Moore-Neighbor Tracing

Idea

The idea behind Moore-Neighbor tracing is simple; but before we explain it, we need to define an important concept: the ***Moore neighborhood*** of a pixel.

Moore Neighborhood

The Moore neighborhood of a pixel, **P**, is the set of 8 pixels which share a vertex or edge with that pixel. These pixels are namely pixels **P1, P2, P3, P4, P5, P6, P7 and P8** shown in ***Figure 1*** below.
The Moore neighborhood (also known as the ***8-neighbors*** or ***indirect neighbors***) is an important concept that frequently arises in the literature.

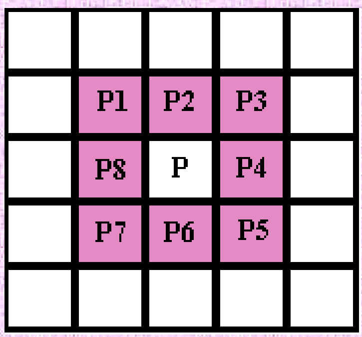


Figure 1

Now we are ready to introduce the idea behind Moore-Neighbor tracing...
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; we'll start at the bottom left corner of the grid, scan each column of pixels from the bottom going upwards - starting from the leftmost column and proceeding to the right- until we encounter a black pixel. We'll declare that pixel as our "**start**" pixel.)

Now, imagine that you are a bug (ladybird) standing on the **start** pixel as in ***Figure 2*** below. Without loss of generality, we will extract the contour by going around the pattern in a clockwise direction. (It doesn't matter which direction you choose as long as you stick with your choice throughout the algorithm).
The general idea is: every time you hit a black pixel, **P**, backtrack i.e. go back to the white pixel you were previously standing on, then, go **around** pixel **P** in a clockwise direction, visiting each pixel in its Moore neighborhood, until you hit a black pixel. The algorithm terminates when the start pixel is visited for a second time.
The black pixels you walked over will be the contour of the pattern.

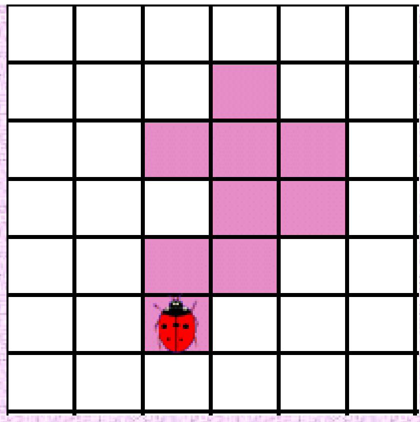


Figure 2

Algorithm

The following is a formal description of the Moore-Neighbor tracing 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.

Define **M(a)** to be the Moore neighborhood of pixel **a**.
Let **p** denote the current boundary pixel.
Let **c** denote the current pixel under consideration i.e. **c** is in **M(p)**.

Begin

- Set **B** to be empty.
- From bottom to top and left to right scan the cells of **T** until a black pixel, **s,** of **P** is found.
- Insert **s** in **B.**
- Set the current boundary point **p** to **s** i.e**. p=s**
- Backtrack i.e. move to the pixel from which **s** was entered.
- Set **c** to be the next clockwise pixel in **M(p)**.
- While **c** not equal to **s** do

  If **c** is black
    - insert **c** in **B**
    - set **p=c**
    - backtrack (move the current pixel **c** to the pixel from which **p** was entered)

  else
    - advance the current pixel **c** to the next clockwise pixel in **M(p)**

  end While

End

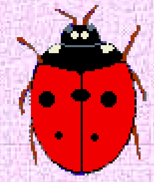
Demonstration

The following is an animated demonstration of how Moore-Neighbor tracing proceeds to trace the contour of a given pattern.

(We have decided to trace the contour in a clockwise direction).

Moore's Algorithm

Demonstration



Analysis

The main weakness of Moore-Neighbor tracing lies in the choice of the stopping criterion, in other words, when does the algorithm terminate?
In the original description of the algorithm used in Moore-Neigbor tracing, the stopping criterion is visiting the **start** pixel for a second time. Like in the case of the Square Tracing algorithm, it turns out that Moore-Neighbor tracing will fail to contour trace a large family of patterns if it were to depend on that criterion.
What follows is an animated demonstration explaining how Moore-Neighbor tracing fails to extract the contour of a pattern due to the bad choice of the stopping criterion:

Demonstration:

A reason to change the stopping criterion

As you can see, improving the stopping criterion would be a good start to improving the overall performance of Moore-Neighbor tracing. There are 2 effective alternatives to the existing stopping criterion:

a) Stop after visiting the **start** pixel ***n*** times, where n is at least 2, OR
b) Stop after entering the **start** pixel a second time **in the same manner you entered it initially.** This criterion was proposed by Jacob Eliosoff and we will therefore call it ***Jacob's stopping criterion.***

Using Jacob's stopping criterion will greatly improve the performance of Moore-Neighbor tracing making it the best algorithm for extracting the contour of any pattern no matter what its connectivity.
The reason for this is largely due to the fact that the algorithm checks the whole Moore neighborhood of a boundary pixel in order to find the next boundary pixel. Unlike the Square Tracing algorithm, which makes either left or right turns and misses "diagonal" pixels; Moore-Neighbor tracing will always be able to extract the outer boundary of any connected component. The reason for that is: for any ***8-connected*** (or simply ***connected***) pattern, the **next** boundary pixel lies within the Moore neighborhood of the current boundary pixel. Since Moore-Neighbor tracing proceeds to check every pixel in the Moore neighborhood of the current boundary pixel, it is bound to detect the next boundary pixel.

When Moore-Neighbor tracing visits the start pixel for a second time in the same way it did the first time around, this means that it has traced the **complete outer contour** of the pattern and if not terminated, it will trace the same contour again. This result has yet to be proved...

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