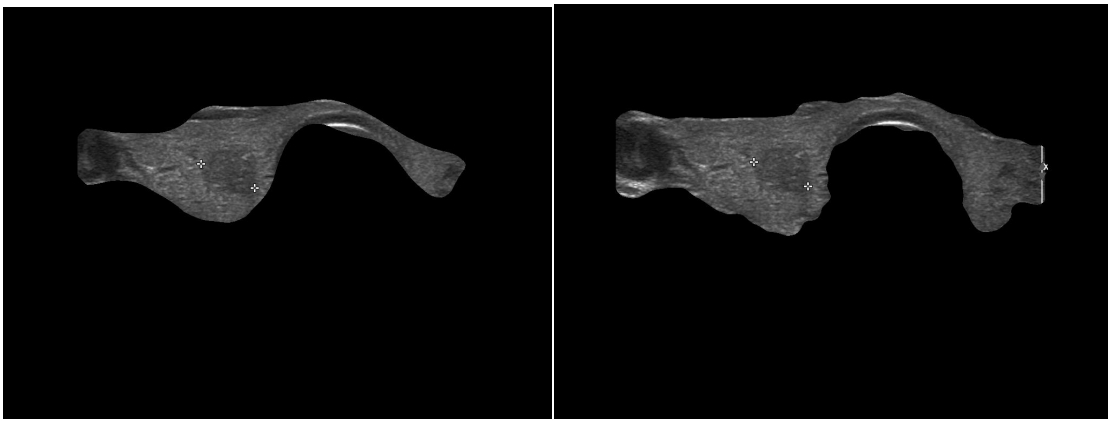
(3) 运行结果

操作方式：
单击鼠标左键标记多个前景种子点；
单击鼠标右键标记多个背景种子点；
按 Q 进行前景提取并切换到下一张待标记图像；
按 W 结束程序。
分别选择了一张横切和纵切的原始图像进行结果展示。
下图为原始图像（55.png，219.png）：

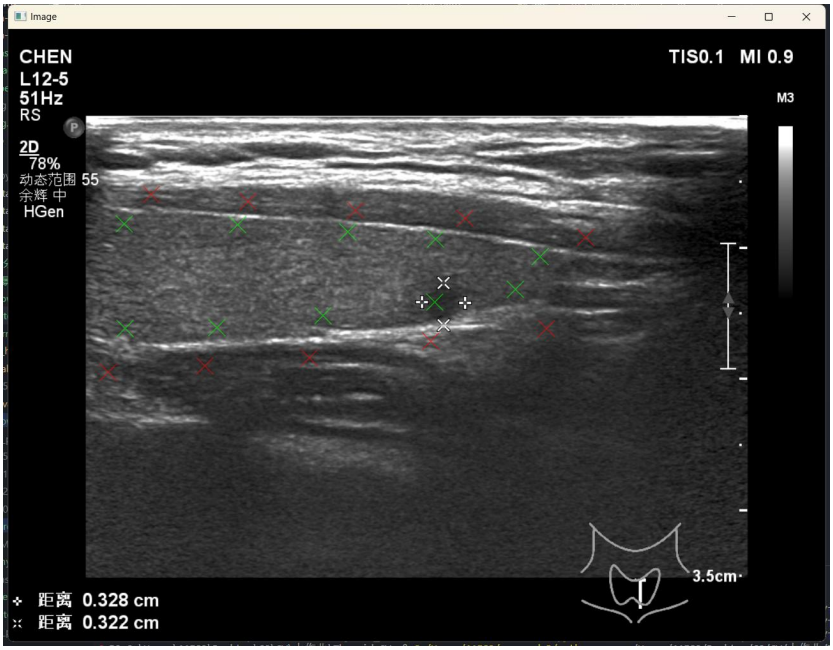


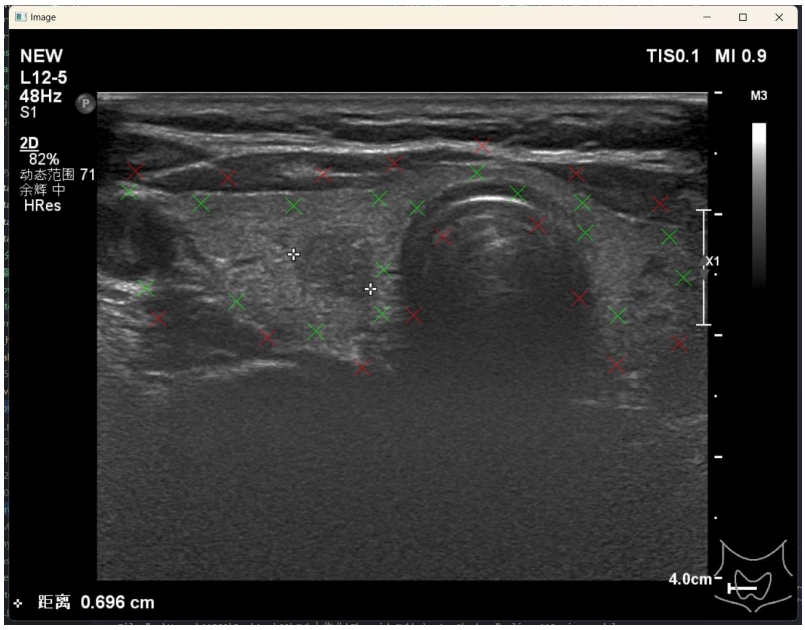
下图为调参过程中参数不合适产生的较差提取结果图：

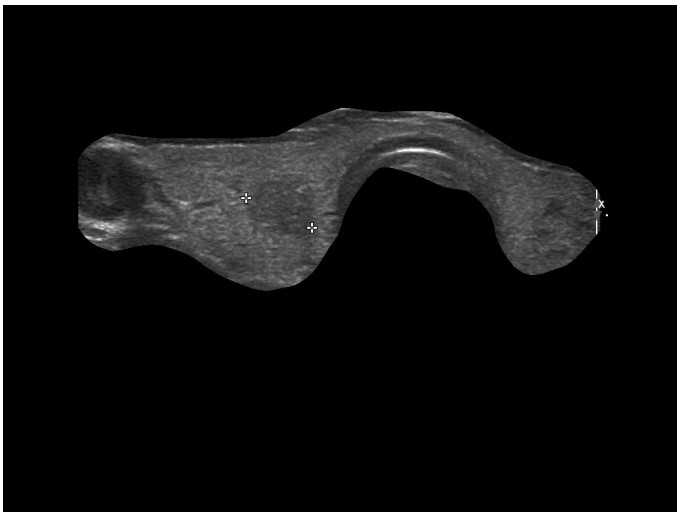




下图为大点情况与调参后较优结果图：







2. 区域增长

(1) 算法原理

增长算法是一种基于像素相似性的图像分割算法，它从种子像素开始逐步生长，将与种子像素相似的像素归为同一区域。算法的基本原理是通过定义像素之间的相似度度量，不断将相似的像素添加到同一区域中，直到达到停止条件。下为主要步骤：

1. 选择种子像素作为初始区域。
2. 计算当前像素与邻 t 域像素的相似度。
3. 判断相似度是否满足阈值条件，满足则将该像素添加到当前区域中。
4. 重复步骤 2.和步骤 3，直到没有符合条件的像素为止。

(2) 实现方案

以下是使用区域增长算法进行甲状腺前景提取的具体实现方案:

1. 图像预处理:对原始图像进行中值滤波，去除噪声。

2. 确定种子点:通过用户交互方式确定甲状腺前景的种子点。
3. 执行区域增长算法:从种子点开始,逐步生长区域,将与种子像素相似的像素归为同一区域。

```
def region_grow(self, img, seeds):
        result = np.zeros_like(img)
        for seed in seeds:
            marks = [seed]
            gray = [np.mean(img[seed[1]-REGION_SIZE:seed[1]+REGION_SIZE+1,
                                seed[0]-REGION_SIZE:seed[0]+REGION_SIZE+1])]
            new_img = np.zeros_like(img)
            cv2.rectangle(new_img, (seed[0] - HALF_REGION_SIZE, seed[1] - HALF_REGION_SIZE),
                          (seed[0] + HALF_REGION_SIZE, seed[1] + HALF_REGION_SIZE), 255, -1)
            while len(marks) > 0:
                marker = marks.pop()
                new_img, new_marks = Growth._traverse_adjacent_pixel(img, new_img, gray, marker[0],
marker[1])
                marks.extend(new_marks)
            result = cv2.bitwise_or(result, new_img)
        return result
```

4. 遍历 8 邻域像素进行区域增长（增长条件：两个区域的灰度共生矩阵相似度大于阈值；区域的灰度与原区域的灰度差值小于阈值）

```
def traverse_adjacent_pixel(img, new_img, gray, x, y):
        new_markers = []
        adjacent = [(x + j * REGION_SIZE, y + i * REGION_SIZE) for i in range(-1, 2) for j in range(-1,
2)]
        for i, j in adjacent:
            if not Growth.in_range(i, j) or (i == x and j == y) or new_img[j, i] == 255:
                continue
            s1, s2, g = Growth.calc_dist_pixel(img, x, y, i, j)
            if s1 > REGION_SIMILARITY_THRESHOLD and \
                    s2 > REGION_SIMILARITY_THRESHOLD and \
                    abs(g - gray) < GRAY_DIFF_THRESHOLD:
                cv2.rectangle(new_img, (i - HALF_REGION_SIZE, j - HALF_REGION_SIZE),
                              (i + HALF_REGION_SIZE, j + HALF_REGION_SIZE), 255, -1)
                new_markers.append((i, j))
        return new_img, new_markers
```

5. 后处理:去除小区域，并平滑分割边界。

```
@staticmethod
    def postprocess(img):
        kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, MORPH_KERNEL_SIZE)
        img = cv2.morphologyEx(img, cv2.MORPH_CLOSE, kernel)
        img = cv2.blur(img, BLUR_KERNEL_SIZE, borderType=cv2.BORDER_REPLICATE)
        img = cv2.threshold(img, POST_THRESHOLD, 255, cv2.THRESH_BINARY)[1]
        return img
```

6. 输出结果:将分割结果保存为图像文件。

(3) 运行结果

操作方式：

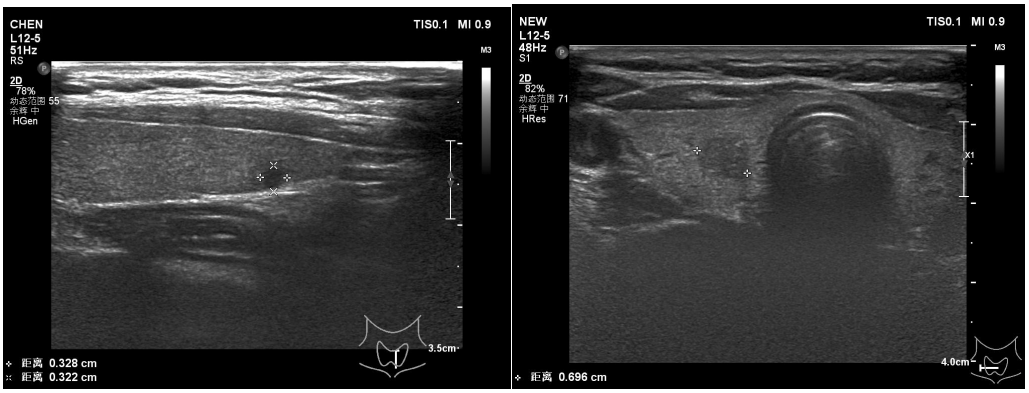
单击鼠标左键标记多个种子点；

按 Q 进行前景提取并切换到下一张待标记图像；

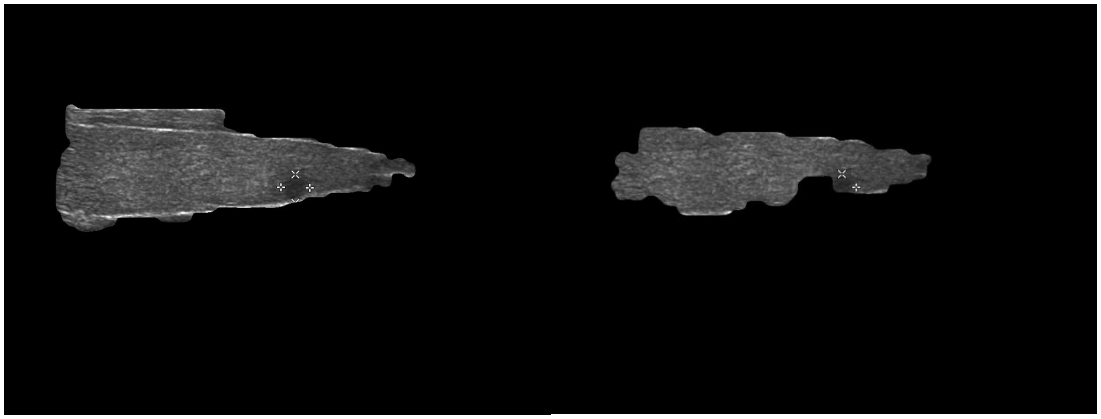
按 W 结束程序。

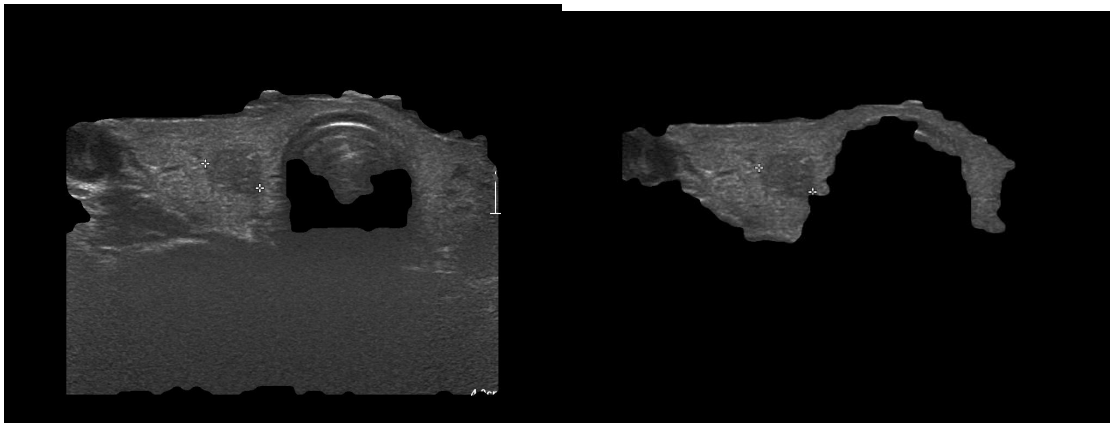
分别选择了一张横切和纵切的原始图像进行结果展示。

下图为原始图像（55.png，219.png）：

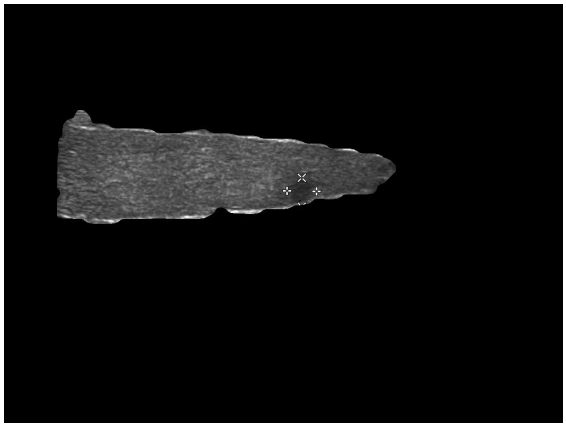


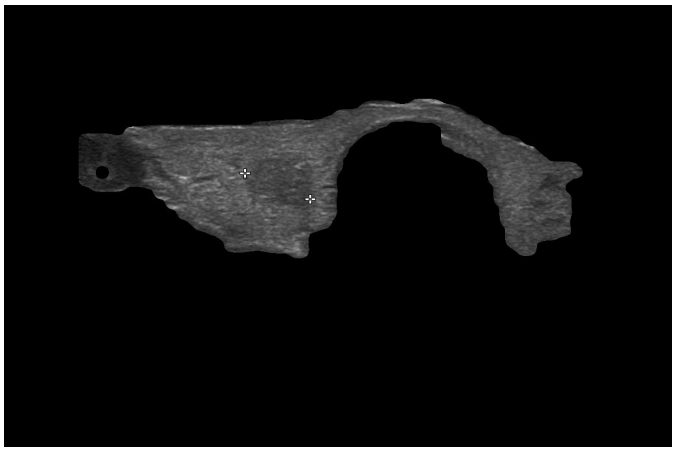
下图为调参过程中参数不合适产生的较差提取结果图：





下图为大点情况与调参后较优结果图：





3. 中间过程

在中间结果中，两种算法都涉及到用户交互，通过选择种子点来指示前景和背景。分水岭算法通过计算分水岭边界将图像分割成多个区域，而区域增长算法通过逐步生长来形成一个或多个区域。最终结果，分水岭算法和区域增长算法都能够提取出甲状腺的前景。然而，分水岭算法可能会产生过分细化的分割边界，导致结果不够平滑（如图 1）;区域增长算法可能会受到种子点选择的影响，导致结果不够准确（如图 2、图 3）。

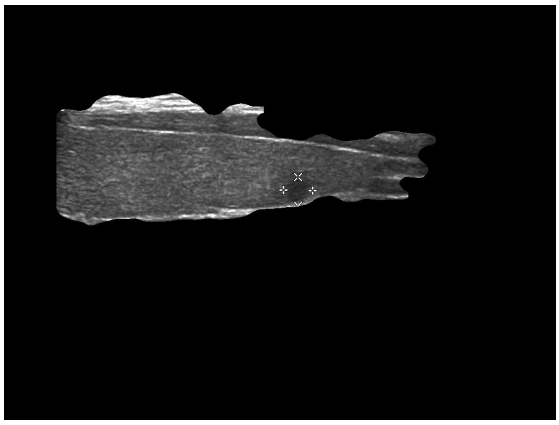


图 1

图 2

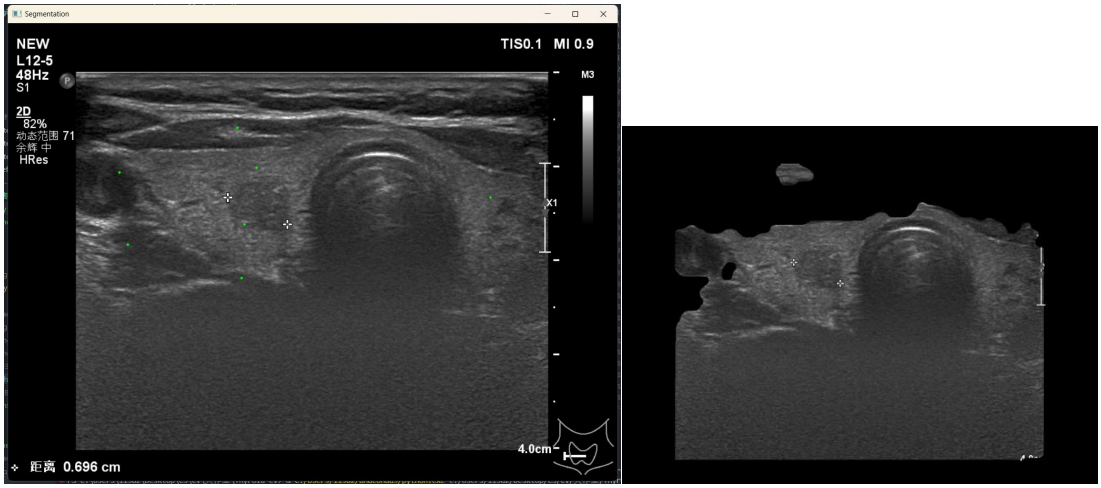


图 3

4. 算法比较&优化

(1) 算法比较

分水岭算法和区域增长算法在甲状腺前景提取中都能够产生合理的结果，但存在一些差异:

分水岭算法在处理复杂图像时可能会产生过分细化的分割边界，需要进行后处理来平滑边界；区域增长算法对种子点的选择较为敏感，可能需要人工干预来调整种子点的位置，以获得准确的分割结果。

分水岭算法具有较好的并行性，适合处理大规模图像数据；区域增长算法在处理图像噪声较多的情况下可能会产生错误的生长，需要进行噪声处理或者相似度度量的优化。

(2) 优化方案

针对以上算法比较中提到的差异，可以考虑以下优化方案:

1. 对于分水岭算法，可以尝试使用更复杂的预处理方法，如边缘增强、颜色空间转换等，以提高分害边界的准确性和平滑性。
2. 对于区域增长算法，可以考虑引入更精确的相似度度量方法，如颜色、纹理、形状等特征的组合,以增强算法对噪声和纹理变化的鲁棒性。
3. 可以结合两种算法的优点，采用混合算法的方式进行甲状腺前景提取,如先使用区域增长算法获取初始分害结果,然后利用分水岭算法进行边界优化和细化。
4. 可以利用机器学习方法，如卷积神经网络(CNN）或者基于图像分割的深度学习模型，进行自动化的甲状腺前景提取。