The Recursive Grass-Fire Algorithm

The algorithm starts in the upper-left corner of the binary image. It then scans the entire image from left to right and from top to bottom, as seen in Fig. 4.28.

At some point during the scan an object pixel (white pixel) is encountered and the notion of grass-fire comes into play. In the binary image in Fig. 7.3 the first object pixel is found at the coordinate (2, 0). At this point you should imagine yourself standing in a field covered with dry grass. Imagine you have four arms (!) and are holding a burning match in each hand. You then stretch out your arms in four different directions (corresponding to the neighbors in the 4-connectivity) and simultaneously drop the burning matches. When they hit the dry grass they will each start a fire which again will spread in four new directions (up, down, left, right) etc. The result is that every single straw which is connected to your initial position will burn. This is the grass-fire principle. Note that if the grass field contains a river the grass on the other side will not be burned.

Returning to our binary image, the object pixels are the "dry grass" and the nonobject pixels are water. So, the algorithm looks in four different directions and if it finds a pixel which can be "burned", meaning an object pixel, it does two things.

Firstly, in the output image it gives this pixel an object label (basically a number) and secondly it "burns" the pixel in the input image by setting it to zero (black). Setting it to zero indicates that it has been burned and will therefore not be part of yet another fire. In the real grass field the fire will spread simultaneously in all directions. In the computer, however, we can only perform one action at the time and the grass-fire is therefore performed as follows.

Let us apply the principle on Fig. 7.3. The pixel at the coordinate (2, 0) is labeled 1, since it is the first BLOB and then burned (marked by a 1 in the lower right corner). Next the algorithm tries to start a fire at the first neighbor (3,0), by checking if it is an object pixel or not. It is indeed an object pixel and is therefore labeled 1 (same object) and "burned". Since (3,0) is an object pixel, it now becomes the center of attention and its first neighbor is investigated (4,0). Again, this is an object pixel and is therefore labeled 1, "burned" and made center of attention. The first neighbor of (4, 0) is outside the image and therefore per definition not an object pixel. The algorithm therefore investigates its second neighbor (4,1). This is not an object pixel and the third neighbor of (4,0) is therefore investigated (3,0). This has been burned and is therefore no longer an object pixel. Then the last neighbor of (4, 0) is investigated (4, -1).

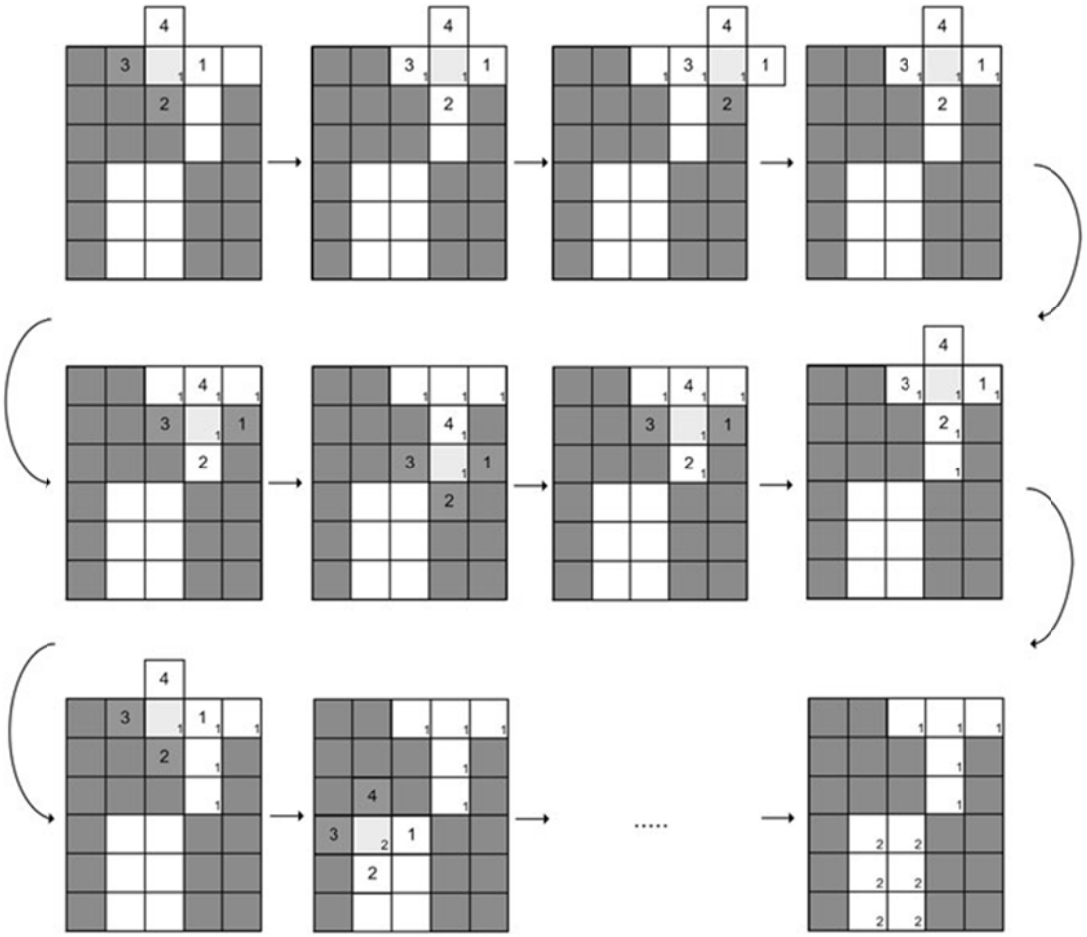

Fig. 7.3 The grass-fire algorithm. The "big" numbers indicate the order in which the neighbors are visited. The small numbers indicate the label of a pixel

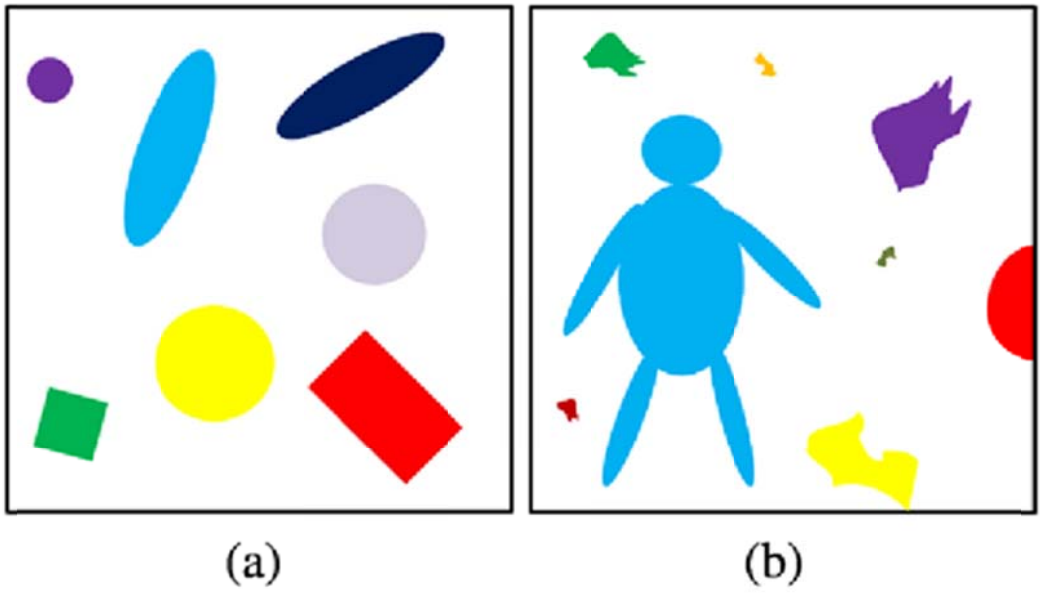

Fig. 7.4 Two examples of extracted BLOBs. Each BLOB has a unique color

This is outside the image and therefore not an object pixel. All the neighbors of (4, 0) have now been investigated and the algorithm therefore traces back and looks at the second neighbor of (3, 0), namely (3,1). This is an object pixel and is therefore labeled 1, burned and becomes the new focus of attention. In this way the algorithm also finds (3,2) to be part of object 1 and finally ends by investigating the fourth neighbor of (2, 0).

All pixels which are part of the top object have now been labeled with the same label 1 meaning that this BLOB has been segmented. The algorithm then moves on following the scan path in Fig. 4.28 until it meets the next object pixel (1, 3), which is then labeled 2, and starts a new grass-fire. The result will be the image shown in Fig. 7.2, where each BLOB has a unique label. In Fig. 7.4 the BLOBs from Fig. 7.1 have been extracted and color coded according to their BLOB label.

The algorithm can be implemented very efficiently by a function calling itself. Such an algorithm is said to be recursive and care should be taking to ensure the program is terminated properly as recursive algorithms have no built-in termination strategy. Another danger is that a computer has a limited amount of memory allocated to function calls. And since the grass-fire algorithm can have many thousands function calls queued up, the computer can run out of allocated memory.

The Sequential Grass-Fire Algorithm

The grass-fire algorithm can also be implemented in a sequential manner. This is less efficient from a programming point of view, but it does not suffer from the problems mentioned above for the recursive grass-fire algorithm.

The sequential grass-fire algorithm also scans the image from top left to bottom right. When an object pixel is found it does two things. Firstly, in the output image it gives this pixel an object label, 1, and secondly it “burns” the pixel in the input image by setting it to zero (black). The next step is to check the neighbors (four or eight pixels depending on the connectivity) and see if any of them is an object pixel. So far this is exactly the same as the recursive grass-fire algorithm, but now comes the difference. If any of the neighbors is an object pixel they are labeled 1 in the output image and set to zero (burned) in the input image. Also, they are placed in a list. Next step is to take the first pixel in the list and check its neighbors. If any are object pixels they are labeled in the output, set to zero in the input and placed in the list. This is continued until all pixels in the list have been investigated. The algorithm then continues

following the scan path in Fig. 4.28 until it meets the next object pixel, which is then labeled 2, and starts a new grass-fire.