ASSESSED EXERCISE 2

Joel Bailey – s5015120

2019

**Image Processing – Skeletonization**

Skeletonization is a method of reducing the foreground elements of a binary image until a skeletal structure of the original foreground elements remains. An ideal skeleton should predominantly preserve the extent and connectivity of the original foreground region, whilst also removing as many of the original foreground pixels as possible. There are number of different methods that can be applied in order to produce a skeleton image. The below implementation uses a successive thinning approach that will erode the foreground pixels until the skeleton structure has been achieved.

To produce a skeleton image the thinning operation is applied to the binary image iteratively. Once the thinning operation can no longer make any changes to the given image, the skeletal structure has been accomplished. Thinning is a morphological operation. This means that it is part of a set of operations that determine each pixel value based on the values of the neighboring pixels. To achieve this, a structuring element (a matrix of binary values) is applied over a region of the image, with the selected pixel value located at the structuring elements origin. Depending on which operation is being performed, the selected region is compared with the structuring element to see if it Fits, Hits or Misses. If the structuring element Fits, then all pixels in the image region match all the values in the structuring element. If the structure element Hits, then at least a single pixel in the image region matches a value in the structuring element. If the structuring element Misses, then no pixels in the image region matches the structuring element. The thinning operation that has been implemented is expressed in Equation (1), where I is the given image and J is the structuring element.

thin(I, J) = I – hit-and-miss(I, J) (1)

To put it simply, the structuring element’s origin is placed over each pixel in the given image. If the structuring element matches (Fits) the selected image region exactly, then the pixel located at the structuring element’s origin will be set to the background (zero). If the structuring element does not fit, then the pixel will be left unchanged. As can be seen, this is a destructive process that deletes pixel data from the image. However, the behavior of this method can still be greatly influenced by the structuring element that is applied. To achieve the skeletal structure, eight structuring elements will be applied to the image in each thinning iteration. Figure 1 represents the first structuring element to be applied, and Figure 2 represents the second structuring element. The origin of both structuring elements is represented by the center value. Both structuring elements are then rotated 90 degrees and applied to the image again in the same order. This is completed until all 90-degree rotations of the structuring elements have been applied (4 x 2 = 8 total structuring elements).

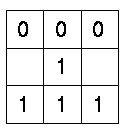
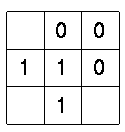


Figure 1. Structuring Element 1 Figure 2. Structuring Element 2

Performing one thinning iteration on the binary image will only remove a certain amount of foreground pixels and will not result in the skeleton structure that we are after. To create the intended skeletal structure the thinning operation will need to be continuously applied to the binary image. Once the thinning operation has no affect on the given binary image, the skeletal structure will have been created. The key benefit of a skeleton image is that it provides a simple and compact representation of a shape that preserves many of the topological and size characteristics of the original shape. Figure 3 and Figure 4 demonstrate the result of the skeletonization process.

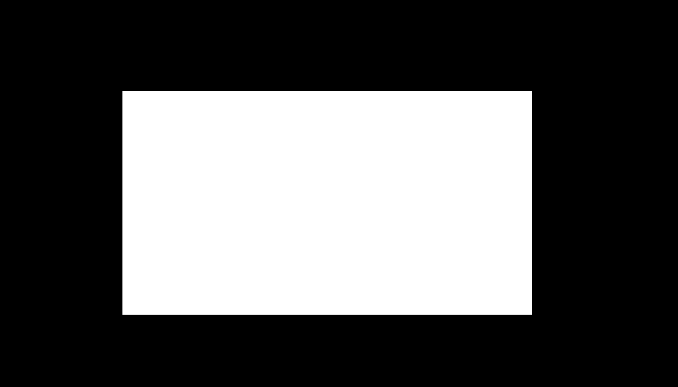
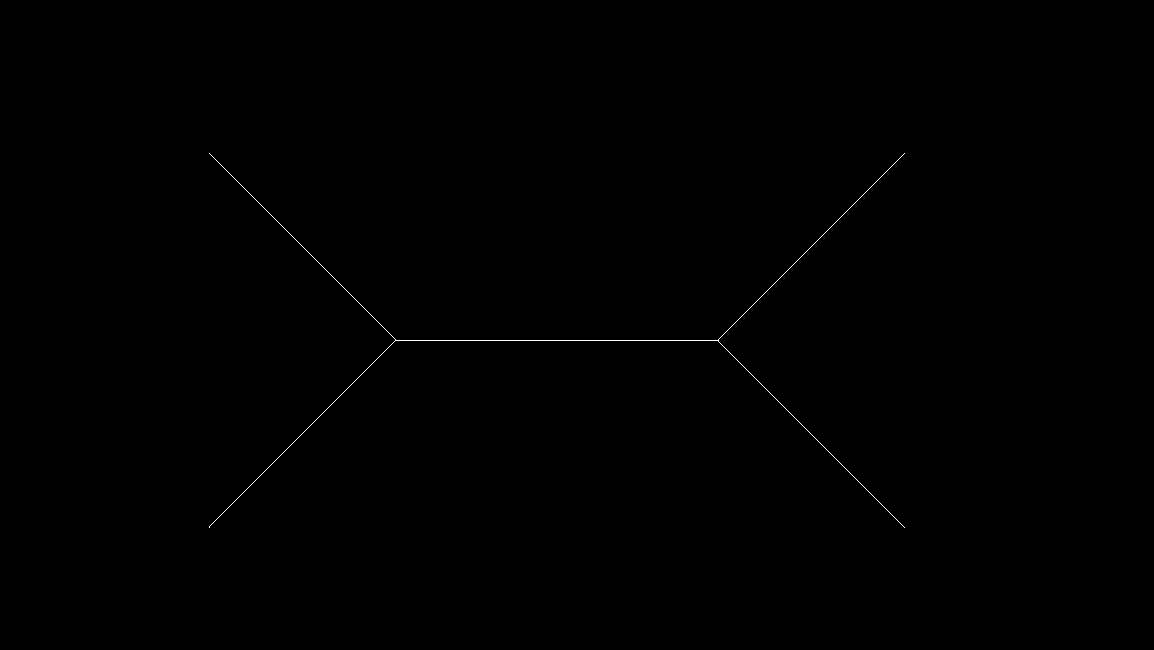


Figure 3. Original Binary Image Figure 4. Skeleton Image

**Implementation of Skeletonization**

Figure 5 displays the pseudocode representation of the algorithm described in the previous section. The skeletonization of each image is achieved using two separate methods. The first method named “thinningItr()” represents one iteration of the thinning operation. In this method each structuring element is created and applied to the image. The resulting image is than compared to the given image to see if the operation had any affect on the given image. If no changes had been detected than the skeletonComplete variable will be set to true, indicating the completion of the skeletonization process. The structuring elements generated in this method use the value 255 rather than 1. This is because in a bitmap image, 255 represents white (the foreground). The structuring element instead uses the value 1 to indicated pixels to ignore. The second method implemented is called “applyMask()”. This method applies a given structuring element to every pixel in the image. As described above, if the structuring element fits, then the pixel located at the origin of the structuring element will be set too the background (zero). This function will then set the current image data to the resulting image. By iterating through these two methods, the skeleton of any binary image can be created.

1. ***Initialize Variables***  
2. currentImgData = [[]]; ***Will be set to binary image data in program.***  
3. skeletonComplete = false;  
4.  
5. thinningItr() {  
6. structEl1 = [[0,0,0],[1,255,1],[255,255,255]];  
7. structEl2 = [[1,0,0],[255,255,0],[1,255,1]];  
8. structEl3 = [[255,1,0],[255,255,0],[255,1,0]];  
9. structEl4 = [[1,255,1],[255,255,0],[1,0,0]];  
10. structEl5 = [[255,255,255],[1,255,1],[0,0,0]];  
11. structEl6 = [[1,255,1],[0,255,255],[0,0,1]];  
12. structEl7 = [[0,1,255],[0,255,255],[0,1,255]];  
13. structEl8 = [[0,0,1],[0,255,255],[1,255,1]];  
14.  
15. original = currentImgData;  
16.  
17. applyMask(structEl1);  
18. applyMask(structEl2);  
19. applyMask(structEl3);  
20. applyMask(structEl4);  
21. applyMask(structEl5);  
22. applyMask(structEl6);  
23. applyMask(structEl7);  
24. applyMask(structEl8);  
25.  
26. If (original == currentImgData) {  
27. skeletonComplete = true;  
28. }  
29. return;  
30. }  
31.  
32. applyMask(mask) {  
33. fit = true;  
34. Img = currentImgData;  
35.  
36. for i=0; i < img.size(); i++ {  
37. for j = 0; j = img[j].size(); j += 3 {  
38. ***Don’t apply mask to padding around boarder***  
39. If (i == 0 || i == (img.size() - 1) || j == 0 || j == (img[i].size() - 1)) {  
40. continue;  
41. }  
42.  
43. ***Iterate through each mask element.***  
44. if (img[i][j] == 255) { ***Check if it is foreground element.***  
45. fit = true;  
46.   
47. for x = 0; x < mask.size(); x++ {  
48. for y = 0; y < mask[x].size(); y++ {  
49. Find index for corresponding pixel value.  
50. index1 = i + (x - 1);  
51. index2 = j + ((y - 1) \* 3)  
52. If (mask[x][y] != 1) {  
53. If (currentImgData[index1][index2] != mask[x][y]) {  
54. fit = false;  
55. break;  
56. }  
57. }  
58. }  
59. if (not fit) {  
60. break;  
61. }  
62. }  
63.  
64. ***If mask fits set origin pixel to background.***  
65. if (fit) {  
66. img[i][j] = 0;  
67. img[i][j + 1] = 0;  
68. img[i][j + 2] = 0;  
69. }  
70. }  
71. }  
72. }  
73. ***Set current image data to resulting image.***  
74. currentImgData = img;  
75. return;  
76. }

Figure 5. Pseudocode implementation of skeletonization.

The pseudocode in Figure 5 has successfully been implemented in C++11. This code can be found in the “skeleton.cpp” file. The two methods shown above are apart of the “Image” class. As stated before these methods are iterated through in order to turn a binary image into a skeleton image. This process can be found in the “createSkeleton()” function.

**References**

[1]"Morphology - Skeletonization/Medial Axis Transform", *Homepages.inf.ed.ac.uk*, 2019. [Online]. Available: http://homepages.inf.ed.ac.uk/rbf/HIPR2/skeleton.htm. [Accessed: 1- May- 2019].

[2]"Morphology - Thinning", *Homepages.inf.ed.ac.uk*, 2019. [Online]. Available: http://homepages.inf.ed.ac.uk/rbf/HIPR2/thin.htm. [Accessed: 1- May- 2019].

[3]"PTC Mathcad Help", *Support.ptc.com*, 2019. [Online]. Available: http://support.ptc.com/help/mathcad/en/index.html#page/PTC\_Mathcad\_Help%2Fexample\_thinning\_and\_skeletonization.html%23wwID0EQHLX. [Accessed: 03- May- 2019].