Representations for Space Planning

CHARLES M. EASTMAN
*Carnegie-Mellon University,*Pittsburgh, Pennsylvania*

Problems involving the arrangement of objects in two- or three-space where the objective function primarily consists of derivatives of the distance between objects or their arrangement are called space planning problems. The representational requirements for this problem area are defined and compared with current computer graphic languages. Four alternative data structures that allow automated space planning are described and compared.

KEY WORDS AND PHRASES: automated design, data structures, computer graphics, computer-aided design, engineering design, architectural design, robots
CR CATEGORIES: 3.20, 3.22, 3.26, 3.41, 3.63

1. Introduction

If the computer is to become more to future engineers and architects than an elaborate machine for carrying out numerical calculations, then means must be developed for it to deal with the nonnumerical aspects of design and planning problems.

Many problems within the general class called "design" emphasize topological and metric considerations. They consist of location problems in which total performance is a function of the distance between the elements being located or their arrangement. This aspect of a design problem can be called its *space planning* aspect. Automation of space planning requires a means for representing this type of problem in a computer. Space planning is also a factor in the development of intelligent robots. To determine appropriate movements, processing must be carried out on an internal map of the space in which the robot is located [1, 2].

Originally, the advent of cathode-ray tubes controlled by computers and the development of computer graphic languages (CGLs) were thought to resolve the major issues in representing and manipulating the space planning aspects of design problems. But the current lack of significant programs dealing with space planning is only one indication that existing CGLs do not facilitate computer augmentation of this type of problem. In this paper computer graphic and other types of data structures that conceivably could be used for space planning are reviewed. The requirements that a representation must satisfy to deal successfully with space planning are offered. Several new data structures for this use are described and evaluated.

* Institute of Physical Planning. This work was supported by the Advanced Research Projects Agency of the Office of the Secretary of Defense (F 44620-67-C-0058) and is monitored by the Air Force Office of Scientific Research.

2. The Problem

The form of external representation normally used by humans to model physical space and to solve space planning problems is a drawing. Particularly used are orthographic projections—plans, sections, and elevations. The elements represented may vary from a microcircuit to a steam turbine, from a unique pattern of specified material, such as furniture or machinery in an enclosed space, to economic zones in a region, depending upon the problem at hand.

To determine how orthographic drawings are interpreted and manipulated, a review was made of a number of carefully reported protocols of designers solving space planning problems [3, 4]. The following insights were gained. Lines on paper are generally used to demark disjoint spatial *domains* within a two-dimensional space. Each Cartesian point within a drawing corresponds to a like point in a proposed or real physical space. Attributes are ascribed to each domain. All points enclosed by a line are assumed to possess similar attributes; a domain is assumed to be homogeneous. All spatial information about each domain —its dimensions, shape, and attributes—is either directly available for processing or easily generated. This is true both for domains that are filled with material and for those that are empty. With all spatial information available for every domain, summations of the dimensions of adjacent domains may be used to determine the distance between any two points or any two domains. They may also be used to determine any desired spatial information for any aggregate set of spatial domains.

The basic operation for generating alternative solutions for space planning problems involved the relocating of a single domain or set of domains. The Cartesian points represented by a domain were altered, though their attributes, shapes, and dimensions remained invariant. Sets of domains, grouped by adjacency, also were relocated. A basic test involved in all relocation operations was an evaluation of a proposed empty domain to determine if the space required to locate the element was completely disjoint from all other filled domains. This could be considered as a test to ascertain whether an empty domain would completely encompass the solid domain it is receiving. Alternatively, it could be a test to check whether already located solid domains had points in common with the solid being located.

Further consideration suggested that any space planning

alternative could be generated and evaluated if the following capabilities were available and facilitated.

(1) Representation of domains of any shape.

(2) Determination of any dimension or attribute of a represented domain.

(3) Identification of any desired set of adjacent domains.

(4) Determination of any dimension or attribute of a set of adjacent domains.

Complex constraints seemed derivable from these types of capabilities. For instance, connectivity between any two points (e.g. to complete a circuit or to identify if accessibility exists between two points) is the search for a given state where a set of adjacent domains also include the given points. The sofa problem, formulated by Howden [5], is the search for a set of adjacent empty domains of a specifiable size and arrangement (possibly complex) that also are adjacent to two specified domains. Similarly, a test to determine if a clear view exists between two points is an evaluation of the attributes of domains (e.g. empty or solid) which are intersected by a vector connecting the two points. Lighting, acoustics, and material costs are also functions whose parameters include distance measures of single or adjacent sets of domains.

The determination of a location for a design element is facilitated by the above capabilities. In the attempt to aggregate a set of empty domains whose total dimensions are equal to or less than those of a solid to be located, overlaps with other solids are automatically excluded. The processing of adjacent domains requires the rules designating adjacency to be well defined. Most commonly, adjacent dimensions are summed parallel to the two Cartesian coordinates.

The above capabilities seemed to define well those needed for space planning. According to the above premises, the power of a representation to handle space planning is determined by the ease in which information about dimensions or adjacencies of a single domain or set of domains can be acquired.

3. Why Existing Graphic Languages Are Unsuitable for Space Planning

The data structures originally conceived to respond to the general needs of computer-augmented design were the CGLs developed for the generation of drawings on a cathode-ray tube. It was implicitly assumed that space planning was facilitated by such representations.

The graphic aspects of all CGLs are generally organized as follows. Physical elements are defined by a list of line segments that identify the spatial boundaries of the element. The line segments are connected head to tail, defining a continuous path. The electron beam of the CRT follows the path defined by these line segments, producing graphic output. Attributes of each element can be appended to the list to complete the specification of the element. The total list for an element is connected end-to-end so as to create a ring. The ring allows entrance at any point in the structure to have access to all other points and for rings to intersect one another. Thus particular attributes of different elements can be strung on a ring that links element rings together. In this way, rings can be used to generate a variety of topological organizations that allow multiple kinds of search and processing. For instance, design elements may be grouped into sets by a connecting ring. Those sets may also be grouped, allowing the definition of a hierarchy of ring structures. A diagram of the typical data structure thus produced is shown in Figure 1. (For a more extensive review of CGLs, see [6, 7, 8].)

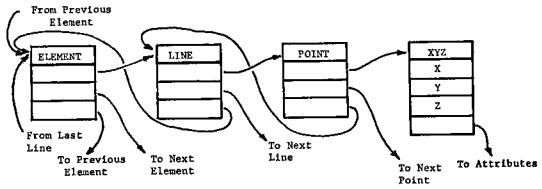

FIG. 1. A typical list structure for representing one line of an element in a computer graphic language.

An element or set of elements in the hierarchical data structure provided by a CGL can be accessed in a variety of ways. The usual operators available for manipulating CGLs include:

(1) Inserting an element at a specified place in the structure.

(2) Deleting an element and collapsing the gap left in the structure.

(3) Merging one ring structure with another (in a specified way).

(4) Performing some operation on all members of a ring or group of rings.

Such a structure and such operations have value in analyzing various kinds of clearly structured systems, such as hydraulic processes or electrical circuits. Elements in the system can be linked by attaching the appropriate ring to it. Analysis of flows through different topological organizations of diverse sets of elements is facilitated. But the analyses and operations required for space planning are not facilitated by such a data structure.

The limitations of using a CGL for space planning derive from the following characteristics.

(1) Only elements located in a space are represented in a data structure. The voids between elements are ignored.

(2) Line segments specify an element. Domains must be derived from these lines.

(3) The specification of each element is ordered in the data structure according to a programmer-defined topological organization, not spatially.

Because no specification exists of the size and shape of empty spaces it is not possible to test directly if an empty space may hold a particular solid. Required is a search of the total data structure for domain overlaps. Four types of algorithms have thus far been identified for checking overlaps. One, the data structure describing existing ele-

ments may be searched for line intersections with the new element. Two, an empty domain can be scanned dimensionally to determine if it can completely encompass the solid it is to hold. Three, the areas within the line segments may be subdivided into convex domains, allowing testing for disjointedness, such as by the method of determinants. And four, a point can be checked for disjointedness with a domain by counting the boundaries crossed by a line segment connecting the point in question and a point exterior to the domain. If an even number of boundaries of the domain are crossed, the point is on the exterior of the domain.

When it is recognized that a location trial is the most common operation in space planning, any representation must place a heavy weight on facilitating this operation. CGLs do not. Other operations such as the calculation of the amount of clear space adjacent to a given element, similarly require scanning the total data structure because adjacent domains are not directly accessible. In general, the shortcoming of CGLs is that they do not hold spatial information in a form related to the processing required for space planning. Significant processing costs are required to generate the information necessary for those operations used most often.

Because of these shortcomings, no significant programs written in a CGL are known that deal with space planning. No extensions of CGLs are known that include facilities needed for space planning.

A variety of other data structures have been developed that more directly respond to the needs of space planning. Each of these structures represents both filled and empty space. Each specifies the adjacency of spatial domains. Each also provides facilities for checking the overlap of domains.

4. Simple Arrays

The only representation widely utilized in mechanically solving space planning problems is the two-dimensional array. In this representation each subscripted variable in a predefined array represents a rectangular unit area, thus a domain. The subscripts of the domain provide the definition of its x and y Cartesian coordinates. The value of a variable in the array denotes its state. If its value is zero, then the domain is empty; if it has the value of

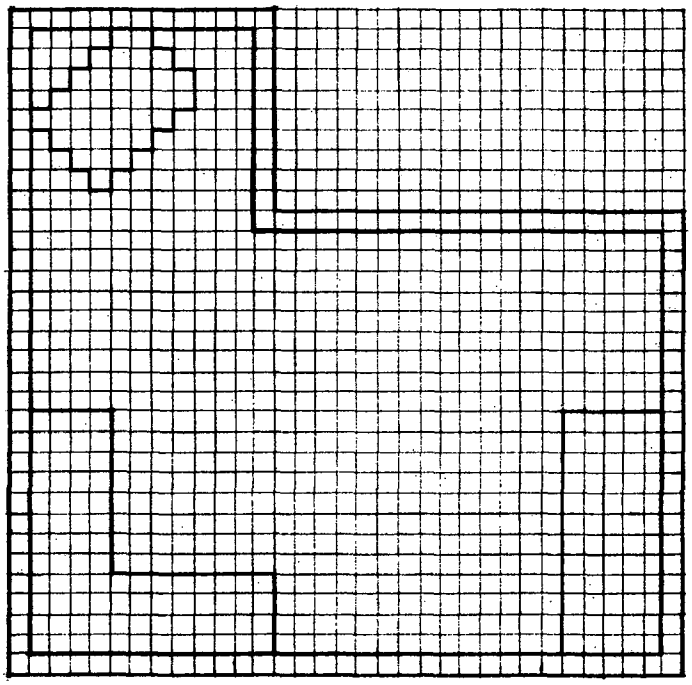

FIG. 3(a). The domains used by an array to represent the room in Figure 2. 1089 domains are used. Maximum error is 8.5 inches.

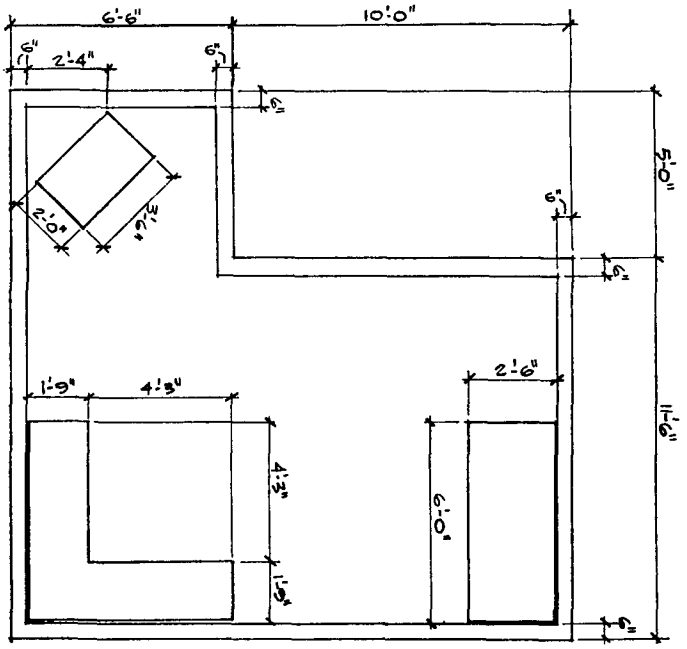

FIG. 2. Orthographic projection of the plan of a room.

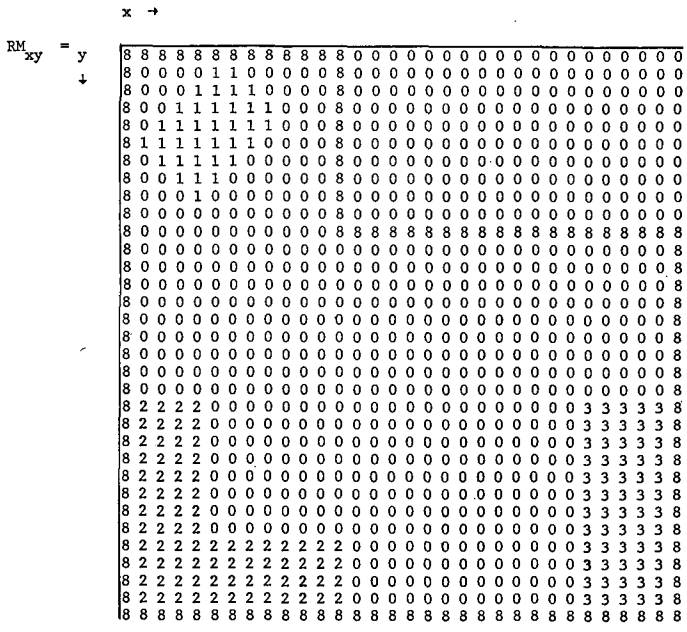

FIG. 3(b). The actual array used to represent the room.

twelve, the contents of all point locations within the domain are "Element Twelve."

In this representation, all domains are rectangular and of a single predetermined size. Both empty and filled domains are represented. Locations are directly addressable according to the subscripts of a domain. Adjacency is identified through the sequential ordering of the subscripts. A single domain size allows the subscripted variables to determine both the location and the dimensions of a set of domains. An example floor plan is shown in Figure 2. The domains used to represent it with a plain array are shown in Figure 3(a). The corresponding array is shown in Figure 3(b).

Operations on such a representation are straightforward. For example, one possible operation for locating an empty space of a given size and mapping a new element defined as an array into it is shown in Figure 4. Written in ALGOL 60, it generates an exhaustive search of the problem space scanning the columns vertically from left to right. (Other sequences for such a search are easily defined.) In this program, an initial location is identified when two of the required empty domains are found. An exact check is made as each domain of the object is mapped into the problem space. The objects are represented by values between ten and fifteen, and their access space by values between five and six. Thus an overlap between two solids or a solid and access space is designated by a value greater than fifteen, but access spaces may overlap. Additional Boolean conditions or evaluation functions can easily be used to evaluate or compare location trials.

The array has been used to represent space in a variety of facilities planning applications [9, 10, 11]. It has been applied in integrated circuit design [12]. It has also been used in experiments involving the design of intelligent robots [1, 13]. Most of these applications involve two-dimensional arrays, though some explorations have been made in three-dimensional applications [11].

The limitations of the array are self-evident. Its dimensional accuracy is determined by the size of the domain. Greater accuracy requires more domains in the representation and thus increases memory requirements. As operations must act on one domain at a time, any increase in accuracy requires a large increase in computing time. Also, all domains must be of square or rectangular shape. Thus irregular forms can only be approximated.[1]

Variations of the basic array representation have been developed that lessen memory requirements and processing time. The simplest compensation is to use arrays that comprise bytes instead of whole words [1]. A more elaborate scheme has been developed at Stanford Research Institute for use as a robot's internal representations of the world [2, 14]. Instead of a single predefined grid, they use a method of subdividing any given rectangular domain into 4×4 grid cells. Each cell can be further subdivided into 4×4 grids recursively. (Currently, their system is limited to three recursions, or a 64×64 array.) Homogeneous domains are not subdivided. Subdivision is only required at the boundaries of elements. A diagram of the domains expressed in such a representation is presented in Figure 5. (The representation shown subdivides domains recursively into 2×2 grids.) We call this representation a *hierarchical array*. The efficiency of the hierarchical array is its ability to define large homogeneous domains efficiently. More detail can be handled with a given amount of memory and processing time than in the plain array.

[1] For a special purpose application using three domain sizes, see [15]. Generally, similar costs and benefits are involved in all representations using predefined domains.

5. String Representations

A major disadvantage of the plain array as a spatial representation is its reliance on a single domain size. The hierarchical array is one attempt at resolution. It utilizes a regular method of decomposition and the sequential summation of nested arrays to determine distance. Other solutions are also possible.

If the domains themselves carry information about their size, then a single size is not necessary. Distances and possibly even location could be determined by summing the appropriate dimensions of adjacent domains. For example, one could make scans of a single unit width across a figure and group all like points into single domains. Like

```
BEGIN FOR A := 1, A+1 WHILE (QX=0) ∨ (QY=0) DO
  BEGIN IF OBJ[A,1] ≥ 10 THEN QX := A;
    IF OBJ[1,A] ≥ 10 THEN QY := A
  END;
  FOR A := 1 STEP 1 UNTIL RMX−OBX + 1 DO
  FOR B := 1 STEP 1 UNTIL RMY−OBY + 1 DO
  BEGIN IF (RM[A+QX−1,B]<10) ∧ (RM[A,B+QY−1]<10)
    THEN
    FOR C := 1 STEP 1 UNTIL OBX DO
    BEGIN FOR D := 1 STEP 1 UNTIL OBY DO
      BEGIN RM[A+C−1,B+D−1] := RM[A+C−1,B+D−1]
        + OBJ[C,D];
        IF RM[A+C−1,B+D−1] ≥ 15 THEN ERR := 1
      END;
      IF ERR = 1 THEN
      BEGIN FOR F := C STEP −1 UNTIL 1 DO
        FOR G := 1 STEP 1 UNTIL OBY DO
        RM[A+F−1,B+G−1] := RM[A+F−1,B+G−1]
        − OBJ[F,G];
        ERR := 0;
        GO TO SCAN
      END
    END;
    GO TO OUT;
    SCAN:
  END;
  OUT:
END;
```

FIG. 4. An ALGOL routine that locates a space and maps one array into another. OBX and OBY are the greatest Cartesian distances of the mapped array OBJ. RM is the space receiving the object. QX and QY are two arbitrarily chosen spaces that must be empty for a space to receive the object. They lead to the more detailed domain by domain evaluation.

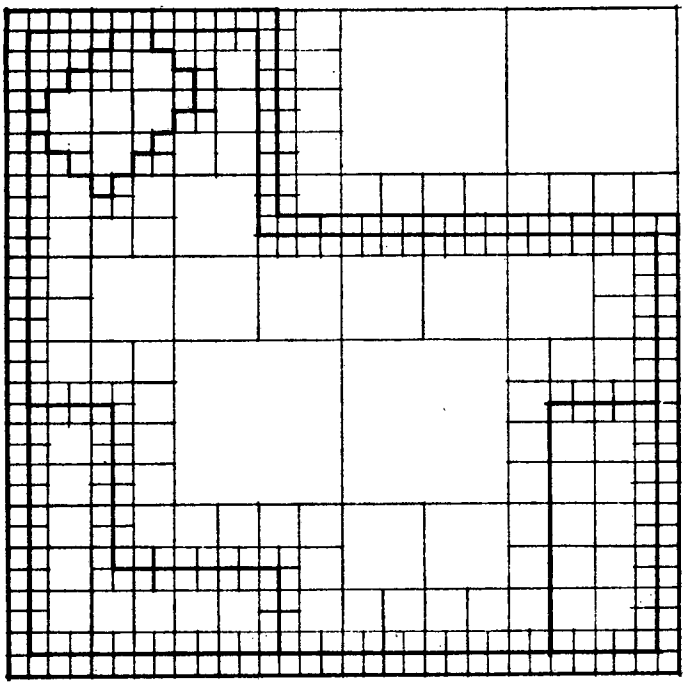

FIG. 5. The hierarchical set of domains defined by the SRI array. It requires 411 domains to represent the room in Figure 2 with a maximum error of 8.5 inches.

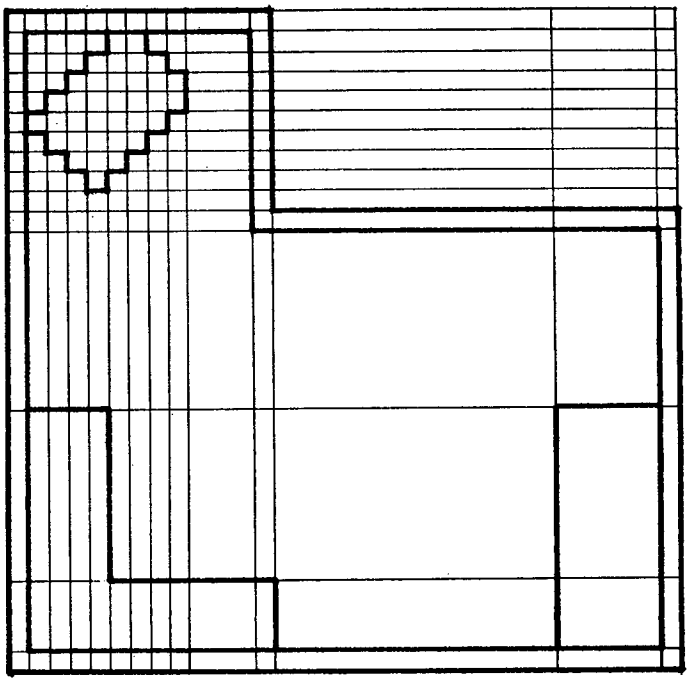

FIG. 6(a). The domains used in the string representation to express the room. It uses 65 domains and has a maximum error of 4.2 inches.

scans then could be grouped. Applying this domain definition technique to the floor plan shown in Figure 2 produces the domains shown in Figure 6(a). The actual data structure is presented in Figure 6(b). In Figure 6(b), a real number represents the relative distance between domain transitions. The letter suffix expresses the state of previous blocks: W, A, B, C, E represent walls, objects A, B and C, and empty space, respectively. The prefix defines the vertical extent of a set of domains. Thus a prefix, along with each symbol string within commas, defines a domain. We call this a *string representation*.

By explicitly dimensioning domains, a string representation is able to express forms more efficiently than the array, e.g. with fewer domains. Horizontal distances are no longer limited to multiples of a predefined domain, but can be real distances measured to any desired degree of accuracy. All domains are rectangular in shape. Skewed elements still require approximation according to a grid.

In a string representation, domains are defined according to a particular strategy for collecting homogeneous point locations. The state of a particular point location is determined by summing row prefixes in the y-coordinate and scanning the appropriate rows in the x-coordinate. The different treatment of each coordinate complicates most search strategies. Many operations such as distance calculations or scanning walls for a spatial attribute require an operator to search one coordinate at a time. Because searches in the two coordinates rely on information organized in different ways, it has been found useful to generate two problem spaces on which to operate, one rotated 90 degrees from the other. After a transformation

```
.5(6.5W, 10.0E)
.5(.5W, 2.1E, .5A, 2.9E, .5W, 10.0E)
.5(.5W, 1.6E, 1.4A, 2.5E, .5W, 10.0E)
.5(.5W, 1.1E, 2.3A, 2.1E, .5W, 10.0E)
.5(.5W, .6E, 3.1A, 1.8E, .5W, 10.0E)
.5(.5W, .3E, 3.1A, 2.1E, .5W, 10.0E)
.5(.5W, .5E, 2.4A, 2.6E, 5W, 10.0E)
.5(.5W, 1.1E, 1.3A, 3.1E, .5W, 10.0E)
1.0(.5W, 5.5E, .5W, 10.0E)
1.0(.5W, 5.5E, 10.5W)
4.5(.5W, 15.5E, .5W)
4.3(.5W, 1.8B, 11.2E, 2.5C, .5W)
1.2(.5W, 6.0B, 7.0E, 2.5C, .5W)
.5(16.5W)
```

FIG. 6(b). The actual data structure used in the string representation.

has been made in one representation, its isomorph is regenerated. The alternative is to have duplicate operations for the two coordinates.

The algorithm in Figure 7 describes the operations made on a string representation that corresponds to the replacement operations for the string array shown in Figure 4. The algorithm generates a depth-first tree search. Upon finding an instance of an empty space large enough in one coordinate to hold the new element, it "builds up" a space by connecting empty spaces to it in the other coordinate. Whenever it fails to find a space of the required size and location to further the buildup, the algorithm backtracks to the previous instance of an empty space and looks for another instance to build upon. When a space of appropriate size is built up, one space is mapped onto the other.

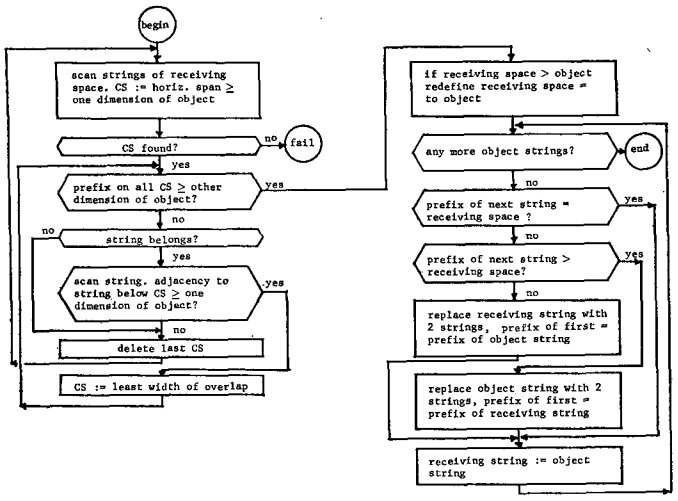

FIG. 7. The search strategy for strings that correspond to the one shown in Figure 4.

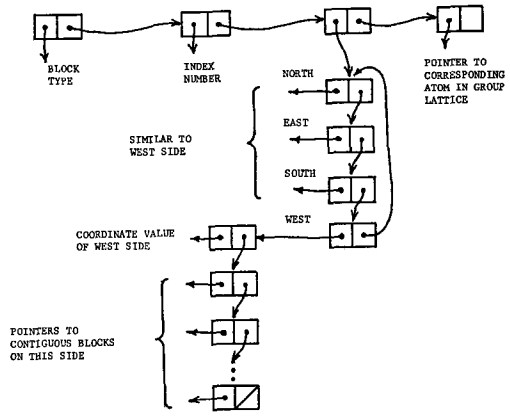

FIG. 8. List structure for a single domain using the adjacency structured representation.

Most operations on the string representation require the building up of individual domains so as to meet particular spatial requirements.

The use of strings to represent space location problems has been explored by the author, using SNOBOL4 as the base language [16].

6. Representations Using Adjacency Structures and Variable Size Domains

In arrays, locations are available through direct addressing in both coordinates. Strings, on the other hand, utilize both direct addressing and adjacency to find the state of a particular point location. Because accessing rules are dissimilar in the two coordinates, different operators are required for each. A single rule of adjacency in both coordinates allowing a single accessing rule suggests a new representation. Specifically, apply the rule that only single domains may be adjacent to each other in either coordinate. A program using this representation has been written by Moran in LISP1.5 [17, 18, 19]. Much of the following description is based upon his efforts.

In such a representation, the four coordinate boundaries of a block are explicitly represented within it on a list. The list structure for representing a rectangular domain is shown in Figure 8. Those blocks adjacent to each other in either coordinate are linked. For instance, the space shown in Figure 2 is connected into domains as shown in Figure 9. A location may be determined by summing the sizes of adjacent domains in both x- and y-coordinates. As in the string representation, finding a particular sized space in an adjacency-structured representation usually requires the building up of domains in both the x- and y-coordinates.

An interesting feature of Moran's program is that, instead of building up groups of domains dynamically during the search for a particular sized space, it has been conceived so as to organize automatically all combinations of domains into an always available group lattice. The organization of this lattice is as follows. All domains of each type, e.g. empty or Type Twelve, form a set. Partially ordered subsets, related by set inclusion, can be topologically ordered for adjacent domains of each type. For example, the space shown in Figure 10 can be defined by the group lattice as shown below. The total form T has as atomic domains A, B, C, D, E, F, G. Using the concepts of least upper bound and greatest lower bound, the domain sets represented by FC, FG, DG, DE, BE, AD, CD, and AB are grouped as unions of adjacent atomic

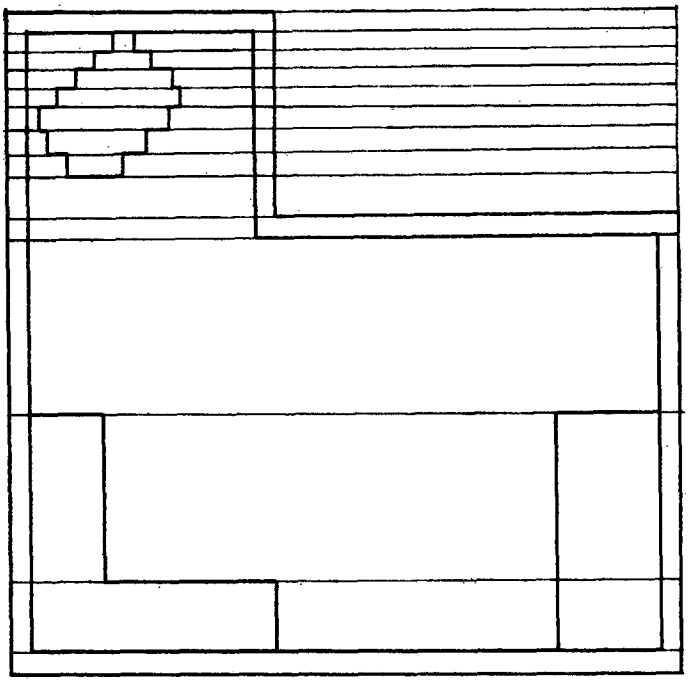

FIG. 9. An adjacency structured representation using rectangular domains and a single rule of adjacency. The maximum error is 8.5 inches when 225 domains are used to represent the plan in Figure 2.

blocks.[2] Further such ordering produces ADG, CDE, CDFG, and ABDE. Each group designates a block set and is itself made up of smaller domains. This lattice is incorporated into the basic definition of the form being processed using the data structure shown in Figure 11. The search for a domain of any size only requires scanning the already organized lattice. It is so organized that only portions of it require searching at any time. The cost of having such information available is paid in the constant bookkeeping required to update it.

The locating of an element consists of redefining the boundaries of a set of blocks in the lattice into a new subset which perfectly matches the space requirements of the element. Replacement is then carried out by redefining all the designated domains in the lattice.

The basic domain and adjacency structure of Moran's program may be implemented without resorting to list-type data structures. The structure involved includes variable sized rectangular domains connected by adjacency relations in the two coordinates. An array can be used to express this structure. Each subscripted variable represents a domain of a different size. Because only one domain may be adjacent to another in the x- and y-coordinates, domain sizes are consistent for each row and column. A zero vector in the x- and y-coordinates can be used to express all domain dimensions. Adjacencies can still be determined from the subscripted variables. The representation shown in Figure 12 is functionally equivalent to Moran's but can be processed with routines readily constructed in common algorithmic languages. We call this representation the *variable array*. The replacement algorithm shown in Figure 7 is applicable to the variable array. While not as efficient as a string representation in the number of domains required to represent a given form, its adjacency structure is the same in both coordinates. The variable array is relatively efficient in both memory and processing requirements.

7. Comparison

Plain arrays, hierarchical arrays, string representations, and variable arrays all explicitly express both empty and filled spatial domains. All also store information about domains in a topological structure based on spatial location. Thus distances or shape information never requires processing of the total representation; only relevant areas need be operated on. Each of the above representations uses a different method to define domains and possesses different memory requirements. Each also explicitly stores different location and shape information, facilitating different kinds of processing. The details of these differences follow.

The plain array represents all space in terms of a single domain size, each domain being addressible according to its absolute location. Because contents are not directly

[2] The greatest lower bound of the two blocks x and y is defined as the largest block z, which is included in blocks x and y. The least upper bound of two blocks x and y is the smallest block z, which includes blocks x and y.

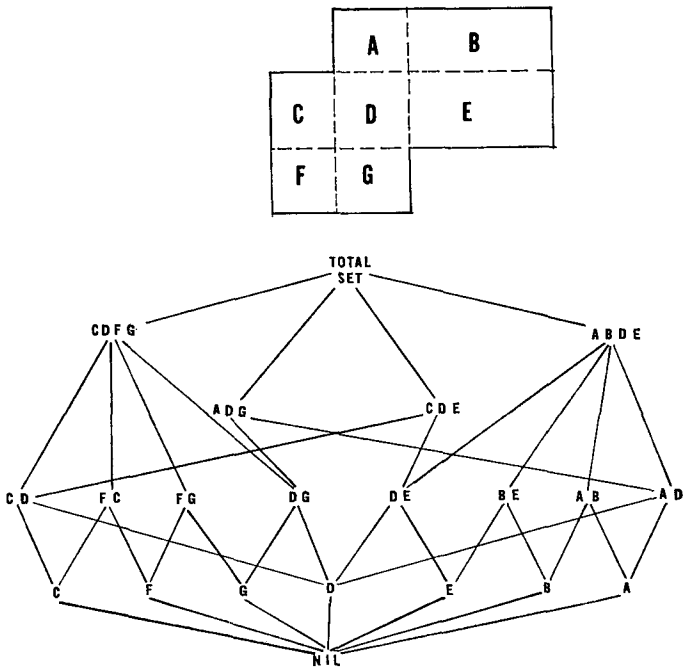

FIG. 10. The domains A, B, C, D, E, F, G are those used to define this form at the top of the figure. The lattice hierarchy and groups make up the form shown.

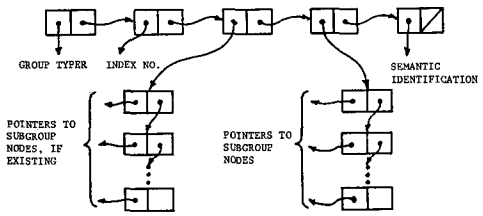

FIG. 11. List structure of one node in the group hierarchy used in Moran's adjacency structured representation.

RM_{XY} =

X → \ Y ↓	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0		.5	.5	.5	.5	.5	.5	.5	.5	.5	1.5	.5	7.0	2.5	.5
1	.5	8	8	8	8	8	8	8	8	8	8	8	0	0	0
2	.5	8	0	0	0	0	1	1	0	0	0	8	0	0	0
3	.5	8	0	0	0	1	1	1	1	0	0	8	0	0	0
4	.5	8	0	0	1	1	1	1	1	1	0	8	0	0	0
5	.5	8	0	1	1	1	1	1	1	1	0	8	0	0	0
6	.5	8	1	1	1	1	1	1	1	0	0	8	0	0	0
7	.5	8	0	1	1	1	1	1	0	0	0	8	0	0	0
8	.5	8	0	0	1	1	1	0	0	0	0	8	0	0	0
9	.5	8	0	0	0	1	0	0	0	0	0	8	0	0	0
10	.5	8	0	0	0	0	0	0	0	0	0	8	0	0	0
11	.5	8	0	0	0	0	0	0	0	0	0	8	8	8	8
12	4.5	8	0	0	0	0	0	0	0	0	0	0	0	0	8
13	4.25	8	2	2	2	2	0	0	0	0	0	0	0	3	8
14	1.25	8	2	2	2	2	2	2	2	2	2	2	0	3	8
15	.5	8	8	8	8	8	8	8	8	8	8	8	8	8	8

FIG. 12. An array of variable sized domains. This array is operationally isomorphic to the adjacency structured representation.

TABLE I. A Comparison of Four Space Planning Representations

	REPRESENTATION			
	Plain array	*Hierarchical array*	*String representation*	*Adjacency structure*
Number of domains required to represent example arrangement	1089	411	65	225
Greatest error in representation of elements not parallel to coordinate	8.5	8.5	4.2	8.5
Greatest error in representation for elements parallel to coordinate	6.0	6.0	0.0	0.0
Domains require definition?	no	yes	yes	yes
Structure similar in both coordinates?	yes	yes	no	yes

addressible, most applications of the array representation also incorporate a secondary memory which stores the location of each element.[3] Though this data structure is simple, the large number of domains it utilizes requires both extensive memory and processing times. In it, the accuracy expressed is proportional to the square of the number of domains used. It has been usefully applied to problems where all elements are highly modular. It is easily implemented in the most common algorithmic languages. The hierarchically subdivided array used at SRI removes some of the above limitations.

The string representation for space planning is a hybrid. While domains are content addressible in both coordinates, direct determination of adjacencies is possible in the x-coordinates only. Sequential processing of domains in the two coordinates differs, and thus requires either dual operators or an operation that rotates the data structure so that a single operator can be applied. While the string representation is the most efficient representation yet discovered, in terms of the number of domains required to express a configuration, it requires more complex operations. When a new element is located, the boundaries of the affected rows must be regenerated. Searching for a particular configuration of space is also time-consuming because of the varying structure in the x- and y-coordinates. The string representation may turn out to be particularly useful in graphic systems using horizontal raster line generation and halftone displays [20].

Moran's adjacency structured representation is completely content addressible. While its structure is consistent in both coordinates, its operations are complex and slow, partly due to the constant updating required of its adjacency and lattice structure. On the other hand, the variable array, which is its isomorph, expresses adjacencies automatically and stores the sizes of domains efficiently. It allows significantly faster processing and less memory. The variable array has been found to be a reasonable compromise allowing detail as well as efficient processing. It is easily implemented in almost any algorithmically oriented language.

The major differences between the representations that have been described are shown in Table I. The relative efficiencies of each are shown. Of the presented space planning representations, only the plain array has been extensively used for space planning. Yet each of the others offers significant efficiencies that should make them superior to the array for particular applications. With appropriate selection, these representations should allow significant space planning applications to be implemented on a computer.

[3] Location addressible means that memory locations are defined according to an absolute Cartesian location. Content addressible means that memory locations are associatively structured and can only be identified by an appropriately structured sequential scan.

8. Extensions

All the presented representations rely on rectangular convex polyhedrons for the expression of domains. They incorporate domains whose edges are perpendicular to the coordinates so that very simple computations may sum distances and areas of adjacent domains. This rule fails to deal with elements located nonparallel with the coordinates. It also cannot accurately deal with curvilinear forms.

To express irregular shapes more accurately, nonrectangular domains must be utilized. One possibility is to use triangular domains which are still convex but allow expression of complex shapes. But to make use of triangular domains, we must develop a strategy to sum adjacent domains. Another approach might be to use line segments, such as are used in CGLs, but subdivide all domains into convex shapes by an appropriate decomposition rule. Again, the summation of adjacent domains is a major issue. These or other extensions of space planning representations will require significant information to be stored about each domain's shape so that summations will still be possible without extensive processing.

In review, it seems that a suitable representation for space planning must include the following characteristics:

(1) Both filled and empty domains must be directly available for processing.

(2) Domains should be structured by adjacency or some other derivative of location. A consistent rule for summing the spatial characteristics of adjacent domains should be available.

(3) Domains should be defined so that overlaps between domains are easily identified. One such rule is that all domains be convex.

Only further efforts at data structure construction will produce a completely general and efficient representation for space planning; nevertheless, a trade-off between complexity and detail is likely to remain.

The representation requirements presented here also suggest that efforts in the linguistic processing of pictures may benefit from a review of the operational characteristics of the picture being processed. By analyzing what operations are carried out on a language, its structure is implied.

The analysis presented here suggests that spatial domains are the primitive element of this particular graphic language. In this light, the common assumption that line segments are the primitives of many graphic languages may require revision.

RECEIVED JUNE, 1969; REVISED OCTOBER, 1969

REFERENCES

1. GROSS, MAURICE, AND NIVAT, MAURICE. A command language for visualization and articulated movements. In *Computer and Information Sciences II*, Julius T. Tou (Ed), Academic Press, New York, 1967.
2. NILSSON, NILS J. A mobile automaton: An application of artificial intelligence techniques. Proc. Int. Joint Conf. Artificial Intelligence, May 1969, Washington, D. C.
3. EASTMAN, CHARLES M. Explorations of the cognitive processes of design, Dep. of Comput. Sci., Carnegie-Mellon U., Feb. 1968, ARPA Rep. DDC No. AD671158, Clearinghouse, Springfield, VA 22151.
4. EASTMAN, CHARLES M. Cognitive processes and ill-defined problems: A case study from design, Proc. Int. Joint Conf. Artificial Intelligence, May 1969, Washington, D. C.
5. HOWDEN, W. E. The sofa problem. *Comput. J. 11*, 3 (Nov. 1968), 299–301.
6. SUTHERLAND, I. E. Sketchpad: a man-machine graphical communication system. Proc. AFIPS 1963 Spring Joint Comput. Conf., Vol. 23, Spartan Books, New York, pp. 329–346.
7. GRAY, J. C. Compound data structure for computer aided design: a survey, Proc. ACM 22nd Nat. Conf. 1967, Thompson Book Co., Washington, D. C., pp. 355–365.
8. THOMAS, E. M. GRASP—a graphic service program. Proc. ACM 22nd Nat. Conf., 1967, MDI Publications, Wayne, Pa., pp. 395–402.
9. ARMOUR, GORDON C., AND BUFFA, ELWOOD. A heuristic algorithm and simulation approach to relative location of facilities. *Man. Sci.* (Jan. 1963), 244–309.
10. LEE, R. B. AND MOORE, J. M. CORELAP—computerized relationship layout planning, *J. Indust. Eng., 18*, 3 (Mar. 1967) 195–200.
11. SIMPSON, M. G., ET AL. The planning of multi-story buildings: a systems analysis and simulation approach. Proc. European Meeting on Statistics, Econometrics and Management Science, Amsterdam, Sept. 1968.
12. BARKEN, ROBERT. A set of algorithms for automatically laying out hybrid integrated circuits. Internal working doc., Bell Telephone Lab., Holmdel, N. J., Aug. 1968.
13. NILSSON, N. J., AND RAPHAEL, B. Preliminary design of an intelligent robot. In *Computer and Information Sciences II*, Julius T. Tou (Ed.), Academic Press, New York, 1967.
14. ROSEN, C. A., AND NILSSON, N. J. Application of intelligent automata to reconnaisance. SRI Project 5953, Third Interim Report, Rome Air Develop. Center, Rome, N. Y., Dec. 1967.
15. FAIR, G. R., FLOWERDEW, ET AL. Note on the computer as an aid to the architect. *Comput. J. 9*, 1 (June 1966).
16. GRISWOLD, R., POAGE, J., AND POLONSKY, I. The SNOBOL4 programming language. Bell Telephone Lab., Holmdel, N. J., Aug., 1968.
17. MCCARTHY, JOHN, ET AL. *LISP1.5 Programmer's Manual.* MIT Press, Cambridge, Mass., 1965.
18. MORAN, THOMAS. Structuring three-dimensional space for computer manipulation. Dep. Comput. Sci. working paper, Carnegie-Mellon U., Pittsburgh, Pa., June, 1968.
19. MORAN, THOMAS. A model of a multi-lingual designer. In *Emerging Methods in Environmental Design and Planning*, G. Moore (Ed.), MIT Press, Cambridge, Mass. (in press).
20. WYLIE, C. ROMNEY, ET AL. Halftone perspective drawings by computer. Tech. Rep. 4-2, Comput. Sci. Dep., U. of Utah, Salt Lake City, Utah, Feb. 1968.

Hansen—cont'd from page 241

The excessive times for the start and removal of an internal process are due to the peculiar storage protection system of the RC 4000, which requires the setting of a protection key in every storage word of a process.

9. Conclusion

Ideas similar to those described here have been suggested by others [4–6]. We have presented our system because we feel that, taken as a whole, it represents a systematic and practical approach to the design of replaceable operating systems. As an inspiration to other designers, it is perhaps most important that it illustrates a sequence of design steps leading to a general system nucleus, namely, the definition of the process concept, the communication scheme, and the dynamic creation and structuring of processes.

We realize, of course, that a final evaluation of the system can only be made after it has been used to design a number of operating systems.

Acknowledgments. The design philosophy was developed by Jørn Jensen, Søren Lauesen, and the author. Leif Svalgaard participated in the implementation and testing of the final product.

Regarding fundamentals, we have benefited greatly from Dijkstra's analysis of cooperating sequential processes.

RECEIVED JULY, 1969; REVISED JANUARY, 1970

REFERENCES

1. *RC 4000 Software: Multiprogramming System.* P. Brinch Hansen (Ed.). A/S Regnecentralen, Copenhagen, 1969.
2. *RC 4000 Computer: Reference Manual.* P. Brinch Hansen (Ed.). A/S Regnecentralen, Copenhagen, 1969.
3. DIJKSTRA, E. W. Cooperating Sequential Processes. Math. Dep., Technological U., Eindhoven, Sept. 1965.
4. HARRISON, M. C., AND SCHWARTZ, J. T. SHARER, a time sharing system for the CDC 6600. *Comm. ACM 10*, (Oct. 1967), 659.
5. HUXTABLE, D. H. R., AND WARWICK, M. T. Dynamic supervisors—their design and construction. Proc. ACM Symp. on Operating System Principles, Gatlinburg, Tenn., Oct. 1–4, 1967.
6. WICHMANN, B. A. A modular operating system. Proc. IFIP Cong. 1968, North Holland Pub. Co., Amsterdam, p. C48.