AD-A095 992 HIGHER ORDER SOFTWARE INC CAMBRIDGE MA F/G 5/8
FOUNDATION OF A KNOWLEDGE REPRESENTATION SYSTEM FOR IMAGE UNDER--ETC(U)
OCT 80 L VAINA, S CUSHING N00039-79-C-0457
UNCLASSIFIED HOS-TR-27 NL

END
DATE FILMED
4-81
DTIC

HIGHER ORDER SOFTWARE, INC.
806 Massachusetts Avenue
Cambridge, Massachusetts 01239

LEVEL II

TECHNICAL REPORT NO. 27

FOUNDATION OF A KNOWLEDGE REPRESENTATION SYSTEM FOR IMAGE UNDERSTANDING

OCTOBER 1980

Final Report Prepared for
Department of the Navy
Naval Electronic Systems Command

NOTICES

DISCLAIMERS

The findings in this report are not to be construed as an official Department of the Navy position, unless so designated by other authorized documents.

The citation of trade names and names of manufacturers in this report is not to be construed as official Government endorsement or approval of commercial produces or services referenced herein.

DISPOSITION

Destroy this report when it is no longer needed. Do not return it to the originator.

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER (14) HOS-TR-27 ✓ 2. GOVT ACCESSION NO. AD-A095992	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) (6) Foundation of a Knowledge Representation System for Image Understanding.	5. TYPE OF REPORT & PERIOD COVERED (9) Interim Report, May [redacted] — Aug [redacted] 80 ✓
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) (10) Lucia Vaina [redacted] Steven Cushing	8. CONTRACT OR GRANT NUMBER(s) (15) N00039-79-C-0457 ✓
9. PERFORMING ORGANIZATION NAME AND ADDRESS Higher Order Software, Inc. / Post Office Box 531, 806 Massachusetts Avenue Cambridge, Massachusetts 02139	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Department of the Navy Naval Electronic Systems Command Washington, DC 20360 (11)	12. REPORT DATE Oct [redacted] 80 ✓ 13. NUMBER OF PAGES
14. MONITORING AGENCY NAME & ADDRESS *(if different from Controlling Office)* (12) 98	15. SECURITY CLASS. *(of this report)* Unclassified
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*

Approved for public release, unlimited distribution.

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

18. SUPPLEMENTARY NOTES

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

knowledge representation, image processing, image understanding, possibility theory, fuzzy algorithms, quantifiers, semantics areal coordinate systems

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

The basis of a knowledge representation system is presented that is able to recognize real-world objects from partial information delivered by human or mechanical "experts", each of which is assumed to have its own information-processing tasks. The system deals directly not with the real world, but with the outputs of the experts' processing, and consists of three component parts, a descriptional component, a category component, and a functional component, in each of which knowledge is structured and accessed from general to specific, making it possible to access exactly

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

34302

as much information as is desired at an appropriate level of detail. Uncertainties are dealt with through the use of possibility theory, which also provides a means for approximate pattern matching. The knowledge representation language includes the use of trivalent quantifiers, whose semantics is explained and elaborated. Areal coordinate systems based on hexagons and squares are examined as possible alternatives to the standard use of bands in image processing.

TABLE OF CONTENTS

1.0 INTRODUCTION

Our overall goal here is to present the basis of a knowledge representation system able to recognize objects in the real world from partial information delivered by various experts.

The first hypothesis is that there is a set of experts, each having his own information processing tasks; these experts process information from the outside world in their own way and then convey it to the knowledge representation system.

Thus, the knowledge representation system does not deal directly with the world, but with inputs that are outputs of the experts' processing. We shall call our system object-views-system (OVS). The information processing task of OVS is to put together the various kind of input information delivered by the experts, to choose among alternatives, to recognize the objects in the world as completely as possible.

We consider that an object, together with its shape, position and location, is characterized by its functionality, by the action it can perform or how it can be acted upon.

Our system has three distinct parts: First, the descriptional part, where the experts information is represented and combined by means of possibility distribution theory.

Second, the category part, which is a kind of skeleton of the architecture of the system. The category part contains classes of objects, and accesses the more particular information. This is useful for holding the system together, for computing similarity between objects, for quickly retrieving desired information in as detailed a form as necessary.

Third, the system has a functional part , where the functionality, the use of the objects, is represented. In the functionality part is represented are the options the object is capable of being used for.

Why is OVS interesting? As a result of the fact that knowledge is structured and accessed from general to particular, OVS provides a means to access as much information as necessary, and to stop at the desired level of detail.

Due to its three component parts, OVS gives complete information about objects in the world, and the modularity of the represention makes it very efficient. Provided that one knows roughly what he is looking for, he can access the relevant module. OVS does not rely on much prestored information. Thus the user need not put much world knowledge into the system. OVS is a recognition system fed by the experts. Relying on possibility theory, OVS has the advantage of dealing with the uncertainties of the real world and provides approximate pattern matching. Of course, the approximation is determined by how much precision is needed to carry through the current computation.

In Section 2, we discuss the OVS system itself, its structure and how it works. In Section 3, we discuss a unique feature of our knowledge representation language, namely, its use of trivalent quantifiers, whose semantics enables us to use expressions like all ships, only submarines, also aircraft carriers, and so on, in a manner approximating that of ordinary English. In Section 4, we give examples to illustrate how an expert in our system--e.g., radar--works. In Chapter 5, we investigate how the efficiency of our experts may be improved by introducing hierarchical areal coordinate systems that replace the bands standardly used in decomposing planar images, and assumed in earlier sections, with hexagons or squares, which provide regular coordinate areas and a natural structure to the decomposed plane.

2.0 OVS SYSTEM

Our goal here is to present the computational constraints involved in the design of a knowledge representation system which is considered to be the basis for the development, in the next couple of years, of concrete higher-level methods of efficient plan generation, goal achievement, goal correction, domain-dependent and goal-oriented question-answering, etc. The last part of the research plan envisaged shall focus on the design of constraints for computer-based languages which would embody the above theoretical results.

Accordingly, (following Marr [1]) we have differentiated three levels for the research. The highest level is the level of the computational theory, which is a theory of the overall computation. This level is crucial; its study determines what the computational problems are that have to be solved and why they are needed. At this level we shall not be concerned with the details of algorithms that must ultimately implement the computational theory.

The next level is the level of the algorithms that are determined by the problem, and the mechanisms involved in carrying them out. Many algorithms that can be designed to carry out the same computation, and the choice of which one to use depends on the mechanisms that are available. The mechanism are usually determined by the nature of the hardware, which constitute the third level.

A great deal of research in Artificial Intelligence (AI) has started with the algorithm level, without worrying about the nature of the computation that the algorithms carry out. The algorithms are not a deep property of a computation, yet the implementation is important because, on the one hand, it reveals details that are not observed during the elaboration of the theory, and, on the other hand, this is what the user requires: to be able to use the computer to do the job

that he cannot do, or that it would be too expensive, or too complicated, for him to do.

Our general research problem is understanding images. What is given is many types of partial information about the real world delivered via several channels. We shall call such specific ways of delivering information about objects expert-based information. The expert-based information is complete from the point of view of each expert, but it is specific, partial information about the real world. Experts can be man or machine, each having some specific knowledge or area of expertise. We can not evaluate the expert's expertise before we have the general representation, and within this framework we could pose problems like improving the expert's skill; in other words, what is needed from the expert to give more complete relevant and useful information to the recognition system? This is an interesting theoretical and practical problem. The basic question would be what is the number of experts needed for obtaining enough information for recognition tasks? Is it more useful to have many experts with a little domain of expertise? Or it is better to have as few as possible, with a wider area of knowledge? To answer these questions, one needs to take into account the real-world possibilities. For example, if we consider the radar as one of the experts, and we know what information it delivers, we have to ask the question: how useful is this information? How can it be improved? What other experts must cooperate with the radar in order to deliver the most useful information? How reliable is the radar information? How well does it communicate with other experts? By this last question we mean what are the experts which understand information delivered by radar? Are the experts communicating with each other, or are they delivering their information to a central processor which goes through the process of putting everything together?

Our hypothesis is that each expert has his area of expertise, and has his own way of obtaining information, which, guided by the needs of the

system, he delivers in an intelligible way. By an intelligible way we mean the fact that the expert's response, or output, is cast in the mold offered by the system. Thus, many times, more than one expert will answer the same question in his own way. This gives the central system the possibility of contrasting and comparing the information.

What we have to do is to recognize the real-world object corresponding to this expert information and design a representation which will be useful for such tasks as computing the similarity between objects from the representation, and identifying only some prerequisite objects, such as destroyers or submarines, for example. The representation must be capable of allowing higher-level tasks involving reasoning, problem solving, etc., to be performed.

Theoretical Basis for Our Research

Given what we want to achieve, i.e., to understand images, and the way in which we argued that the research must proceed (on the computational level and on the algorithmic level) we believe that the most appropriate tools would be those offered by artificial intelligence approaches and by possibility theory, or fuzzy sets in general.

What makes our research unique is our use of both these sets of tools, and their application on both levels of investigation.

Most AI research in representation of knowledge concerns level two of the approach, the algorithm. We rely mostly on theoretical ideas laid down by Marr [2]; Marr and Poggio [3], Marr and Ullman [4], who provided a computation theory for low-level vision.

Our problem is different, however; we do not try to understand vision in general, nor do we worry about the human visual system. Our goal

is to understand images. The overall picture of the world that we are moving in is that of expert's information about the real-world (ex. radar image, etc.). Thus, the input information to our problem is preprocessed information by specific experts. What we want to do is to recognize the objects in as much detail as is needed for the current purpose, to predict their actions, to describe their state, and to predict their next state. The amount of data involved might be extremely large. This leads us to the second tool - fuzzy sets and possibility theory. On the level of computation we shall use possibility theory to master the huge amount of information and avoid getting cluttered in useless detail. On the level of the algorithm we shall deal with fuzzy algorithms. A "fuzzy algorithm" is "an ordered set of fuzzy instructions which upon execution yields an approximate solution to a specific problem" (Zadeh [5]). Fuzzy algorithms are extensively used in everyday life, when one drives a car, searches for an object, cooks food, etc., and also more abstractly in pattern recognition and decision making.

The Representation Issue

Criteria for efficient recognition:

Marr and Nishihara [6], in their study of how to represent three-dimensional shape information, laid down the following three criteria to be satisfied by a representation in order for it to be useful for recognition.

(1) Accessibility, (computability)

The representation should be easy to compute from the image.

(2) Scope and uniqueness

The representation is used for recognition, so the description of a shape must be unique.

(3) Stability and sensitivity

To be useful for recognition the degree of similarity between two shapes must be reflected in their descriptions, but at the same time small differences must be observed. Stability is increased by using a lower resolution. This leads to the idea of a hierarchical representation as we shall see later.

The representation

The representation should provide a description of a shape that is unique. Thus, we have to have a canonical form for the representation. A canonical form is a computable function which transforms any expression "a" into a unique equivalent expression f(a) such that for any two expressions a_1 and a_2, a_1 is equivalent to a_2, if and only if $f(a_1)$ is equal to $f(a_2)$. With such a function one can avoid the combinatorial search for an equivalence chain connecting the two expressions and merely compute the corresponding canonical forms and compare them for identity. Thus the canonical form provides an improvement in efficiency over having to search for an equivalence chain for each individual case.

For this we have to assume that the function f is effectively computable.

Definition: A function f is computable if and only if there is an effective procedure which, given an n-tuple $(x_1 \ldots x_n)$ for its variables, will produce $f(x_1 \ldots x_n)$.

It is useful to remember that the following conditions on a function f are equivalent:

(a) f is computable
(b) When viewed as a relation f is a decidable relation.

Design of the Representation

Primitives:

The complexity of the primitives used by a representation is constrained by the type of information that can be derived reliably by processing prior to the representation. Thus, in the case of the radar-expert we include that its output information, i.e., the values for the length, orientation, width, and velocity. An important point here is that size and similarity are crucial aspects of a representation's primitives that influence the information it makes explicit. Thus, information about features much larger or much more general than the primitives used is difficult to access since it is represented only implicitly in the configuration of a large number of smaller or more detailed items. On the other hand, features that are much smaller than the primitives are omitted from the description because they are totally inaccessible.

Modularity:

An important question is how the information is organized by the representation. All that is needed for a representation is some scheme of associations, together with a set of information processes, that will extract the appropriate information about connections. The main structures to provide appropriate direct representation of most things of interest to us are:

1. List structures. Lists are symbol structures constructed from a single relation (next, or prior) linking one symbol token with the next. Some of the symbols in a list can contain symbols that are lists. Thus, a hierarchical structure of lists can be built up.

2. Attribute-value associations. These structures contain sets of attributes and values, which are both symbols. The attributes have a unique value in each association. There are many possibilities, from no organization at all to fancy modular organization. A modular organization seems especially suitable because, as Marr [7] puts it, it can make sensitivity and stability distinctions explicit, by arranging for all constituents of a given module to lie at roughly the same level of stability and sensitivity.

We consider a hierarchical representation, from general to particular.

Coordinate system

The importance of the coordinate system used by the representation was well argued in Marr & Nishihara [6], Marr [8], Vaina [9] and Marr and Vaina [10]. Basically we have two types of coordinate systems:

(1) Viewer-centered locations are specified relative to the viewer;

(2) Object-centered locations are specified in a coordinate system defined by the viewed object.

Viewer-centered descriptions are easier to produce but not very successful because they depend on the vantage point from which they are built. Thus, one has to test distinct views on distinct objects, and this is very inconvenient, requiring a very large store in memory. An alternative to this is to use object-centered coordinate systems and to focus on the computation of a visual description which is independent of the vantage point. This involves finding the axes for the representation. We propose to have a combined hierarchical representation object, centered on the top of the hierarchy and having slots, with viewer centered information attached at the bottom. Thus, we envi-

sion an object represented by a set of views, which form the object-views-system

(OVS).

Accessing the Information:

One of the most important design questions facing a recognition system concerns how to structure the knowledge base of facts and rules so that appropriate items can be efficiently acccessed. This question has two parts - firstly, it refers to an appropriate indexing system, and secondly, it refers to the way of accessing the information. Our system is a recognition system, and thus involves two things: (1) a collection of stored representations and (2) various indices into the collection that allow a newly derived representation to be associated with one in the collection. In finding an object in the representation, first, a small set of elements is retrieved, so designed as to fit the specification for what is being sought. This is followed by a "read" matching process in which each candidate is matched against the retrieved pattern, using different mechanisms.

Indexing:

How many types of links are needed? In most AI systems for knowledge representation there is the assumption that both for retrieval and for matching, a resourcing is a single set of links among the data. This hypothesis that a single set of links is enough to handle all the tasks of the system is also basic in systems that use complete indexing, or Conniver, or Lisp. Systems like KRL [11], on the other hand, have a built in associative link system used only for retrieval. Our system is conceived as having an indexing mechanism which allows the user to catalog any item under a key.

We classify the objects represented hierarchically according to the precision of the information they carry, and the index is based on this classification. Thus, the top-most level contains the most undifferentiated representation available. Lower in the hierarchy the representations become more differentiated. Thus, we differentiate between ships and islands. When a new object is identified, it is related to a representation in the collection by starting at the top of the hierarchy. The more information is derived, the more specific we get in the representation. When, sometimes, a part of an object is recognized, then a reverse-index provides information about what the whole object is likely to be. The reverse-index plays a crucial role in our representation system, because there are many sources of identification (experts), each delivering partial information. There is another type of index, index by default , which provides specific information about the object before its representation was derived from the expert's total information. Thus, the index by default accesses hypotheses as to what the object could be, or in the event that the object should be A, then the a_1, a_2 identified would be B. This index by default is very important for speeding up the recognition of an object, before its actual recognition from the information delivered by the experts.

It may be useful to construct other indices, like cross-indices, which access information conditionally, etc., but this lies outside of the scope of this report.

The main point of this section is to suggest that we view recognition as a gradual process proceeding from the general to the specific, which constrains the derivation of the description of the object from the information delivered by the experts. The information available for establishing the correspondence between the object and the model increases as the recognition process procedes: it starts with very little and very general information, and grows with each new step.

Pattern Matching:

Usually a pattern-matching process involves a pattern describing some requirements and a datum. Descriptions can be viewed as symbolic property lists. Each property refers to a different universe of discourse. Each element in the datum expressing a property is matched with an element in the pattern expressing a property referring to the same universe of discourse. The purpose of pattern recognition is to assign a given object to a class of objects similar to it. Zadeh [12], assumes that such a class is often a fuzzy set F. A recognition algorithm, when applied to an object p, yields the grade of membership F(p) of p in a class F.

One of the most attractive ways of defining a fuzzy pattern class is to assign to each class a deformable prototype. The grade of membership of a given object in the class depends on the deformation energy necessary to make the prototype close to the object and the remaining discrepancy between the object and the deformed prototype. Kotoh and Thiramatsu [13] propose an interesting approach for the representation of fuzzy pattern classes. A feature is viewed as a fuzzy partition of the pattern space, and each member corresponds to a fuzzy value of this feature. For instance, if the possible fuzzy values of the feature "size" are "small", "medium," and "large," those values realize a fuzzy partition provided that the orthogonality condition $\mu_{small}(m(p)) + \mu_{medium}(m(p)) + \mu_{large}(m(p)) = 1, \forall p$.

A fuzzy pattern class is expressed by a logred expression of feature values which correspond to different features: for instance, the class of objects whose size is "medium", width "narrow", and weight "heavy".

Let F for example be a fuzzy pattern class defined by the fuzzy feature-values $F_1, F_2, \ldots F_r$ where F_i is a fuzzy value of feature i. An object p is thus characterized with respect to the class F by a

membership value $\mu_{Fi}(m_i(p)$ denoted $\mu_i(p)$ for simplicity. $\mu_F(p)$ is then built by aggregating the $\mu_i(p)$ in some manner. For example an interesting way would be a subjective aggregation where features are of unique importance. The choice of an aggregation depends upon the scope of the matching. There are several fuzzy pattern classes $F^1,..F^S$ and the recognition problem is to assign a given object p to a definite class.

In presenting the model, we shall consider its structural and computational hypotheses.

Structural hypotheses concern the layout or configuration of information that is prestored in the representation. They address the representational problem: how is knowledge represented?

Computational hypotheses concern processes that locate, transfer, transform, compare, or modify symbols in the structure of the representation.

The Object-Views-System: An expert-based knowledge representation system.

A basic principle of our knowledge representation system shall be the need for modularity. We represent knowledge as collections of separate, simple fragments. This idea exists already in many AI knowledge representation systems, such as frames [14], "thread memory" [14], [15], or Hendrix's system [16]. A view is a data-structure for representing an object or a situation. A view is a network of nodes and relations that is loop free. The meaning of the relation defines the relational module in the system. Thus, in the case in which the relation is "is-a", we would be in the category module; when the relation is "used-for", we are in the functional module, and when the relation is "has-is" we are in the descriptive module. The descriptive module

is the most in contact with reality. Thus, it contains information delivered by the various experts. The information in the descriptive module is stored in the form of attribute-value associations. The information in the category module and the functional module is stored in the form of lists. The common rule for each module is that the information is stored from general to particular, i.e.:

(a) In the descriptive module, the object has, first, the properties inherited via the category module, from a more general term; and only after exhausting these does it have its current properties.

(b) In the category module, we have the information stored from general to more particular, through the relation "is-a".

(c) In the functional module, we have first the more general uses and then the more specific ones. Representing uses is achieved by representing the actions in which the object can be involved. The elements of the functional modules are similar in the representation to Minsky's frames [17]. The top level represents things that are always true about the supposed object or situation. The lower levels have many slots that must be filled with the specific instances. The values given to the slots actual representation. Thus, we could say that, before the identification of the object and its function, we have "views-types", which become "views-tokens" after the identification. When reasoning is carried through the representation, planning, etc., it happens on both levels - in the type and token views level. Reasoning developed on the "view-types" informs about the general plan, or strategy. Reasoning developed on the "view-tokens" informs about what really is obtained. Reasoning on types gives "types of reasoning", like various plans, etc. Reasoning on tokens is more difficult, because it is more specific with more data and conditions. Each "reasoning-chain-token" belongs to one

or several "reasoning-chain-types." Each slot can specify the conditions that must be satisfied by its assignment. This specification is made by markers.

Collections of related "views" constitute multiple-descriptions or multiple-functions, depending on the module we are in.

The skeleton of the representation that keeps everything together is the category module. However, we may want to keep track of the relations between the elements in this module too. Thus, we shall define a relation which displays the similarity among the members of a category. As might be expected this relation will be a possibility-distribution, and, given the universe of discourse, will show how its elements are refinement, or a participation of a more general form.

The most important role of the skeleton is to provide a replacement view, when it turns out that the proposed view is not a suitable representation. This is done by a matching a process controlled by the information associated with the "view", and by the system's current goals.

Constraints

I. In general we are provided only with partial information about the object or the situation. Based on this partial information, the representation must be evoked. Thus, we must provide a means to reconstruct the object from partial information about it.

II. The process of recognition is goal oriented; thus, the system has several procedures, one (or a class) of which is considered the current goal of the system. This is used to decide which terminals and conditions must be made to match reality.

III. Some of the terminals need "adjustment". This means that they cannot retain the default information and must have new values assigned based on current information.

IV. Redundant information is used in the representation system to trade off memory space for computation depth. The choice for where to put redundancy influences the structure of the representation and influences the mode of search and deduction.

Remark

By providing this modular representation, we avoid the declarative vs. procedural dilemma for the data structure. However, this question would come up only when we would be closer to the programming level.

Principles for effective communciation

We have seen that the data base in the descriptional module is provided by the experts. Our representation relies very much on the expert's expertise in detecting and transmitting some specific information. The experts are supposed to furnish information with some degree of approximation or tolerance. This degree depends upon their expertise, or the need for precision in the information. There are a few principles [18] which the experts should follow in order to be efficient and reliable informants, as follows:

(1) MANNER: Be perspicuous

1a: avoid obscurity of expression
1b: avoid ambiguity
1c: be brief
1d: be orderly

This can be rephrased as instructing speakers and addressees to use, and interpret each other as using the same language. This means, for us, that the various experts' messages must be expressed in a language compatible with the knowledge representation language. When ambiguous expressions are used, they should be treated as having just one m,eaning and not different meanings. By "be orderly", different things are meant in the temporal domain and in the spatial domain.

RELEVANCE: be relevant

QUALITY: try to make your contribution one that is true.

(a) do not say what you believe is false
(b) do not say that for which you lack adequate evidence.

A simplified version of this principle would be the following:

(2) QUALITY: say only that which you know is most suitable for the question you want to answer.

(3) QUANTITY: (a) make your contribution as informative as is required for the current purposes.
(b) do not make your contribution more informative than is required.

These principles, called "maxims for conversation" [19] shall be elaborated more in detail in a future report where we plan to consider the contribution of experts and their interaction.

How to deal with so many sources of knowledge?

The descriptional module contains information delivered by various experts - humans or machines. How is this information to be stored

the most usefully? In general, that there are two types of knowledge representation models: network and set-theoretic models. Network models, exemplified by the work of Collins and Quillian [20], Rumelhart [21], Lindsey [22], Norman [23], Hendrix [16], Vaina and Greenblatt [14], Vaina [15], Fahlman [24], etc. assume that words and their conceptual counterparts exist as independent units in the representations connected in a network by labelled relations. In the set theoretic-models, concepts are represented by sets of elements. We make the assumption that the features associated with a given category vary in the extent to which they define the category. This helps us to keep the number of features manageable, and also to operate with some kind of "basic features." We consider as basic features, the features which are relevant to the recognition or identification of the seen object.

However, we can consider a variation in the defining quality of the features; some characterize the object more relevantly than others. These features are not totally independent. Thus, if the value of the size of the object is small and the value of location with respect to the perceiver very far, then the value of size is kept as questionable until more information is obtained; or until the proportion between the values of location and size can be computed. Thus, we shall consider the description module as a set theoretical representations of knowledge. In fact, it forms a matrix; the columns consituting the features, and the row, the experts. Thus, each element in the matrix is a value associated by an expert with a feature. How do we take into account all these values? We propose to consider the name of the feature. Are the name of the variable in the possibility distribution theory and the values associated by the experts or of the possibility distribution?

Now, if X is a feature, taking values in a universe of discourse U, then by the possibility distribution of X, denoted by πX, is meant the

fuzzy set of possible values of X, with the possibility distribution function πX: U $\longrightarrow$ [0,1] defining the possibility that X can assume a value u. Thus,

$$x\ (u) = \text{Poss}\ [X = u].$$

with $\pi X(u)$ taking values in the interval [0,1].

To connect the possibility distribution to fuzzy sets, Zadeh formulates the following postulate [25]:

Possibility Postulate : In the absence of any information about X other than that conveyed by the proposition,

$$p = X \text{ is } F$$

the possibility distribution of X is given by the possibility assignment equation

$$\pi_X = F.$$

This equation implies that

$$\pi_X(u) = \mu_F(u)$$

where F(u) is the grade of membership of u in F, i.e., the degree to which u fits one's subjective perception of F. In the case of a proportion $p = N \text{ is } F$, we associate a possibility assignment equation $\pi_X = F$, where X is a variable that is explicit or implicit in N. Thus, we have

$$N \text{ is } F \longrightarrow \pi_X = F$$

When X is implicit in N, then the possibility assignment equation first identifies X, and second characterizes its possibility distribution. Thus: p = "Enterprise is a big ship." X might be

$$X = \text{Size(ship(Enterprise))}.$$

and then the possibility assignment equation

$$\pi\text{size(ship(Enterprise))} = \text{BIG}.$$

This formalism is interesting because, aside from the fact that we can cope with vagueness, it enables us to establish the referent and the value of the features with respect to the referent. There, the referent is "ship", which is rather general. In the category module we see that a subclass of ships could be "fishing boat", and we see that the value of the feature could change. Thus, we would have,

$$X = \text{size(fishing boat(Enterprise))}.$$

$$\text{size(fishing boat(Enterprise))} = \text{very, very big}.$$

In the case of features whose values depend on each other like the example with size and location, we can use a conditional possibility distribution [26]. Thus, if X and Y are variables taking values in U and V, respectively, then the conditional posssibility distribution of Y given X is induced by a proposition of the form "If X is F then Y is G," and it is expressed as $\delta(Y/X)$,

$$\pi_{(Y/X)}\,(v/u) = \text{Poss}\{Y = v \mid X = u\},$$

where we express the conditional possibility distribution function of Y given X. X is considered to a more defining, basic, or principal feature than Y. If the distribution function of X is known, as well as the conditional distribution of Y given X, then we can build the joint distribution function of X and Y by:

$$\pi(X,Y)(u,v) = \pi_X(u) \wedge \pi(Y/X)(v/u).$$

Efficiency:

We have seen earlier, that the principle of quantity requires us to make the information as detailed as needed. Thus, if the goal is to detect ships, the experts need not convey detailed information about islands, or restaurants on islands. But, the information required varies with the current goals of the system. How could this be handled?

Following Vaina [27], we propose the following: Let $A = A_1, A_2$ be a set of terms labelling such features as size, location, height, etc. Every $A_i, \epsilon i\ (1,h)$ is represented by a set of terms $A_i = \overset{m}{\underset{j=1}{z}} A_i\ j \epsilon$ $(1,m)$ and A_{ij} takes values in a universe of metadiscourse labelled by ij, $Vij = \overset{z}{i}\ \overset{z}{j}\ Aij$. By the universe of discourse we take here the real world data, the domain of values of the features as they are associated by experts. We will call the real world information OL-1 (object language 1), and the set of features OL-2. The metalevel ML is the common metalanguage for OL-1 and OL-2. Thus, if $Ai = Size = \overset{m}{\underset{j=1}{z}} Aij$ = {big, small, medium, not too big, not too small, etc.}, and the universe of discourse in OL-1 would be the interval [0, 1000] feet. The metalanguage mediates the relation between OL-2 that is the set of characteristic features used by the system and the experts to characterize the object, and OL-1 the real world data perceived by the experts. We consider that the "tolerances" used by experts are sensible.

In general, the goal of the system selects a subset of OL-2 that is relevant. The experts associate values for each feature required. Thus, the input to the system would be a possibility distribution associated as a value with each feature.

Sometimes, the goal - as in a problem solving situation - is to select those real-world objects which have some desired value of the relevant features. For example, if the feature one looks at is size, and the

question is to select those objects in the real world whose size is big, then the experts, after evaluating the object from the point of view of size, have to express to what degree it can be considered "big." This computation is carried out on the level of metalanguage ML, where the system attributes a desired value to each feature, and the expert shows to what degree the object satisfies the requirement when perceived and evaluated through the expert's expertise.

Functional Module:

The "views" in the functional module represent objects categorized by their stereotypical uses. They contain slots, which can be filled with other expression fillers which may themselves be "views." Thus, we might have a "view" representing a typical "aircraft carrier" with slots like "airplanes", "owner", "location", etc. A particular "aircraft carrier" is to be represented by an instance of this "view," obtained by filling in the slots with specific information. Thus, the top level contains general information about "aircraft-carriers", and then, the lower levels are specific to the extent that a particular ship represented is an "aircraft carrier." The system gets this particular information from the experts, who perceive the real-world information and process it using their expertise. The name "aircraft-carrier," for example, is in the category module and a pointer goes from it to the specific views in the "functional module." Thus, what one knows to start with is that the "aircraft-carrier" is a kind of ship. Then to find more information about what this means from the functional or use point of view, one goes to the related "views-system" in the function module, and to the information in the descriptional module, looking for what the experts know in their representation about aircraft-carriers. Then all this knowledge is made active, and the system might try either to identify the "aircraft-carriers" in the real world, or to answer questions about them, etc.

Another way of looking at the views in the functional module is as bundles of properties. Thus, an aircraft-carrier can be represented as x(airplanes(x_1,Y_3) of.... where the free variables Y_1, Y_2... correspond to the slots. The slots' information is represented in the descriptional module in the expert's universe. We have seen how we deal with knowledge represented by more than one expert. "x" is the identifier of the -expression, it gets bound to the variables Y_2 expressing knowledge about a specific object. Thus, for example x = Entreprise.

Towards a Computational Theory of Image Understanding

We consider understanding images to be a two-stage process. Image analysis extracts features from the raw intensity values recorded in an image and converts these features into a convenient symbolic representation. Scene analysis interprets the symbolic features produced by image analysis according to some externally-defined goal.

Image analysis defines what can be considered as forced by the data in the subsequent interpretation. Scene analysis, on the other hand, is an exercise in problem solving. At this stage one is free to invoke whatever prior knowledge is available to aid in image interpretation.

In early artificial intelligence research, there was assumed to be a sharp line between image analysis and scene analysis. The purpose of image analysis was to generate a two-dimensional line drawing of the scene (Binford and Horn [28]. The purpose of scene analysis was to interpret these two-dimensional line drawings in terms of the three-dimentional objects which motivated them (Roberts [29], Winston [30]).

As the field matured the information leading from image analysis to scene analysis became stronger (Falk [31]), leading to a substantial reduction in the computation required in image analysis. Recent

work (Marr [8]) has shown that there is a great deal of information about three-dimensional shapes contained in image intensities and that this information can be computed without the help of higher-level knowledge.

Image Analysis is a hard problem

A. Data Compression

One of the major goals of image analysis is to extract features of intensity that are important and to throw everything else away. In other words, one of the first challenging steps in the image analysis endeavor is to choose a useful method of data compression. The features one wants to use in a data compression process are those which can be conveniently defined in terms of properties of images. The image properties need to be simple to compute.

B. Loss of Information

Data compression is a means of choosing useful information and neglecting the rest. But, because images are defined in two dimensions, while objects exist in a three-dimensional world, information is lost in the projection of a three-dimensional object into a two-dimensional image. The mapping from object-space to image-space is many to one. It is not conceivable to analyze two-dimensional images without some specific assumptions about the three-dimensional nature of the objects that motivate the image. A particular object feature may appear quite differently depending on the direction of viewing. As Marr [8] puts it, images have a viewer-centered representation, and this can become a tedious and frustrating problem for recognition of objects. Moreover, projection introduces two-dimensional image features which have no direct correlation with any three-dimensional object property. For

example, neighboring points in an image do not necessarily correspond to neighboring points on objects.

Ways of Avoiding the Difficulties

A. Principle of the Least Commitment:

In most real world situations, it is advisable to keep generality for as long as possible. In other words, one does not want to decide too soon upon the values of primitives, or their relevance to anything in general - the decisions are made when more information is available, when the trade-off between what is desired and what it is possible to compute is established. A too early commitment does not ensure the way to the solution, but rather a narrows down of the possibilities.

B. Organization of Information

One should be able to classify objects into broad categories, and also discriminate in detail categories as needed. Thus, the most general discrimination would be between ships and islands, for example, but the recognition system must be able to distinquish types of ships, like combatant ships from transporation, for example, and moreover, one should be able to classify within types (such as destroyers or aircraft carriers).

An important requirement is to identify the nationality of ships, or their function - do they belong to the enemy, or to one of the allies? Are they used for intelligence, or for destroying? At the same time we do not want to get more detail than is needed. Thus, for example, once an island is identified, if the target is a ship, one should not pursue further the information about the island. This leads to the idea that information must be accessed from general to particular.

3.0 THE SEMANTICS OF TRIVALENT QUANTIFIERS

A key feature of our knowledge-representation language is the use of quantifiers in its sentences. In particular, we include both standard bivalent quantifiers and non-standard trivalent ones; the latter allow a third "truth value" other than true or false, arising from the use of dynamic semantic interpretation. In this report, we illustrate the semantics of these quantifiers by giving a sketch of a language which contains them. This language can be viewed as defining part of the semantic requirements on the knowledge-representation language itself, i.e., those requirements imposed on the language by its use of trivalent quantifiers.

1. Basic Symbols

We distinguish four kinds of symbols in our language: constants, variables, predicates, and quantifiers. Constants serve as proper names; each constant denotes one specific individual object. Variables act like proper names whose specific reference can change; a single variable can denote different individual objects, depending on the requirements of the context. Predicates serve to identify classes of objects and relationships among such classes; the number of classes involved in such a relationship is reflected as the number of argument places in the predicate. Quantifiers serve to relate predicates, specifically in terms of their reference; in general a quantifier will focus on one class of predicates and express a relationship between it and another such class, the two classes being separated by a semicolon [32].

The symbols in (1), for example, are constants; each denotes

(1) PT109, U.S.S. Enterprise

exactly one object, the PT109 and the U.S.S. Enterprise, respectively. The symbols in (2) are variables; either could denote either of these

(2) x,y

objects depending upon the context. The symbols in (3) are predicates; the first three are one argument predicates and thus

(3) Submarine( ), Carrier( ), Threatened ( ), Under(,)

denote classes of lists of length 1, namely, the classes of submarines, of carriers, and of threatened things, respectively; the fourth is a two-argument predicate and thus denotes a class of lists of length 2, namely, the class of pairs of objects such that the first is under the second. The symbols in (4) are quantifiers; the first two are standard symbols in logic, meaning "whatever" and "there is" or

(4) ∀, ∃, All, Only, Many, The

"there are", respectively; the others mean essentially what they mean in English.

2. Formulas

Formulas of the language are formed by combining quantifiers, predicates, variables, and (perhaps) constants in meaningful ways. Formula (5), for example says that whatever is a submarine is

(5) (∀x)(Submarine(x); Threatened(x))

threatened. Formula (6) says that all submarines are threatened; it

(6) (All x)(Submarine(x); Threatened(x))

differs from (5) in assuming that there are, in fact, submarines in the field of interest and thus that the asserted threat is actual, and not merely potential. Formula (7) says that there are (one or more)

(7) (∃x)(Submarine(x); Under(x,PT109))

submarines under the PT109; we could say that there is exactly one by using a different quantifier. Formula (8) says that the submarine

(8) (The x)(Submarine(x), Under(x,PT109); Threatened(x))

under the PT109 is threatened; it assumes, as part of the meaning of "The" that there is, in fact, exactly one such submarine.

The role of the variable "x" in each case is to relate the quantifier to the predicate and, in particular, to the appropriate argument place in the predicate; for example, if the PT109 were itself a submarine, then formula (9) would make just as much sense as formula

(9) (The x)(Submarine(x), Under(PT109,x); Threatened (x))

(8), because it says that the submarine the PT109 is under is threatened. More complicated meanings can be expressed by introducing additional variables for additional quantifiers; formula (10), for

(10) (The x)(Submarine(x),(The y)(Carrier(y),Under(x,y);Threatened(x)))

example, says that the submarine under the carrier is threatened, while formula (11) says that there is a threatened submarine under the

(11) (∃x)(Submarine(x),(The y)(Carrier(y),Under(x,y);Threatened(x)))

carrier, and formula (12) says that there is a submarine under the

(12) $(\exists x)$(Submarine(x),(The y)(Carrier(y),Under(x,y);Threatened(y)))

threatened carrier.

3. Models and Satisfaction

We take as models of our language the set of pairs $\langle D,R\rangle$, where D is the set of objects on or in the ocean and R is a function that assigns members of D to constants and lists of members of D to predicates, the length of a list being equal to the number of argument places in the predicate. A model, in other words, is a possible state of the ocean, as derived, for example, from a radar scan. Given a model M, we say that a formula is true in M or satisfied in M, written $M \models (A)$, if it is true or satisfied in M no matter what values are assigned to free variables, i.e., variables not associated with ("bound by") a quantifier. The latter notion, i.e., satisfied by M given an assignment of values to variables, is defined by rules like (13)-(18), where f is a function that assigns members of D to variables,

(13) $M \models (x_1 = x_2)[f]$ iff (i.e., if and only if) $f(x_1) = f(x_2)$;

(14) $M \models (P(x_1,\ldots,x_n))[f]$ iff $(f(x_1),\ldots,f(x_n)) \in R(P)$;

(15) $M \models (A\&B)\ [f]$ iff $M \models (A)[f]$ and $M \models (B)[f]$;

(16) $M \models (\neg A)\ [f]$ iff it is not the case that $M \models (A)[f]$;

(17) $M \models ((\forall x)A)[f]$ iff $M \models (A)\ [f']$ for whatever assignments f' for M are like f except perhaps (i.e., at most) at x;

(18) $\underline{M} \models ((\forall x)(B;A))\ [f]$ iff $\underline{M} \models (\underline{A})\ [\underline{f}']$ for whatever assignments $\underline{f}'$ for $\underline{M}$ are like $\underline{f}$ except perhaps at $\underline{x}$ for which $\underline{M} \models (\underline{B})\ [f']$

and "$\underline{M} \models (\underline{A})\ [\underline{f}]$" is read *f satisfies A in M* or *M satisfies A given f* [33].

B and A in (19) are lists of formulas, the *relativization*

(19) $(\forall x)\ (B;A)$

formulas and the *principal formulas*, respectively, of (19). Intuitively, $M \models (A)$ can be read as saying that formula A is true of the state of the ocean that is represented by M. Formula (5), for example, is true of a given state of the ocean if whatever submarines are present in that state are, in fact, threatened.

Given rules (15) and (16), we can define "$B \supset A$", read *if B then A*, as "$\neg(B \ \& \neg A)$"; in other words, "if B not A" is true in a model if and only if it is not the case that B is true and A is not.

Given this definition, formula (5), for example, is logically equivalent to formula (20), which contains only the single complex

(20) $(\forall x)(\text{Submarine}(x) \supset \text{Threatened}(x))$

predicate "(Submarine⊃Threatened)()", rather than the two predicates "Submarine()" and "Threatened()". In general, however, it is not possible to reduce a two-predicate formula to a one-predicate one, for quantifiers other than "∀" [32].

4. Dynamic Rule Application

Given rules (16)-(18), we can define "$\exists x$" as "$\neg(\forall x)\neg$", the rightmost "$\neg$" applying only to the rightmost predicate in "$(\forall x)(B;A)$". This enables us to talk about the satisfaction of formulas with "$\exists$", without having to give an explicit satisfaction rule for them. We can also define satisfaction for quantifiers like "All", "Only," and "The" without giving explicit satisfaction rules for them, by reading rule (18) in a non-standard way.

Rules like (13)-(18) are standardly viewed as applying "statically", i.e., as passively describing a state of affairs, a relationship that obtains between a formula and a model, but they can also be viewed as applying "dynamically", i.e., as actively assigning to formulas the values of a feature [+ satisfied]. On the static view, such rules are inherently bivalent, because of the "iff" that occurs in their formulation: either $M \models (A)$ [f] or not, depending on the content of the relevant rule. The dynamic view, however, allows for a third "truth value", because of the possibility that a rule gets stopped in the course of its execution, before it gets to assign an appropriate value. We can say that A is true (in M (given f)), if A is assigned the value [+ satisfied] by the relevant rule; false, if it is assigned the value [- satisfied]; and neither (true nor false), if the rule gets stopped for some reason before it can make an assignment. Given a rule like (4), a formula and its negation will be assigned opposite [satisfaction] values, assuming that they get assigned at all. In the event that a rule stops, neither the formula nor its negation gets assigned a value, and neither of them is true. Satisfaction simply is not an issue for such a formula or its negation in M (given f).

In particular, we assume that (18) applies by testing (f')s first against B in (19) and then against A and that tests of both kinds must actually have taken place in order for the rule application to have been successfully carried through to completion. If (18)

finds assignments f' that satisfy B in M and then finds that each of those satisfies A, then it assigns the feature value [+ satisfied (by f in M)] to (19). If it finds (f')s that satisfy B but then finds that some of those fail to satisfy A, then it assigns the feature value [-satisfied] to (19). If it fails to find any (f')s at all that satisfy B, and so has nothing to test against A, then it simply grinds to a halt without rendering any judgement as to the relation between (19) and [+ satisfied]. In the first case, (19) is true under f in M of the state of the ocean that is represented by M. In the second case, its negation is true of that state of the ocean: (19) itself is false. In the third case, (19) is simply irrelevant to that state of the ocean: neither it nor its negation is true, because satisfaction is not an issue for them for the model and assignment in question.

To get definitions for "All", "Only", and "The", we consider the case in which B in (19) contains no free variables. If B does not contain the variable of quantification (i.e., x) free, but does contain other free variables, then an assignment f will assign values to those variables, and every f' that differs from f at most at x will assign exactly the same values to those variables, since f' also differs from every other f' at most at x. It follows that B is satisfied by every f' or by none according as B is satisfied or not by f. Since f' differs from f at most at x, in other words, f' is f, as far as B is concerned, because B does not contain x and so is blind to the difference between f and f'. Under our dynamic reading of (18), however, whether or not there is an f' that satisfies B is what determines whether or not satisfaction by f is an issue for (19), so satisfaction by f will be an issue for (19) according as f itself does or does not satisfy B. This is because f itself is an f', differing from itself at most at x, just as every other f' differs from f at most at x.

If B not only lacks the variable of quantification, but is entirely devoid of free variables, then even f itself is irrelevant to the

satisfaction of (19) by $\underline{f}$. If $\underline{B}$ contains no free variables at all, then it is blind to the differences not only among the various ($\underline{f}'$)s, but also among the various ($\underline{f}$)s, before we even get around to sorting them into ($\underline{f}'$)s. This means that the satisfaction of $\underline{B}$ in $\underline{M}$ depends not on any assignment, but only on the model itself, and that this is the case also, therefore, for whether satisfaction is an issue for (19).

It follows that we can get the correct meanings for "All" and "Only", for example, by defining them as in (21) and (22), in which $\underline{B}$ is stipulated to contain no

(21) (All $\underline{x}$)($\underline{B}$;$\underline{A}$) = ($\forall\underline{x}$)(($\exists\underline{x}$)$\underline{B}$, $\underline{B}$; $\underline{A}$)

(22) (Only $\underline{x}$)($\underline{B}$;$\underline{A}$) = ($\forall\underline{x}$)(($\exists\underline{x}$)($\underline{B}$;$\underline{A}$),$\underline{A}$;$\underline{B}$)

free occurences of $\underline{x}$. For example, by these definitions, formulas (23) and (24), respectively, are equivalent to formulas (25) and (26)

(23) (All x)(Submarine(x); Threatened(x))

(24) (Only x)(Submarine(x); Threatened(x))

(25) ($\forall$x)(($\exists$x)Submarine(x), Submarine(x); Threatened(x))

(26) ($\forall$x)(($\exists$x)(Submarine(x); Threatened(x)), Threatened(x); Submarine(x))

because -- using (27) for (25) and (28) for (26) --, these formulas get

(27) (a) ($\exists$x) Submarine(x)
(b) ($\forall$x)(Submarine(x); Threatened(x))

(28) (a) $(\exists x)$(Submarine(x); Threatened(x))
(b) $(\forall x)$(Threatened(x); Submarine(x))

assigned [+ satisfied], if both (a) and (b) are true; [- satisfied], if (a) is true but (b) is not; and neither, if (a) is not true [34].

(29) $(\not\forall x)$(Submarine(x); Threatened(x))

In contrast to formula (29), for example, which is true even if there are no submarines on the ocean at all, formula (23) is true only if there are, in fact, such submarines. Similarly, formula (24) is true only if there are in fact, submarines that are threatened. The significance of this point becomes clearer in connection with formula (30), which also requires the existence of threatened submarines, to be

(30) $\neg$(Only x)(Submarine(x); Threatened(x))

true according to (26). Formula (30) tells us both that submarines are threatened and that other things are threatened as well, without specifying what these latter things are.

5. Model Selection

The semantic effect of a list of formulas in our language is to narrow down the class of possible states of the ocean (i.e., models) to the intersection of the classes of those that satisfy each formula individually. This contrasts with more standard accounts that view formulas as building up a model that contains the objects mentioned in the formulas [35]. An advantage of our account is that it leaves all options open except those it is explicitly desired to rule out. Given a report consisting of a sequence of formulas like the one in (31), for example, a constructive interpretation would conclude

(31) $(\exists x)$ Carrier (x)
$(\exists x)$(Submarine(x); (The y)(Carrier(y)); Under(x,y)

that there were no destroyers in the area because destroyers are not mentioned in the report. Our selective interpretation would leave this question open because the presence of destroyers has not been ruled out in the report. This seems clearly to be the safer option.

Our account also provides a double check on consistency through its use of trivalent quantifiers. Suppose that there are sentences $\underline{s}_j$ and $\underline{s}_k$ in a report such that $j<k$ and such that the formula that represents the meaning expressed by the negation $\neg\underline{s}_j$ of $\underline{s}_j$ occurs as a relativization formula--without free variables--of the formula that represents the meaning expressed by $\underline{s}_k$. Since $\underline{s}_j$ rules out all models that satisfy $\neg\underline{s}_j$, the relativization formula of $\underline{s}_k$ is not satisfied, so satisfaction is not an issue for $\underline{s}_k$. In the report suggested in (32), for example, a warning will result from $\underline{s}_k$, because all models in

.
.
.

(32) $\underline{s}_j$: $\neg(\exists x)$(Submarine(x); Threatened(x))

.
.
.

$\underline{s}_k$: (Only)(Submarine(x); Threatened(x))
[=$(\forall x)((\exists x)$(Submarine(x); Threatened(x)), Threatened(x); Submarine (x)}] [=26]

(rejected by s_j)

.
.
.

which it can be either true or false have already been rejected by s_j.

6. Control Maps and Fuzzification

This framework can be refined in various ways, in particular, by replacing predicates with a functional notation and allowing the resulting functions to have internal structure [36]. Once this move is made, the way is open to the use of HOS control maps for representing this structure in a systematic way and to the fuzzification of the language through the use of linguistic variables. Developing these two options is the focus of currently on-going work.

4.0 IMAGE ANALYSIS

An image is a two-dimensional array of intensity values. The first goal in analyzing an image is to describe the significant intensity changes. A change in intensity is usually the result of change in illumination, or a discontinuity in the depth or orientation of a surface, such as that which occurs along the boundary between two objects separated in depth. Many times real world objects impose on the intensity changes a variety of spatial organizations, reflected in the structure of the image. These structures are captured by a set of tokens that correspond to oriented edge or boundary segments. Thus, we have:

(1) bars: parallel edge pairs
(2) blobs: closed contours which can be defined from a cloud of tokens, for example.

A given intensity change allows for computing the two-dimensional orientation in the image, the width, i.e., the distance across which the intensity is changing, and the length, i.e., the distance along the orientation of the intensity change over which other properties remain roughly uniform.

The physical changes that are the direct motivation of changes in intensity are edges. But, unfortunately, there may not exist a one-to-one correspondence between intensity changes detected at a particular scale, and edges in the physical world. Thus, we need a method for edge detection.

The Intensity Function As Edge Detector

A change in intensity is the phenomenon that we detect and describe in an image; edges are the physical changes that motivate the changes in intensity. To deal with intensity changes, we have to find a function

to apply to the image which will allow us to extract the changes. This function has to be able to deal with uniform changes as well as with abrupt changes. Also, changes can take place on different scales, for example, there are local changes and macro-changes. The local changes take place over short distances; the macro-changes are gross, general changes in the image. Thus, we can formulate two requirements in the formal definition of the intensity function:

(1) the intensity change incorporates the scale at which the change occurs.
(2) Since changes in the physical world are taking place in space, the function has to be spatially localized.

The main goal of the intensity function is to localize the discontinuity in the image intensity. This is achieved by finding peaks in the first directional derivative of intensity, or equivalently, zero-crossings in the second directional derivative (a zero-crossing is a place where the value passes from positive to negative). Zero-crossing provides a natural way of moving from a continuous "analog" representation, such as two-dimensional intensity values, I(x,y) to a discrete "symbolic" representation without loss of information. (Logan, thereom [37]). But it is impractical to consider the orientation, so we should try to find a function which is not a directional operation. The only nondirectional linear second-derivative operator is the Laplacian operator. The equivalent of the zero-crossing of the second directional derivative taken perpendicular to an edge coincides with the zero-crossings of the Laplacian along that edge. (Marr and Hildreth [38]). Thus, we can detect intensity changes occuring at all orientations using the non-oriented location operator.

Intensity Function

Remarks:

The descriptions obtained from a single channel are sets of zero-crossing contours (blobs, bars) with symbolic descriptions of orientation, slope, and size, attached to the segments of the contours or to the entire contour. By size, we understand the length and width of the contours. Primitive tokens can be combined to form larger scale tokens that reflect larger scale structure in the image. Thus, at some intermediate level we group together the tokens with the same orientation, and the difference in orientation obviously would determine the representation. One would expect that in one of the large objects, the degree of organization of the 'images' is not very high at the smaller scale, and become more interesting at the larger scale.

Marr and Hildreth [38] consider three main steps in the detection of zero-crossing,

(1) a convolution with D^2G where D^2 is the second directional derivative operator, and G is the Gaussian.
(2) the localization of zero-crossing, the location of the value zero of D^2G.
(3) checking the alignment and orientation of a local segment of zero-crossings.

When certain conditions are satisfied these steps might be made more economical. A zero-crossing segment in a Gaussian filtered image is a linear segment l of zero-crossings in the second-directional derivative operator whose direction is perpendicular to l.

The amplitude associated with a zero-crossing segment is the slope of the directional derivative token perpedicular to the segment.

The set of zero-crossing segments and their amplitudes constitutes a primitive symbolic representation of the changes in a region of image's spectrum. To cover the whole spectum requires the reapplication of the analysis over a sufficient number of channels. Logan (1977) [37], (quoted from [39]) shows that if a one-dimensional analytic function is (a) bandpass of bandwidth one octave less, and (b) has no free zeroes, i.e., complex zeroes in common with its Hilbert transform, then the function is completely determined (up to an overall multiplicative constant) by its zero-crossings.

Thus, this theorem says that if condition (a) is satisfied then the zero-crossing completely determines the convolution values.

Constraint: For a reliable representation of an image based on zero-crossings, which allows recovery of sharp intensity change, we have to filter it through a set of independent bandpass channels with one octave bandwidth. Thus, the masics that approximate the second directional derivative operator, following Logan's result, should be a bandpass with one octave bandwith.

Improving efficiency: Orientation is an important property of an edge, but it is more efficient and less costly to compute it. Often the convolution of an image with the nonoriented Laplacion operator was done, and a map of intensity changes at all orientations was obtained.

Thus, the Gaussion distribution satisfies the localization, space and frequency properties, and the Laplacian satisfies the property of having one orientation dependency.

Conditions on the intensity function.

1. Condition of Linear Variation: The intensity function near and parallel to the line of zero-crossing should be locally linear.

2. Condition of Linearity: The intensity function should be linear along, but not necessarily near, the line of zero-crossing.

If these two conditions are satisfied, then an intensity change at any orientation will coincide with a line of zero-crossings in the output of the Laplacian, along the orientation of the change.

Intensity function: We have seen that the intensity function has to satisfy two basic requirements. These two requirements are conflicting, and Barcewell [40] relates them by the uncertainty principle, which says $\Delta x \, \Delta\omega \geq 1/4\pi$, where Δx is the spatial variance, and $\Delta\omega$ is the frequency variance. The Gaussian is the only distribution [N] which optimizes this relation:

(1) $G(x) = 1/\sigma 2 \exp(-x^2/2\sigma^2)$,

with Fourier transform

(2) $G(\omega) = \exp(-\sigma 2\omega^2)$

where σ is the number of picture elements. In two dimensions,

$G(x,y) = \sigma 2\exp(-r^2/2\sigma^2)$

where $r^2 = x^2 + y^2$, the distance from the center of the mask. Then the Laplacian of the Gaussian distribution is

$$^2 G = \delta 2G/\delta x^2 + \delta^2 G/\delta y^2$$

$$= [2 - r^2/\sigma^2] \exp(-r^2/2\sigma^2)$$

then, the intensity changes may be detected by looking for the zero values in the convolution $\nabla^2 G*I$. Where I is the image (I(x,y)) and '*' is the convolution operator. Thus, we seek the zero crossing in

$f(x,y) = \nabla^2(G * I(x,y))$, and by the derivative rule for convolutions

$f(x,y) = \nabla^2 G * I\ (x,y)$.

Thus, in order to detect intensity changes, the following steps are involved:

(1) the image $I(x,y)$ is convolved with the two-dimensional Gaussian operator.

(2) if the condition of linear variation holds, intensity changes in $G * I$ are characterized by zero crossings in the orientation-independent differential operator, the Laplacian obtained by searching for zero values in the convolution $\nabla^2 G * I$.

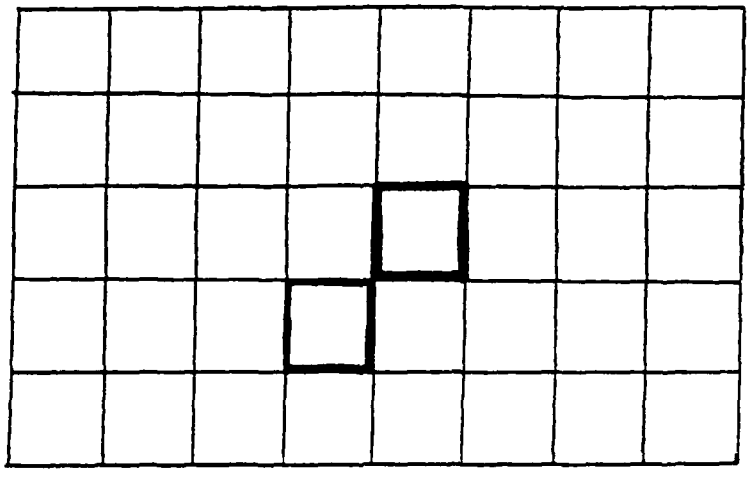
5.0 AREAL COORDINATE SYSTEMS FOR PLANE DECOMPOSITION

Image processing tends standardly to be based on decomposition of the plane into narrow bands that support the representation of images by distinguishing edges and differential intensities of various planar regions. In this report we examine two alternative methods of decomposing the plane that replace bands with squares and hexagons. Using regular figures like squares and hexagons maximizes the uniformity of a representation by supporting a natural arithmetic encoding of those figures, their higher-level aggregates, and their geometrical relationships, such as vector addition, rotation, and distortion. In particular, we will compare the areal coordinate systems that result from square- and hexagon-coverings of the plane with respect to a number of criteria that appear to be relevant to the representation of image-derived information.

1. "Slippage"

A covering of the plane with squares allows for "slippage" in that the correspondence between adjacent rows or columns of squares can be entirely arbitrary. In Figure 1a, for example, the squares are arranged in a uniform grid pattern, while in Figures 1b and 1c, successive rows are displaced with respect to each other to a greater or lesser degree. In Figure 1b, this displacement is uniform in that each row of squares is displaced by exactly one half of a side length, while in Figure 1c, the displacement looks quite arbitrary, leading to an apparently very complex pattern of square arrangement. It follows from this fact that a specification of side length and location alone of one square is far from sufficient to determine the general pattern of a plane covering. If we assume uniform (or null) displacement of successive rows, then a specification of side length and location of two non-co-row or columnar squares suffices to determine the entire covering, as shown in Figures 1a and b. Without such an assumption,

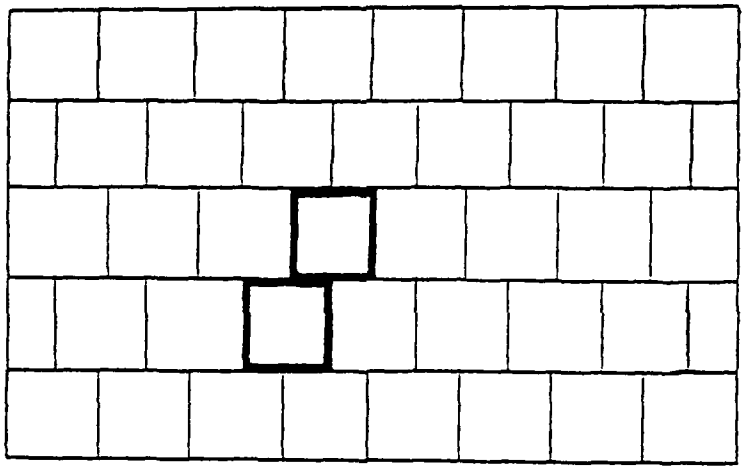
(a)



$d = 0$

null
displacement

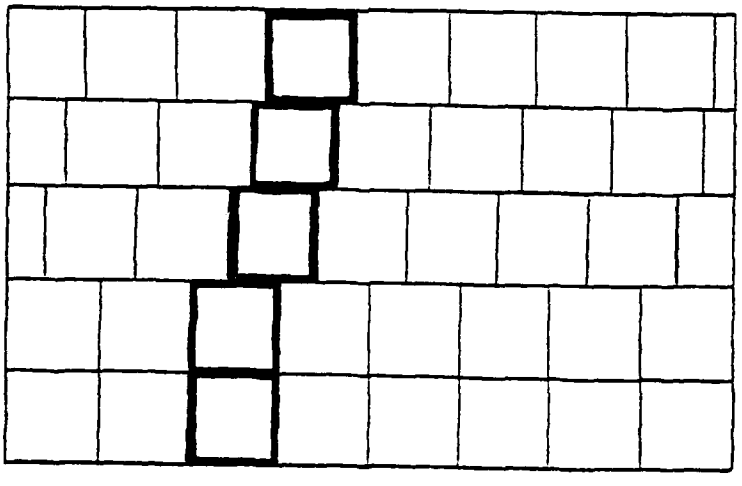
(b)



$d = \frac{s}{2}$

uniform
displacement

(c)



$d = \frac{s}{2^{n-1}}$

more complex
displacement

Figure 1: Horizontal Slippage in Square-Coverings of the Plane

however, nothing can be determined about the pattern of the covering without a specification of the location of one square in each row. The displacement of successive rows in Figure 1c, for example, is given by $d = s/2^{n-1}$, where s is the length of a side, n is the number of a row above the bottom one, and d is the fraction of side length a row is displaced to the right. Clearly, more complex patterns can easily be generated by such slippage, leading to abstractly interesting, but less and less useful and practical square coverings of the plane. As shown in Figure 2, this is not a problem in hexagon-coverings of the plane. Once a single hexagon is located in the plane, the entire rest of the covering is thereby also determined. All that needs to be specified to characterize a hexagon-covering, in other words, is the side length and location of one hexagon.

2. Uniform Adjacency

Square-coverings of the plane have non-uniform adjacency, as shown in Figure 3. In the case of null displacement, some squares meet on a side and some meet at a point, as shown in Figure 3a. In the case of uniform displacement, all squares meet on sides, but some meet on a full side, while others meet on portions of a side, as shown in Figure 3b. If more complex displacement is allowed, then adjacency is even more non-uniform, as shown in Figure 3c, since even these portions are of different lengths in the various meetings because of the differences in displacement from row to row. As shown in Figure 4, this is not a problem in hexagon-coverings of the plane, because each hexagon meets each of its neighbors on a full side and shares exactly one sixth of its boundary with each neighbor. Adjacency is strictly uniform, in other words, in hexagon-coverings of the plane.

3. Uniform Hierarchical Structure

Square-coverings of the plane are uniform in hierarchical structure only in the unique case of null displacement. As shown in Figure

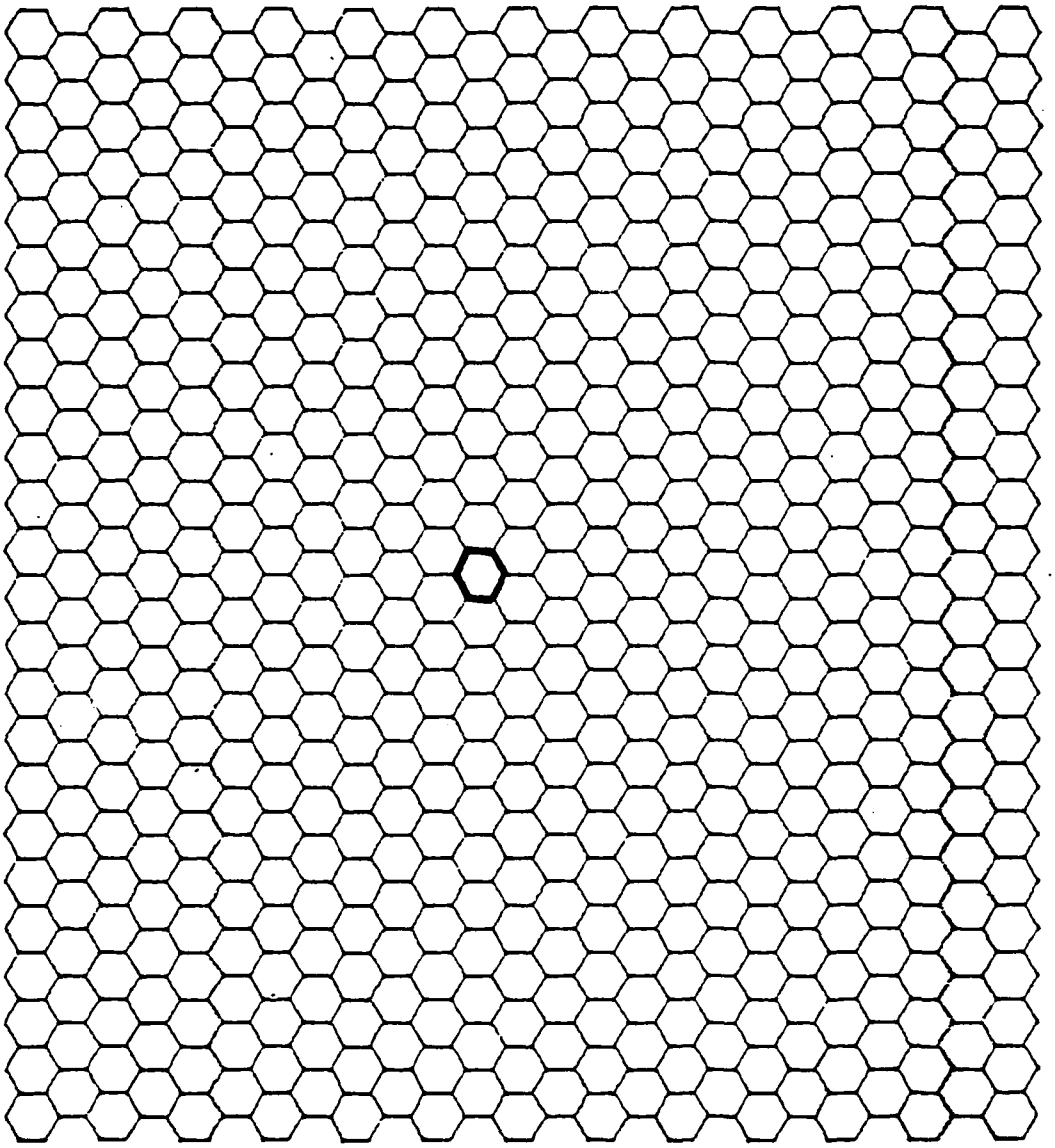
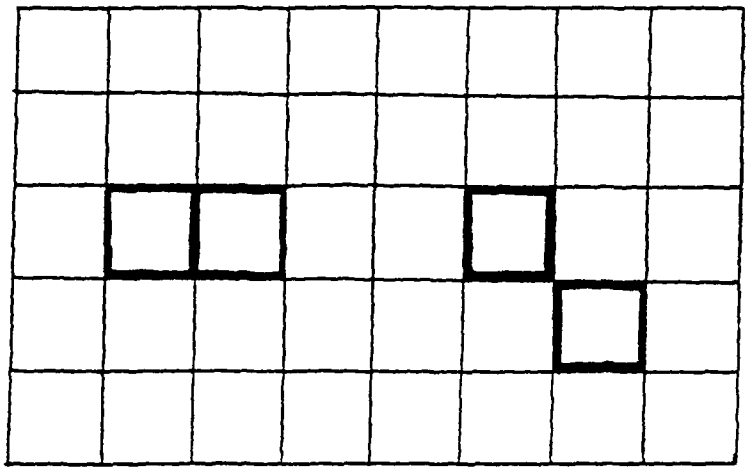
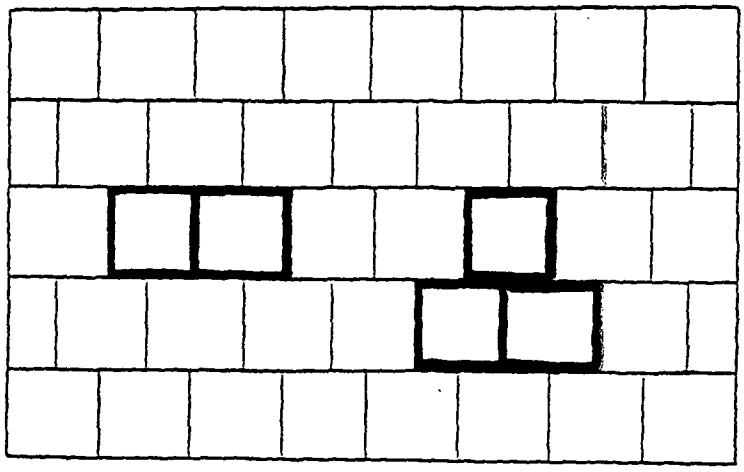


Figure 2: Absence of Slippage in Hexagon-Coverings of the Plane

(a)



(b)



(c)

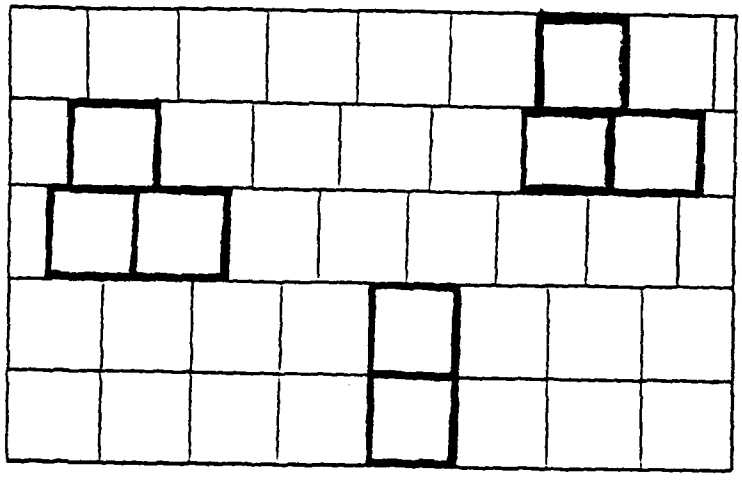


Figure 3: Non-Uniform Adjacency in Square-Coverings of the Plane

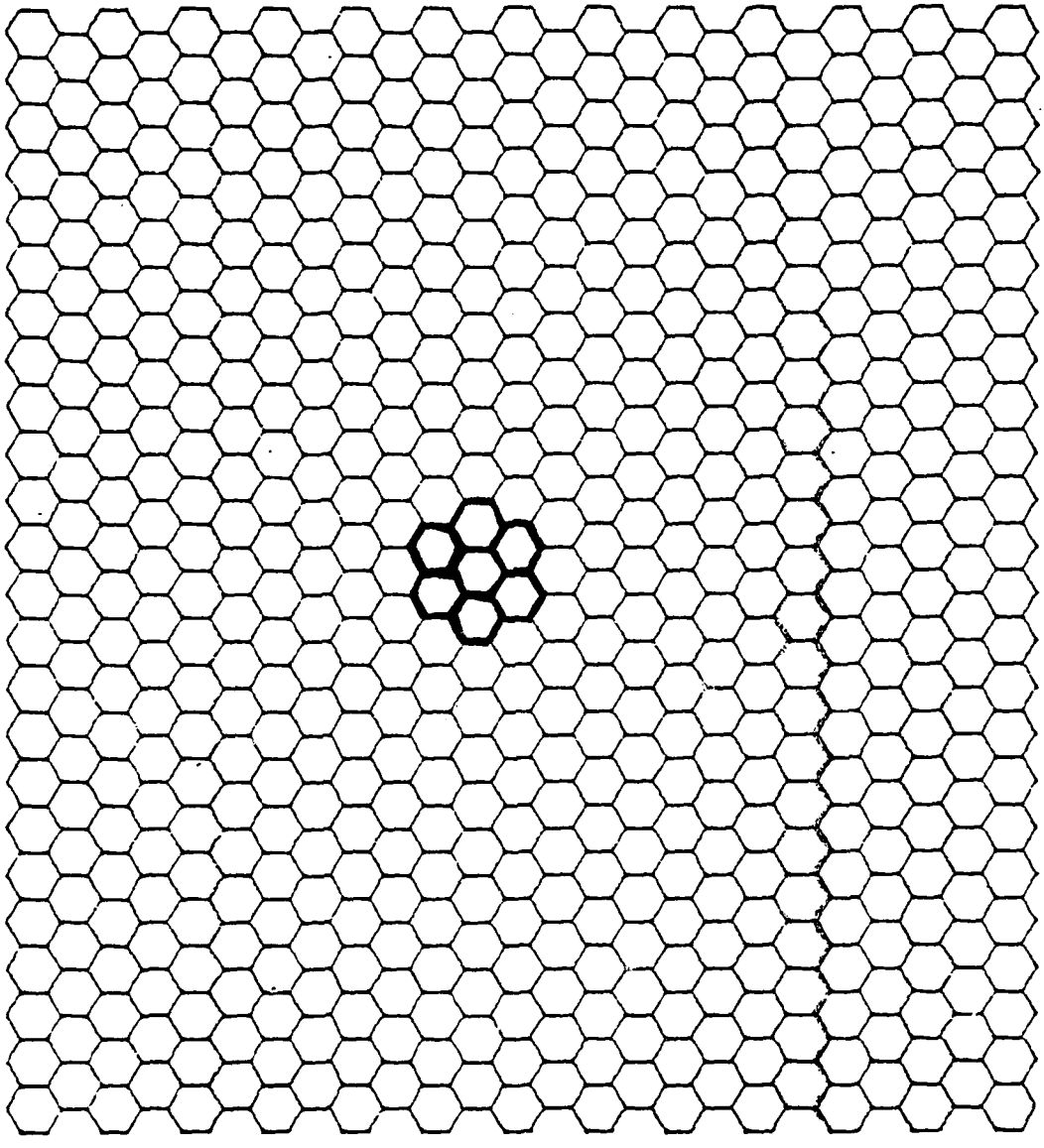


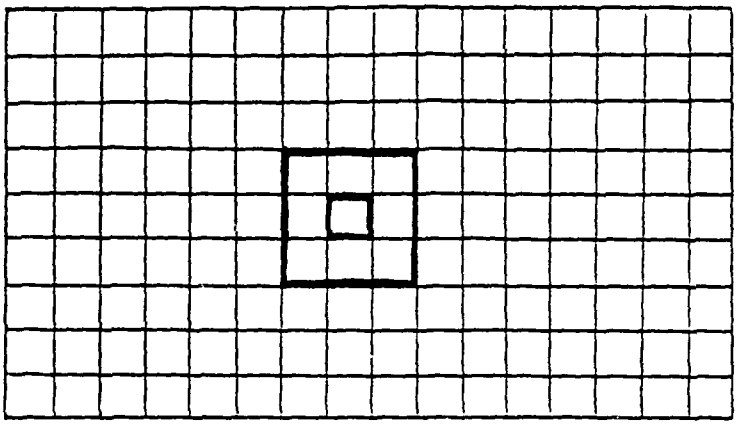
Figure 4: Uniform Adjacency in Hexagon-Coverings of the Plane

5a, an aggregate consisting of a square and all its immediate neighbors itself constitutes a square, as does the aggregate consisting of that square and all of its immediate neighbors. This is not the case, however, with uniform or more complex displacements. As shown in Figure 5b for the case of d = s/2, the aggregate that consists of a square and all of its immediate neighbors is not a square, or even a rectangle, but a twelve-sided figure with sides not even of equal length, because of left- and rightward extensions induced by the displacement itself. Hexagon-coverings of the plane are also non-uniform in hierarchical structure in that an aggregate consisting of a square and all its immediate neighbors is not itself a hexagon, but a "snowflake"-like figure with eighteen sides, as shown in Figure 6. In contrast to square-coverings with d = s/2, however, these sides are all of equal length and the aggregate itself is symmetric around its center, resulting in invariance of the covering as a whole under any number of sixty-degree rotations of the plane in any direction. The square-aggregate in Figure 5b is symmetric around its horizontal and vertical axes, but not around its center, resulting in invariance of the covering as a whole only under rotations of 180° or its multiples.

4. Uniform Packing

Square-coverings of the plane allow uniform directly vertical, horizontal, and diagonal packing of first-level aggregates only in the case of null displacement. As shown in Figure 7a, such aggregates pack uniformly with no gaps in the covering in each of these three directions. Directly vertical packing is impossible in the case of horizontal displacement, because of the displacement itself, as can be seen in Figures 7b and c. As shown in Figure 7b, directly horizontal packing necessarily leaves gaps in the covering, and the same is true of directly diagonal packing, as shown in Figure 7c. Uniformity is possible with first-level square aggregates only in displaced packing. As shown in Figure 7d, potential gaps in the covering can be elimi-

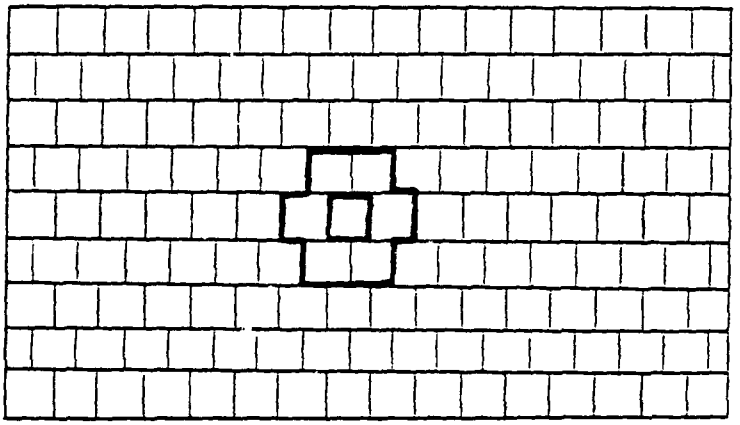
(a)



d = 0

uniform
hierarchical
structure

(b)



$d = \frac{s}{2}$

non-uniform
hierarchical
structure

Figure 5: First-Level Aggregates in Hierarchical Structure of Square-Coverings of the Plane

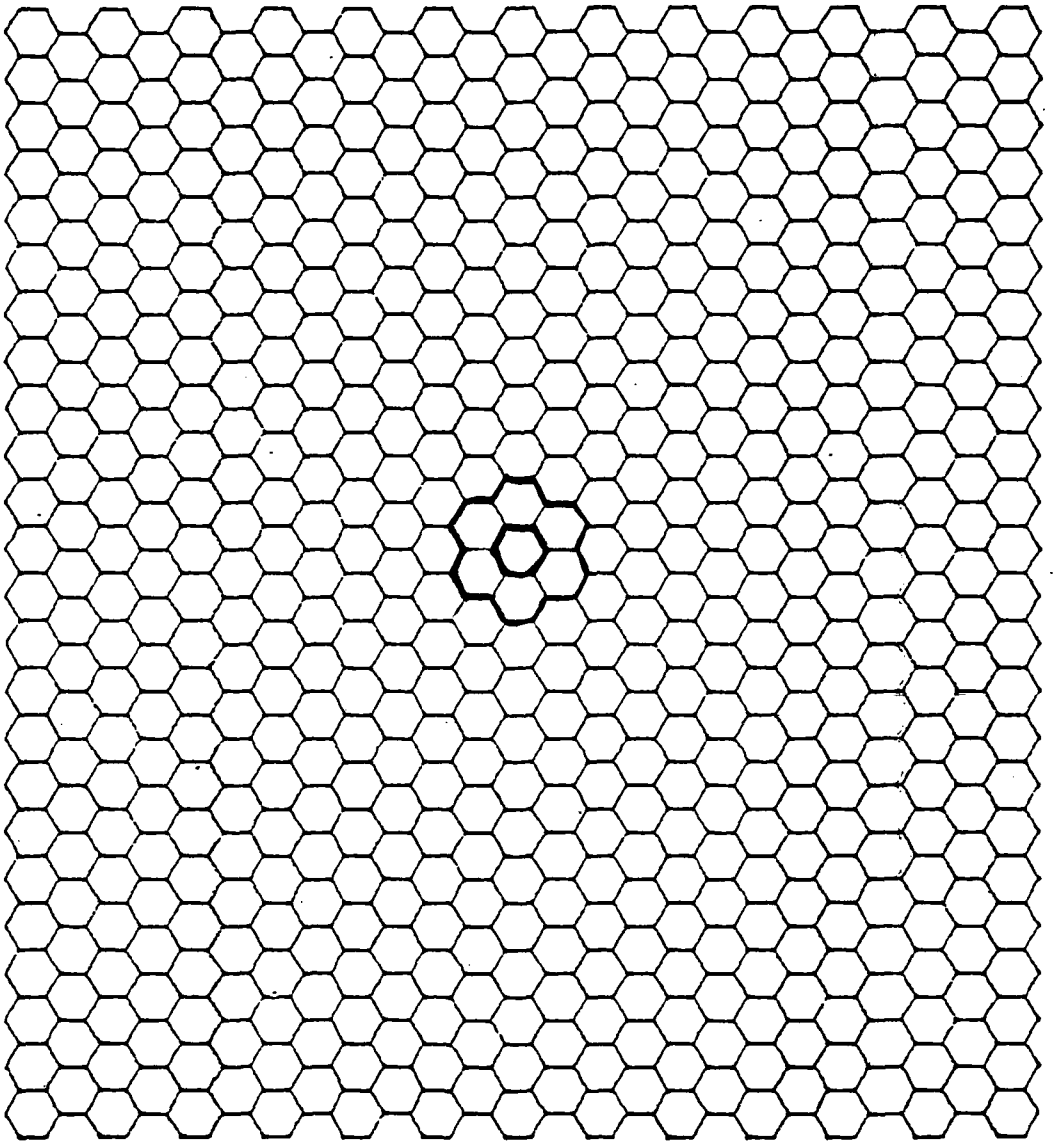
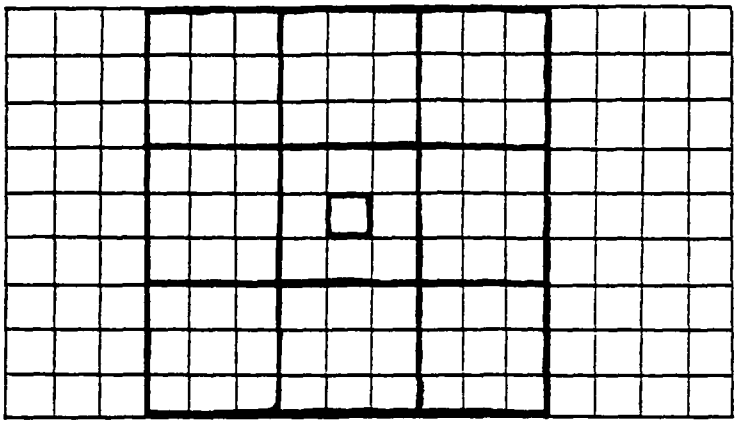


Figure 6: First-Level Aggregate in Hierarchical Structure of Hexagon-Coverings of the Plane

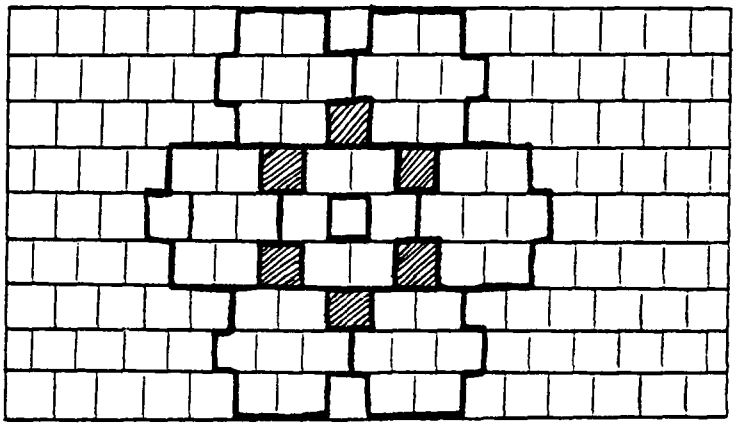
(7a)



$d = 0$

uniform packing, directly vertical, horizontal, and diagonal

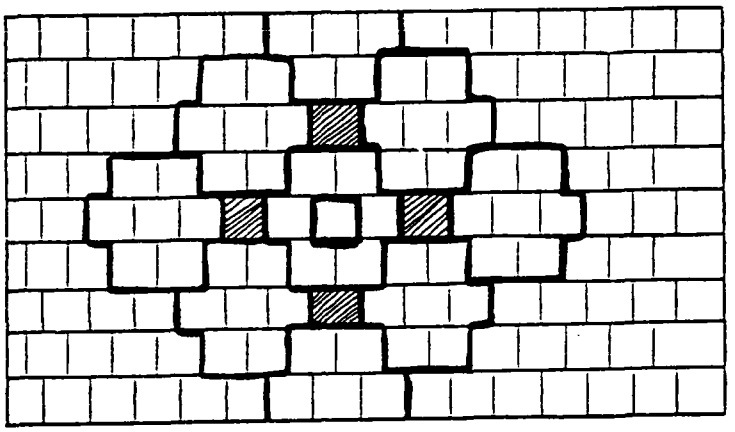
(7b)



$d = \frac{s}{2}$

non-uniform directly horizontal packing

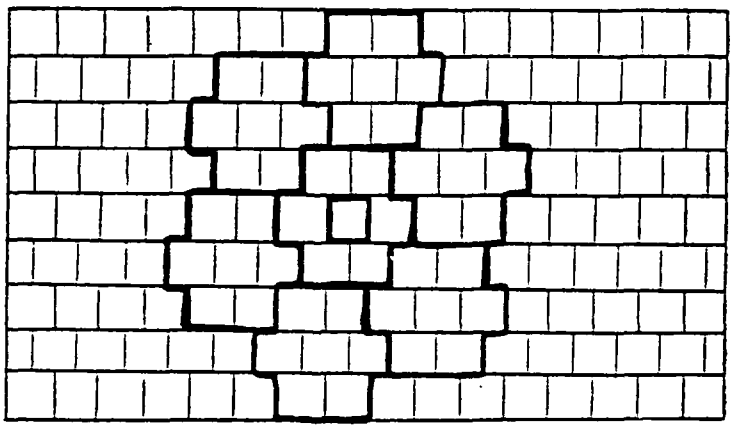
(7c)



$d = \frac{s}{2}$

non-uniform
directly
diagonal
packing

(7d)



$d = \frac{s}{2}$

uniform
displaced
packing

Figure 7: First-Level Packing in Square-Coverings of the Plane

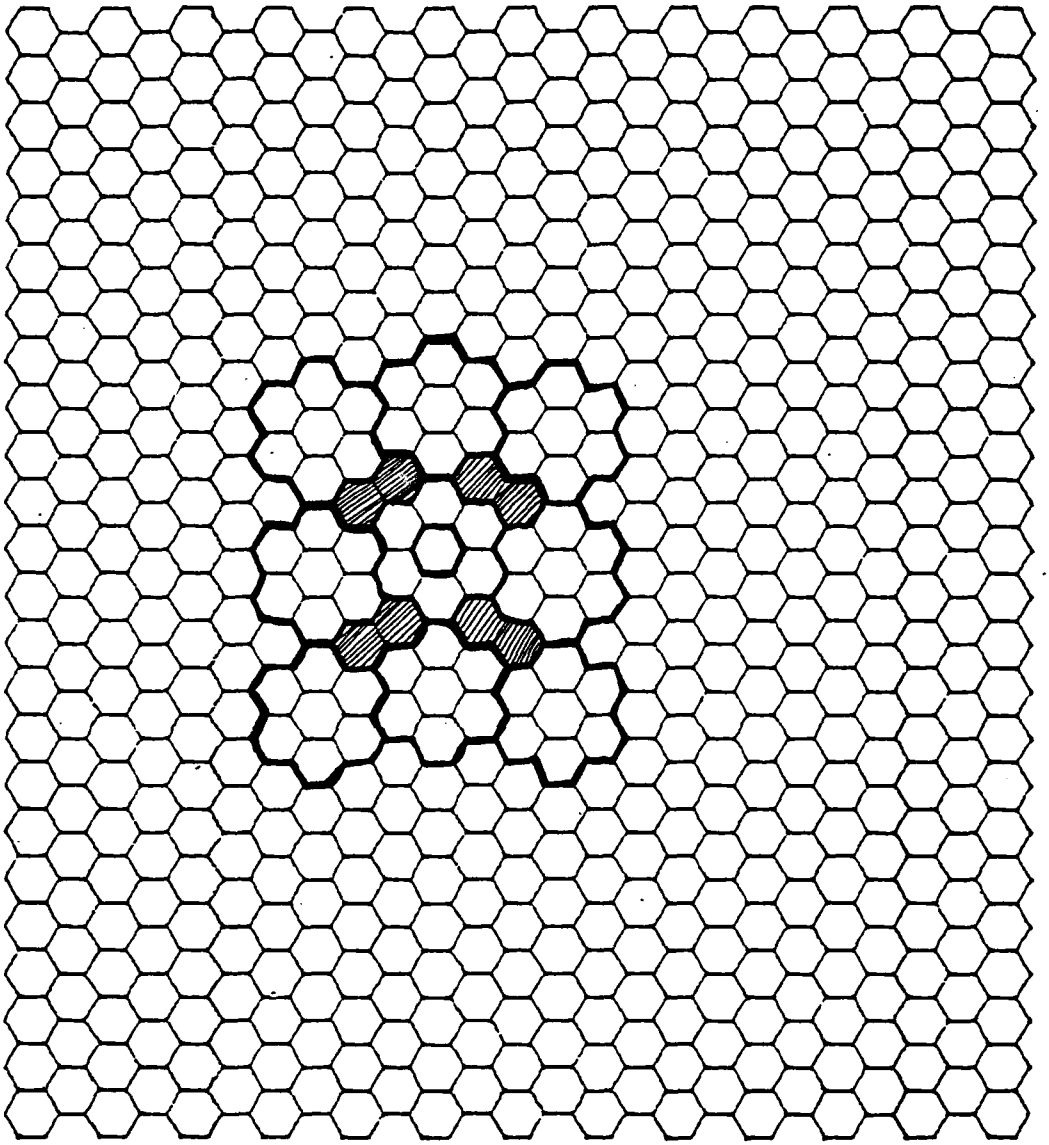
nated only by having the surrounding aggregates displaced around the center aggregate in the direction of the displacement of the squares themselves. With hexagon-coverings of the plane, the situation is similar, directly horizontal packing being impossible this time because of the fact that first-level aggregates must always meet in a side, just like the hexagons that make them up. As shown in Figures 8a and b, respectively, directly horizontal and diagonal packing

are necessarily non-uniform with a different gap pattern left in the covering in each case. As in the case of square-coverings of the plane, uniform packing is possible with hexagon-coverings only by having the surrounding aggregates displaced with respect to the center one, as shown in Figure 8c.

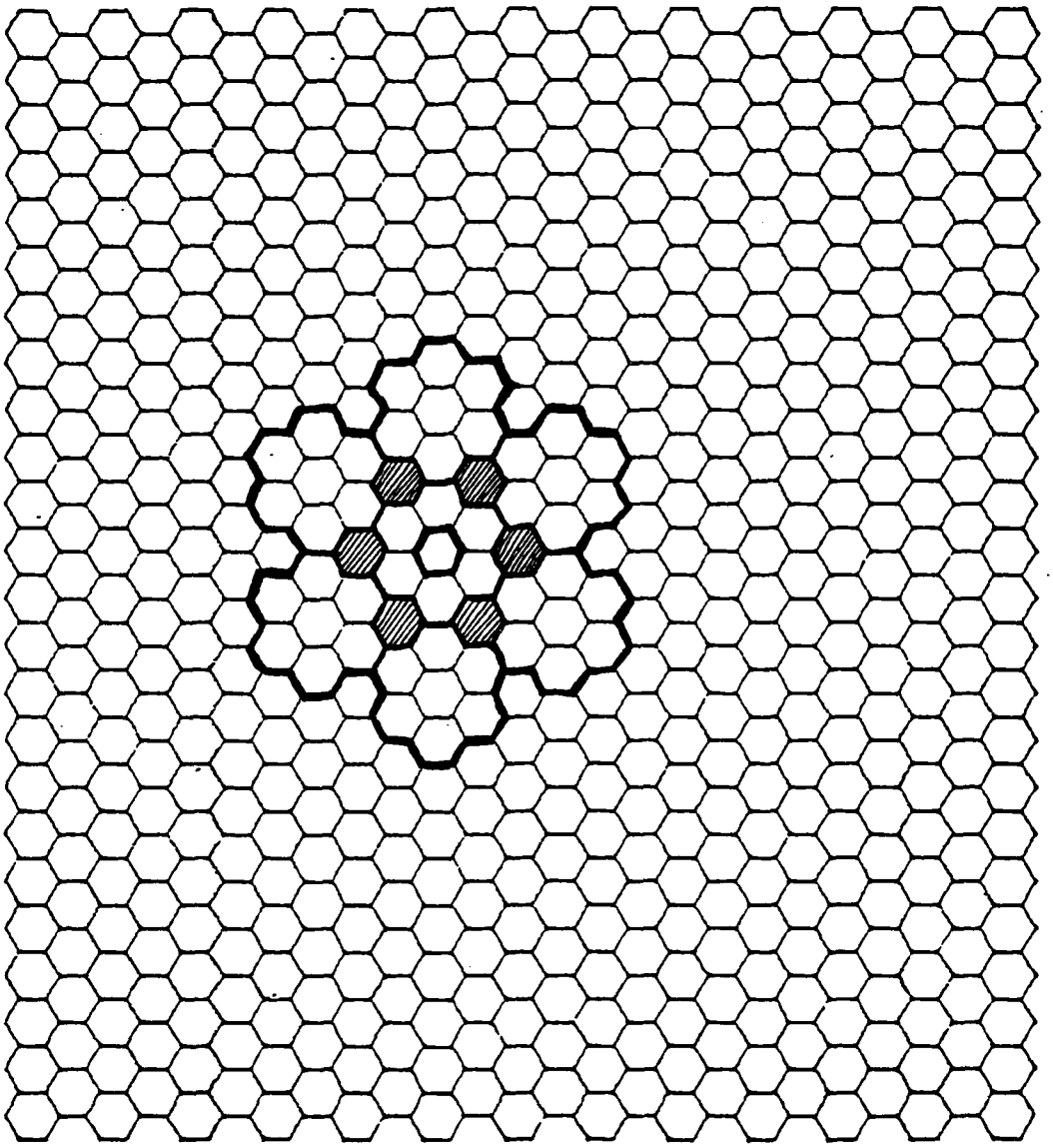
5. Arithmetic Encoding of Hierarchical Structure

5.1 Hexagon Arithmetic

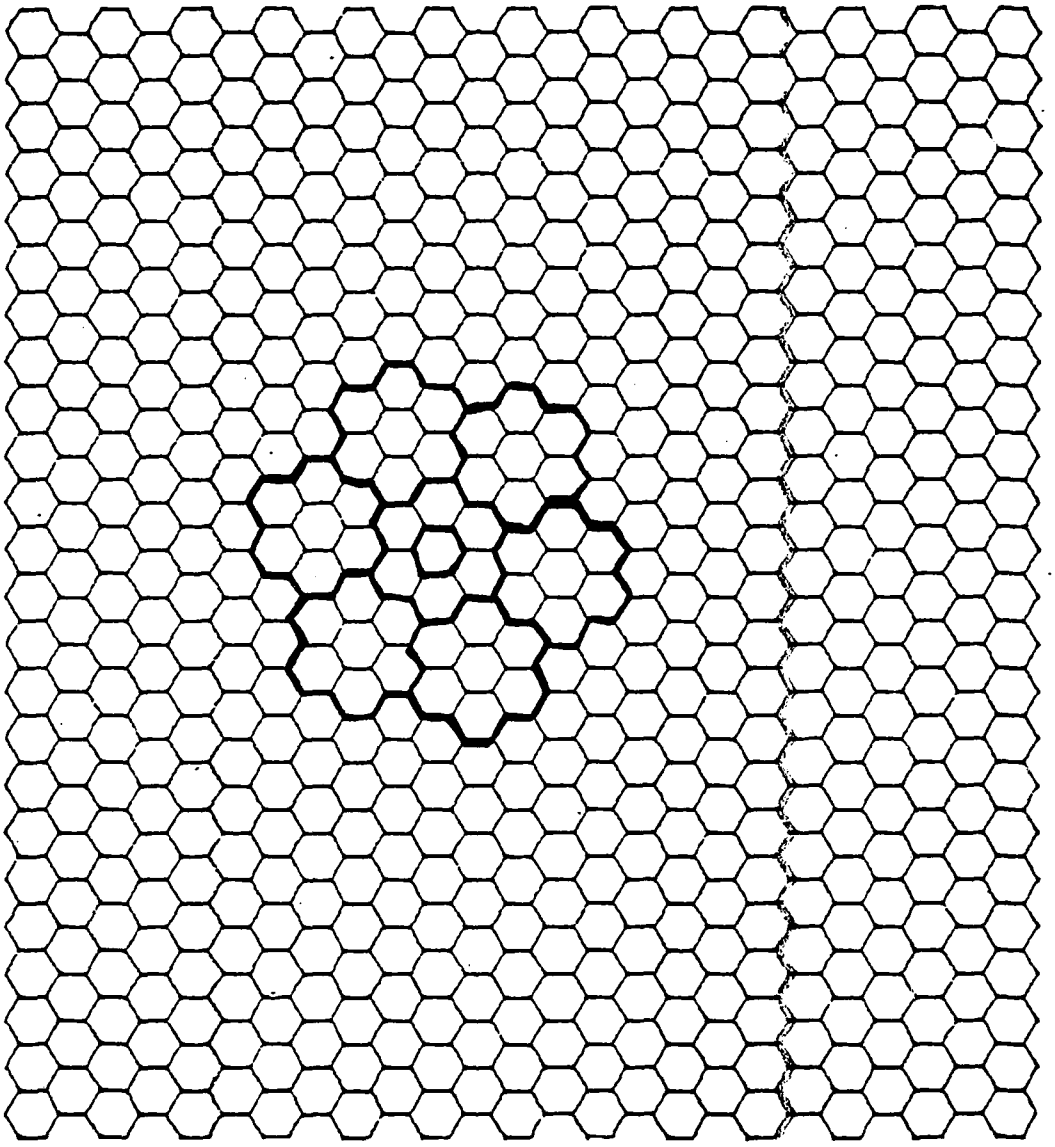
Hexagon-coverings of the plane can be naturally encoded in arithmetic form in a way that gives useful geometric interpretations to the basic arithmetic operations. First, we choose one hexagon as our "origin" and assign it the "coordinate" 0, as show in Figure 9a. Second, we assign numbers from 1-6 to each of the six directions emanating from 0 and assign the number of each direction as a coordinate to the immediate neighbor of 0 in the respective direction; in particular, we assign these numbers so that the coordinates of opposite hexagons add up to 7, as shown in Figure 9b. Finally, we assign numbers in the same way to aggregates of each level, using a new leftward place value for each level of aggregates, as shown in Figure 9b. Given the numbering scheme, addition of hexagons can be defined as in Figure 10a and multiplication as in Figure 10b, with all the usual laws of addition and multiplication (associative, commutative, distributive, etc.) assumed to hold, as illustrated, respectively, in Figures 11a and b.



(8a) Non-Uniform Directly Horizontal Packing

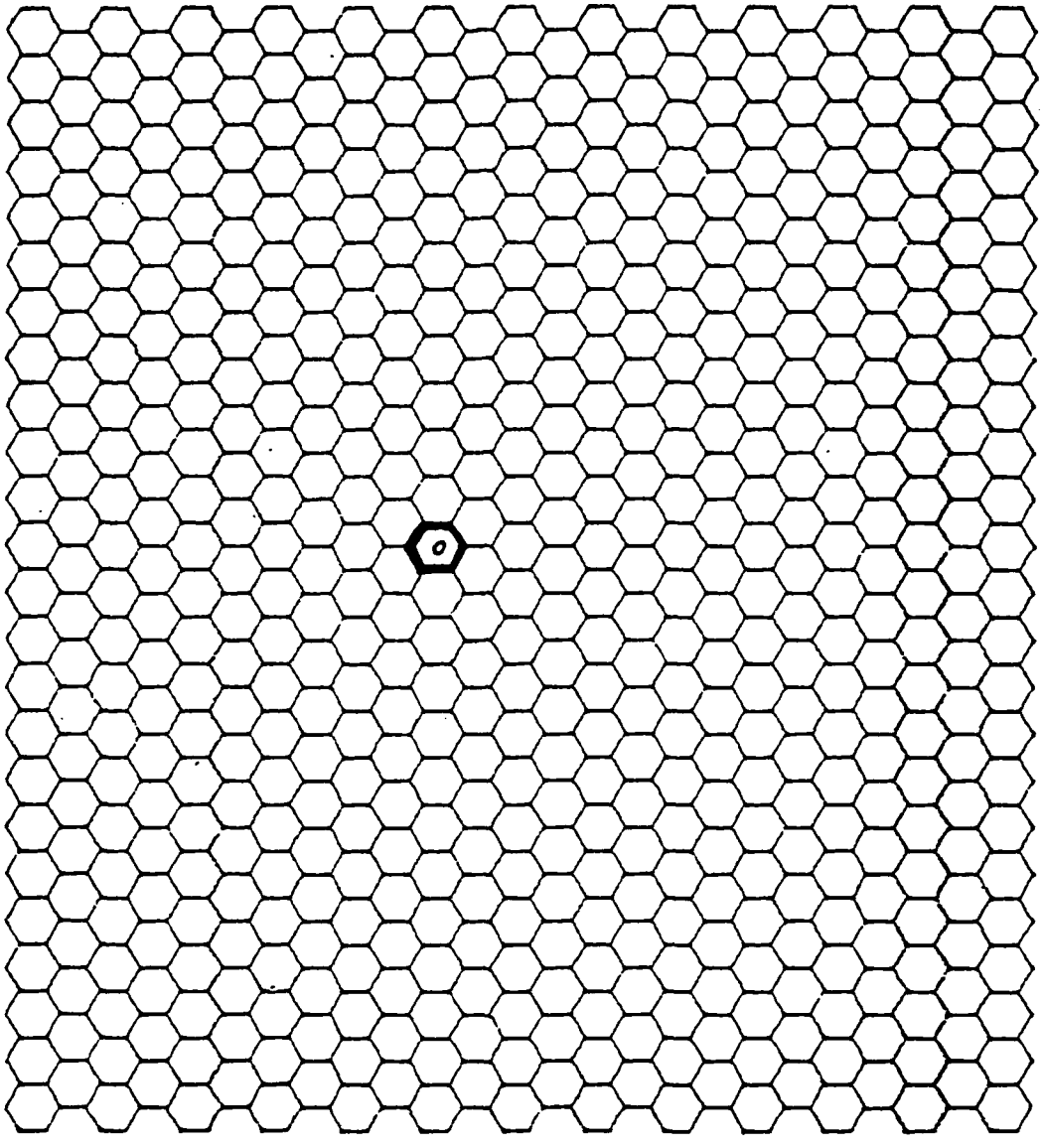


(8b) Non-Uniform Directly Vertical Packing

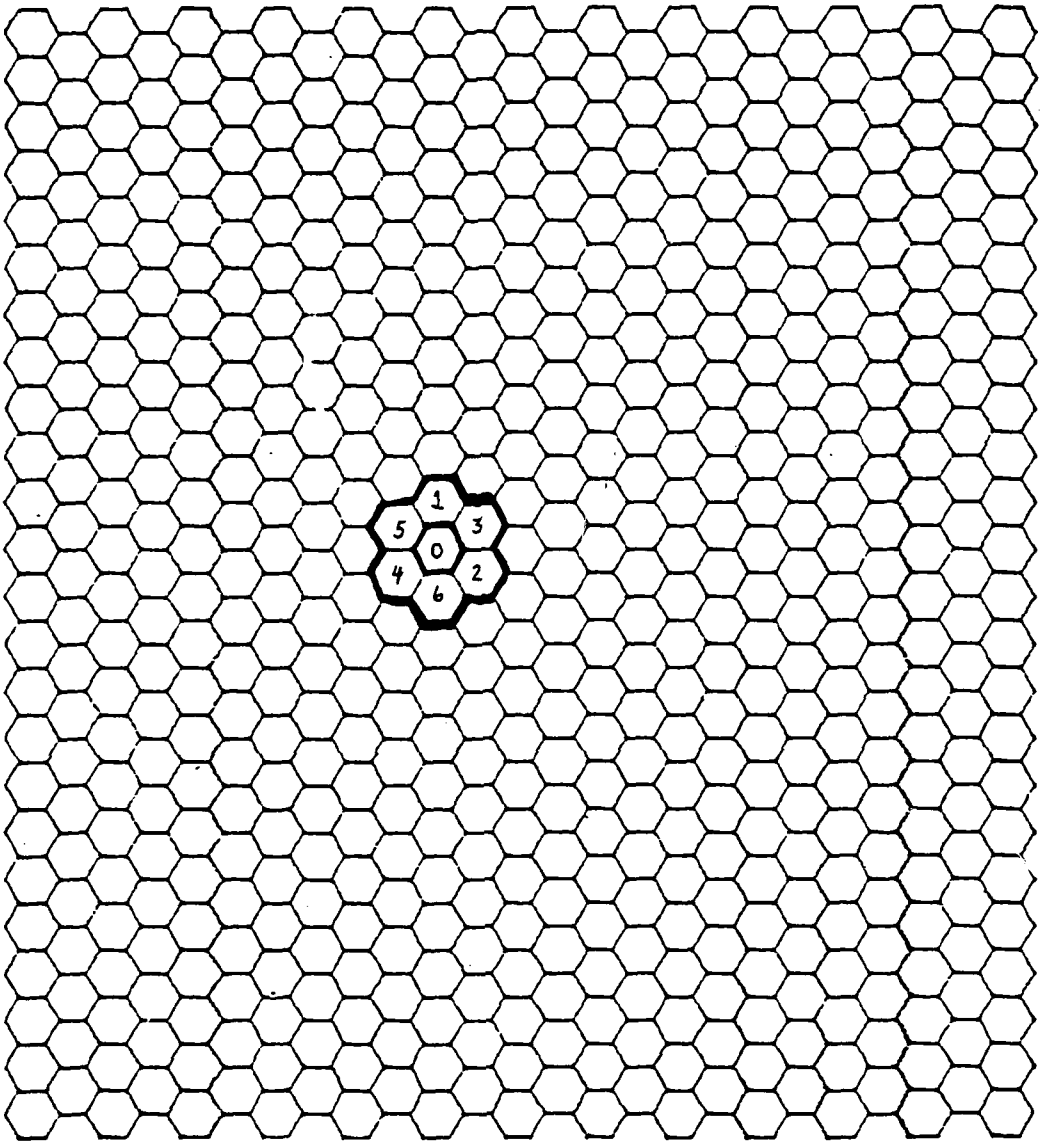


(8c) Uniform Displaced Packing

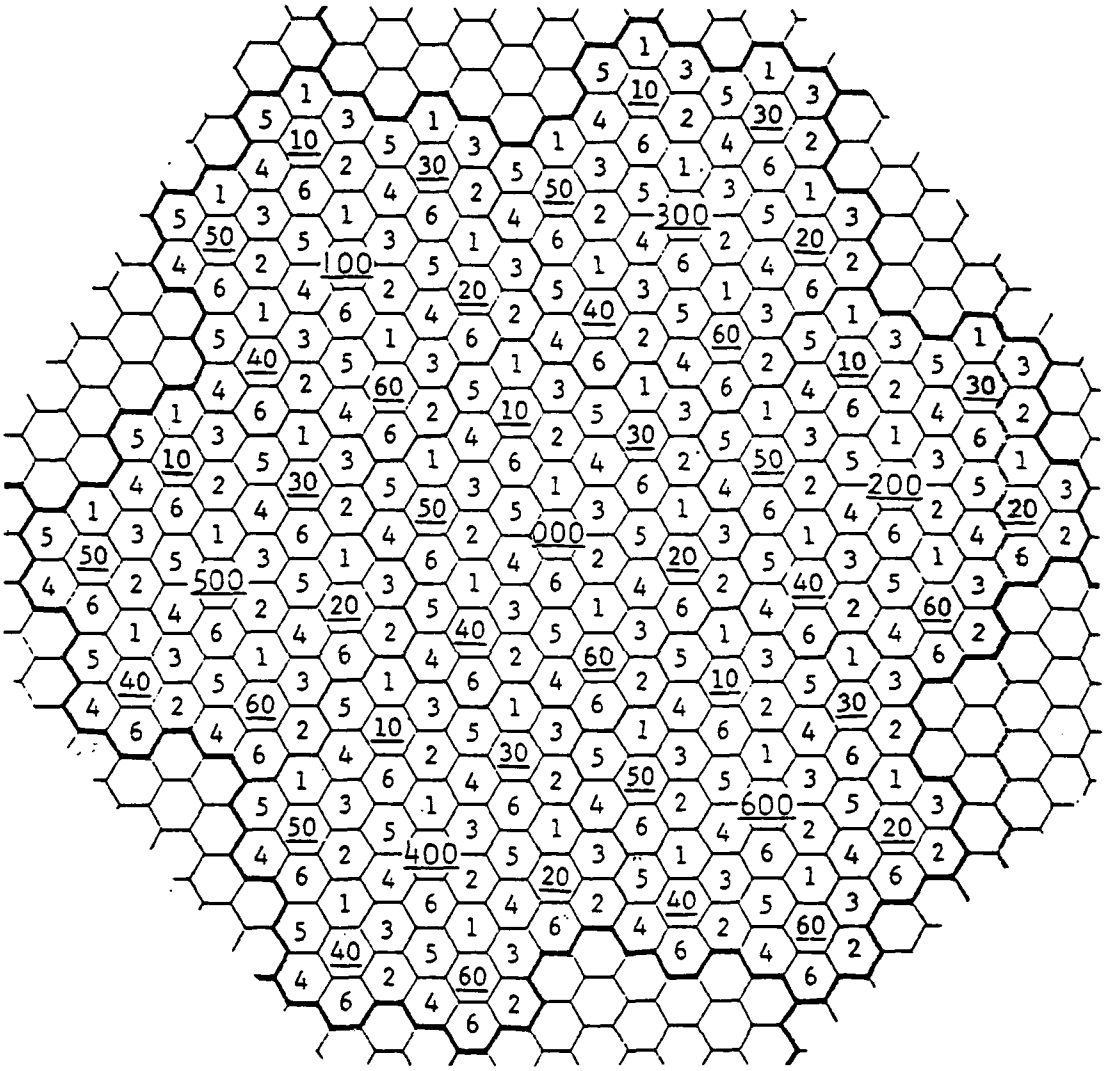
Figure 8: First-Level Packing in Hexagon-Coverings of the Plane



(9a) Arbitrary Choice of Origin



(9b) Numbering of Directions and Immediate Neighbors of Origin



(9c) Coordinates of the First Three Levels of Aggregates

Figure 9: Coordinate System of Hexagon-Covering

(a) The Hexagon Addition Table

+	0	1	2	3	4	5	6
0	0	1	2	3	4	5	6
1	1	12	3	34	5	16	0
2	2	3	24	25	6	0	61
3	3	34	25	36	0	1	2
4	4	5	6	0	41	52	43
5	5	16	0	1	52	53	4
6	6	0	61	2	43	4	65

(b) The Hexagon Multiplication Table

x	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6
2	0	2	4	6	1	3	5
3	0	3	6	2	5	1	4
4	0	4	1	5	2	6	3
5	0	5	3	1	6	4	2
6	0	6	5	4	3	2	1

Figure 10: Hexagon Arithmetic Tables

(a) A Sample Hexagon Addition Problem

PROBLEM: Add the numbers 123 and 461.

```
   (2) (3)
    1   2   3
  + 4   6   1
  -------------
    0   4   4
```

Note: () Denotes a carry.

3 + 1 = 34 or (3)4

(3) + 2 = 25, 25 + 6 = 24 or (2)4

(2) + 1 = 3, 3 + 4 = 0.

(b) A Sample Hexagon Multiplication Problem

```
   25
  x 64
  ------
   16      (= 25 x 4)
  52       (= 25 x 6)
  ------
  536      (sum using System X addition)
```

Figure 11: Sample Hexagon Arithmetic Problems

The addition of two digits results in a base digit, which is the sum of the two digits modulo 7, and a carry-digit, which, when non-zero, indicates the direction in which the addition carries the sum into the next row of surrounding hexagons, as shown in Figure 12. Multiplication of two digits is simply their product modulo 7. Geometrically, hexagon addition is a hierarchical version of standard vector addition, as shown for the example of Figure 11a in Figure 13a. Hexagon mulitplication adds angles, measured from the vertical, and multiplies distances from the origin, as shown in Figure 13b, and so can be used to represent the stretching and rotating of image patterns in the plane.

5.2 Square Arithmetic

The arithmetic encoding of square-coverings of the plane is richer than that of hexagon-coverings because of the potential variations in grid patterns discussed in earlier sections. Square-coverings with null displacement cannot be encoded as straightforwardly as hexagons because of the absence of uniform adjacency, which we discussed in Section 2. As shown in Figure 14, numbering a null-displacement square-covering in a manner analogous to that of the hexagon-covering we discussed in Section 5.1 leads to an addition table that lacks the appropriate modulus property. Since there are eight squares surrounding any given square, base digits should add modulo 9, just as hexagon base digits add modulo 7, but this is not borne out in fact. An appropriate numbering is given in Figure 14a and the addition table derived from it (by the vector parallelogram law) in Figure 14b. As the table shows, 8 plus 4, for example, is 82, but 8 plus 4 is not 2 mod 9.

A geometrically natural encoding of null-displacement square-coverings can be obtained, if we drop the requirement that a higher-level aggregate must consist of a lower-level aggregate plus all its immediate

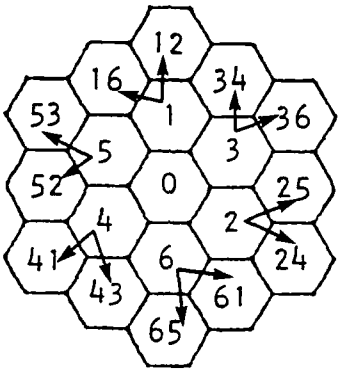
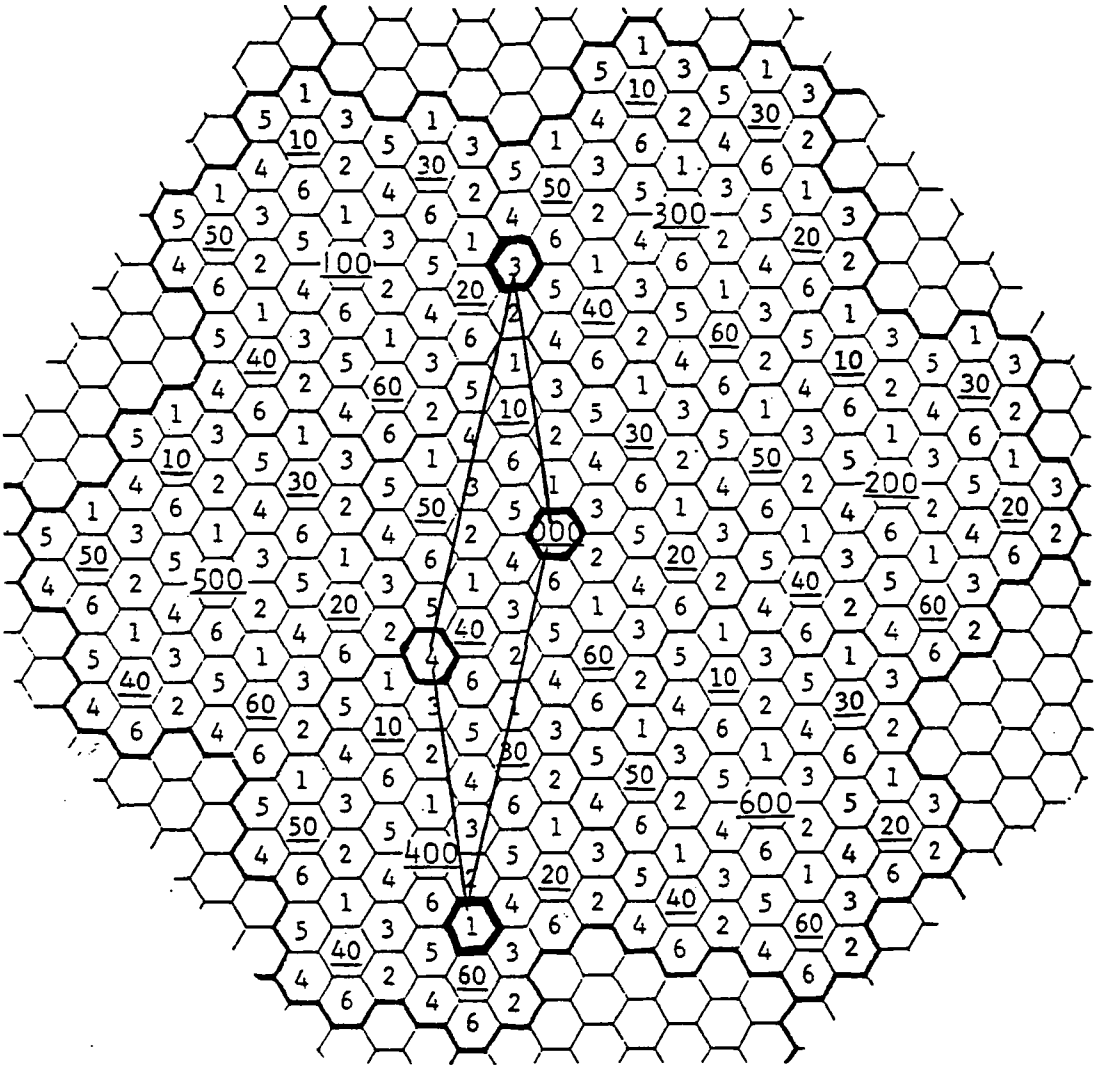
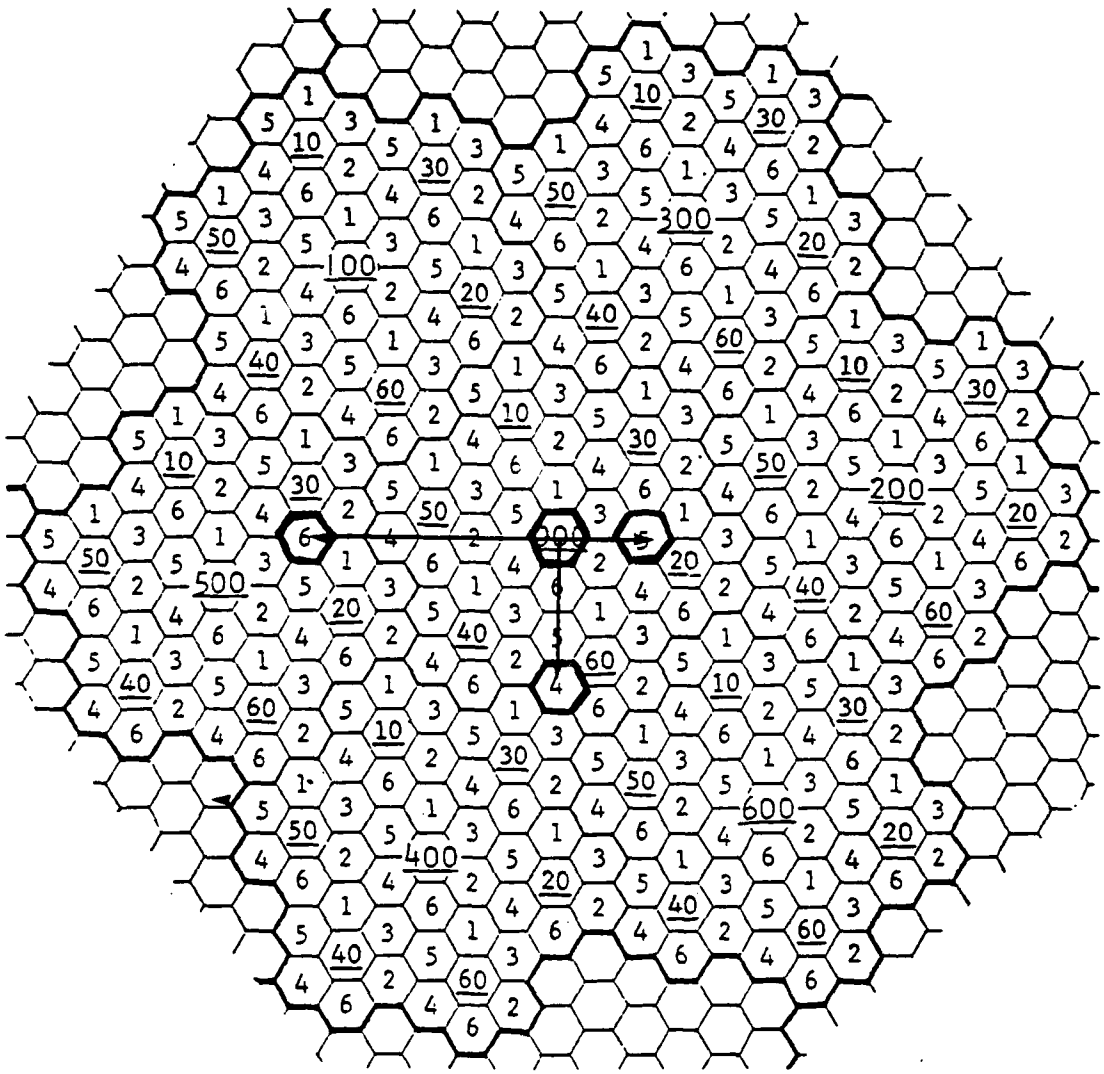


Figure 12: The Significance of Carry-Digits in Hexagon Arithmetic

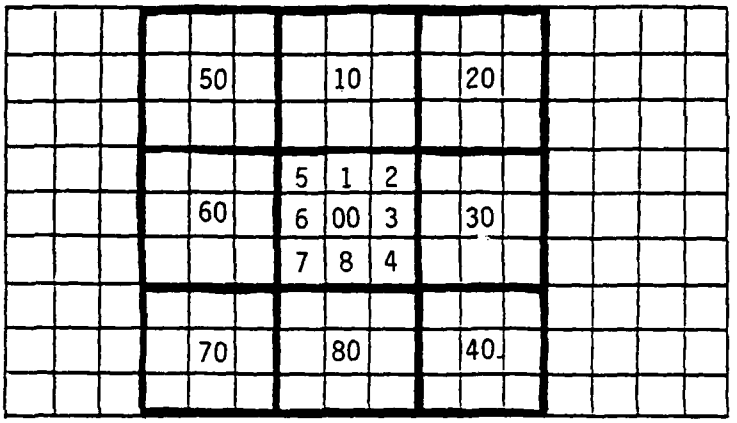


(13a) Addition: A Hierarchical Version of the Standard Vector Parallelogram



(13b) Multiplication: Adds Angles from the Vertical and Multiplies Distances from the Origin

Figure 13: Geometric Interpretation of Hexagon Arithmetic



(a) An Appropriate Numbering

+	0	1	2	3	4	5	6	7	8
0	0	1	2	3	4	5	6	7	8
1	1	18	14	2	3	17	5	6	0
2	2	14	27	35	36	18	1	0	3
3	3	2	35	36	37	1	0	8	4
4	4	3	36	37	45	0	8	81	82
5	5	17	18	1	0	54	62	63	6
6	6	5	1	0	8	62	63	64	7
7	7	6	0	8	81	63	64	72	85
8	8	0	3	4	82	6	7	85	81

(b) The Corresponding Addition Table

Figure 14: Null-Displacement Square Addition

neighbors and, instead, allow first-level aggregates to consist of a square plus those immediate neighbors that meet it at an edge, rather than at a point, as shown in Figure 15. Such aggregates pack uniformly in two ways, upward climbing and downward climbing as shown in Figure 16, and a suitable arithmetic can be constructed for either packing by having the coordinate numbers climb in the direction opposite to those of the aggregates themselves, as shown in Figures 1 and 18 for the example of downward climbing coordinate numbers. As shown in Figure 17, downward climbing numbers with upward climbing aggregates results in a suitable arithmetic encoding, with base digits adding modulo 5, but as shown in Figure 18, the same numbering fails to yield an appropriate arithmetic when the aggregates are also downward climbing.

Uniform displacement in square-coverings of the plane results in an arithmetic that is intermediate in a number of ways between null displacement and hexagons. As with hexagons, each square has exactly six immediate neighbors, each of which meets the square at an edge, but, in contrast to hexagons, some meet at a full edge and some only at a half edge, because of the displacement itself, as shown in Figure 19. As with square-coverings with null displacement, two uniform packings of first-level aggregates are possible, upward climbing and downward climbing, as shown in Figure 20, and a suitable arithmetic can be defined only when the climbing of the aggregates is opposite to that of the numbering, as shown in Figures 21 and 22. According to Figure 21, 5 plus 4 for example, is 42, in accordance with the fact that 9 is congruent to 2 mod 7, but, according to Figure 22, 5 plus 4 is 43, which is not consistent with this fact. If, for convenience, we refer to hexagon arithmetic as arithmetic of type X (for heXagon) and to square arithmetic with null displacemnt as arithmetic of type T (for the shape of its first-level aggregates), then it makes sense to refer to square arithmetic with uniform displacement as arithmetic of type $\frac{X-T}{2}$ to indicate its intermediate status. Whether this suggestive

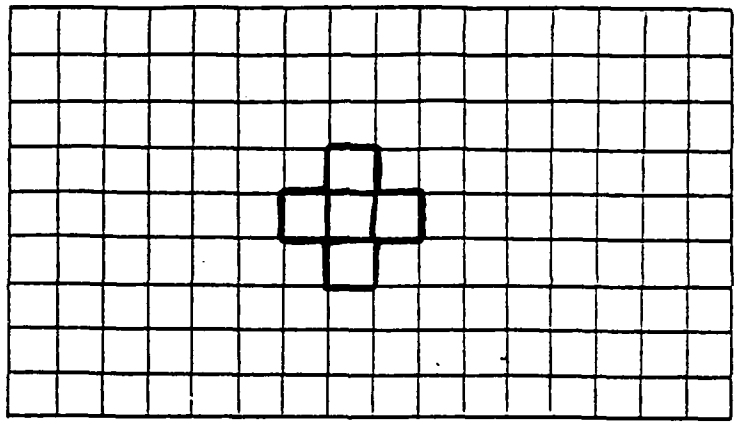
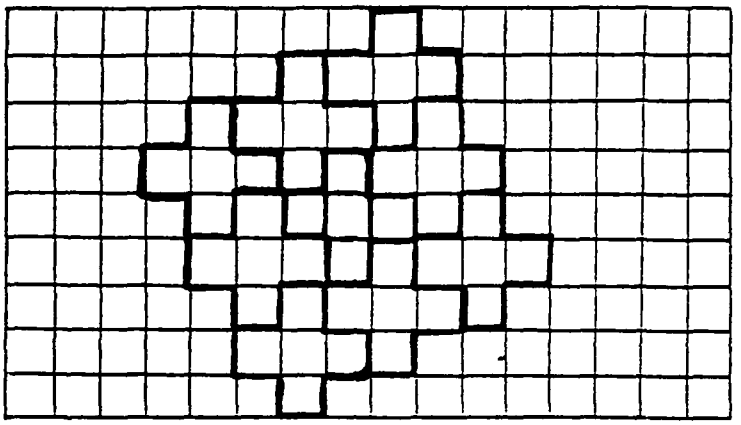
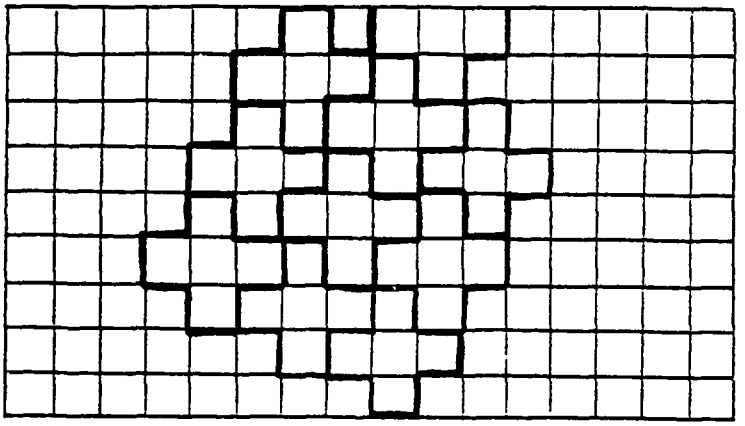


Figure 15: Arithmetically Encodable First-Level Square Aggregate: Null Displacement

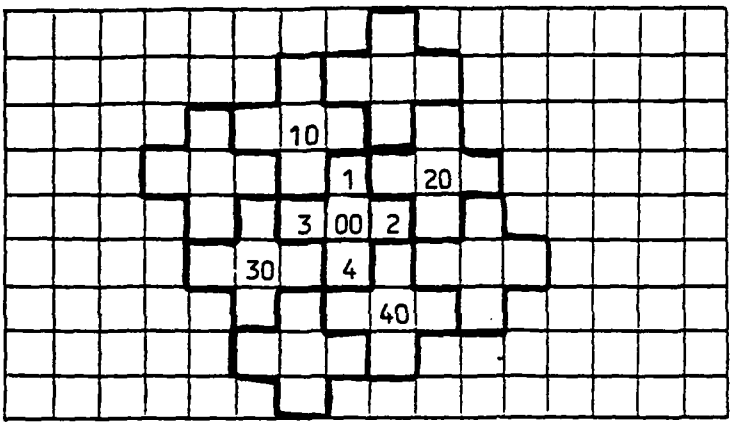


(16a) Upward Climbing



(16b) Downward Climbing

Figure 16: Two Uniform Packings of First-Level Square Aggregates: Null Displacement

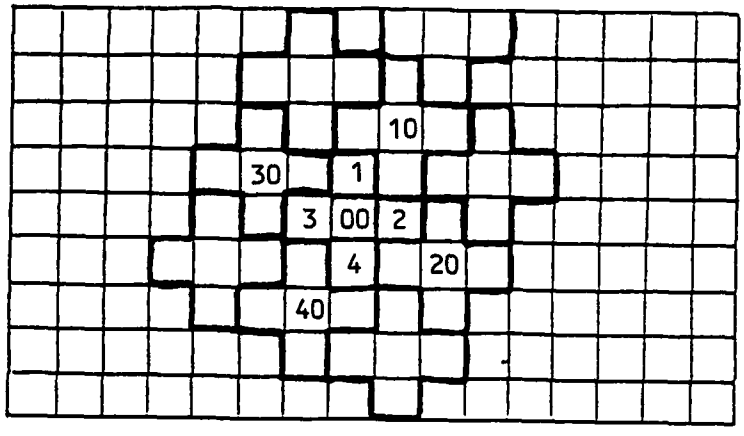


(17a) Numbering of Squares and Aggregates

+	0	1	2	3	4
0	0	1	2	3	4
1	1	12	23	14	0
2	2	23	24	0	41
3	3	14	0	31	32
4	4	0	41	32	43

(17b) Addition Table: Base Digits Add Modulo 5

Figure 17: Arithmetic When Numbers and Aggregates Climb in Opposite Directions: Null Displacement



(18a) Numbering of Squares and Aggregates

+	0	1	2	3	4
0	0	1	2	3	4
1	1	13	14	32	0
2	2	14	21	0	23
3	3	32	0	34	41
4	4	0	23	41	42

(18b) Addition Table: Base Digits Do Not Add Modulo 5

Figure 18: Arithmetic When Numbers and Aggregates Climb in the Same Direction: Null Displacement

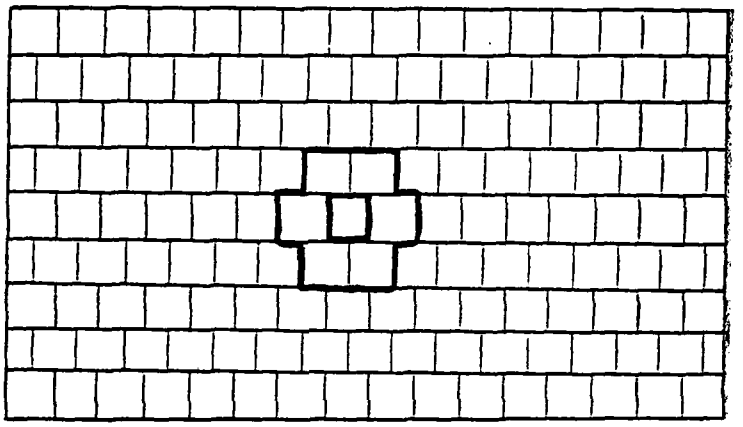
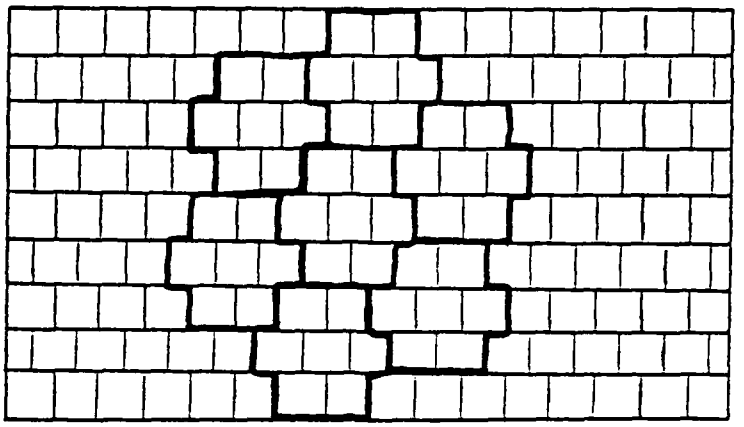
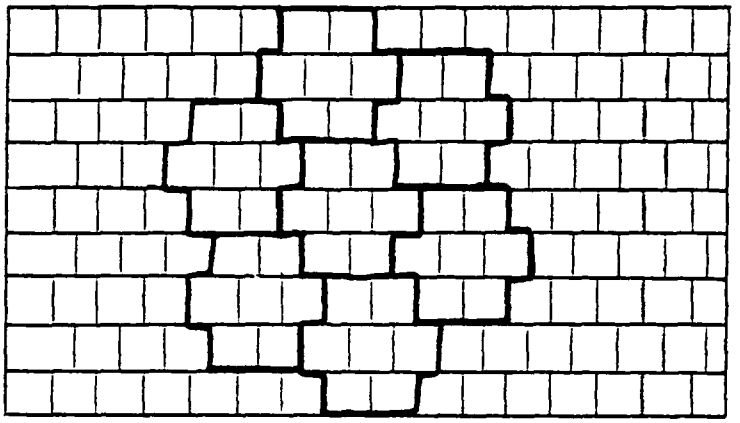


Figure 19: First-Level Aggregate in Square-Coverings with Uniform Displacement

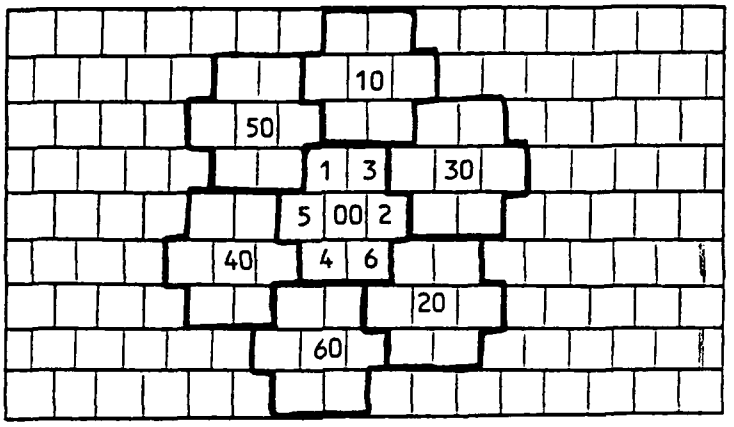


(20a) Upward Climbing



(20b) Downward Climbing

Figure 20: Two Uniform Packings of First-Level Square Aggregates: Uniform Displacement

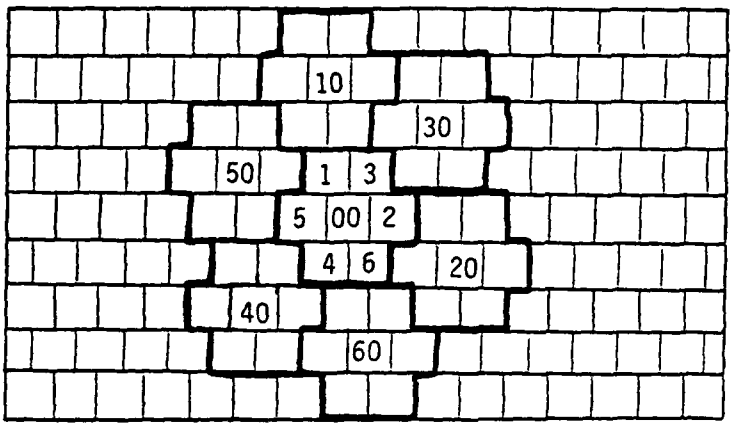


(21a) Numbering of Squares and Aggregates

+	0	1	2	3	4	5	6
0	0	1	2	3	4	5	6
1	1	52	3	14	5	56	0
2	2	3	34	35	6	0	21
3	3	14	35	16	0	1	2
4	4	5	6	0	61	42	63
5	5	56	0	1	42	43	4
6	6	0	21	2	63	4	25

(21b) Addition Table: Base Digits Add Modulo 7

Figure 21: Arithmetic When Numbers and Aggregates Climb in Opposite Directions: Uniform Displacement



(22a) Numbering of Squares and Aggregates

+	0	1	2	3	4	5	6
0	0	1	2	3	4	5	6
1	1	14	3	16	5	52	0
2	2	3	21	34	6	0	25
3	3	16	34	35	0	1	2
4	4	5	6	0	42	43	61
5	5	52	0	1	43	56	4
6	6	0	25	2	61	4	63

(22b) Addition Table: Base Digits Do Not Add Modulo 7

Figure 22: Arithmetic When Numbers and Aggregates Climb in the same Directions: Uniform Displacement

notational device generalizes in any natural way to other areal coordinate systems is an intriguing question theoretically, but would seem to be of little practical interest. The wide range of coordinate systems themselves, however, would seem to be of very great practical interest, especially if they can be integrated into a dynamic coordinate system that shifts among them in response to applicational needs.

6. Squares and Hexagons in Higher Order Software

Systems are defined in Higher Order Software (HOS) by giving formal specifications of the data types, functions, and control structures that make them up. Data types are the basic kinds of objects that play a role in a system, functions are the activities these objects are involved in, and control structures are the relationships that obtain among these activities. A data type is specified by giving a set of primitive operations that map into and out of the data type (and others) in accordance with a given set of axioms. A function is specified by, first, defining an operation in terms of a tree structure, or "control map," that expresses its decomposition into suboperations and ultimately into the primitive operations on data types and, second, assigning specific variables as inputs and outputs of the operation so defined. Control structures are specified by giving a control map in which variable operation names appear, so that it is the relationships among them that get represented by the tree structure, rather than the operations themselves. With respect to an areal coordinate system, such as square- or hexagon-coverings of the plane it is the squares or hexagons themselves, which we will call "places" to emphasize their areal character, that provide the basic data type, with things like distance measuring and border determination in the class of operations, structures, and thus functions, when appropriate variables get assigned.

Square-coverings of the plane with null displacement can be specified in terms of primitive operations that represent the four cardinal directions in the plane, as shown in Figure 23. Each primitive operation assigns each place its immediate neighbor in the corresponding direction, as characterized in the axioms in the figure. The first axiom, for example, says that the immediate northern neighbor of the immediate southern neighbor of a place is that place itself, unless that place is a southern border--i.e., it lacks an immediate southern neighbor, stated in the first partition--, in which case the North operation rejects, because no such neighbor exists. North and South, in other words, are inverse operations, except for the case in which one of them rejects. The second axiom, in contrast, characterizes East and South as orthogonal, rather than as inverses, because it says that the immediate eastern neighbor of the immediate southern neighbor of a place is the immediate southern neighbor of its immediate eastern neighbor, except, again, in the case in which one of these operations rejects and the corresponding neighbor does not exist. Hexagon-coverings of the plane can be specified in a similiar way, using six directions, rather than four, as shown in Figure 24. Since square-coverings of the plane with uniform displacement also involve six immediate neighbors of each square in forming first level aggregates, and thus six basic directions, they too can be characterized by the same data type specification as can hexagon-coverings of the plane.

Both hexagon-coverings of the plane and square-coverings with either null or uniform displacement can use exactly the same structure for determining border regions. As shown in Figure 25, a boolean-valued structure produces an output of True, if its input place has no immediate neighbor in the F direction, and False, if its input place has an immediate neighbor in the F direction, where F is any function (operation) from places to places and thus, in particular, any of the basic direction primitive operations of the data type. Null-

```
DATA TYPE:  PLACE;
PRIMITIVE OPERATIONS:
place_2 = North(place_1)
place_2 = South(place_1)
place_2 = East (place_1)
place_2 = West (place_1)

AXIOMS:
   WHERE p IS A PLACE
North (South(p)) = 1p OR K_REJECT (2p);
East  (South(p)) = South(East(1p)) OR K_REJECT (2p);
West  (South(p)) = South(West(1p)) OR K_REJECT (2p);

PARTITION OF p IS
   1p| Equal (South(p), REJECT) = False,
   2p| Equal (South(p), REJECT) = True;
South(North(p)) = 3p OR K_REJECT (4p);
East (North(p)) = North(East(3p)) OR K_REJECT (4p);
West (North(p)) = North(West(3p)) OR K_REJECT (4p);

PARTITION OF p IS
   3p| Equal (North(p), REJECT) = False,
   4p| Equal (North(p), REJECT) = True;
East (West(p)) = 5p OR K_REJECT (6p);
North(West(p)) = West(North(5p) OR K_REJECT (6p);
South(West(p)) = West(South(5p) OR K_REJECT (6p);

PARTITION OF p IS
     5p| Equal (West(p), REJECT) = False,
     6p| Equal (West(p), REJECT) = True;
West (East(p)) = 7p OR K_REJECT (8p);
North(East(p)) = East(North(7p) OR K_REJECT (8p);
South(East(p)) = East(South(7p) OR K_REJECT (8p);

PARTITION OF p IS
   7p| Equal (East(p), REJECT) = False,
   8p| Equal (East(p), REJECT) = True;

END PLACE;
```

Figure 23: HOS Specification of Data Type PLACE: Square-Coverings of the Plane

DATA TYPE: PLACE;

PRIMITIVE OPERATIONS:

$place_2$ = North($place_1$)
$place_2$ = South($place_1$)
$place_2$ = NorthEast($place_1$)
$place_2$ = NorthWest($place_1$)
$place_2$ = SouthEast($place_1$)
$place_2$ = SouthWest($place_1$)

AXIOMS:

WHERE p IS A PLACE

North(South(p)) = ^{1}p or K_{REJECT} (^{2}p);
NorthEast(South(p)) = SouthEast(^{1}p) OR K_{REJECT}(^{2}p);
NorthWest(South(p)) = SouthWest(^{1}p) OR K_{REJECT}(^{2}p);
SouthEast(South(p))= South(SouthEast(^{1}p)) OR K_{REJECT}(^{2}p);
SouthWest(South(p)) = South(SouthWest(^{1}p) OR K_{REJECT}(^{2}p);

PARTITION OF p IS

^{1}p | Equal (South(p), REJECT) = False
^{2}p | Equal (South(p), REJECT) = True

South(North(p)) = ^{3}p or K_{REJECT}(^{4}p)
NorthEast(North(p)) = North(NorthEast(^{3}p)) OR K_{REJECT}(^{4}p)
NorthWest(North(p)) = North(North West(^{3}p)) OR K_{REJECT}(^{4}p)
SouthEast(North(p)) = NorthEast(^{3}p) OR K_{REJECT}(^{4}p)
SouthWest(North(p)) = NorthWest(^{3}p) OR K_{REJECT}(^{4}p)

PARTITION OF p IS

^{3}p | Equal(North(p), REJECT) = False
^{4}p | Equal(North(p), REJECT) = True

North(NorthEast(p)) = NorthEast(North ^{5}p)) OR K_{REJECT}(^{6}p)
South(NorthEast(p)) = SouthEast(^{5}p) OR K_{REJECT}(^{6}p)
NorthWest(NorthEast(p)) = North(^{5}p) OR K_{REJECT}(^{6}p)
SouthEast(NorthEast(p)) = NorthEast(SouthEast(^{5}p) OR
K_{REJECT}(^{6}p)
SouthWest(NorthEast(p)) = ^{5}p OR K_{REJECT}(^{6}p)

Figure 24: HOS Specification of Data Type Place: Hexagon-Coverings of the Plane

```
PARTITION OF p IS

        5p | Equal (NorthEast(p), REJECT) = False
        6p | Equal (NorthEast(p), REJECT) = True


    North(NorthWest(p)) = NorthWest(North (7p) OR K_REJECT (8p)
    South(NorthWest(p)) = SouthWest(7p) OR K_REJECT(8p)
    NorthEast(NorthWest(p)) = North(7p) OR K_REJECT(8p)
    SouthEast(NorthWest(p)) = 7p OR K_REJECT(8p)
    SouthWest(NorthWest(p)) = NorthWest(SouthWest(7p)) OR
                                          K_REJECT(8p)


PARTITION OF p IS


        7p | Equal (NorthWest(p), REJECT) = False
        8p | Equal (NorthWest(p), REJECT) = True


    North(SouthEast(p)) = NorthEast(9p) OR K_REJECT(10p)
    South(SouthEast(p)) = SouthEast(South(9p)) OR K_REJECT(10p)
    NorthEast(SouthEast(p)) = SouthEast(NorthEast(9p)) OR
                                          K_REJECT(10p)
    NorthWest(SouthEast(p)) = 9p OR K_REJECT(10p)
    SouthWest(SouthEast(p)) = South(9p) OR K_REJECT(10p)


PARTITION OF p IS


        9p | Equal (SouthEast(p), REJECT) = False
        10p | Equal (SouthEast(p), REJECT) = True


    North(SouthWest(p)) = NorthWest(11p) OR K_REJECT(12p)
    South(SouthWest(p)) = South(SouthWest(11p)) OR K_REJECT(12p)
    NorthEast(SouthWest(p)) = 11p OR K_REJECT(12p)
    NorthWest(SouthWest(p)) = SouthWest(NorthWest(11p)) OR
                                          K_REJECT(12p)
    SouthEast(SouthWest(p)) = South (11p) OR K_REJECT(12p)


PARTITION OF p IS

        11p | Equal (SouthWest(p), REJECT) = False
        12p | Equal (SouthWest(p), REJECT) = True

END PLACE;
```

Figure 24: HOS Specification of Data Type Place: Hexagon-Coverings of the Plane

(Continued)

STRUCTURE: b = F-border(p);
WHERE b IS A BOOLEAN
WHERE p IS A PLACE;

$b = K_{False}({}^{1}p)$ OR $K_{True}({}^{2}p)$;

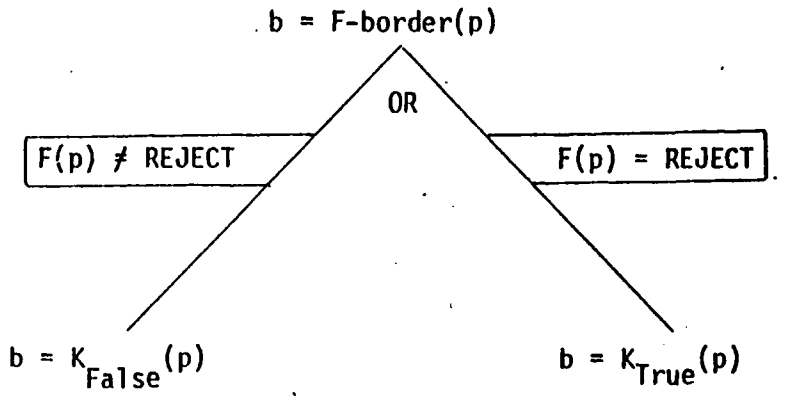
PARTITION OF p IS

${}^{1}p|$ Equal(F(p), REJECT) = False,

${}^{2}p|$ Equal(F(p), REJECT) = True;

SYNTAX: b = (p is a/an F border);
END F-border;

(a) AXES Syntax for the Border Structure



Syntax: b = (p is a/an F border)

F-border is the border structure

F can be any function from places to places

(b) Control Map for the Border Structure

Fig. 25 Structure That Tests Whether a Place is a Border Place in Any Direction

displacement square-coverings, on the one hand, and uniform-displacement square-coverings and hexagon-coverings, on the other hand, differ with respect to this structure only in that there are four of these basic directions for the former and six for both of the latter. This becomes relevant for an operation that decides whether an input place is a border region in any of the basic directions. As shown in Figure 26, the hexagon case is slightly more complex, though essentially of the same form, because of the presence of the two additional directions.

Very different control maps would be required, however, for the determination of the distances between places in the hexagon- and null-displacement square-covering cases. To determine the distance between two arbitrary places, we first determine their relative orientations by sending out search rays from each place in each of the basic directions until all intersections occur, and then counting the number of places in each direction. As shown in Figure 27, all four intersections produce the same count in the square case, but different counts arise from the six intersections in the hexagon case. Determining distance in the square case would thus involve a simple application of the Pythagorean theorem, whereas determining distance in the hexagon case requires the use of the appropriate version of the law of cosines. If x and y are the counts to an intersection, then the distance d is given, in the square case, by $d^2 = x^2 + y^2$, but in the hexagon case, by $d^2 = x^2 + y^2 - xy$ or $d^2 = x^2 + y^2 + xy$ depending on whether the angle opposite the d line is 60^o or 120^o. Determining this angle for a chosen intersection is an important part of the distance operation in the hexagon case that has no counterpart in the square case.

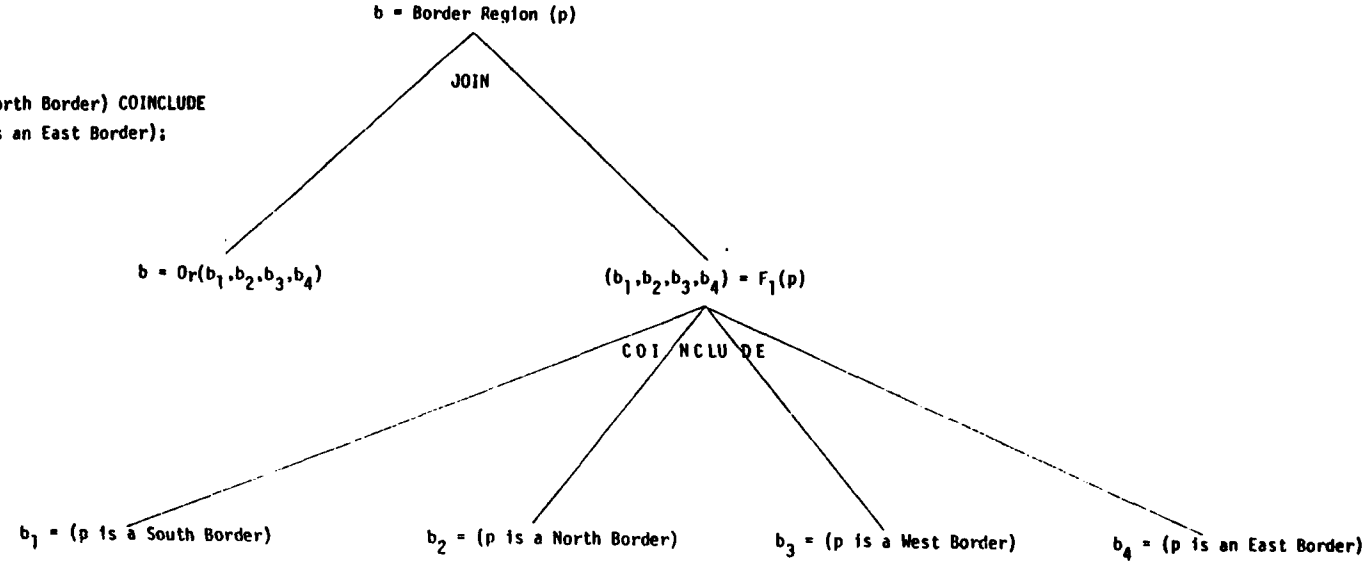
OPERATION: b = Border Region (p)
  WHERE b, b_1, b_2, b_3, b_4 ARE BOOLEANS;
  WHERE p IS A PLACE;

b = Or (b_1,b_2,b_3,b_4) JOIN (b_1,b_2,b_3,b_4) = F_1(p);

b_1 = (p is a South Border) COINCLUDE b_2 = (p is a North Border) COINCLUDE
  b_3 = (p is a West Border) COINCLUDE b_4 = (p is an East Border);

END Border Region;

(i) AXES Syntax for Operation Border Region



(ii) Control Map for Operation Border Region

(a) Null-Displacement Square Case

Fig. 26 Operation That Tests Whether a Place is a Northern, Southern, Eastern or Western Border

OPERATION: b = Border Region (p)

WHERE b, b_1, b_2, b_3, b_4, b_5, b_6 ARE BOOLEANS;

WHERE p IS A PLACE;

b = Or (b_1,b_2,b_3,b_4,b_5,b_6) JOIN (b_1,b_2,b_3,b_4,b_5,b_6) = F (p);

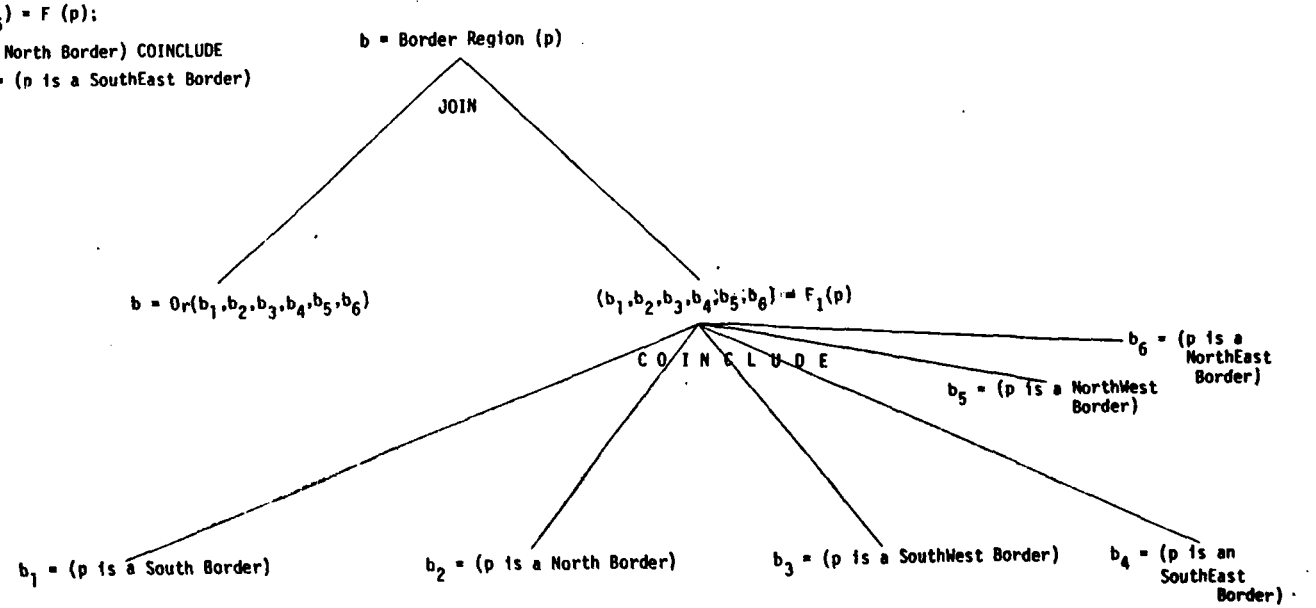
b_1 = (p is a South Border) COINCLUDE b_2 = (p is a North Border) COINCLUDE

b_3 = (p is a SouthWest Border) COINCLUDE b_4 = (p is a SouthEast Border)

COINCLUDE b_5 = (p is a NorthWest Border)

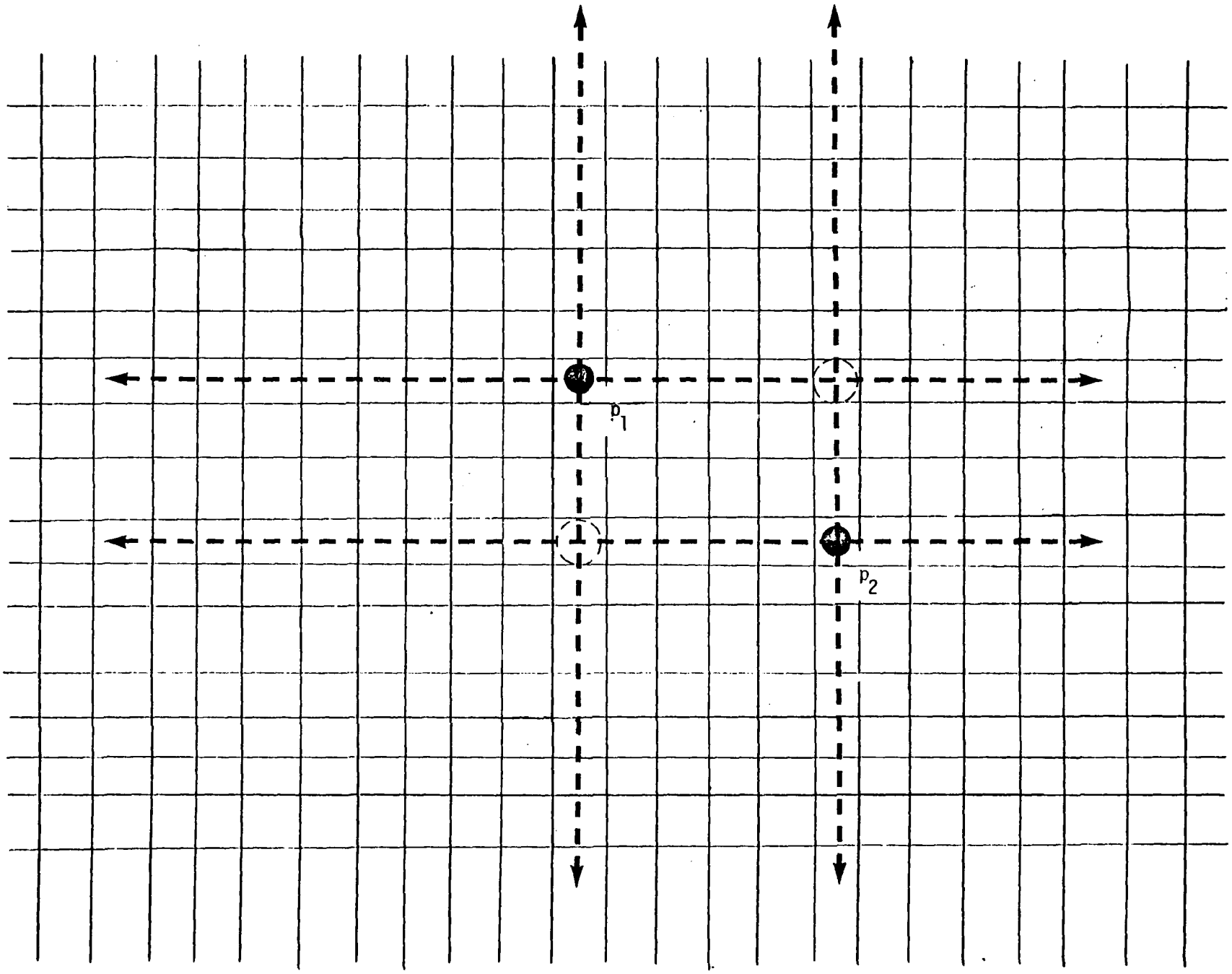
COINCLUDE b_6 = (p is a NorthEast Border);

(i) AXES Syntax for Operation Border Region



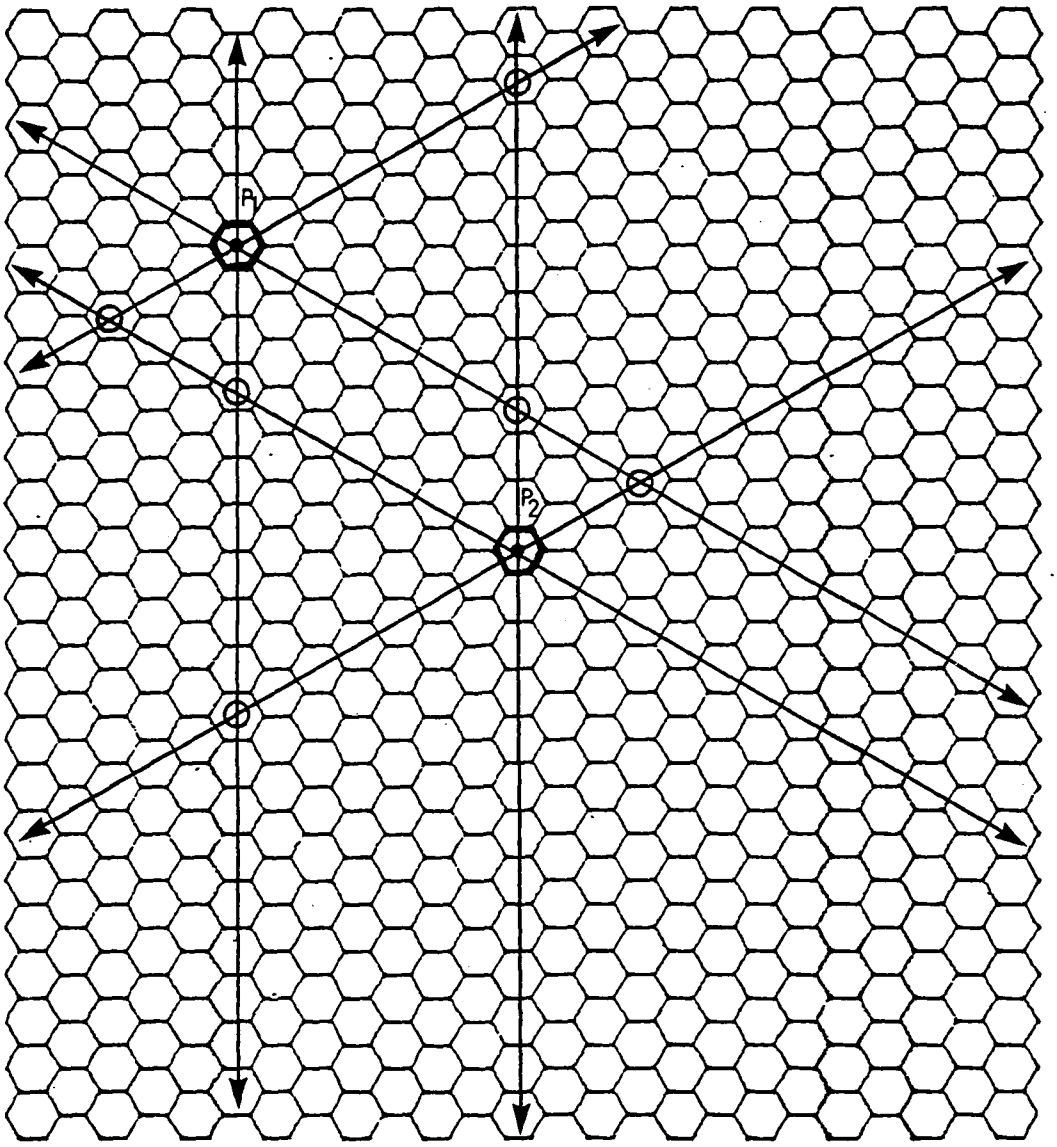
(ii) Control Map for Operation Border Region

(b) Hexagon Case



(a) Four Intersections in the Square Case: Same Count in Each Case

Figur 27: Search Rays for De ermination f Distance Between Arb'trary Place



(b) Six Intersections in the Hexagon Case: Different Counts in Each Case (Pairwise)

REFERENCES

[1] Marr, D., "Representing Visual Information," MIT, AI Laboratory, 374, Memo No. 1, MIT, Cambridge, MA, 1976.

[2] Marr, D., "Artificial Intelligence - A Personal View," in Artificial Intelligence 9 (1977), North-Holland.

[3] Marr, D. and Poggio, T. (1979), "A Computational Theory of Human Steres Vision," in Proc. R. Soc., B. 204, pp. 301-328, London.

[4] Marr, D. and Ullman, S., (1979), Directional Selectivity and Its Use in Early Visual Processing," MIT, AI Laboratory, Memo 524, MIT, Cambridge, MA.

[5] Zadeh, L.A., "Fuzzy sets as a basis for a theory of possibility," Fuzzy Sets and Systems, Vol. 1, pp. 3-28, 1978, North Holland.

[6] Marr, D. and Nishihara, K., (1977), "Representation and Recognition of the Spatial Organization of Three Dimensional Shapes," MIT AI Laboratory, Memo 416, MIT, Cambridge, MA.

[7] Marr, see [8].

[8] Marr, D., VISION (1981) Freeman, S-F.

[9] Vaina, L., (1979), "Image Understanding Methods to Deal with Uncertainty in Radar Image Analysis," Proc. New Orleans Workshop, 21-21 February 1980.

[10] Marr, D. and Vaina, L., (1980), "Representing Moving Shapes," MIT, AI Laboratory Memo, MIT, Cambridge, MA.

[11] Bobrow, D. & Winograd, T., "An Overview of KRL, a Knowledge Representation Language," Cogn. Science, pp. 13-45, 1977.

[12] Zadeh, L.A., "PRUF - A Memory representation language for natural languages," Int. J. for Man-Machine Studies, Vol. 10, pp. 395-460, 1978.

[13] Kotoh, K. & Thiramatsu, K., "A representation of pattern classes using the fuzzy sets," Systems, Computers, Controls, pp. 1-8, (1973).

[14] Vaina, L. & Greenblatt R., "The use of thread and memory in anomia and childhood concept learning," MIT AI Laboratory, WP 196, MIT, Cambridge, MA., 1979.

[15] Vaina, L., "Towards a computational theory of semantic memory," MIT AI Laboratory Memo 456, MIT, Cambridge, MA., 1980.

[16] Henrix, G., Partitioned Smeantic Networks, Speech Understanding Research 1976, SRI.

[17] Minsky, M.L., "A Framework for Representing Knowledge," in Winston (ed.), The Psychology of Computer Vision, 1975.

[18] Vaina, L., Lecture logico - mathematique de la narration, E.R.M.E., Paris, 1977.

[19] Gorice, P., "Logic and Converstation," U.P. Cole and J.L. Morgan, "Syntax and Semantics": Speech Acts, Vol. 3, 1975.

[20] Collins, A. and Quillian, M., "Experiments on semantic memory and language comprehension, Gregg, W., (ed.), "Cognition in learning and memory," New York: Wiley, 1972.

[21] Rumelhart, D. and Norman, D., "Active Semantic Networks as a Model of Human Memory."

[22] Lindsey, P.M. and Norman, D.A., "Human Information Processing," New York. Academic Press, 1977.

[23] Norman, D.A., "Models of Human Memory," New York, Academic Press, 1979.

[24] Fahlman, NETL: "A System for Representing and Using Real World Knowledge," MIT Press, Cambridge, MA., 1979.

[25] Zadeh, L., "Approximate Reasoning," UCCS - Berkley, Memo, 1977.

[26] Vaina, L. & Hintikka, J., (eds.), Cognitive Constraints on Communication: Representations and Processes. Reidel-Dordrecht, Holland (1981).

[27] Vaina, L., Semiotics of With, Versus(1).

[28] Binford, T. & Horn, B., "The Binford-Horn Line Finder," MIT, AI Laboratory WP 16, Cambridge, MA., 1971.

[29] Roberts, L.G., Machine Perception of Three-Dimensional Solids, in J.G. Tippet (ed.) Optical and Electro-Optical Inform. Processing., pp. 159-197, MIT Press, Cambridge, MA., 1965.

[30] Winston, P., Learning structured descriptions from examples, Winston, P. (ed.), The Psychology of Computer Vision, MIT Press, Cambridge, MA., 1975.

[31] Falk, G., "Computer Interpretation of Imperfect Line Del? is a Three Dimensional Scene" PG. Thesis, AI Memo 131, California, 1970.

[32] Cushing,S. "The Formal Semantics of Quantification," UCLA Ph.D. dissertation, available from University Microfilms, Ann Arbor, Michigan and Indiana University Linguistics Club, Bloomington (1976).

[33] van Fraassen, B.C. Formal Semantics and Logic. New York: The Macmillan Company (1971).

[34] Cushing, S. "Semantic Considerations in Natural Language: Crosslinguistic Evidence and Morphological Motivation," Studies in Language, 3,181-201 (1979).

[35] Stenning, K. "Articles, Quantifiers, and their Encoding in Textual Comprehension," In Freedle, R.O. (ed.), Discourse Production and Comprehension. Hillsdale, N.J.: Lawrence Erlbaum Associates (1977).

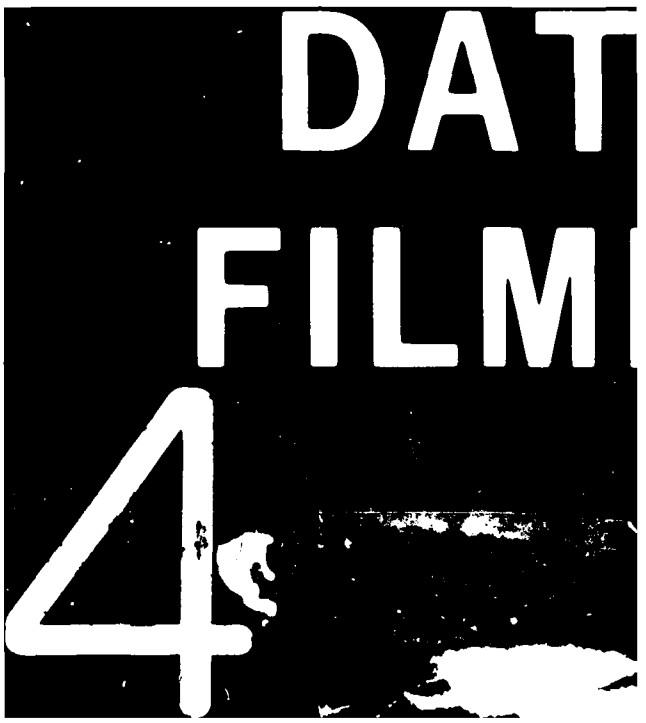
[36] Cushing, S. "Lexical Functions and Lexical Decomposition: An Algebraic Approach to Lexical Meaning," Linguistic Inquiry, 10, 327-345 (1979).

[37] Logan, B.F., Information in the Zero-Crossings of Bandpass Signals. Bell System Technical Journal J6, (487-510), (1977).

[38] Marr, D. and Hildreth, E., Theory of Edge Detection. MIT AI Laboratory Memo 518, (1979).

[39] Marr, Poggio and Ullman, Bandpass Channels, Zero-Crossings and Early Visual Information Processing, J. Opt. Soc. Am. (in press), (1980).

[40] Barcewell, The Fourier Transform and its Application, New York, McGraw-Hill, (1965).


DAT
FILM
4