Theo Pavlidis' Algorithm

Idea

This algorithm is one of the more recent contour tracing algorithms and was proposed by Theo Pavlidis. He published it in his book ***Algorithms for Graphics and Image Processing*** in 1982, chapter 7 (section 5). It is not as simple as the Square Tracing algorithm or Moore-Neighbor tracing, yet it is not complicated (a property shared by most contour tracing algorithms).
We will explain this algorithm using an approach different from the one presented in the book. This approach is easier to comprehend and will give insight into the general idea behind the algorithm.

Without loss of generality, we have chosen to trace the contour in a clockwise direction in order to be consistent with all the other contour tracing algorithms discussed on this web site. On the other hand, Pavlidis chooses to do so in a counterclockwise direction. This shouldn't make any difference towards the performance of the algorithm. The only effect this will have is on the relative direction of movements you'll be making while tracing the contour.

Now let's proceed with the idea...

Given a digital pattern i.e. a group of black pixels, on a background of white pixels i.e. a grid; locate a black pixel and declare it as your "**start**" pixel. Locating a "**start**" pixel can be done in a number of ways; one of which is done by starting at the bottom left corner of the grid, scanning each column of pixels from the bottom going upwards -starting from the leftmost column and proceeding to the right- until a black pixel is encountered. Declare that pixel as the "**start**" pixel.
We will not necessarily follow the above method in locating a **start** pixel. Instead we will choose a **start** pixel satisfying the following restriction imposed on the choice of a start pixel for Pavlidis' algorithm:

Important restriction regarding the direction in which you enter the start pixel
You actually can choose ANY black boundary pixel to be your **start** pixel as long as when you're initially standing on it, your left adjacent pixel is NOT black. In other words, you should make sure that you enter the **start** pixel in a direction which ensures that the left adjacent pixel to it will be white ("left" here is taken with respect to the direction in which you enter the **start** pixel).

Now, imagine that you are a bug (ladybird) standing on the **start** pixel as in ***Figure 1*** below.
Throughout the algorithm, the pixels which interest you at any time are the 3 pixels in front of you i.e. **P1, P2** and **P3** shown in ***Figure 1***.
(We will define **P2** to be the pixel right **in front** of you , **P1** is the pixel adjacent to **P2** from the left and **P3** is the right adjacent pixel to **P2**).

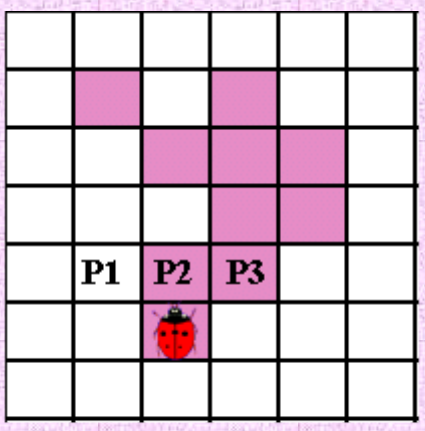

Figure 1

Like in the Square Tracing algorithm, the most important thing in Pavlidis' algorithm is your "sense of direction". The left and right turns you make are with respect to your current positioning, which depends on

the way you entered the pixel you are standing on. Therefore, it's important to keep track of your current orientation in order to make the right moves.
But no matter what position you are standing in, pixels P1, P2 and P3 will be defined as above.

With this information, we are ready to explain the algorithm...

Every time you are standing on the current boundary pixel (which is the **start** pixel at first) do the following:

First, check pixel **P1.** If **P1** is black, then declare **P1** to be your current boundary pixel and **move one step forward followed by one step to your current left** to land on P1.
(the **order** in which you make your moves is very important)
Figure 2 below demonstrates this case. The path you should follow in order to land on **P1** is drawn in blue.

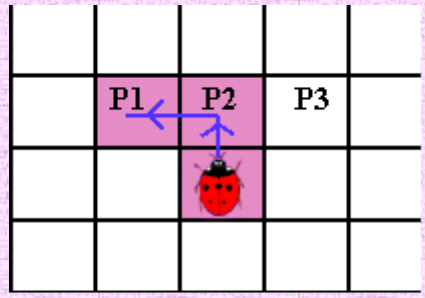

Figure 2

Only if P1 is white proceed to check P2...

If **P2** is black, then declare **P2** to be your current boundary pixel and **move one step forward** to land on **P2**.
Figure 3 below demonstrates this case. The path you should follow in order to land in **P2** is drawn in blue.

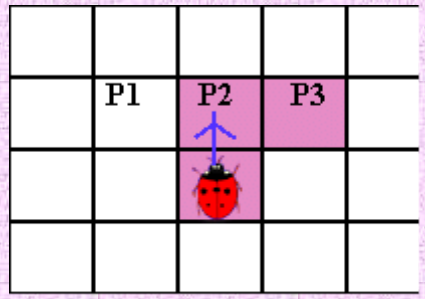

Figure 3

Only if both P1 and P2 are white proceed to check P3...

If **P3** is black, then declare **P3** to be your current boundary pixel and **move one step to your right followed by one step to your current left** as demonstrated in Figure 4 below.

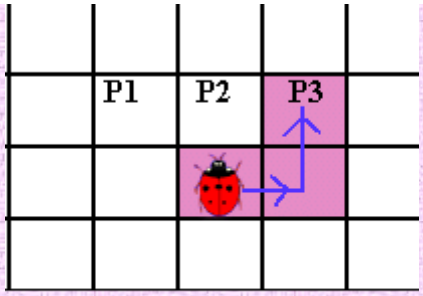

Figure 4

That's it!!
3 simple rules for 3 simple cases. As you've seen, it's important to keep track of your direction as you turn since all moves are with respect to your current orientation.
But haven't we forgotten something?!
What if all 3 pixels in front of you are white?
Then, you rotate (while standing on the current boundary pixel) 90 degrees clockwise to face a new set of 3 pixels in front of you. Afterwards you do the same check on these new pixels as you've done before.
You may still ask: what if all of **these** 3 pixels are white?!
Well, then rotate again through 90 degrees clockwise while standing on the same pixel.
You can rotate 3 times (each through 90 degrees clockwise) before checking out the whole Moore neighborhood of the pixel. If you rotate 3 times without finding any black pixels, this means that you are standing on an **isolated pixel** i.e. not connected to any other black pixel. That's why the algorithm will allow you to rotate 3 times before it terminates.

Another thing: ***When does the algorithm terminate?***
The algorithm terminates in 2 cases:
a) as mentioned above, the algorithm will allow you to rotate 3 times (each through 90 degrees clockwise) after which it will terminate and declare the pixel an isolated one, OR
b) when the current boundary pixel is your **start** pixel, the algorithm terminates "declaring" that it has traced the contour of the pattern.

Algorithm

The following is a formal description of Pavlidis' algorithm:

Input: A square tessellation**, T,** containing a connected component **P** of black cells.

Output: A sequence **B** $(b_1, b_2, ..., b_k)$ of boundary pixels i.e. the contour.

Definitions:

- Define **p** to be the current boundary pixel i.e. the pixel you are standing on.
- Define pixels **P1, P2** and **P3** as follows: ***(also see Figure 1 above)***
- **P2** is the pixel in front of you adjacent to the one you are currently standing on i.e. pixel **p**.
- **P1** is the left adjacent pixel to **P2**.
- **P3** is the right adjacent pixel to **P2**.
- Define a **"step"** in a given direction as moving a distance of one pixel in that direction.

Important Note: At all times, imagine that you are a bug moving from pixel to pixel following the given directions. "forward", "left" and "right" are with respect to your current positioning on the pixel.

Begin

- Set **B** to be empty.
- From bottom to top and left to right scan the cells of **T** until a black **start** pixel, **s,** of **P** is found ***(see Important restriction concerning direction you enter start pixel above)***
- Insert **s** in **B**.
- Set the current pixel, **p**, to be the starting pixel, **s**.
- Repeat the following

  If pixel **P1** is black
    - Insert **P1** in **B**
    - Update **p=P1**
    - Move one step forward followed by one step to your current left

  else if **P2** is black
    - Insert **P2** in **B**
    - Update **p=P2**
    - Move one step forward ***(see Figure 3 above)***

  else if **P3** is black
    - Insert **P3** in **B**
    - Update **p=P3**
    - Move one step to the right, update your position and move one step to your current left ***(see Figure 4 above)***

  else if you have already rotated through 90 degrees clockwise 3 times while on the **same pixel p**
    - terminate the program and declare **p** as an **isolated** pixel

  else
    - rotate 90 degrees clockwise while standing on the current pixel **p**

  Until **p=s** (End Repeat)

End

Demonstration

The following is an animated demonstration of how Pavlidis' algorithm proceeds to trace the contour of a given pattern. Remember that you are a bug (ladybird) walking over the pixels; notice how your orientation changes as you turn left or right. We have included all possible cases of the algorithm in order to explain it as thoroughly as possible.

Theo Pavlidis' Algorithm

Demonstration

Analysis

If you are thinking that Pavlidis' algorithm is the perfect one for extracting the contour of patterns, think again...
It's true that this algorithm is a bit more complex than say, Moore-Neighbor tracing which has no special cases to take care of, yet it fails to extract the contour of a large family of patterns having a certain kind of connectivity .

The algorithm works very well on 4-connected patterns. its problem lies in tracing some 8-connected patterns that are not 4-connected.
The following is an animated demonstration of how Pavlidis' algorithm fails to extract the contour of an 8-connected pattern (that is not 4-connected) by missing a large portion of the boundary.

Demonstration:

An 8-connected Counterexample

There are 2 simple ways of modifying the algorithm in order to improve its performance dramatically.

a) Change the stopping criterion
Instead of terminating the algorithm when it visits the start pixel for a second time, make the algorithm terminate after visiting the start pixel a third or even a fourth time.
This will improve the general performance of the algorithm.

OR

b) Go to the source of the problem; namely, the choice of the start pixel
There is an important restriction concerning the direction in which you enter the start pixel. Basically, you have to enter the start pixel such that when you're standing on it, the pixel adjacent to you from the left is white. The reason for imposing such a restriction is:
since you always consider the 3 pixels in **front** of you in a **certain order**, you'll tend to miss a boundary pixel lying directly to the left of the start pixel in certain patterns.

Not only the left adjacent pixel of the start pixel is at risk of being missed, but also the **pixel directly below that pixel** faces such a threat (as demonstrated in the counterexample above). In addition, the pixel corresponding to pixel **R** in ***Figure 5*** below will be missed in some patterns. Therefore, we suggest that a start pixel should be entered in a direction such that the pixels corresponding to pixels **L, W** and **R** shown in ***Figure 5*** below, are white.

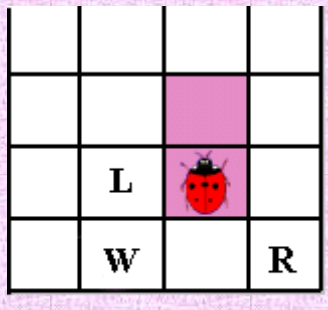

Figure 5

In this way, patterns like the one in the above demonstration will be correctly traced and the performance of Pavlidis' algorithm will greatly improve.
On the other hand, finding a start pixel which satisfies the above restriction could be tough and in many cases such a pixel won't be found. In that case, the alternative method for improving Pavlidis' algorithm should be used, namely: terminating the algorithm after visiting the start pixel for a third time.

All comments and questions are welcomed...
abeer.ghuneim@mail.mcgill.ca