621-630.

[RM89] Rosenfeld, A. and Melter, R. A., “Digital geometry,” *The Mathematical Intelligencer,”* vol. 11, No. 3, 1989, pp. 69-72.

[SK76] Sklansky, J. and Kibler, D. F., “A theory of nonuniformly digitized binary pictures,” *IEEE Transactions Systems, Man & Cybernetics,* vol. SMC-6, Sept.,1976, pp.637-647.

```
            which $p=b$ was entered);
            else advance the current pixel $p$ to the next clockwise pixel in M($b$).
        end while
    End
```

While the Moore-neighborhood contour tracing algorithm will work correctly on the example of Fig. 5 whether the pixel marked S is black or not, it is not guaranteed to work correctly on all 8-connected patterns. It is left as an exercise for the reader to find suitable counter-examples. However, performance can again be improved by waiting to encounter the starting pixel for a third time. These algorithms are useful nevertheless, particularly in most practical situations where the “thickness” of the patterns is much greater than one pixel. In such situations counter-examples are much harder to find. A more serious drawback is that if the pattern **P** has holes the contour tracing algorithms described above only deliver the outer boundary of **P** and not the boundaries of the holes. Such would be the case for example for patterns such as the letters O, P, B, Q, D and R.

In order to make precise statements about the situations in which these algorithms are guaranteed to compute the entire boundary of connected sets care must be taken in defining the connectivity of the background and the “holes” [RM89], [Ro79]. A special case of interest is the that when not only the pattern is 4-connected but the background (set of white pixels) is also 4-connected. It is left as an exercise for the reader to determine if the Moore-neighborhood contour tracing algorithm will work correctly on such images. Finally we mention that the contour tracing algorithms described above are inherently sequential and a mistake made early in the game stays or amplifies the problems later on. One can attribute the weaknesses of these algorithms to their sequential nature although their weaknesses may be overcome by additional sequential procedures. However, we shall see in later chapters that with parallel algorithms we can avoid such problems in a simpler and more elegant manner.

4. References

[De72] Deutch, E. S., “Thinning algorithms on rectangular, hexagonal, and triangular arrays,” *Communications of the ACM*, vol. 15, No. 9, Sept. 1972, pp. 827-837.

[De70] Deutch, E. S., “On parallel operations on hexagonal arrays,” *IEEE Trans. Computers,* vol. C-19, Oct. 1970, pp. 982-983.

[FT83] Fejes Toth, “New results in the theory of packing and covering,” in *Convexity and Its Applications*, P. M. Gruber and J. M. Wills, eds., Birkhauser, Basel, 1983, pp. 318-359.

[GS87] Grunbaum, B. and Shephard, G. C., *Tilings and Patterns*, W. H. Freeman and Co., 1987.

[Po40] Polachek, H., “The structure of the honeycomb,” *Scripta Mathematica*, vol. 7, 1940, pp. 87-98.

[Ro79] Rosenfeld, A., “Digital topology,” *American Mathematical Monthly*, vol. 86, 1979, pp.

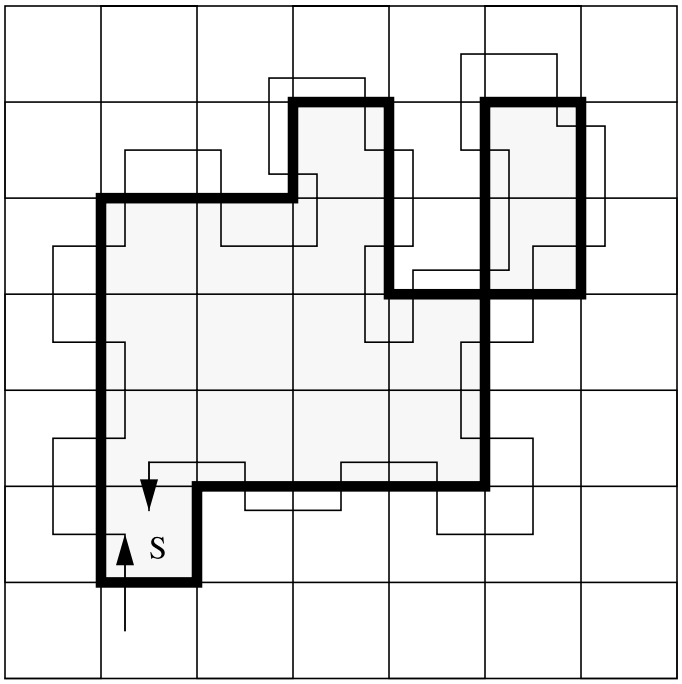

Fig. 5   Square tracing an 8-connected pattern.

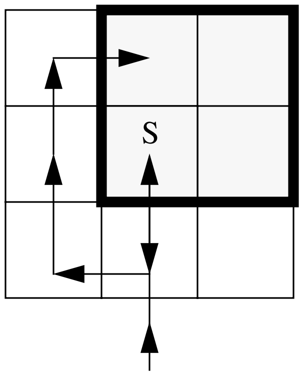

Fig. 6 Searching for the next black contour pixel in Moore-neighborhood contour tracing.

rent pixel *p* is black, the Moore neighborhood of *p* is examined in a clockwise manner starting with the pixel from which *p* was entered and advancing pixel by pixel until a new black pixel in **P** is encountered as illustrated in Fig. 6. The algorithm is described more precisely below.

ALGORITHM *Moore-Neighborhood-Tracing*

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.

Begin

{*Initialization*: find a pixel in **P**, initialize **B**, define M(*p*) to be the Moore neighborhood of the current pixel *p*}

1. Set **B** to be empty.

2. From bottom to top and left to right scan the cells of **T** until a pixel *s* of **P** is found (until the current pixel *p=s* is a black pixel of **P**), insert *s* in **B**. Let *b* denote the current boundary point, i.e., *b*=*s*.

3. *Backtrack* (move the current pixel to the pixel from which *s* was entered) and advance the current pixel *p* being examined to the next clockwise pixel in M(*b*).

while *p* is not equal *to s* ***do***

if *p* is black, insert *p* in **B**, set *b*=*p* and *backtrack* (move the current pixel to the pixel from

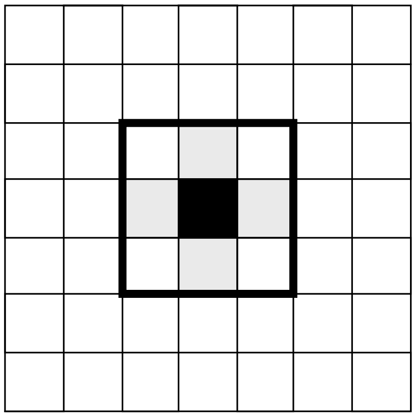

Fig. 4 A pixel in a square tessellation and its *eight*-neighbors.

```
        Update $p$.
        else turn right (visit the right adjacent pixel of $p$); Update $p$.
    end while
End
```

Note: The notion of left *and* right *in the above algorithm is not to be interpreted with respect to the page or the reader but rather with respect to the direction of entering the "current" pixel during the execution of the scan.*

Fig. 5 shows an example of how the square tracing algorithm starting at the pixel marked S produces a sequence of boundary pixels in the case **P** is 8-connected. It would be nice to be able to prove that Algorithm *Square-Tracing* always works correctly, i.e., that for all connected patterns it extracts the entire set of boundary pixels as defined above. Unfortunately this is not the case. Note that in Fig. 5 the 8-connected pattern consists of two separate 4-connected components, i.e., two 4-connected components that are not 4-connected to each other. If the black pixel marked S were marked white instead then the algorithm would only trace one of the 4-connected components of **P**. Since there are simple procedures for converting 8-connected patterns to 4-connected ones without appreciably changing their shape it would be nice to be able to prove that the square tracing algorithm always works correctly at least for 4-connected patterns. It is left as an exercise for the reader to provide an example of a 4-connected pattern that is *not* completely contour traced either by the square tracing algorithm. However, there are ways of modifying the algorithm to obtain an increase in the chance that the algorithm will terminate correctly in practice. For example one may try to make the algorithm continue executing when S is encountered for the second time to determine if an un-explored portion of **P** is discovered and terminate only after S is encountered again for the third time.

More popular modifications of Algorithm *Square-Tracing* to handle 8-connected patterns discard the *left-turn-right-turn* rule altogether. One such procedure is known as *Moore-neighborhood tracing*. The eight neighbors of a pixel shown in Fig. 4 are also referred to in the literature as the Moore neighborhood of a pixel. In the Moore neighborhood tracing algorithm, when the cur-

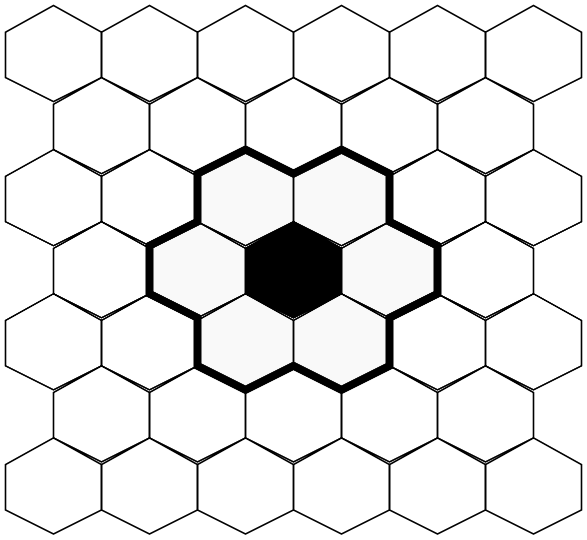

Fig. 3 A pixel in a hexagonal tessellation and its six neighbors.

useful definitions of “border” points which are intimately related to the boundary. We say that a black pixel of **P** is a 4-*border* point if at least one of its 4-neighbors is white. We say that a black pixel of **P** is an 8-*border* point if at least one of its 8-neighbors is white.

One particularly simple procedure for contour tracing connected components of binary pictures is called *square-tracing* and is described below. The idea is to imagine you are a bug crawling along the black and white boundary pixels and making 90 degree turns to the left or right with respect to the direction in which you are pointing at any one given moment depending on what the color of the pixel you are standing is.

```
ALGORITHM Square-Tracing
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.
Begin
    {find a pixel s in P and initialize B}
    1. Set B to be empty.
    2. From bottom to top and left to right scan the cells of T until a pixel s of P is found (until
       the current pixel p being examined is the black starting pixel s of P). Insert s in B. Turn left
       (visit the left adjacent pixel of p). Update p.
    while p not equal to s do
        if the current pixel p is black insert p in B and turn left (visit the left adjacent pixel of p);
```

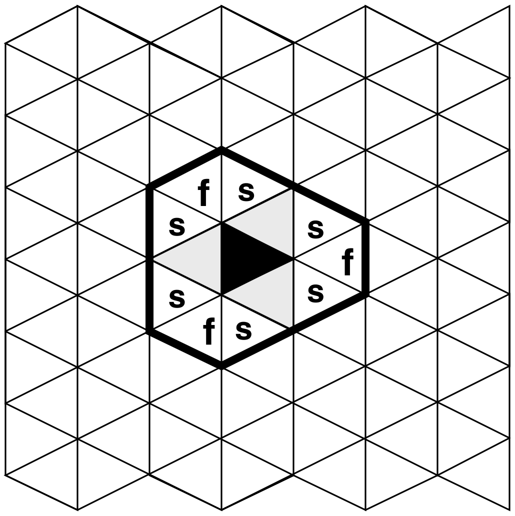

Fig. 2 A pixel in a triangular tessellation and its three types of neighbors.

each type of tessellation **T** we can define an object or pattern in **T** as a *connected component* of black pixels. Thus one of the first stages in a pattern recognition system is to actually find all the objects in the digitized field of view **T**. Consider for example an hexagonal tessellation **T** and let **P** denote the set of pixels in **T** that are black. We say that two pixels in **T** are neighbors if they share an edge in common. We say that **P** is *connected* (or a connected component) if for every pair of pixels p_i, p_j in **P** there exists a sequence of pixels $p_i, ..., p_j$ such that (1) it is contained in **P** and (2) every pair of pixels adjacent in the sequence are neighbors of each other. We can obtain analogous definitions of connectivity for square arrays. However, because we now have two different types of neighbors we obtain two different types of connectivity. Accordingly let **P** be a set of black pixels in a square tessellation **T**. We say that **P** is *4-connected* (or a 4-connected component) if for every pair of pixels p_i, p_j in **P** there exists a sequence of pixels $p_i, ..., p_j$ such that (1) it is contained in **P** and (2) every pair of pixels adjacent in the sequence are *4-neighbors* of each other. Similarly, We say that **P** is *8-connected* (or an 8-connected component) if for every pair of pixels p_i, p_j in **P** there exists a sequence of pixels $p_i, ..., p_j$ such that (1) it is contained in **P** and (2) every pair of pixels adjacent in the sequence are *8-neighbors* of each other. Obviously a 4-connected pattern is an 8-connected pattern but not vice-versa.

3. Contour Tracing

One method of finding and analyzing the connected components of **T** is to scan **T** in some manner until a border pixel of a component **P** is found and subsequently to trace the boundary of **P** until the entire boundary is obtained. The boundary thus obtained can be stored as a doubly-linked circular list of pixels or as a polygon of the boundary of the pixels in question. Such procedures are generally termed *contour tracing*. There is frequently no standard formal definition given of the boundary of a connected pattern. In practice the boundary is what seems appropriate as the boundary to humans and algorithms are judged in accordance to how well they agree with human perception. However in order to make more precise statements about our algorithms there are two

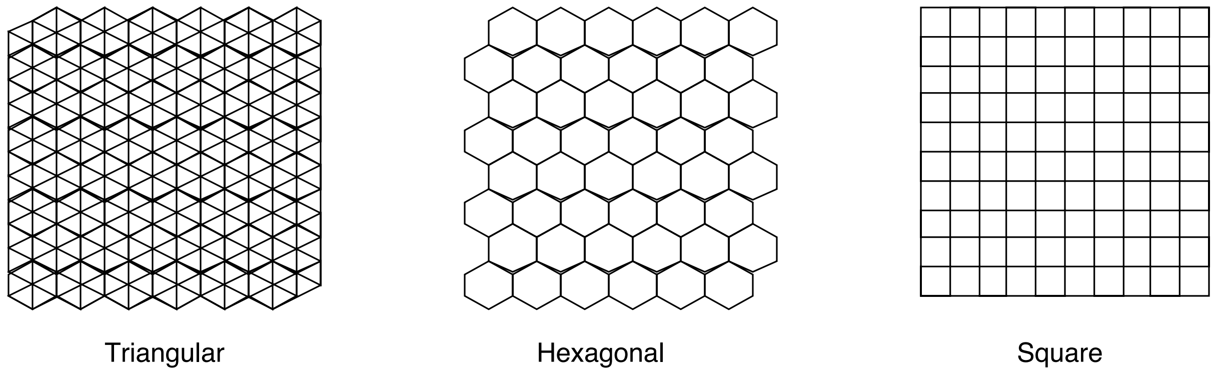

Fig. 1 Illustrating the three types of regular tessellations.

view the resulting grid as a graph, all the nodes in this graph have degree six whereas the degrees are four and three in square and hexagonal tessellations, respectively. More importantly however are the neighborhood properties of a single pixel. Consider the pixel marked black in Fig. 2 and call it *p*. *T*he *neighborhood* of *p* is defined as the set of pixels that intersect *p*. We see that *p* has 12 neighbors and these can be classified into three groups: those that have an edge in common with *p* (shown shaded in Fig. 2), and those that only have a point in common. Of the latter we can distinguish between those that are “facing” a vertex of *p* (marked *f)* and those that are “skewed” with a vertex of *p* (marked *s*). The complicated nature of this situation and its complicating effect on the resulting image processing algorithms designed to work on such tessellations makes this type of tessellation un-attractive although much research was done on all three types of tessellations during the early days of pattern recognition [De72].

1.2 Hexagonal Tessellations

From the point of view of the neighborhood properties of a single pixel, the hexagonal tessellation is the most elegant as can be seen from Fig. 3 where a black pixel *p* is shown surrounded by its six neighbors. All the six neighbors belong to the same equivalence class in the sense that they share one edge with *p*. Besides their use in image processing hexagonal arrays are notoriously present in nature singularly evident in the honeycomb of the bees. As a result many fascinating mathematical properties of these tessellations and their 3-dimensional relatives have been the subject of investigation for two thousand years [Po40].

1.3 Square Tessellations

Square tessellations are the most widely used in practice. Fig. 4 illustrates a black pixel *p* and its neighborhood. We see that *p* has 8 neighbors and these can be classified into two groups: those that have an edge in common with *p* (shown shaded in Fig. 4), and those that only have a point in common. The shaded group are called the *4-neighbors* of *p* whereas the totality of neighbors is called the *8-neighborhood* of *p*. Thus a 4-*neighbor* is an *8-neighbor* but not vice-versa.

2. Connectivity

One of the most important concepts in image processing is the notion of *connectivity*, In

CHAPTER 2

GRIDS, CONNECTIVITY AND CONTOUR TRACING

Godfried Toussaint

ABSTRACT

This chapter introduces the three basic types of grids used to represent a digital pattern: *triangular, hexagonal* and *square*. The notions of *4-connectedness* and *8-connectedness* are defined and their impact on tracing the boundary of a digital pattern is analyzed.

1. Tessellations

A *tessellation* of the plane is essentially a partitioning of the plane into regions. Typical examples of bounded regions that form tessellations are a quilted bedspread and a tiled bathroom floor. There exists considerable variation in the terminology of tessellations. In the mathematical literature the words *tiling* and *mosaic* are often used and one also encounters the words *paving* and *parqueting* [GS87]. More formally we say that a *planar tessellation* **T** is a countable family of closed sets **T** = $\{T_1, T_2, ...\}$ which cover the plane without *gaps* or *overlaps*. As in a well constructed bathroom floor we would not want some tiles to be missing (gaps) or to be lying on top of others (overlaps). More precisely, the union of all the sets T_1, T_2,... (also referred to as *cells*) constitutes the entire plane and the intersection of the interiors of every pair of cells is empty. It is useful to define two related notions here. A *packing* is a family of sets which has no overlapping. A *covering* is a family of sets that has no gaps. Clearly, a tessellation is both a packing and a covering. Furthermore, when the object of interest is not the entire plane but a subset of it as for example a simple *polygon* then the term tessellation is replaced by *partition*. Coverings and packings of the plane are of interest to mathematicians and a good survey on recent results in this area can be found in [FT83]. Partitions and other coverings of polygons, on the other hand, are very useful in computer science and will be of interest when we study feature extraction and the measurement of shape.

Here we are interested in tessellations as models for the representation of digital images and therefore it is convenient to let all the cells (*pixels* in this context) be the same shape and size. Such tessellations are termed *monohedral*. This means that every pixel in **T** is congruent either directly or reflectively to one fixed shape *T* called the *prototile* or *protocell*. This is of course not a necessary condition for obtaining effective computer vision systems. The tessellations induced by the distribution of rods and cones in the retinas of our eyes are certainly not monohedral. Some research has also been done on image processing algorithms on non-monohedral tessellations [SK76]. However, our primary concern is with a restricted version of monohedral tessellations. If we also insist that each cell be a *regular* polygon, i.e., a polygon with all its sides and angles equal, then we obtain the so-called *regular* tessellations. It turns out that there exist only three different types of such tessellations: triangular, hexagonal and square as illustrated in Fig. 1.

1.1 Triangular Tessellations

Triangular tessellations are the most complicated of the three in some sense. For one, if we