An experimental program for architectural design

By W. M. Newman*

The solution by computer of problems in architectural design is made much more feasible by the use of a display and light pen for input and output. In the program described here, the light pen is used to create designs made up out of industrialized units, and the computer is then capable of performing calculations on the design and displaying the results on the screen.

This program was written to show how a computer-driven display could be used as an input/output device when designing with modular building units. It was written for the PDP-7 computer at the Cambridge University Mathematical Laboratory, which is equipped with a Type 340 Display and Light Pen (see **Fig. 1**). The user of the program can add wall units, windows, doors, etc., to his design by typing the appropriate request on a teleprinter. A picture of the unit then appears on the screen wherever the light pen is pointing, and can be moved about the screen until it is in its correct position. The user can then insert the unit permanently in the design, and the program automatically aligns the unit with the modular grid. By means of the light pen, units can be erased, duplicated or moved to new positions. The designs are stored in the computer in a form which enables them to be compactly recorded on punched tape, and which allows the program to perform numerical processing on the design, including calculating areas.

In the first part of this paper the operating techniques and general capabilities of the program are outlined, and the second part describes some of the programming methods adopted.

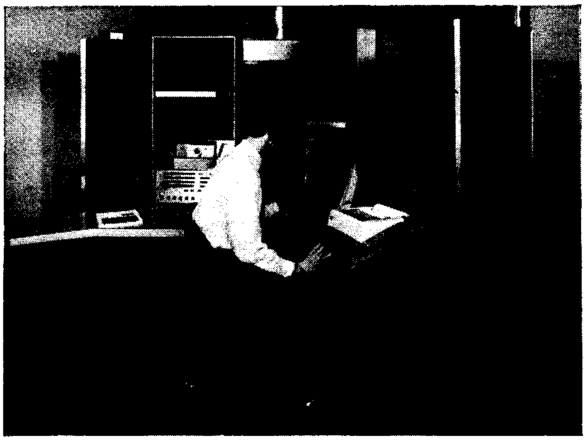

Fig. 1.—The PDP-7 computer

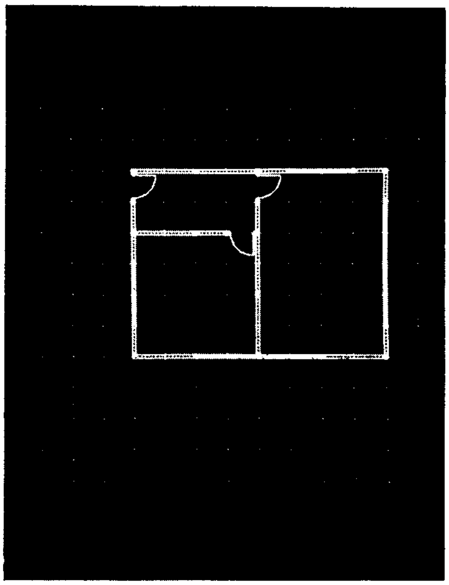

Fig. 2.—A simple building design on the screen

1. General description

In order to demonstrate the techniques of the program, a hypothetical modular building system was devised; it consists of about twenty different units, including walls, windows, floor slabs, etc., which can be assembled together on a 3-foot square grid. A typical design is shown in plan view in **Fig. 2,** illustrating two basic rules which govern the system: all walls, windows and doors must lie along grid lines, and each such unit must be provided with a supporting column at each end. The system was inspired by the IBIS Industrialised Building System (Williams, 1965), which it closely resembles.

The 10-inch square display screen of the 340 is treated as a "window" on to a very large drawing board. Designs can cover an area measuring 768 ft × 768 ft at full scale, or about 15 ft × 15 ft at the scale used (roughly $\frac{1}{4}$ in. to 1 ft). The screen itself shows only a small portion of this area, measuring 12 grid squares

* *Computer Science Section, Imperial College, London.*

in each direction. Designs can cover four floors, any one of which can be displayed as a horizontal section at either floor level or wall-unit level.

The user operates the program with the aid of the light pen and the teleprinter keyboard. All commands are given via the keyboard, and the light pen is used only to indicate positions on the screen. The program starts in an inactive state called *Base Mode*, with the display operating but the light pen disabled. The light pen can be enabled, preparatory to using it to delete or duplicate an item, by pressing the space bar on the teleprinter, and a pen-tracking routine is then called into operation. The pen is finally disabled by giving a command to return to Base Mode, generally by pressing the space bar again. Thus alternate presses on this key will enable and disable the light pen.

Apart from those controlling the light pen, the four most important commands at the user's disposal are *duplicate*, *shift*, *erase* and *rotate*. Each operation involving these commands is terminated by pressing the space bar to regain Base Mode and cause the new item and its position to be stored. It is not necessary to position the item exactly before inserting it: when the space bar is pressed the item "jumps" to the nearest modular position. This permits a tolerance of over $\frac{1}{4}$ inch in specifying item positions. Items can also be inserted by typing an *item code* preceded by a colon. Each of the units has such a code, consisting of two or three alphanumeric characters. On typing the colon, the routine for finding and tracking the light pen is called, and as soon as the item code has been correctly typed, a picture of the item appears on the end of the light pen.

The display "window" can be moved one grid space in any of four directions by pressing the appropriate key; +, −, < and > were chosen as the most suitable symbols to represent movement up, down, left and right, respectively. To change from one floor to another involves typing a string of three characters—PC2 for example. The first letter denotes by P or F whether a normal plan view or a horizontal section at floor level is required, and the final number gives the floor level. The use of C or F as the middle letter allows the user to choose between clearing the screen before displaying the new level or keeping a picture of the old level as a guide. The latter feature is essential on starting to design a new floor, as the user cannot otherwise be sure he is placing the units correctly in relation to the previous level. To avoid confusion, the previous level is displayed at reduced brightness.

Complete designs can be punched out in a compact form on paper tape, and the same designs can be read in again to any part of the "drawing board"; these two operations are initiated by typing PUNCH and READ, respectively. A large design in which a small group of units occurs repeatedly can be rapidly assembled by reading the group off a previously prepared tape into the required positions.

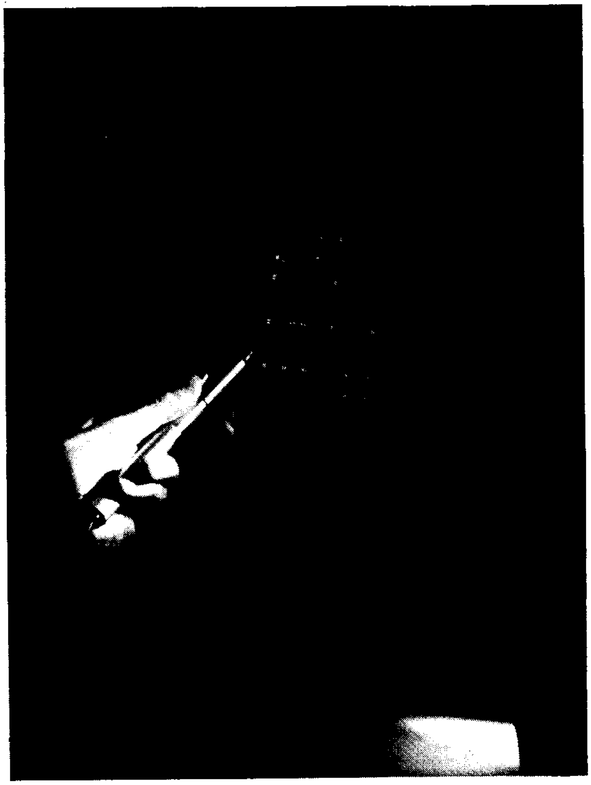

Fig. 3.—The area of the indicated room is calculated and displayed on the screen

It was not originally considered that complicated numerical routines lay within the scope of the project. However, to demonstrate that the system would permit such extensions, two numerical routines were included. The first, which is far from being complex, types out an inventory of the units currently contained in the design; the second, called by typing "A", calculates the area of the enclosed space within which the light pen is pointing. The value appears on the screen near the end of the pen, as shown in **Fig. 3,** and the user can request a typed record of this value.

2. Details of the program

The 340 Display reads its data directly from the core storage of the PDP-7, by autonomous transfer using hesitations or "cycle stealing". The program must give the display a starting address when initiating the transfer, and the 340 then proceeds to read words sequentially from that address and to interpret them as display data. This will continue without interfering with the current program until the 340 reaches a "stop" word, when the display halts and a peripheral flag is set. By means of the interrupt facility the program can then restart the display.

The 340 can operate in a variety of different modes, and interprets its data as points, vectors, etc., according

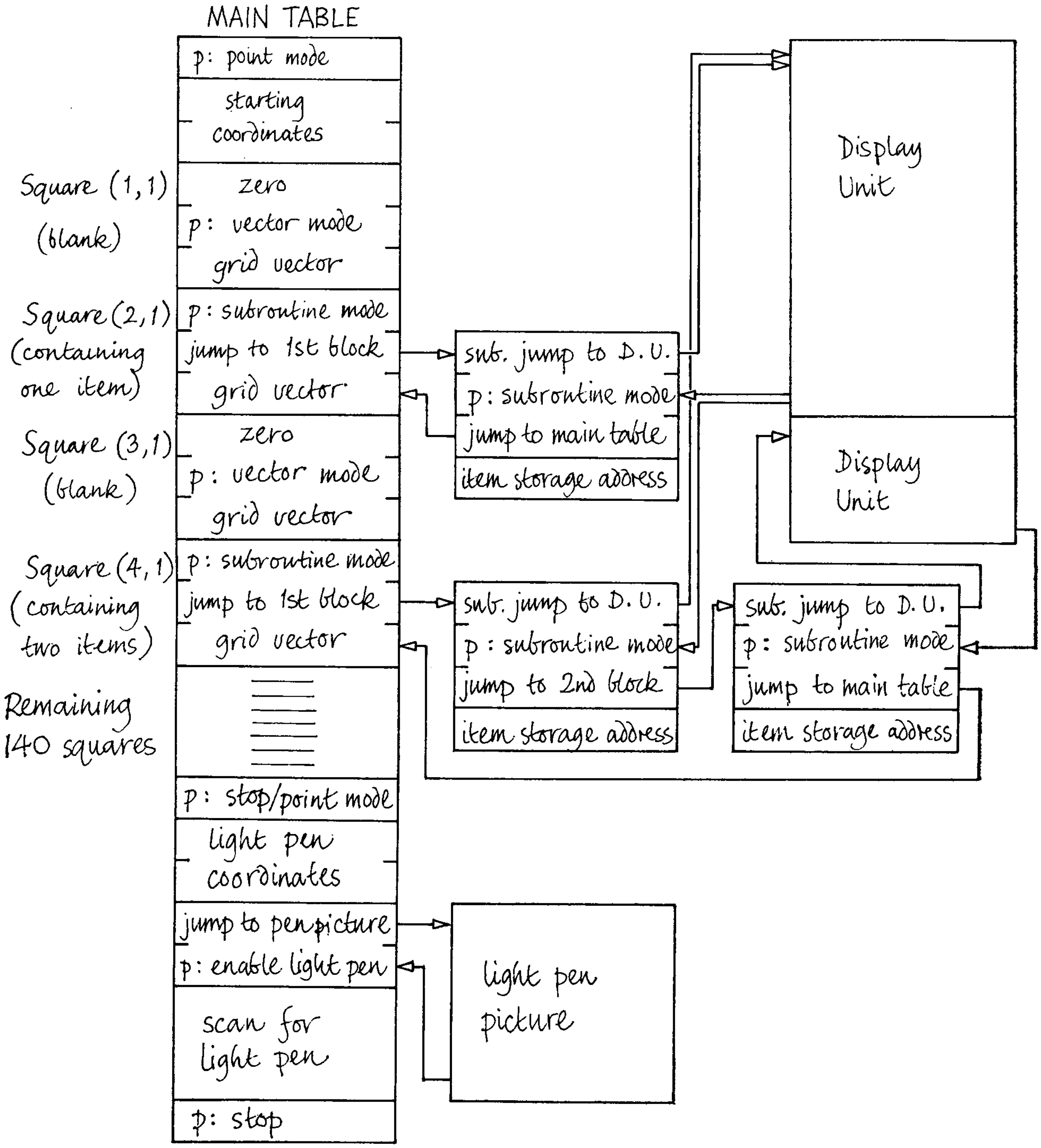

Fig. 4.—Display table organization. "p:" denotes a parameter word, setting a fresh mode, light pen status, etc., as indicated. In this example, the first items of both display lists are identical, and therefore share the same display unit

to the current mode. An additional *subroutine mode* facility permits jumps to be inserted in the display data "table"; these can be either true subroutine jumps, each with a return jump stored elsewhere in core, or simple non-return jumps.

The light pen (Sutherland, 1963; Stotz, 1963) is a simple photo-electric device connected to a flag which can be tested by the program. Tracking is performed in this particular program by means of a small spiral raster; as soon as a new pen position is picked up, the centre of the raster is moved to this spot.

The simplest possible table of display data contains all the necessary points and vectors in a single block of store, but requires frequent reorganization to close gaps caused by deletions. Such re-arrangements can only be carried out between stopping and re-starting the display, and may cause unwelcome delays or flicker. Other reasons for discarding this type of table were the difficulty of tracing individual entries in the table, and the necessity of dealing with large units whose pictures might extend off the edge of the screen. Such edge violations cause an interrupt to the PDP-7, and are best avoided in the interest of keeping a fast display.

A self-organizing table was therefore used, employing multiple list-structures for storing the subroutine jumps to the unit pictures. Each list is associated with one square on the display, and is constructed from a variable number of four-word blocks (Comfort, 1964). Each picture subroutine is made up entirely of visible and invisible vectors; the starting point of the picture coincides with the finishing point, and the whole picture is contained within one grid square. A number of these subroutines are necessary to portray one of the larger units, and must be inserted in the lists of a set of adjacent squares.

Fig. 4 shows diagramatically the arrangement of the display table. Each of the 144 grid squares within the display window has a three-word entry in the main table, and if a square contains any items, a list of jumps to the appropriate *display units* (i.e. picture subroutines) is attached to this point in the table. Each jump is placed at the head of a four-word *display block*, and is followed by a "pointer" or jump to the next display block. New display blocks are added at the top of the appropriate lists, and deletions can be made by moving the pointer from one block to the one above.

Changes within the display table therefore consist almost entirely of alterations to pointers in these lists and to the parameter word which precedes each pointer. This parameter word is necessary for selecting subroutine mode, and is also used to set brightness and scale and to indicate in which segment of the grid square the next display unit in the list lies. Normally, therefore, a pointer is moved together with its parameter word, and these two words are placed in a *display entry block* to await insertion when the display stops. A large number of entries may be made to the display, and the entry blocks involved are chained together to form an Entry Block List. Like display blocks, entry blocks are of four words each, and therefore there is a continuous need for four-word blocks, particularly as blocks of the same size are used for permanently storing the position of each item. These blocks are all supplied from the same Available Block List, and as soon as a block is no longer needed it is returned to this list. In this way the program dispenses with the need for any periodic garbage-collecting routines.

It has been possible to devote almost the whole of the upper half of the 8192-word store to the Available Block List, which therefore contains 1000 blocks when the program is started. Items are stored in nine *Floor Storage Lists*, representing the nine permissible levels in a four-storey building. When an item is added to the design, its type and position are stored in a four-word block fetched from the Available Block List, and this block is added at the head of the appropriate Floor Storage List. Thus items are stored in these lists in their order of insertion; a more associative method of storage would be advisable if more processing were to be done on the design.

Moving the display window

Since the items are stored in lists, rather than in core locations corresponding to their actual positions, a complete list search is necessary to discover which items, if any, occupy a given grid square. The routine for setting up a fresh display, when the user requests a new level, is organized into two subroutines, and makes extensive use of list searches. The first subroutine searches the appropriate Floor Storage List for items which lie wholly or partly within the display window, and each such item when found is inserted in the display by the second subroutine. The latter subroutine is equipped to deal with items extending across more than one grid square: in these cases there is a separate display unit for each square which the item occupies, and before inserting each of these units in its appropriate display list the subroutine checks that the unit lies within the display window. If it does not, the display unit is omitted.

Horizontal movements of the display window are achieved by moving all the display lists up or down the main display table. Only two alterations need be made to each list: the pointer to the first display block is moved to a new place in the table, and the return jump in the last block is adjusted. Before moving the pointers, all the display blocks in the 12 "leading" squares are returned to the Available Block List, and after all moving is complete the 12 "trailing" squares are filled by the two subroutines described in the preceding paragraph.

Duplicating and erasing items

In order to duplicate an item on the screen, the program has a routine for recognizing at which item the light pen is pointing. It does so by searching through the display list that belongs to the square under the pen.

For this purpose the aim of the pen must be more accurate than when making insertions: the small diamond-shape indicating the pen position must lie within or partly within the item boundary. This is the reason why the parameter word preceding each display list pointer contains positional information—the list is simply searched until a display unit is found which occupies the same segment of the grid square as the light pen. It is then quite easy to place the same set of display units on the pen, ready for insertion.

On typing "E" to request an erasure, the same routine as above is used to identify the item under the pen. Each of the item's display units is then replaced by a blank display unit, so that the item disappears. The addresses of the display units removed in this operation are temporarily stored in a "Scrap List" made up of four-word blocks, so that if the user wishes he can replace them by pressing the "return" key. On the other hand, if the user presses the space bar, indicating that he wishes to confirm the deletion, the information in the Scrap List is sufficient to identify the item involved and to remove its display and storage blocks permanently.

Moving an item is the logical combination of all the operations involved in duplicating and erasing. A similar Scrap List is set up, so that the item can be returned to its original position by typing "return".

Calculating areas

The program computes areas of enclosed spaces by completely traversing the perimeter of the space, meanwhile performing the integral $\int y\,dx$. When the starting point on the perimeter is reached, the current value of the integral is converted to square feet and displayed on the screen under the light pen. The method of traversing the perimeter is quite simple, but is fairly time-consuming because of the large number of list searches involved.

The grid square is too large a unit for accurate determination of areas, so each of the four segments of a square is treated separately. The "tartan" grid which thus arises is illustrated in **Fig. 5**, where it is shown superimposed by a *test grid* whose equally-spaced lines bisect the segments of the tartan grid. The perimeter point follows the lines of the tartan grid during the traverse, and its path, which determines the area, is established by a succession of tests made at the intersections of the test grid (*test points*). The purpose of each test is to find whether the test point is vacant or occupied by an item, and each item in the local Floor Storage List is therefore checked against the coordinates of the test point until an item occupying the point is found, or until the end of the list is reached.

Once a starting point on the perimeter has been found, the traverse algorithm takes over, using the following rules:

(i) Each test step is from the most recently found vacant test point to one of the four adjacent points.

(ii) If a test point is found to be occupied, the next step is at 90° *clockwise* to the previous step, and the perimeter point is moved in the same direction to the next tartan grid point.

(iii) If a test point is vacant, the next step is at 90° *anti-clockwise* to the previous step, and the perimeter point is not altered.

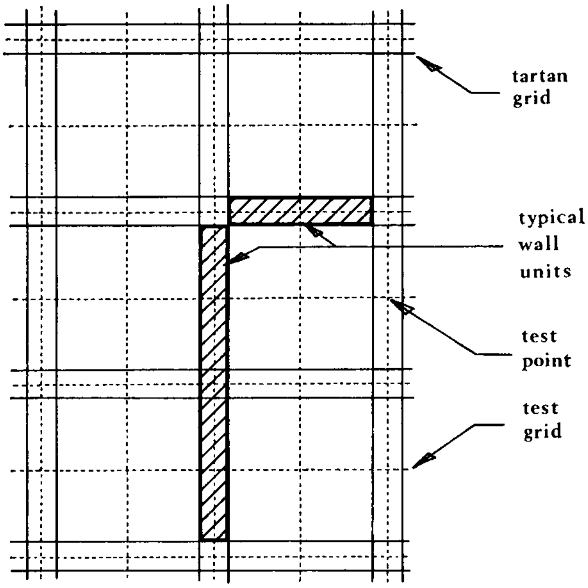

Fig. 5.—**Grids used in area computation**

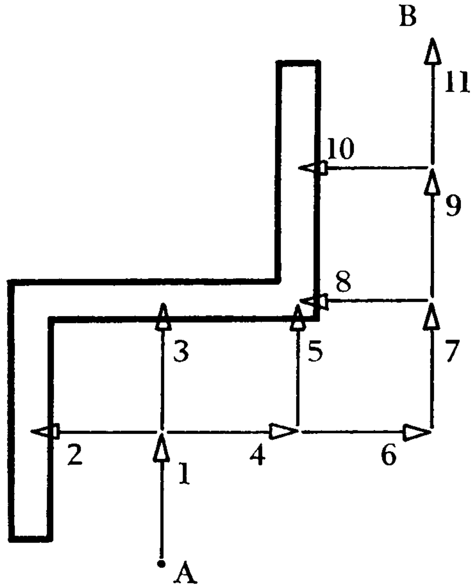

Fig. 6.—**Fragment of a traverse during area computation**

Fig. 6 shows the test steps involved in a typical section of a traverse, starting at A and finishing at B.

Each time the perimeter point is moved, $y\,dx$ is added to the area integral, and the traverse is complete when the starting point is reached. The initial routine for finding the starting point consists of searching in the $+y$ direction from the position of the light pen until an occupied test point is found.

This method as it stands cannot find the area of an obstructed space such as a courtyard containing a free-standing structure. However, because the program is continuously scanning in the $+y$ direction from the position of the light pen, the user can "present" these obstructions to the program by moving the pen beneath them. Their area is then automatically subtracted from the displayed total.

Acknowledgments

I wish to express my thanks to Professor M. V. Wilkes for generously allowing me to make extensive use of the PDP-7 computer, and to Mr. N. E. Wiseman and Dr. R. Penrose for advice and assistance in developing the program.

References

WILLIAMS, A. (1965). "IBIS Development Project," *Architect and Building News*, Vol. 227, p. 156.

SUTHERLAND, I. E. (1963). "Sketchpad: A Man-Machine Graphical Communication System," *AFIPS Conf. Proc.*, Vol. 23, p. 329.

STOTZ, R. (1963). "Man-Machine Console Facilities for Computer-Aided Design," *AFIPS Conf. Proc.*, Vol. 23, p. 323.

COMFORT, W. T. (1964). "Multiword List Items," *Comm. Assoc. Comp. Mach.*, Vol. 7, p. 357.

Book Review

Learning Machines, by Nils J. Nilsson, 1965; 132 pages. (Maidenhead: *McGraw-Hill Publishing Company, Ltd.*, 80s.)

The scope of this book is more restricted than the title suggests, since (*a*) only machines for pattern classification are treated and (*b*) even within this, detailed consideration is given only to that part of the classification task which remains when the coordinates of the pattern space have already been chosen. Restriction (*a*) does not seriously reduce the range of application of the treatment, since learnable tasks other than pattern classification can usually be decomposed in such a way that pattern classification plays a part. Restriction (*b*) is more serious, and is partially acknowledged in a section headed "The problem of what to measure".

The problem of choosing suitable pattern-space coordinates is not just that of deciding what to measure, however, at least in the case of machines using relatively simple types of discriminant function. The detailed mathematical treatment in the book is restricted to these. Such machines can only be successful if the representations of equivalent patterns are reasonably well clumped together in the pattern space. In most non-trivial applications it will therefore be necessary to process the input data before presenting it to the learning machine.

In the case of more complex machines, such as one consisting of more than two layers of adjustable elements, it is conceivable that useful learned behaviour could be achieved using the raw data as input. Part of the machine (e.g. those layers nearest the entry points of the input pathways) might learn to perform appropriate coordinate transformations. What training algorithms would be useful in such a machine is still an open question, though it is the faith of most workers under the banner of "Cybernetics" that suitable ones can be found. The difficulty of finding them is emphasized by the present book where, even in the case of two-layer machines, the only training algorithm given is one which modifies only one of the layers.

The restrictions on the scope have been emphasized because they are not implicit in the title, and because restriction (*b*) is often underemphasized in the literature on pattern-classifying devices. The book is useful in spite of them and is an admirable review of a body of very sound mathematical work which has been stimulated by studies of perceptrons and related devices. A variety of types of discriminant function is treated, as well as training algorithms and their respective convergence proofs.

One chapter is devoted to layered machines. Since the study of these, particularly where the number of layers is large, is presumably the key to further progress, it is useful to have this summary of what has been achieved. It turns out to be remarkably little.

The book presents a large amount of useful information in a clear and readable way. It is well produced, as it should be at the price, and will be valuable to anyone interested in machine learning or pattern recognition.

A. M. ANDREW