Chapter 2

REAL-TIME SEQUENTIAL PROCESSING OF BINARY PICTURES

Our goal in this chapter is to present the fundamental constraints we shall have to comply with in our subsequent analyses, and to introduce some basic notions and general notation. Our constraints are imposed by material limitations in both the hardware and software realizations of most general-purpose picture processing systems.

§2.1 The Fundamental Constraints

We shall deal with binary pictures, *i.e.*, binary pixel arrays corresponding to an image of a given size, whose local gray levels have been binary quantized with a given spatial resolution. For most images of interest the conversion to binary represention gives rise to a huge amount of binary information. For instance, with a resolution of 8 pixels per millimeter, an ordinary A4 sheet of paper gets converted into around 4 million pixels, hence 4 million bits. This simple observation calls for the following comments. In the first place, the significant information contained in an ordinary document, or for that matter, in ordinary black and white pictures, can usually be compressed into a much smaller number of bits. In the second place, if, for our present purposes, we were to store the complete pixel array in memory, we would not only waste much storage space, but also exclude for ever the possibility of implementing a fast, light weight and effective picture processing system. Therefore, it is desirable to keep memory requirements as low as possible.

We shall assume that the acquisition device "reads" the picture sequentially in a single raster scan, as is the case for facsimile readers, TV cameras, laser scanners, etc.

These elementary considerations suggest an analogy with human reading. The reader scans successively all the lines of a document from left to right, and interprets every character, word, or sentence as soon as it has been read. We shall also require that some interpretation of the contents—say, the features—be completed at the times these features have been completely scanned.

These observations induce the fundamental restrictions we shall have to comply with:

- ▸ The image is read sequentially in a single raster scan.
- ▸ Only a bounded part of the picture must be kept in memory.
- ▸ The feature of the image must be detected (and output) as soon as they have been completely scanned.

Let us now examine these constraints, together with their practical consequences, in some detail.

2.1.1 Sequential processing

We assume that the picture is spatially quantized by a quadruled grid G with M rows and N columns. We write $p(i,j)$ for the pixel at the intersection of row i and column j. We use $x(i,j)$ to denote the binary value corresponding to its colour (we use 1 for black, 0 for white) in the picture. For $t = iN + j$ we can write

$$\begin{aligned} q(t) &= p(i,j), \\ y(t) &= x(i,j), \end{aligned} \tag{2.1}$$

where

$$t = 0, \ldots, MN - 1; \qquad i = 0, \ldots, M - 1; \qquad j = 0, \ldots, N - 1.$$

Then, saying that the picture is acquired by a single raster scan means that the pixels $q(t)$ are "read" in increasing order, and the input signal is the flow of the values $y(t)$, $t = 0, \ldots, MN - 1$. Let us note that this already excludes from consideration the combination of direct and reverse raster scans, or the use of such special purpose acquisition devices as flying spot scanners which are sometimes used for line or contour following purposes [1],[2].

2.1.2 Limited memory

With a sequential acquisition system, it is quite convenient to store in a working memory a *bounded* sequence of pixel values $y(t)$, for example, a small number of rows, or a window centered around a pixel. We shall adopt the common practice of storing *three* rows of pixel values, and analyzing the properties of a pixel by looking at the 3×3 window centered around it.

The major advantage of this approach is that it is easily amenable to both hardware and software implementations. Figure 2.1 displays an actual hardware implementation. Squares represent delay flip-flops, and the whole construction is a shift register of length $2N + 3$. Note that when the 3×3 window in Figure 2.1 is centered on, say, one vertical border of the image, *i.e.*, when $j = 0$ or $N - 1$, some of the pixels in the window belong to the opposite border of the picture. We will not dwell on the subject right now, but we might anticipate that such situations will deserve special consideration.

Let us now turn our attention to the software implementation of Figure 2.1. It differs sligthly from its hardware equivalent in that the basic elements of the input file are rows, *i.e.*, arrays of length N of binary values. We take as window three successive rows, and the local 3×3 windows are obtained by scanning these rows from left to right. Our program handles only the rows numbered $1, \ldots, N - 2$, and adds automatically two blank rows numbered 0 and $N - 1$.

The windowing procedure, centered on pixels from row i is called *window(i)*. The code can be found in Appendix B, Section B.3. The procedure *window* constructs three arrays *frow*, *srow*, and *trow* (whose names are mnemonics for first, second and third row) corresponding to the rows indexed by $i - 1$, i, and $i + 1$. Note that, for $i > 1$, *frow* and *srow* can be obtained by a mere shift of *srow*

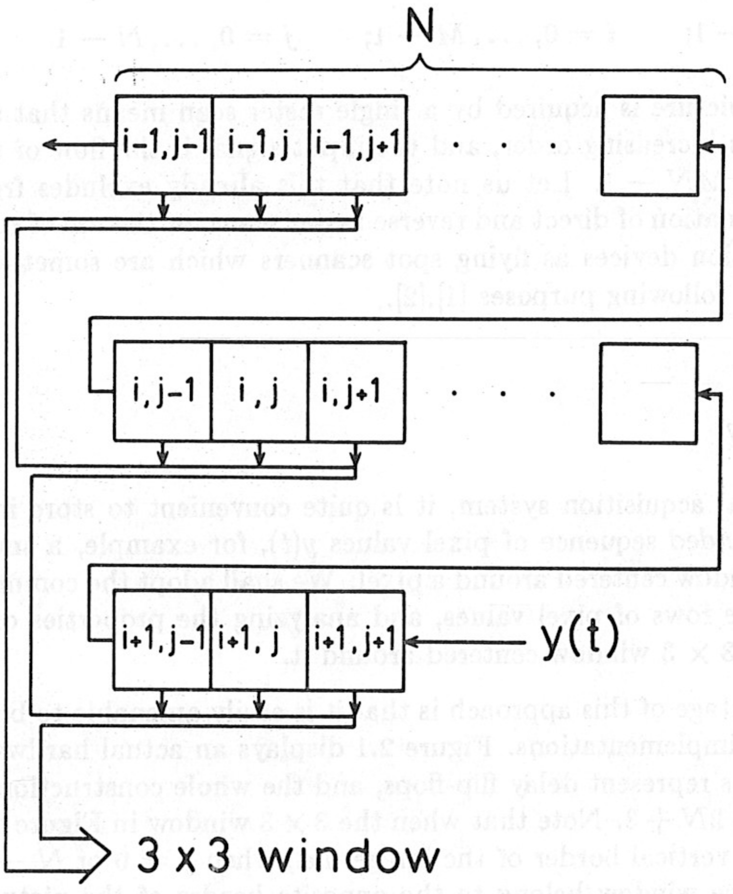

Figure 2.1. Hardware implementation of 3 × 3 window

and *trow* respectively. This is the approach which is followed in the procedure *transitiontothenextrow* (see Section B.2. in Appendix B).

Now, as the rows of the picture are acquired, the algorithm builds different structures corresponding to the features of the image, and outputs them in real-time. Features can be connected components as well as surrounding relations between them. Additional memory space will be necessary to store the description of these features at the time they are being discovered. The memory space required for these purposes will be discussed in due time.

2.1.3 Real-time processing

Let us presently examine the significance of our real-time requirement.

As was just said, the features of interest are the connected components of the image, *i.e.*, the set of black pixels, their contents, and, possibly, the surrounding relations between them. The description of these features will be stored in appropriate data structures. Our goal will be to be able to output these data structures as soon as the corresponding features have been completely disclosed. The words "as soon" should not be taken to mean "in a fixed time" for it is clear that the time required to transfer a data structure in, say, some output buffer, is proportional to the size of that data structure. Feature detection and some analysis will be carried out at the time that the rows on which they appear are processed. At the moment when a feature has been completely disclosed, for instance, when the scan is on the last pixel of a connected component, an output procedure is activated which requires a time linear in the number of bits necessary to describe the component in question. In particular, the time requirement for the whole algorithm must be proportional to the size of the picture. This corresponds to what is usually meant by the words "real-time processing".

In a number of applications, the real-time requirement is a crucial one. When it is combined with dynamic memory management, it permits to avoid congestion of the main memory. It is also frequently the key to an harmonious cooperation of serial processes.

§2.2 Some Definitions and Notations

The description of the algorithm needs a minimum of mathematical formalism concerning pictures on a square grid. As already mentioned, G designates a quadruled grid with M rows and N columns. The elements of G are the *pixels* (or *pels*). The image on G is a partition of the pels of G into two tones, black and white. We write F for the set of black pels and we call it the ***figure***. We write B for the set of white pels and we call it the *background*.

For reasons already alluded to, we shall constantly make the *Frame Assumption* which states that the pixels of rows 0 and $M-1$ and of columns 0 and $N-1$ are all in B, [3]. In this way, we avoid the problems occurring when the window is centered on any of these pixels. In actual fact, our program automatically turn these two rows and columns into white thereby preventing accidental violations of the frame assumption due to noise or any other cause.

It is well-known that two adjacency relations can be defined on a square grid, namely, the 4-adjacency and the 8-adjacency, [4]. Let us recall, however, that one must always take opposite adjacencies on the Figure F and the background B. Our program leaves the choice of the adjacency on the figure as an option to the user. We will write k for the number 4 or 8 representing the adjacency chosen for F, and k' for the adjacency which applies then to B. In fact, $k' = 12 - k$.

Further definitions and notations will be introduced as we proceed through the following chapters.

REFERENCES

[1] N.G. Altman, "Automatic digitizing of engineering drawings," *Proc. Electro 78, Electronic Show and Convention*, Session 22/3, pp. 1–3, May 1978.

[2] E.C. Greanias, P.F. Meagher, R.J. Norman, and P. Essinger, "The recognition of handwritten numerals by contour analysis," *IBM Journal,* vol. 7, pp. 14–21, Jan. 1963.

[3] C. Ronse, *Digital Processing of Binary Images on a Square Grid,* Philips Res. Rept. [R.454], June 1981.

[4] A. Rosenfeld, "Adjacency in digital pictures", *Information and Control,* vol. 26, pp. 24–33, 1974.