Associative processing of line drawings

by NEIL J. STILLMAN and CASPER R. DEFIORE

Rome Air Development Center (EMBIH)
Griffiss Air Force Base, New York

and

P. BRUCE BERRA

Syracuse University
Syracuse, New York

INTRODUCTION

The marriage of computer graphics and an associative memory is a natural union. This is evidenced by the widespread use of software simulations of associative memories in today's most flexible graphical systems. The content-addressability of a hardware associative memory makes conventional addressing schemes superfluous and eliminates the need for pointers required to link related data, vastly reducing system overhead. The parallel retrieval and update functions possible with a hardware associative memory remove any need for multiple storage which is so prevalent in current systems and simultaneously increases processing speed. The capability of implicitly storing relations between data further decreases the storage requirements, while increasing flexibility.

After examining current graphical data structures, all of which rely on a maze of pointers or multiple storage of information to represent the naturally relational graphical data, and reviewing the fundamentals of associative memories, a data structure utilizing an associative memory to process line drawings is presented.

BACKGROUND

One of the first systems to allow graphical communication with a computer, SKETCHPAD,[18,8] utilized two-way pointers. The data about drawings were actually structured in two separate forms. The first was a table of display spot coordinates designed to make display as rapid as possible, while the other was a ring structure designed to contain the topology of the drawing and facilitate its modification. Each entity consisted of n consecutive storage locations, with standardized locations for information about the various properties of each entity type. All references to a particular entity block were linked together by a string of pointers originating within that block and pointing to the succeeding and preceding members of the string. Different rings thread through several levels in an element providing several paths to the same information. Sutherland comments that his ring structure was not intended to pack the required information into the smallest possible storage space and that some redundancy was included in the ring structure to provide faster running programs. SKETCHPAD placed a higher priority on speed than on the ability to store huge drawings.

Another ring-oriented data structure is CORAL,[19,8] (Class Oriented Ring Associative Language). It stores data in blocks of arbitrary but fixed length. The blocks represent objects which can be connected by rings; each object can belong to more than one ring allowing the multi-dimensional associations required for graphical data structures. Unlike SKETCHPAD, CORAL isn't limited to two-way pointers; all ring elements have a forward pointer to the next element in the ring, and pointers to the ring start (type identifier) are alternated with back pointers for all ring elements. By alternation of the less useful pointers, CORAL retains the flexibility afforded by each pointer type, but requires only half the space and does not incur a significant time loss. Since efficiency was not a major consideration during the system's development, storage space and processing time produce a high overhead.

Similarly, DAC-1[11] and its successor APL[6,8] at General Motors use blocks of entity descripters, each of which describes an entity and its properties, linking blocks in current use together in a ring structure. By decomposing a picture into entities a hierarchical structure is obtained.

ASP[12,8] (Associative Structure Package), is another ring implemented data structure, but differs from the others discussed in that it is a dual ring structure. All elements belong to two rings; the "upper" ring being those elements possessing the same property; and the "lower" ring being a series of rings of elements related to the master element by different properties. The ASP structure allows interrogation in seven associative forms (Feldman[16]). Lang concludes that the user, depending upon his application, should determine how the rings are to be implemented, i.e., with only forward pointers, or with backward and/or ringstart pointers. If the rings are small, forward pointers are probably sufficient while if the rings are large other pointers should perhaps be introduced.

A slightly different approach is the data structure of GRAPHIC-2,[3] basically a directed graph with no closed loops. The structure contains four types of blocks; nodes and branches of fixed length and leaf and data blocks of arbitrary size. By convention, only the leaf blocks can contain displayable material, while the other blocks provide structural information. Because space is a scarce resource in the GRAPHIC-2 computer, an abbreviated pointer system is utilized, including neither back pointers nor ringstart pointers. To quote Christensen, "Tracing one's way through the structure therefore may require more time, but time is a resource that is more readily available in GRAPHIC-2." The directed graph is used also by Cotton and Greatorex[4] in their remote computer graphics system, and serves as the basis for the graphics data structure used at the University of Utah.[2]

Van Dam and Evans,[20] in an effort to reduce the size of a given graphical item to the absolute minimum, have kept their data structure as pointer-free as possible. The general structure of an item is a block containing (1) a set of "keys" which name or identify the information within an item, (2) elements which may contain any type of information, i.e., data, program, or both, and (3) a table of contents which associates the keys with their related items and thereby allows the system to locate elements within an item. An item is retrieved by providing a "description," a logical expression combining keys, elements, and conditions on the values of elements. Schemes for keeping part of the picture in tree form and part in reduced form (points and lines) are being considered for future implementations. The fact that the points and lines form makes lightpen pointing impractical but is nevertheless considered implies that the storage space is at a great premium.

The improvement of these systems centers about two tasks regardless of the data structure discussed: the processing of data at a faster rate, and the storing of data in the smallest possible space. These two goals, to date, have been incompatible, i.e., processing speed has been gained at the expense of storage and storage can be minimized only at the cost of processing time. With the advent of an "associative memory," speed and storage compression become compatible. The parallel search capability speeds processing while the content-addressability, which eliminates conventional addresses and therefore data pointers and all housekeeping functions associated with them, both increases speed and decreases storage requirements.

FUNDAMENTALS OF ASSOCIATIVE MEMORIES[21]

An associative memory has three main features not possessed by conventional memories: (1) word-parallel access of the entire memory, (2) word-parallel performance of its basic operations in the entire memory, and (3) the inclusion of comparison as a basic operation. In addition, word operations may be performed either bit-serial or bit-parallel. Bit-serial operation will compare sequentially against a 1 bit by n word slice of memory (where n = number of words in AM) across the word while bit-parallel operation will compare the entire memory m bits by n words (m = number of bits in a word) simultaneously. It appears that this distinction is of minor consequence, introducing only the element of a time delay without affecting the three prime features noted above.

The fundamental operations of any memory are reading from, and writing into, any word or bit location. An associative memory adds comparison to these two universal requirements. It is the parallel execution of the primary operations that sets the associative memory apart. A natural associative memory strategy is that of two-block partitioning, i.e., in order to perform an operation on *some* members of the associative memory, the members that are not to be operated on must be segregated. This is accomplished by an initial operation performed in parallel on all locations of the associative memory to flag the members of interest. Then, for example, a parallel write could be used to zero out all flagged words simultaneously, or a parallel read could be used to read the contents of the flagged words simultaneously into an external buffer. In the comparison operation, the reference word (comparand) is simultaneously compared with all flagged words. The comparand is not restricted to an entire word but may be any arbitrarily specified field in the word.

The provisions of word-parallel access and simultaneous comparison make the conventional concept of a memory address obsolete. Formerly, when only one word of a computer could be accessed at a time, information was stored in an orderly fashion in uniquely numbered storage locations. In the associative memory, information is retrieved by content, not location, hence the term "content-addressable memory." An associative memory is ideally suited to cross-referencing because unlike a conventional memory which must maintain a separate index for each characteristic, information may be retrieved on any combination of characteristics.

The instruction capabilities of associative memories are usually grouped into two categories; search instructions and arithmetic functions. The search instructions allow simultaneous comparison throughout any portion of memory (i.e., any number of words) and upon any portion (field) of a word (i.e., any number of bits). The search instructions[9] include the following: equality, inequality, maximum, minimum, greater than, greater than or equal, less than, less than or equal, between limits, next higher, and next lower. The Boolean operations AND, inclusive OR, exclusive OR, and complement may be performed between fields to provide complex query capability. Arithmetic operations of addition, subtraction, multiplication, division, increment field, and decrement field are indispensable in such graphical operations as scaling and translation.

In summary, an associative memory is ideally suited to perform operations on large amounts of data since it can operate on all members of the data simultaneously, in the time of a single operation, the only constraint being memory size. An associative memory therefore, in theory, has a speed advantage in proportion to the number of words of data to be processed.

GRAPHICS AND THE ASSOCIATIVE MEMORY

Ring structures yield answers to questions such as "What are the coordinates of Square X?," and its converse "(X_1, Y_1), (X_2, Y_2), (X_3, Y_3), (X_4, Y_4) are the coordinates of what square?" by virtue of their forward and backward pointers. There are more than two ways to pose a query however. Consider the question "What is the relationship, if any exists, between point X and point Y?" or "What pairs of objects are associated by the relationship SIDE OF?" In all, there are seven associative forms[16] of a query as shown in Figure 1. Since the relation is not explicitly stored in any of the previously discussed data structures, there is no way of answering questions phrased in forms 4, 5, 6, or 7. In order to answer questions in the last four ways, conventional concepts of data processing must be abandoned. The new structure must store, in addition to the objects, the relationship associating them. This task has been accomplished in similar ways by Rovner and Feldman,[15] Ash and Sibley,[1] and Levien and Maron.[13] In these approaches the "triple" (form 1) ATTRIBUTE OF OBJECT = VALUE is the basic element of the data structure. Levien and Maron add a fourth parameter by giving each "triple" a name identifier. This parameter may be used as an element in another "triple." In these approaches, each "triple" is stored at least three times to *simulate* an associative memory and enable queries in all seven associative forms to be more efficient than in a single listing.

Searching is minimized by using a hashed addressing scheme which will translate the query directly into the address of the answer (or linked to the answer). A hashed addressing scheme does, however, produce conflict situations, i.e., more than one pair of elements can hash to the same address, producing a conflict that must be resolved, for example, by a chained search of other answers until the desired answer is identified. The tradeoff is between tolerable conflict and the size of the addressable space.

Two major improvements to the present-day simula-

	ATTRIBUTE OF	OBJECT	=	VALUE
(1)	SIDE OF	SQUARE 1	=	LINE 1
(2)	SIDE OF	SQUARE 1	=	?
(3)	SIDE OF	?	=	LINE 1
(4)	?	SQUARE 1	=	LINE 1
(5)	SIDE OF	?	=	?
(6)	?	?	=	LINE 1
(7)	?	SQUARE 1	=	?

Figure 1—Associative forms of a query

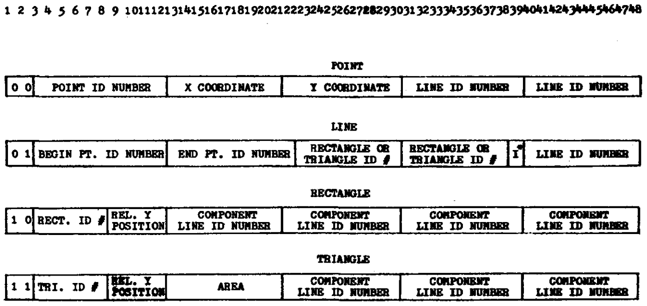

Figure 2—Multi-relational graphics data structure

tions of the associative memory would be the elimination of multiple storage of all relations and the removal of the conflict situation caused by hashing. Both are provided by a true "associative memory," which processes a random list of triples in any of the seven query modes in the most efficient manner possible, having stored it only once while not requiring an addressing scheme. Even though the data triples are stored only once, by the nature of the word-parallel, bit-serial operation with masking of an associative memory, all seven associative form questions can be answered with equal ease.

In the above scheme each triple takes up one word in the associative memory, i.e., the data are structured one relation per word. Storing one relation per word, however, doesn't even begin to take advantage of the power of an associative memory. Utilizing a Control Data 1604B computer interfaced with a prototype associative memory built by Goodyear for Rome Air Development Center[9] and containing 2048 forty-eight bit words, a complete two dimensional line-drawing graphics system can be implemented. Imposing the following constraints on the system will assure that any drawing will fit completely in the associative memory. The maximum number of points per drawing will be 1024, and the maximum number of lines will be 512. Also, a maximum of 64 rectangles and 64 triangles can be defined. Two bits of each word are required to specify the entity type. Six bits defines a unique ID number for each rectangle or triangle, while it takes nine bits and ten bits to specify lines and points respectively with a unique identifier. The four relations

SIDE OF SQUARE X = LINE W

SIDE OF SQUARE X = LINE X

SIDE OF SQUARE X = LINE Y

SIDE OF SQUARE X = LINE Z

which are stored three times (approximately 12 words) in LEAP[15] and similar systems, and once (four words) in the same system using an associative memory can be stored in one word by placing the nine bit codes of the four lines in the word identifying the rectangle which they compose. This particular example therefore requires about 10 percent of the storage requirement of any system in existence today. Triangles, lines, and points are defined similarly (See Figure 2). The limits specified above provide for identification of all entities in 1664 words, leaving 336 words unused. A point may belong to more than two lines but only space to specify two is provided. At absolutely no overhead to the system another word may be used, repeating the first 30 bits of the point record, and specifying the identifiers of two additional lines. This may also be done with a line which belongs to more than two rectangles or triangles. Exclusive of time factors external to the associative memory (which would be incurred conventionally as well) it would take, for example, less than 60 microseconds to retrieve the record of a specific rectangle.[9]

Possibly the most notable feature of the planned implementation is its extremely fast update capability. Scaling and translation, which are merely multiplication by, and addition of, a constant respectively are accomplished, in their entirety and regardless of the complexity of the picture, in the same time that a conventional memory processes one coordinate. This fact that retrieval and update functions are completely independent of picture complexity (as long as the picture is contained completely in the AM) sums up the greatest advantage of the associative memory. Another notable feature is that the system overhead per picture, again regardless of the complexity of the picture, is always four words.

The update or modification of a picture is most dependent on "pointers" or "relations" and it is threading through all these that consumes most of the time in conventional approaches. The elimination of this maze due to the content addressing of the associative memory provides, for example, the deletion of a line and therefore all objects to which it belongs (i.e., rectangles, etc.) in about 140 microseconds.[9]

CONCLUSIONS

Feldman's simulator[7] raised the question whether it would pay to build hardware associative memories for general purpose use since it should be feasible to build a software system which loses a factor of about two in storage, and three-to-five in time, against an associative memory of the same basic speed.

It should be noted, however, that according to Minker,[14] present day relational data systems technology has emphasized retrieval to the exclusion of maintenance, i.e., update capability. Maintenance functions depend primarily on the "pointers" or "relations" and therefore associative memories will exert their maximum influence in this area.

In addition, the work of Sibley[1,17] is patterned after Rovner's[15] "triples" using hashed addressing. His view is that software simulations of associative memories "for the moment . . . are a stopgap measure."

Feldman's estimate of a loss of a factor of two in storage to an associative memory seems very conservative in light of the new data structure introduced above. Figures as to the time advantage of such a system will have to await implementation of the data structure but it is expected that timing results will show Feldman's estimate of a saving of three-to-five in time to also be very conservative.

The software simulated associative memory using hashing is limited to an exact match operation, and all other search strategies must be built on multiple use of the exact match operation, due to the fact that hashing requires a completely specified field on which to apply the hashing algorithm. On the other hand, a hardware associative memory has about a dozen different basic search capabilities indicating that a hardware associative memory is far more flexible than a software simulation of an associative memory.

As an example, consider the problem of finding all lines of length between four and six inches. Let the name and length be specified for each line. In the simulated associative memory if hashing is done by name only, or by name and length, the question cannot be answered; if hashing is by length only *and* the lengths are integral then an exact match on lengths four, five, and six will yield the answers. However, if the lengths are continuous between four and six, then again, for all intents and purposes, the simulated associative memory cannot yield an answer. In contrast, a hardware associative memory would do a single between limits parallel search and arrive at a complete solution in less than twice the time required for an exact match.

As integrated circuits come into widespread use and the price of an associative memory drops to about twice that of a conventional memory[5] more and more people will begin to examine its unique advantages.

REFERENCES

1 W L ASH E H SIBLEY
TRAMP—An interpretive associative processor with deductive capabilities
Proceedings ACM National Conference 1968 pp 143-56

2 S CARR
Geometric modeling
University of Utah Technical Report 4-13 1969

3 C CHRISTENSEN E N PINSON
Multi-function graphics for a large computer system
Proceedings Fall Joint Computer Conference Vol 31 1967 pp 697-711

4 I COTTON F S GREATOREX JR
Data structures and techniques for remote computer graphics
Proceedings Fall Joint Computer Conference Vol 33 Part I 1968 pp 533-544

5 C DEFIORE
Fast sorting
Datamation Vol 16 No 8 August 1 1970 pp 47-51

6 G G DODD
APL—A language for associative data handling in PL/I
Proceedings Fall Joint Computer Conference Vol 29 1966 pp 677-684

7 J A FELDMAN
Aspects of associative processing
MIT Technical Note 1965-13 April 1965

8 J C GRAY
Compound data structure for computer-aided design—A survey
Proceedings ACM National Conference 1967 pp 355-365

9 *Handbook of operating and maintenance instructions for the associative memory*
Vol II—Associative Memory Programming Manual
Goodyear Aerospace Corporation Akron Ohio
GER-13738 March 1968

10 A G HANLON
Content-addressable and associative memory systems—A survey
IEEE Transactions on Electronic Computers August 1966

11 E L JACKS
A laboratory for the study of graphical man-machine communication
Proceedings Fall Joint Computer Conference Vol 26 1964 pp 343-350

12 C A LANG J C GRAY
ASP-A ring implemented associative structure package
Communications of the ACM Vol 11 No 8 August 1968 pp 550-555

13 R E LEVIEN M E MARON
A computer system for inference execution and data retrieval
Memorandum RM-5085-PR Rand Corporation Santa Monica California September 1966

14 J MINKER J D SABLE
Relational data system study
Auerbach Final Report—Contract F30602-70-0097 July 1970

15 P D ROVNER J A FELDMAN
The leap language and data structure
January 1968

16 P D ROVNER J C FELDMAN
An algol-based associative language
Communications of the ACM Vol 12 August 1969 pp 545-555

17 E H SIBLEY D G GORDON R W TAYLOR
Graphical systems communications—An associative memory approach
Proceedings Fall Joint Computer Conference Vol 33 Part I 1968 pp 545-555

18 I E SUTHERLAND
Sketchpad—A man-machine graphical communication system
Proceedings Spring Joint Computer Conference Vol 23 1963 pp 329-346

19 W R SUTHERLAND
On-line graphical specification of computer procedures
Tech Report 405 Lincoln Laboratory Cambridge Massachusetts 1966

20 A VAN DAM D EVANS
A compact data structure for storing, retrieving, and manipulating line drawings
Proceedings Spring Joint Computer Conference Vol 30 1967 pp 601-610

21 A WOLINSKY
Principles and applications of associative memories
Third Annual Symposium on the Interface of Computer Science and Statistics Los Angeles California January 1969