AD-A096 569 ILLINOIS UNIV AT URBANA COORDINATED SCIENCE LAB F/G 9/4
A CELLULAR AUTOMATA APPROACH TO COMPUTER VISION AND IMAGE PROCE--ETC(U)
SEP 80 G D HADDEN N00014-75-C-0612
UNCLASSIFIED T-100 NL

1 OF
AD
A096 569

LEVEL

REPORT T-100 SEPTEMBER, 1980

COORDINATED SCIENCE LABORATORY

# A CELLULAR AUTOMATA APPROACH TO COMPUTER VISION AND IMAGE PROCESSING

GEORGE DANIEL HADDEN

UNIVERSITY OF ILLINOIS – URBANA, ILLINOIS

(6) A CELLULAR AUTOMATA APPROACH TO
COMPUTER VISION AND IMAGE PROCESSING.

(14) T-100 By

(10) George Daniel Hadden

(9) Doctoral thesis,

Thesis Advisor: David L. Waltz

(11) Aug 80

(12) 155

| | |
|---|---|
| Accession For | |
| NTIS GRA&I | ☒ |
| DTIC TAB | ☐ |
| Unannounced | ☐ |
| Justification ~~Per letter on file~~ | |
| By | |
| Distribution/ | |
| Availability Codes | |
| Dist | Avail and/or Special |
| A | |

This work was supported in part by the Office of Naval Research under contract N00014-75-C-0612 new
(15)

099900

# A CELLULAR AUTOMATA APPROACH TO COMPUTER VISION AND IMAGE PROCESSING

George Daniel Hadden, Ph.D.
Coordinated Science Laboratory and
Department of Electrical Engineering
University of Illinois at Urbana-Champaign, 1980

This dissertation describes a system called HEXVIS, which performs operations on scenes. All operations are carried out in a hexagonal array of cellular automata-like objects which corresponds to that scene. The system can perform (for instance) the following tasks: recognition of edges, corners, and axes of symmetry, texture discrimination, determination of areas, and generation of Voronoi tessellations.

First, the scene is embedded in the hexagonal array, then, the cells pass messages describing the cells' contents to their neighbors which, in turn, pass them on. As these messages pass through cells, they can interact with each other and with the contents of the cell in which they find themselves. The cells all perform the same operation or group of operations in parallel on their visiting messages. As a consequence of these operations, the states of some cells change in a way which indicates that they correspond to "interesting" parts of the scene.

This process can be repeated recursively using the altered states of the cells as new messages to be broadcast.

In addition a new algorithm is presented which shrinks binary scenes in a hexagonal array. It is proved that all scenes with (at most) simply connected holes are transformed into a set of isolated points, each corresponding to a connected object in the original scene. This shrinking algorithm is embedded in HEXVIS.

# A CELLULAR AUTOMATA APPROACH TO COMPUTER VISION AND IMAGE PROCESSING

BY

GEORGE DANIEL HADDEN

B.S., Purdue University, 1973
M.S., University of Illinois, 1977

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Electrical Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 1980

Thesis Advisor: Professor David L. Waltz

Urbana, Illinois

## ACKNOWLEDGMENTS

I wish to express my gratitude to a number of people:

To my advisor David Waltz for his many helpful suggestions including that of passing messages in a hexagonal array;

To my colleagues at the Coordinated Science Laboratory -- especially Tim Finin and Harry Tennant -- for long hours spent in inspiring conversation about our work;

To my parents, whose financial support during my undergraduate years, and emotional support always, helped me discover the joy in learning;

To my children, Julie and Jenny, for giving me hugs when I needed them;

And most importantly, to my wife and best friend, Carrie Kostroski, without whose love, support, and patience, I might never have finished.

## TABLE OF CONTENTS

## LIST OF FIGURES

# CHAPTER 1

# INTRODUCTION

## 1.1 Introduction to the System

This dissertation describes a system called HEXVIS which performs vision and image processing operations on a static scene. All operations are carried out in parallel in a hexagonally tessellated array of cellular automata-like objects which corresponds to that scene. Operations which have been examined include (among others) the following: edge, corner, and symmetry axis finders, a binary shrinking algorithm, area determiners, region growing algorithms, and texture discriminators and recognizers.

The cells communicate by passing messages according to a set of production rules. This process is user programmable and can be repeated recursively using the altered states of the cells as new messages to be broadcast.

## 1.2 The Basics

HEXVIS differs from a traditional cellular automata system in two important ways. First, the automata (cells) are not finite state machines. This difference manifests itself in two ways: 1) the amount of information which can flow through a cell is unbounded and 2) the cells can execute unrestricted programs which use that information as data. The memory available to each cell has also been assumed to be unbounded, although it is clear that if the system were to be realized in hardware, memory would be limited. For most purposes, each cell may be assumed to be a reasonably powerful computer (i.e., powerful enough to execute LISP, say).

A second difference is one of convention. Most of the literature on cellular automata concerns itself with arrays which are rectangular. My system uses a hexagonally tessellated cellular array. The reasons for this decision will be presented in Chapter 2.

As an aside, it turns out that a hexagonal array can be thought of as a rectangular array with an odd neighborhood definition (as shown in figure 1). In fact, I make use of this relationship in simulating the hexagonal array.

```
0 - 0 - 0 - 0 - 0 - 0        0 - 0 - 0 - 0 - 0 - 0
| / | / | / | / | / |         \ / \ / \ / \ / \ / \
0 - 0 - 0 - 0 - 0 - 0          0 - 0 - 0 - 0 - 0 - 0
| \ | \ | \ | \ | \ |   <-->   / \ / \ / \ / \ / \ /
0 - 0 - 0 - 0 - 0 - 0        0 - 0 - 0 - 0 - 0 - 0
| / | / | / | / | / |         \ / \ / \ / \ / \ / \
0 - 0 - 0 - 0 - 0 - 0          0 - 0 - 0 - 0 - 0 - 0
```

Figure 1  Simulation of a Hexagonal Array
With a Rectangular Array

HEXVIS begins with a rectangular array of picture elements. It processes this information and embeds it in the hexagonal array. The "processing" referred to consists of extracting local gradient and laplacian information (for instance), forming appropriate list structures and inserting them into the cells. The mechanics of this process will be discussed in detail later on.

The hexagonal array is much more complex than the original rectangular grey level array. Each of its elements can contain an arbitrary LISP list structure. In general this list structure has at least two constituents: 1) the "state" of the cell (where "state" here has a meaning slightly different from the one used in cellular automata theory), and 2) the messages resident in the cell. Furthermore, the cells are designed to accommodate user defined additions. Such things as the cell's coordinates, a record of previous states of the cell, and the original information contained in the cell can all be useful to the cell in its computation. These additions are easily made.

Each cell (with a few exceptions) forms a message describing the cell which is to be broadcast to the other cells in the array. The messages pass from cell to cell in a synchronous way. That is, time in the system is quantized. At each time period, each message in the array moves one step into one or more of the neighbors of its host cell.

Each message is again a LISP list whose first element is a directional component called the "heading" which specifies in which direction the message is travelling. Initially, of course, the message is travelling in <u>all</u> directions, and this is indicated by the message's heading.

A set of production rules is associated with the headings. These rules specify, for a given heading, where (i.e., in which of the host cell's neighbors) and with what new heading the attached message will appear at the next time period.

For instance, a message at time zero (which has a heading specifying all directions) will appear in all six neighboring cells at time one. Each of the six copies of the original message will have a different heading indicating that it is travelling outward from the cell which originated it. The net effect of these productions is that a message spreads outward from its originating cell like a ripple from a pebble thrown into a pond. A detailed description of message passing and the production rules appears in chapter 2.

The rest of the message contains the information from the message's originating cell: data such as gradient direction and magnitude, laplacian, and, in higher layers (later in the processing of the picture), texture information, shape description, etc. As these messages travel around the cellular world, the information they carry can interact in various ways with its environment. The simplest type of interaction is one in which messages resident in each cell are compared in a pairwise manner. As it turns out, this particular class of interactions is quite powerful. Its members can find edges and corners, fill in incomplete lines, and infer axes of symmetry, among other things.

At the other extreme of complexity are interactions involving complex statistical analysis of messages integrated over many time units. Textural discrimination is an example of a task requiring this type of higher level processing.

## 1.3 Why?

What good is this system? There are three reasons HEXVIS is worthwhile. First of all, it represents an interesting information processing paradigm in its own right. The evolution of a cellular automata-like system based on state changes which are brought about by message propagation and interaction is a significant departure from (and extension to) the traditional cellular automata model. It is reminiscent in some ways of Hewitt's and Baker's "actor" formalism [Hewitt, 1977] in that both systems have parallel processes which communicate through passing of messages (although HEXVIS is synchronous).

Secondly, the system is general enough to be an abstract vehicle for parallel algorithms. It includes as a subset all of cellular automata and could be used to model even asynchronous processes.

Lastly, the system is potentially quite fast. Most vision and image processing systems (including this one as it is now implemented) perform many trivial and not so trivial operations on vast amounts of data in a serial manner. Obviously, if these operations were done in parallel, the systems' speeds would be much improved. HEXVIS and its algorithms are designed with this fact in mind. As more and more sophisticated parallel machines become available, a system like HEXVIS will become more attractive. Furthermore, at the rate chip densities have been increasing, it is at least conceivable that in the near future, the entire HEXVIS system could be housed on one chip. Some applications that spring to mind are: smart television cameras which could (say) follow a person around the stage, microscopes, telescopes and televisions with built-in real time image enhancement, and mobile household devices.

## 1.4 Outline of the Dissertation

After introducing the topic in chapter 1, I will describe the basic model in chapter 2. Chapter 3 covers related work while chapters 4 and 5 discuss specific algorithms which have been implemented in HEXVIS. In chapter 6, I present a new algorithm (and its proof) which shrinks hexagonal binary scenes to a set of points. Chapter 7 contains some concluding remarks and suggestions for future work. Finally, an appendix discusses the history of cellular automata.

# CHAPTER 2

# THE BASIC MODEL

## 2.1 The Hexagonal Grid

The cells in HEXVIS are arranged in a hexagonal pattern on the plane. Each cell can directly communicate only to its six neighbors. Communication with cells further away is done indirectly by propagating messages which start at each cell at time zero (the system is synchronous) and spread outward in a hexagonally shaped wavefront. The message propagation is governed by certain rules which are described later in this chapter. Each cell is a reasonably powerful computer and can use the contents of the messages which pass through it in its computation in order to discover interesting information about the scene.

### 2.1.1 WHY HEXAGONS?

There are only three regular tessellations of the plane: triangular, square, and hexagonal. It is interesting to note that the hexagonal and triangular tessellations are duals in the sense that if one draws a line from the center of a hexagonal cell to the center of each of its neighbors, a triangular tessellation is obtained. The reverse happens if one starts with a triangular tessellation: one obtains a hexagonal tessellation. The square tessellation is its own dual.

A triangular scheme has no recognizable advantages over the others. In fact, it seems to combine the worst features of square and hexagonal schemes, while offering the advantages of neither. There are two undesirable features which a triangular array has in common with the hexagonal case. The first problem is that representing a triangular (or hexagonal) array in the inherently square arrays of LISP (as in most computer languages) requires extra overhead to keep track of the correspondence between the two. The other problem that the representation of triangular and hexagonal arrays share is that the computation to determine a cell's neighbors depends on where in the array the cell resides.

Triangular and square arrays, on the other hand, share the problem of having more than one sort of neighbor. In the case of the square array, each cell has four edge neighbors and four corner neighbors. The triangular case is even worse: three edge neighbors, three of one type of corner neighbors, and six of another type of corner neighbors.

The choice between a hexagonal and a square representation, however, is not quite so clear. There are several advantages to using a hexagonal array of cells rather than a square one. First, the shape of the messages' "wavefront" is a hexagon. In a square array the messages' wavefront would propagate in a square (or diamond) shape. (Rosenfeld [1979] shows a propagation scheme for square arrays whose wavefront is in the shape of an octagon, but for various reasons, this scheme can not be cast in a message passing mold similar to my own.) Since a hexagon is closer in shape to a circle than a square is, the distance at a particular time from a cell to any of the messages which it has generated does not vary as much as in a square propagation scheme. (The ratio in a square of shortest to longest distance is 1/(sqrt 2) or .707, whereas in a hexagon, that ratio is (sqrt 3)/2 or .866 -- much closer to unity.) In a hexagonal array, then, if two messages which have traveled for the same amount of time meet, they have traveled approximately the same distance -- a useful fact for finding axes of symmetry, centers of circles, etc.

Further advantages derive from the fact that six is the maximum number of isotropic neighbors a cell can have in any planar tessellation. This is important for two reasons. First, it is desirable for a cell to have as many neighbors as possible in order to distribute the information more evenly. Second, we would like all of these neighbors to look alike to the cell, hence the wish for isotropy. As an example of a cellular world without isotropy, consider a square tessellation in which each cell is defined to have eight neighbors. The neighbors on the diagonal are different than the neighbors on the four sides: they are farther away, they share no borders, etc.

There is also a problem with ambiguity in the connectivity of a square array which is not present in the hexagonal array. The two traditional (and natural) choices for neighborhoods in the square array are the four-cell and the eight-cell neighborhood. In the former case, the question of whether removing or adding a point will alter the connectivity of the scene can not be assured without knowing the state of the corner cell, as well. Figure 2 illustrates this. If D, E, and B are in the "one" state, we can not know whether removing E will disconnect B and D without knowing the state of A.

```
A B C
D E F
G H I
```

Figure 2 Connectivity in a Square Array

In the hexagonal tessellation, there is no such ambiguity. All the neighbors of a cell share an edge with that cell. Furthermore, two neighbors of a cell which share adjacent edges of the cell also have an edge in common with each other.

Another advantage of a hexagonal array is this: Since vertices in a hexagonal array have only three edges, a region of the plane can be encoded as a bit string. To illustrate, imagine a bug crawling along one of the edges. When it comes to a vertex, it can either turn to the right or to the left (assuming it will not back up, of course). This information can be encoded as a series of zeros and ones as the bug crawls around the perimeter of the region. In a square array, on the other hand, the bug has three choices at each vertex (right, left, and straight ahead), doubling the number of bits per choice.

An interesting aside is that a hexagonal array actually closely approximates the pattern of rods and cones on the human retina [Lindsay, 1972]. Thus, a low-level vision model using hexagonal arrays might stand a better chance of modeling human vision than one using a square array.

Of course, there are some <u>disadvantages</u> to using hexagonal arrays instead of square ones. As I pointed out before, array simulation, addressing schemes, and neighboring cell calculation are more difficult and time consuming because of the inherently rectangular nature of computer arrays (at least in LISP). Pictures need some preprocessing in order to be loaded into hexagonal arrays because camera hardware in general produces pictures in the form of a rectangular grid. These difficulties are not insurmountable (as will be shown) and are more than outweighed by the advantages.

### 2.1.2 Mappings Between Rectangular and Hexagonal Arrays

Since most television cameras are set up to output pictures in a square array, how can we transform this to a hexagonal world? Also, since computer arrays are rectangular in nature, what is an efficient way of storing these hexagonal arrays in the computer? Two transformations are involved here.

As shown in figure 3, to derive a rectangular array from a hexagonal one, every other row is moved to the right and stored in a row in the rectangular array. The lines between the cells in both parts of the figure represent neighbor relationships. The only problem with this method is that the coordinates of the neighborhood set of a cell depend upon in which row the cell resides. (For instance, in row one a cell's zeroeth neighbor is directly above it in the square array while in row two, the corresponding neighbor is above and to the right.) Consequently calculation is slowed somewhat.

An alternate scheme is to only use every other element of the array. However, I felt that the resultant increase in speed would not make this waste of space worthwhile. Yet another scheme would be to move every row to the left as in

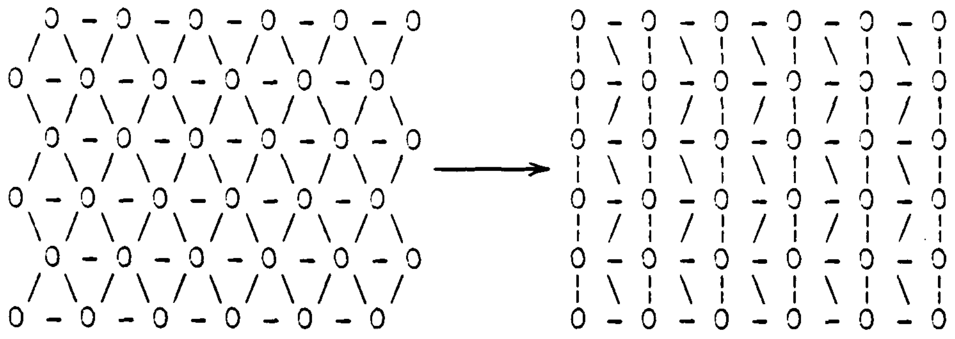


Figure 3 Hexagonal to Square Conversion

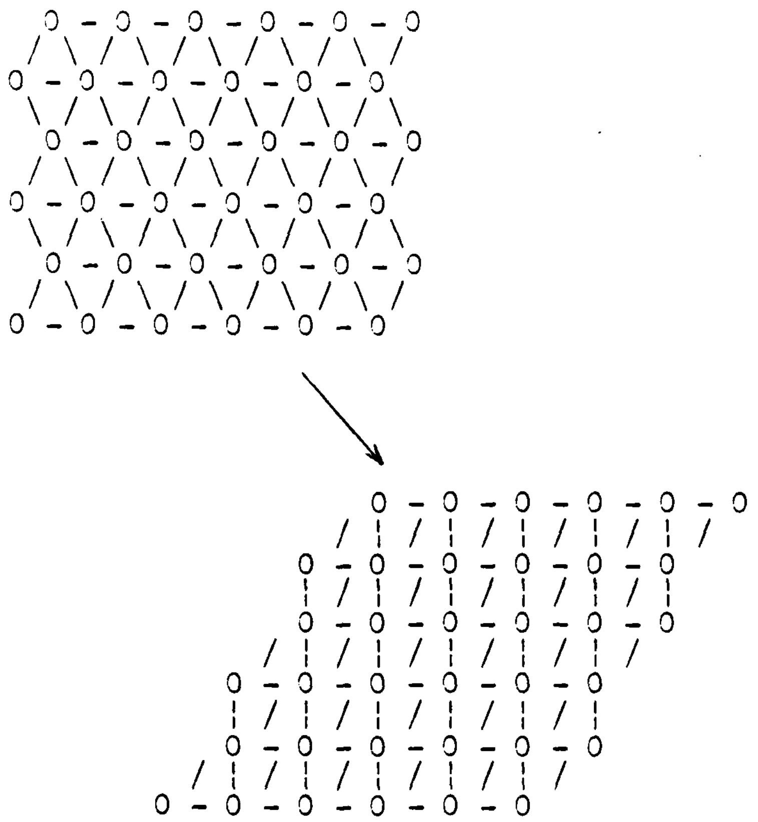


Figure 4 Hexagonal to Square Conversion

figure 4. Then every neighborhood set would be computed similarly, but pictures would be skewed -- that is, if we wanted to store a picture in a rectangular array according to this method, the hexagonal array corresponding to the original scene would have to be a parallelogram slanted to the left, hence, so would the original scene.

Ahuja [private communication] has suggested a variation of this latter method which maps rectangular pictures to rectangularly stored hexagonal arrays and still allows the neighborhood relation to be the same for all cells. In this case the parallelogram which results from shifting every row to the left is cut on the left side. The smaller piece is then fitted onto the other side of the parallelogram to make a rectangle. The neighborhood function must use modulo functions in order to be correct for all cells.

```
. X . X . X . X .

. . . . . . . . .

X . X . X . X . X

. . . . . . . . .

. X . X . X . X .

. . . . . . . . .
```

Figure 5 Mapping a Hexagonal Array to a Square Array

There are a number of possible methods of getting a hexagonal model from a picture stored in a square array. The most reasonable methods all involve picking every fourth picture cell as in figure 5. (The X's correspond to the sites in the rectangular array where the hexagonal cells will be located.) The problem is, however, what to do with the other three-quarters of the picture. Can it contribute in some way to the hexagonal model? In fact, it can; the left-over cells can be used to derive gradient information. For example, suppose the picture cell in question and the cells above and below it (remember, we are talking about the rectangular array of grey levels from the camera) have a brightness level of 0.5 on a scale of 0 to 1. Suppose also that the cell to the right and the two cells above and below that one have a brightness level of 0 and that the three corresponding cells on the left have a brightness level of 1. This situation implies that there is a brightness gradient at that cell pointing to the left. This information can be encoded into the cell in the hexagonal world and broadcast in messages from that cell. Note that this technique is a modification of the Sobel operator [Duda, 1973].

Another possible use of the left-over cells would be to simply average their brightness levels with the brightness level of the central cells. As with all averaging techniques, this would have the effect of eliminating some noise in the picture, but, at the same time it would degrade the resolution.

Notice that there are two distinct ways of picking the central cells from a rectangular array (see figure 6). I chose the method shown in figure 6a. My reason for making that seemingly arbitrary choice has to do with a peculiarity of the vision system at the Coordinated Science Laboratory. This system produces a rectangular array of picture cells such that each cell represents an area which is approximately 1.2 times as wide as it is high. When the central cells are chosen as in figure 6a, the hexagons are much closer to being regular than they would be if chosen as in figure 6b. (The important dimensions are shown on the figure with the vertical distance between rectangular cells normalized to 1.0 and the horizontal distance to 1.2.)

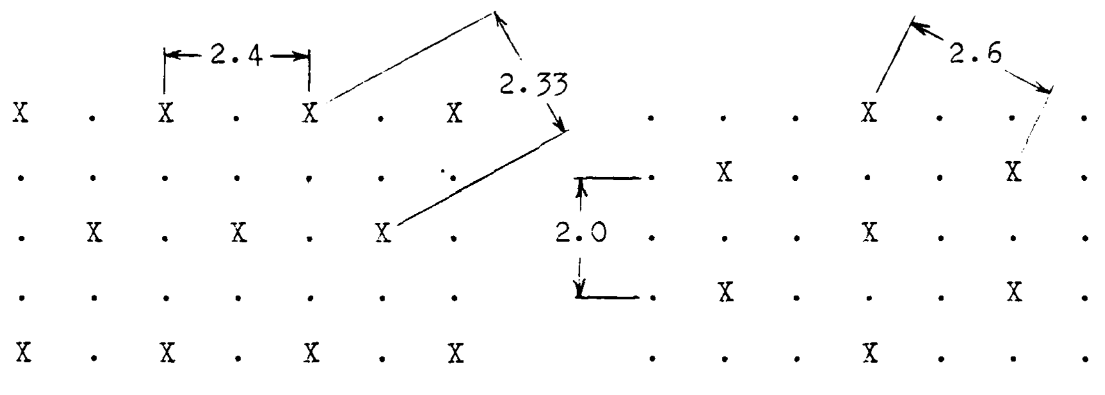


Figure 6 Two Ways of Choosing a Hexagonal Array from a Rectangular Array

## 2.2 Message Passing

### 2.2.1 Production Rules

Messages carry information from one part of the cellular world to another. They spread outward from their originating cells in all directions like a ripple from a pebble thrown into a pond. How is this done? How do the many copies of a message know in which direction to travel and in which cell or cells to appear at the next time unit? This is accomplished by using a set of production rules which operate on the heading of the message. The "heading" is a part of each message copy. It determines in which direction a message is traveling. By examining a heading in a message copy and the set of production rules, HEXVIS can determine into which cells the message is to be propagated next and what headings to give those copies in the cells in which they appear. The workings of headings and production rules are described in detail in section 2.3.3.

The rules are designed to fulfill three goals: (1) given enough time, messages from a single originating cell should be able to pass through all other cells in a unique way; (2) the messages should form a number of distinct and equal rays emanating from the originating cell; and (3) the shape of the propagating messages, i.e. the "wavefront", should be as close to a circle as possible. The four rule propagation techniques discussed in this section satisfy all of these goals to varying degrees.

Figure 7 shows graphically one of the message propagation schemes. This figure is an example of a type of diagram common in this section. Since it may not be clear intuitively exactly what these figures represent, I will explain them.

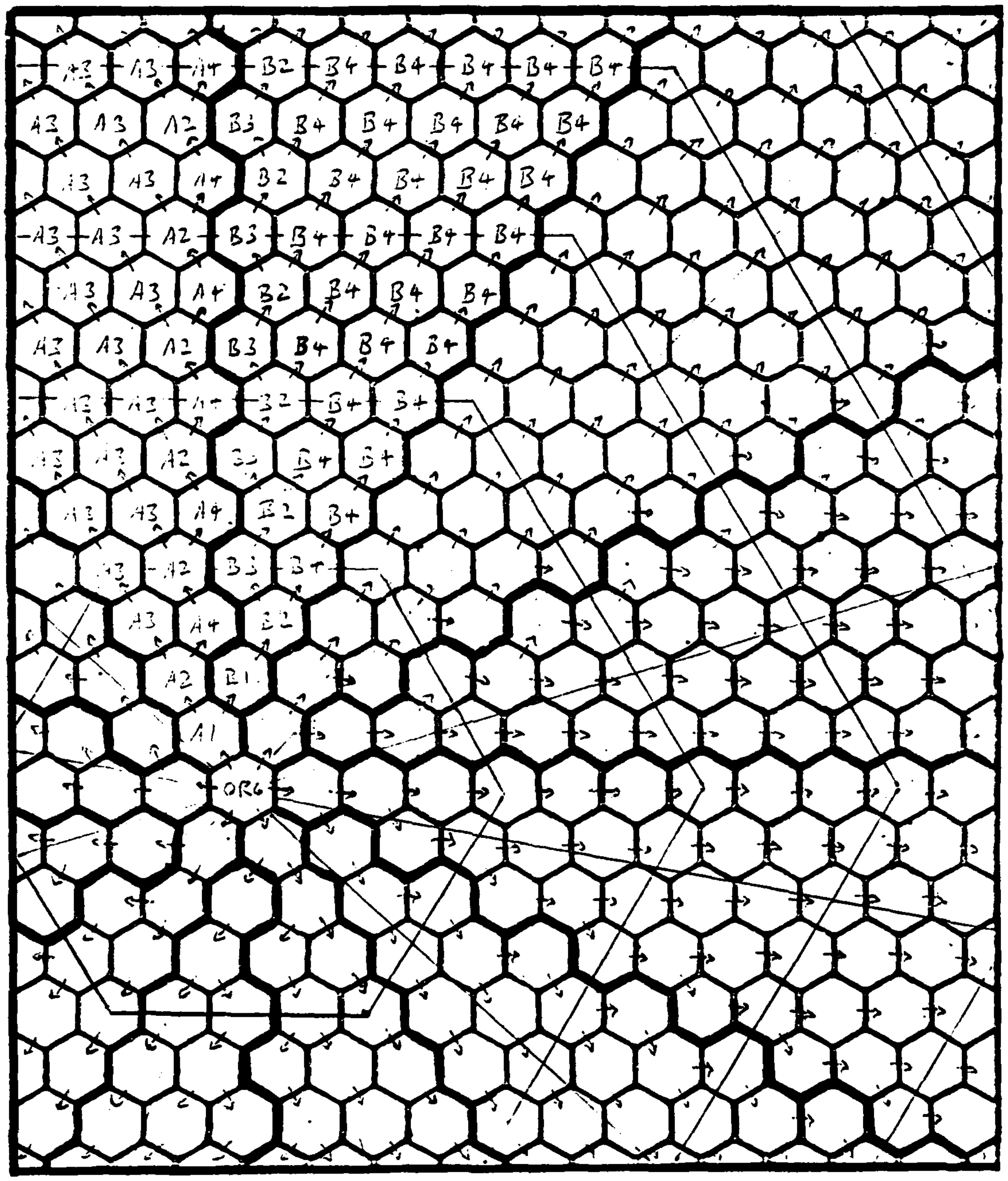


Figure 7 Propagation Method 1

Notice the cell labelled "ORG" with six arrows emanating from it. I will call this cell "C" (the cell is not called "ORG" because that is the name of the heading of its message at time zero) and be concerned for purposes of this explanation only with the copies of the message that originate there (keeping in mind that potentially all of the cells could have produced messages). Notice that these six arrows point to the six adjacent cells. This indicates that the original message will, at Time = 1, have appeared in these six cells. Each of these six messages will of course have a different heading so that the production rules can distinguish them. However, they are in all other respects duplicates of the original message.

In general then, when a cell in the diagram has one or more arrows pointing into other cells, it will transfer any messages it contains <u>which originated from cell C</u> (it may contain others) to the cells at the other end of the arrows by the following time period. For example, from Figure 7 we can infer that if at time period T there is a message with heading B2 in a cell, then at time T + 1, both the cell to the upper left and the cell to the upper right will have a copy of that message with headings B3 and B4 respectively. At time T + 2 then a cell which had a message with B3 in it will produce that same message with the new heading "B2" in its upper right neighbor. And so on.

Not all of the rules are included in these diagrams. However, they can be inferred by rotating the figure sixty degrees clockwise and "adding 2" to the letter modulo L until the sector of interest is available. So, for example, we can see that H2 in figure 7 produces an H3 in its lower right neighbor and an H4 in its lower left neighbor even though this is not shown in the diagram. Each concentric hexagon centered on cell C is formed from lines which run through all the cells which have C's message at the same time period. This is called the "wavefront" of the message.

Remember that in actual operation this process is going on for many -- perhaps all -- of the cells in the array. It is quite possible, for example, for one cell to have ten to twenty resident messages which must be parceled out in various combinations to its neighbors.

Again referring to figure 7, there are two sets of lines running more or less radially outward from C. The heavy crooked lines delimit the sectors. They represent divisions which define the twelve directions in which the message can travel, i.e. all messages between two adjacent lines (again recall we are talking only about the messages from cell C) are said to be travelling in the same direction even though at the cell-to-cell level they change direction frequently. Notice that messages never change their direction, that is, if a message copy produces another in an adjacent cell, the new message keeps the direction of the old message. The straight lines through the middle of each sector represent exactly that -- the center of the sector.

The labels in the cells (ORG, A2, B4) are the names of headings. These are solely for human convenience; the program uses bit strings to determine message routing; in this way processing speed is greatly increased.

The design of the production rules underwent several iterations. Figure 7 shows a representation of the original scheme proposed by Waltz [1978]. We will call this "method 1".

Method 1 has two very attractive advantages over the present method of message propagation: First, the set of production rules is simpler (a total of seventy-two rules versus three hundred forty-two rules in the present system). Using it would undoubtedly produce a significant increase in the speed of the system. The second advantage is that the angle subtended by each of the twelve sectors is exactly equal. In the present system, adjacent sectors differ by a small amount, causing slight variation in behavior depending on direction being considered.

Now for the disadvantages: Notice that each segment is situated so as to cover the area between adjacent numbers of an imaginary superimposed clock. Why is this a problem? The answer stems from the fact that many objects in the world have important visual features which are oriented either vertically or horizontally. This is a fact which is characteristic not only of artifacts, but also of many natural objects and phenomena (e.g., trees and the horizon).

Notice that in method 1, the boundaries between some of the segments are oriented in a horizontal direction and others are oriented in a vertical direction. Because of this, a direction which should be interpreted as pointing directly up could be labeled as either 11:00 or as 12:00 depending on slight perturbations of the scene. A similar problem exists for the horizontal direction. Furthermore, this problem can not be easily removed; interpretation of some directions will always be ambiguous due to the fact that the directions are quantized. We can, however, choose quantization regions which group directions we wish to regard as similar. One solution I considered was to rotate the cellular world by fifteen degrees, but the consequent problems of mapping a non-rotated square array into a rotated hexagonal array are very messy unless the camera taking the picture is also rotated by the same amount, thus making this an unsatisfactory solution.

Method 2 is shown in figure 8. It is quite a bit more complex than method 1. It does solve the main problem of method 1, but it produces some new problems of its own.

The problem of rule proliferation was discussed somewhat above. This method has a total of one hundred ninety-two rules. The motivation for dropping this method and

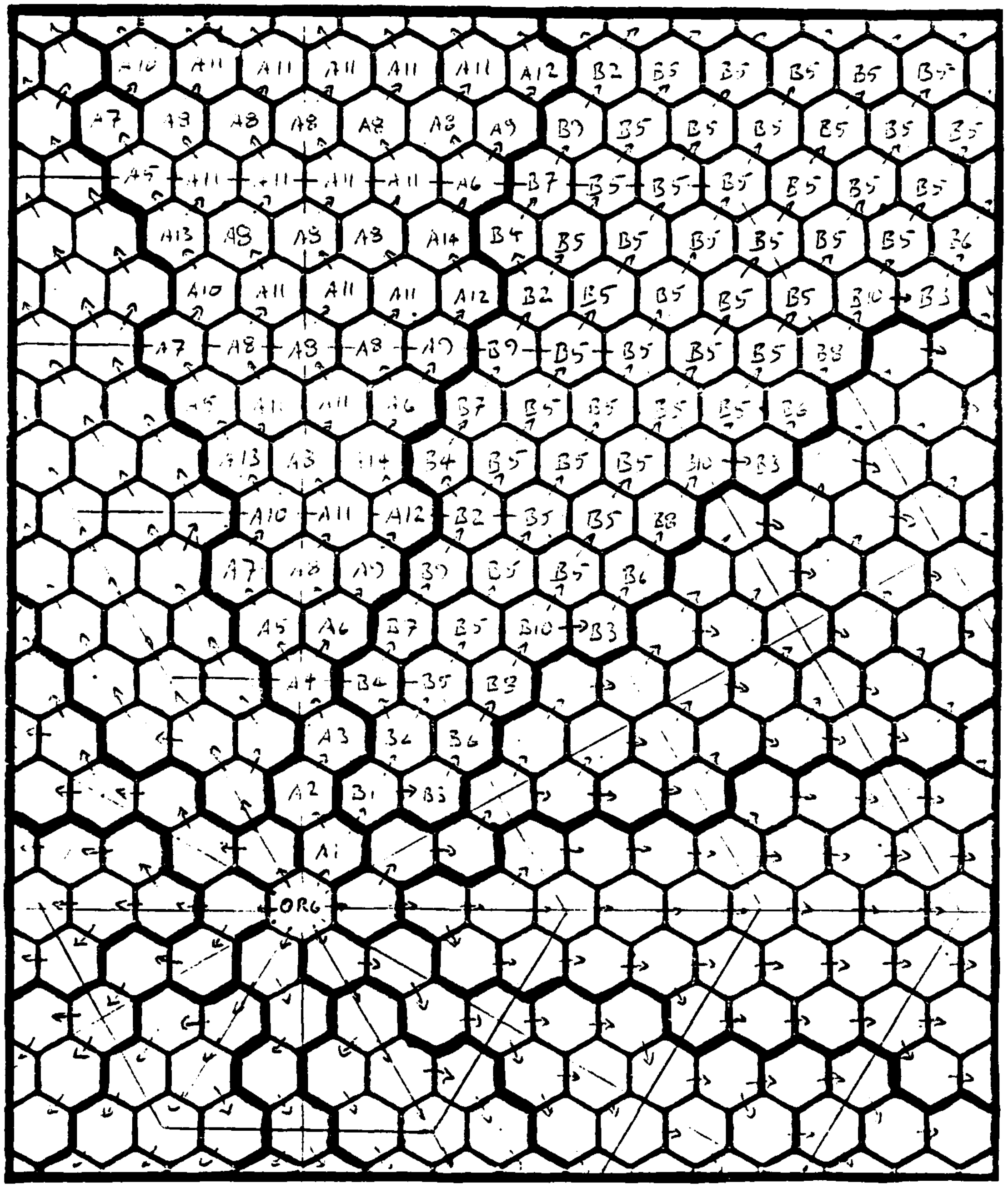


Figure 8 Propagation Method 2

redesigning the propagation rules again, however, is based on some more subtle effects.

First of all, look at the 12:00 and 1:00 sectors. Notice that close to cell C, the 12:00 sector is much thinner than the other. At Time = 3 there is only one message copy in the 12:00 sector but the 1:00 sector has three. This disparity continues throughout the message's expansion, although it becomes less significant; there are never more copies with evenly numbered directions than with odd. Table 1 summarizes this. This asymmetry is both unaesthetic and somewhat hazardous -- "hazardous" because of the possibility that some messages that should meet have a greater chance of missing each other, since they cover a smaller arc than they should.

Table 1 Numbers of Messages in Even and Odd Sectors

| time | no. of copies in 12:00 sector | no. of copies in 1:00 sector |
|---|---|---|
| 2 | 1 | 1 |
| 3 | 1 | 2 |
| 4 | 1 | 3 |
| 5 | 2 | 3 |
| 6 | 3 | 3 |

Another problem with method 2 is also related to symmetry. All of the sectors have a slight shift in the clockwise direction, so that there is a left-right asymmetry.

Figure 9 shows method 3, the current rule propagation scheme. This method has a number of nice features lacking in method 2. The only disadvantage it has relative to method 2

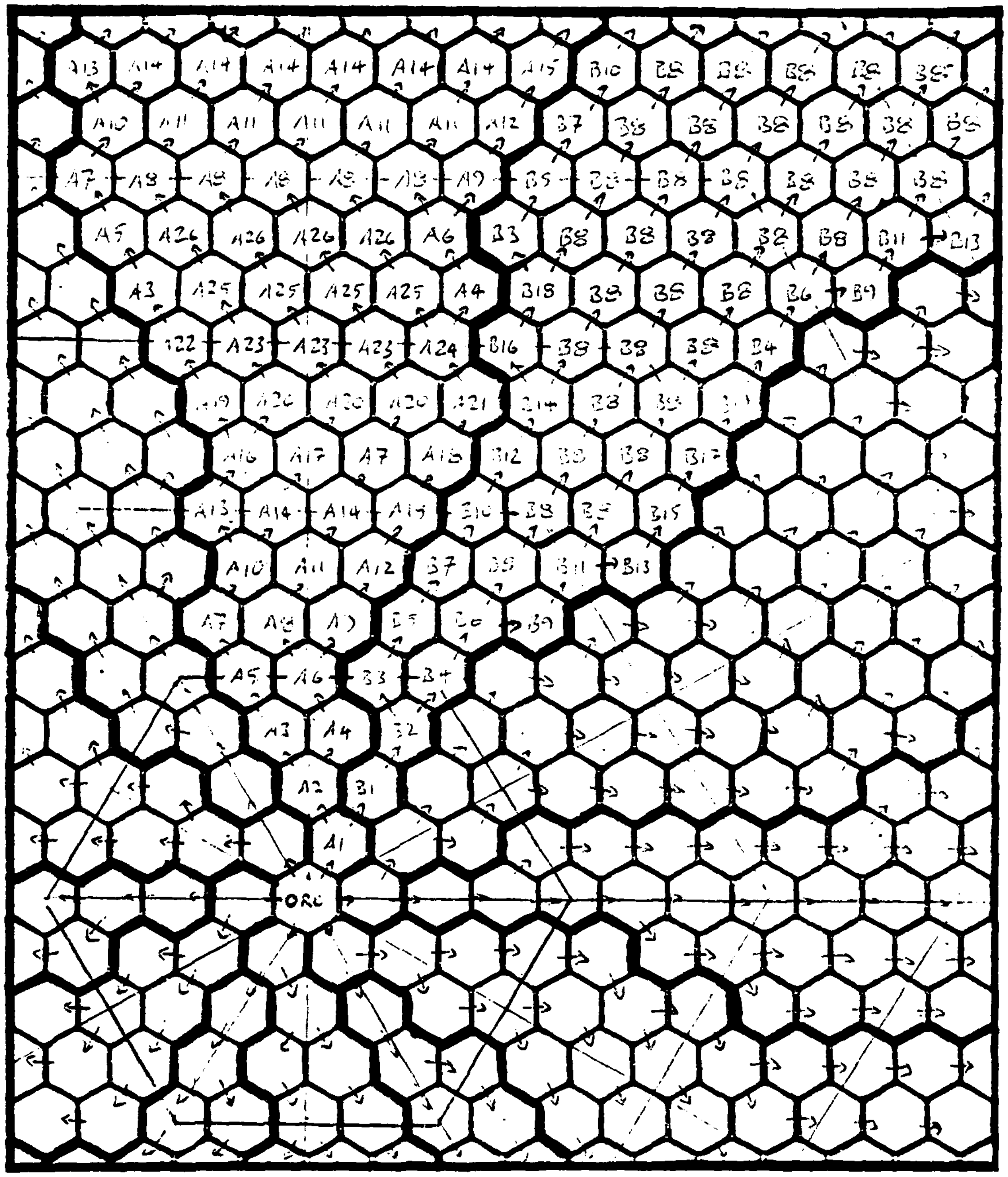


Figure 9 Propagation Method 3

is that it has a larger number of rules -- three hundred forty-two to be exact.

It is desirable to have the number of messages in each sector grow evenly. A message in method 2 has the same number of copies for three time units and then grows by two in only two time units, although this was not mentioned as a problem (there are good propagation schemes that have this property). It turns out that the property of even growth (i.e., where the number of copies in first the odd sectors then the even sectors is incremented) and the property of one sector always having a smaller or equal number of copies than the other always appear together. Another method I discovered while writing this chapter is almost the same as method 3, but combines this inequality "problem" with another aspect of the differences between even and odd sectors to come up with an even better set of rules. I will discuss this method next. In method 3, I chose even growth. At the time it seemed to be the more useful and aesthetically pleasing property.

Figure 10 shows method 4. As stated above, this method combines two adverse effects of message propagation with the effect that they cancel each other out. Recall that earlier I mentioned that the even and odd numbered sectors are different angular sizes. Odd numbered sectors are on the whole smaller because the wavefront "turns a corner". The difference is a matter of only a degree or so. If we use the

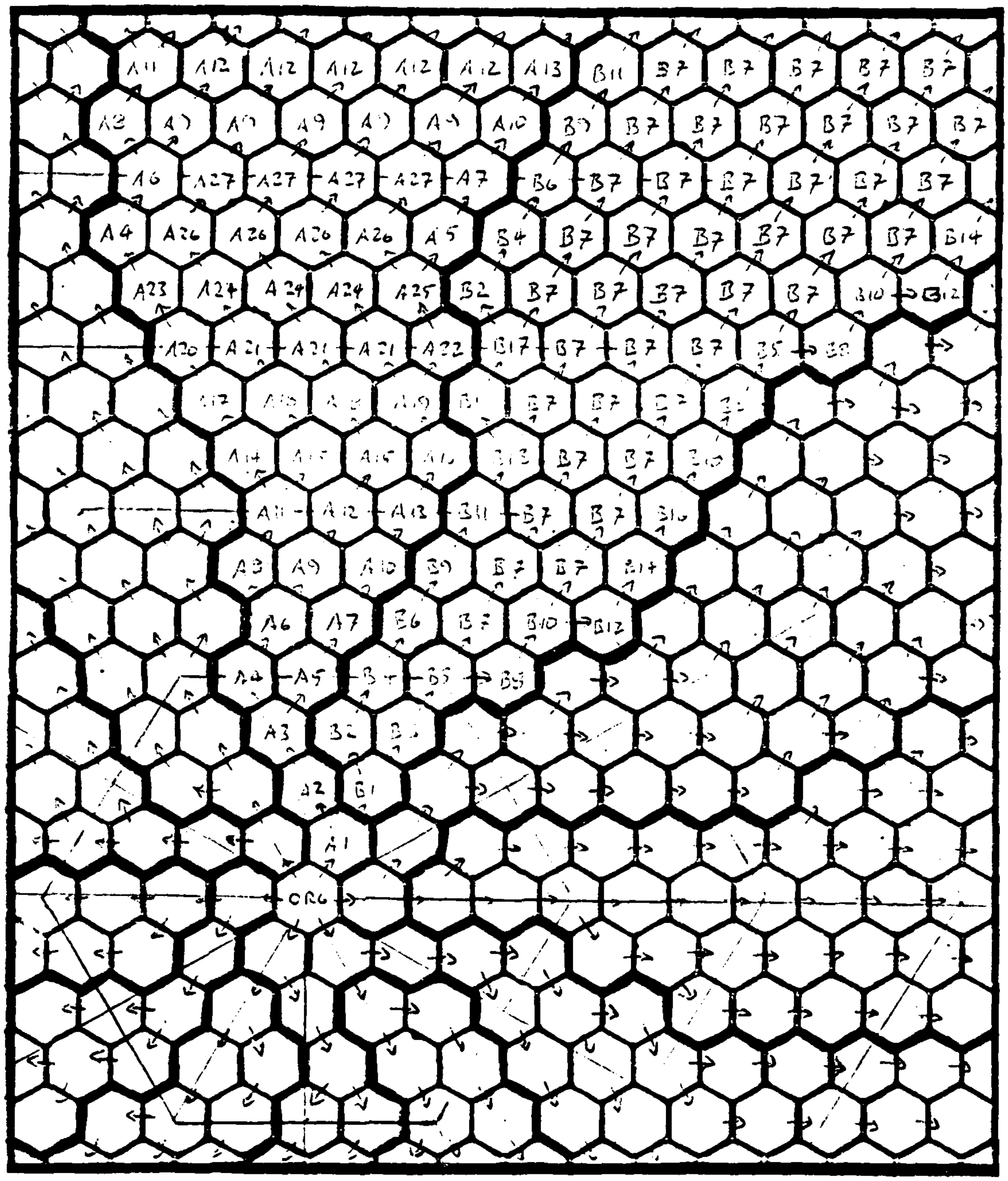


Figure 10  Propagation Method 4

technique of always letting the odd group of sectors have greater or equal numbers of message copies at any time period, the effect is to expand the width of the sector. This effect in the long run (after several time periods) tends to become insignificant, but in the short run, it is strong enough to counteract the inequality problem. Compare figure 9 and 10. The sectors in the latter look much more equal. Method 4 incorporates this idea and, as an added bonus, uses the same number of production rules as does method 3.

## 2.2.2 Variations

Variations on the propagation rules can be introduced at a higher level by the function which alters the state of the cell. This function (which will be discussed in more detail later) is available to all cells and is applied to the cells' contents. It operates by returning as a value the cell as it appears at the following time period, including the new messages which are to be resident in the cell. Since the updating function is free to return any value as the new cell, it has the power to introduce new messages or destroy old ones at its discretion. In particular, the updating function can have the effect of introducing higher level propagation rules. Some examples of higher level rules which are useful for image processing with HEXVIS are the following: 1) propagate for a fixed distance, 2) propagate through similar regions only ("similar" having a fluid definition), 3) propagate along line segments or boundaries (special case of 2), 4) propagate until similar region, 5) propagate until meeting some other message, and 6) propagate normal to edge.

## 2.2.3 Dodecagonal Propagation

There exists another family of propagation rules which I have examined in some detail. Its members possess the desirable quality of producing wavefronts which expand in the form of a dodecagon (a twelve-sided figure). However they also have the undesirable quality of causing messages to propagate into the same cell.

Recall one of the goals specified for the design of the propagation rules is to have the wavefront of the messages expand in a manner as close to a circle as possible. The hexagonal propagation gives a fairly good approximation to a circle (the ratio of the smallest distance to the largest distance is .866, a circle being 1.0). In the discussion, I compared this with a square -- the only other possibility allowed in the type of propagation rules under consideration (a triangular scheme propagates as an irregular hexagon).

In general, of course, a polygon with more sides will be closer to a circle. Specifically, the ratio for a dodecagon is .966. This is very attractive for my purposes.

Unfortunately, though, this method has a drawback. In order to "flatten the corners" so to speak of the hexagonal wavefront, it is necessary to hold back some of the message copies. This effect is produced (as shown in figure 11) by letting some rules propagate messages into the same cell (called "message doubling"). Notice that this is a violation of one of the other goals specified, i.e., that of having messages reach each cell in a unique way. (If a message copy produces a message copy at the next time unit in the same cell, then there are two paths to that cell.) In addition, if a message is resident in a cell for two consecutive time periods, we have to worry about how to count it in the cell's operations: do we count it twice?

It seems probable from my investigation of these propagation rule schemes that methods could be devised to propagate in a wavefront which is arbitrarily close to a circle. This I will save for future examination.

## 2.3 Data Structures

HEXVIS makes use of a number of internal data structures. In this section, I will examine them and explain their uses.

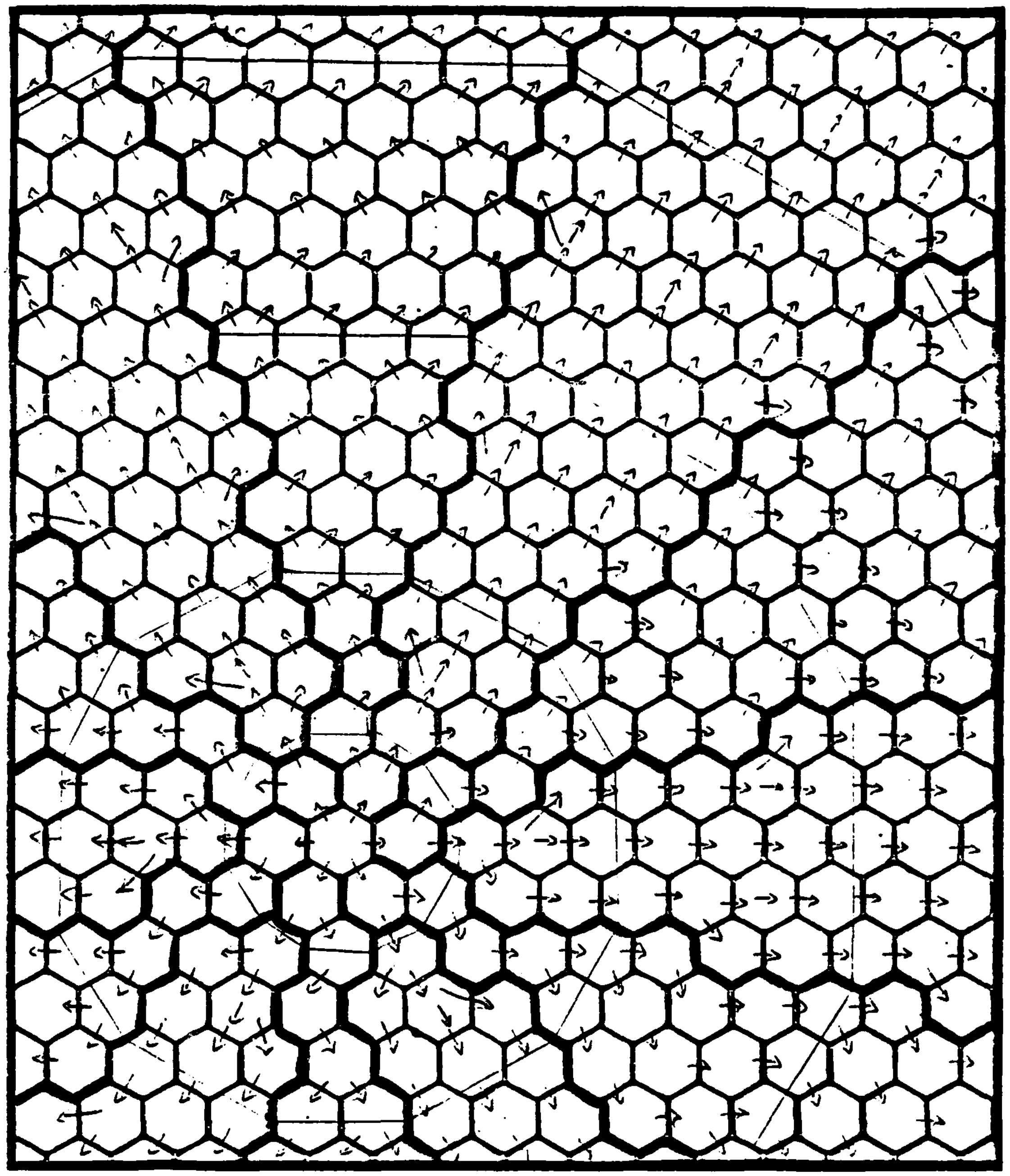


Figure 11 Dodecagonal Propagation Scheme

## 2.3.1 Arrays

First of all I will describe the cellular array itself. As I indicated before, the hexagonal array is simulated by a rectangular LISP array. The array can have arbitrary X and Y dimensions in order to accommodate any size picture, but it has a third dimension which is always equal to two. Perhaps a more useful way of looking at this is that there are two copies of the cellular array. The reason for this is so the parallelism of the system can be maintained. If a cell were to change as a result of some operation, it would have inappropriate messages for its neighbors. Therefore, the updated versions are entered in their correct positions in the currently unused array while examining the cells in the current array for their input. When all of the cells have been visited, the arrays exchange roles and the process continues.

Another array in the HEXVIS system is the picture array. It serves as an input buffer for the raw picture data from the camera. In this implementation, its dimensions are a constant 42 by 238 This size array can contain a picture whose dimensions are 252 by 238 pixels, by storing six pixels in each array entry with six bits of grey level data per pixel.

### 2.3.2 Cells

The cells themselves are the elements of the cellular array. Their structure is that of a list with at least two elements. The first element is called the "state". Its main use is to indicate that something important has happened in the cell as a result of the processing. For instance, if the cell acquires evidence that it "lives" on an edge, the state would be modified to reflect that fact. Its state might be something like the following: ((3 6) (2 9)), indicating that there were three pairs of messages which matched in the six o'clock direction and two in the nine o'clock direction.

The second element of the cell is a list of the messages which are resident in the cell. This list of course changes at each time step as the old messages leave and the new ones enter the cell.

Other storage in the cell is optional. Such items as the cells coordinates, the original data, and results from previous layers of processing can be included.

### 2.3.3 Messages

Each message is also a list. The first element is an atom called the "heading" which governs the path a message takes in travelling through the cellular world. This process is discussed in some detail later so I won't describe it here, but I will describe the structure of the heading.

Headings have two representations: one for human consumption and one for LISP's innards. First, for the humans: Recall that the directions in HEXVIS are quantized into the twelve clock directions. Saying that a message "has a direction" (say 3) means that (unless the time is 0) it was produced by a message which was to its left at the last time period and that it will produce one (or possibly two) messages which will be to its right at the next time period. The headings are represented by a letter followed by a number, e.g., "C12". The letter in the headings representation corresponds to the clock direction in which the message is traveling. The number distinguishes the thirty or so headings having a particular direction.

The machine representation corresponds closely to the human one. To compute it, I first convert the letter to a number: A to 0, B to 1, and so on until L which is changed to an 11. This number is multiplied by 32 (i.e. shifted left 5 bits) and added to the number which follows the letter in the human representation. For example, consider again the heading "C12". The "C" is converted to a 2, multiplied by 32 and added to 12 to yield 76, which is the machine representation.

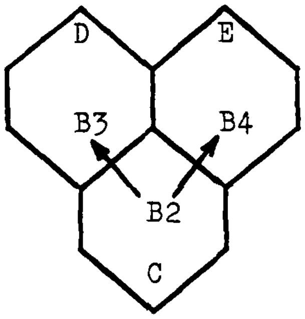
One of the messages, "ORG", has no direction, so I arbitrarily give it the machine representation 372, a number which is larger than any of the other machine representations.

The reason there are two representations is that compiled MACLISP code. is very fast at integer arithmetic, rivaling FORTRAN. Using a numeric domain rather than a symbolic one for the machine version of the headings lends a great increase in speed to the simulation. Humans, on the other hand, have trouble making sense of numbers like 76 or 372 in this context, so a more mnemonic representation is called for.

The rest of the message varies greatly depending on what sorts of tasks are being performed and what stage of the processing (what layer) the system finds itself. For instance, if we want the system to find edges, the rest of the message contains gradient information which is derived from the grey level array. If we are performing a shrinking or thinning operation, we need only carry an indication of whether the originating cell is part of the "figure", or part of the "ground".

### 2.3.4 Production Rules

The production rules which govern message propagation are stored as entries in an array, indexed by the heading of the message (in computer representation, of course). Each entry is divided into six five-bit fields, each of which corresponds to a neighbor. The idea is this: if a cell C sees a message M in neighbor N with heading H, it looks at the H-th entry in the production array. Within this entry, C looks at the N-th five-bit field. If the field is zero, C knows that this message will not be propagated to C at the next time period. However, if the field is not zero, C constructs a heading with the same clock direction (see above) but appends the number it found in the five-bit field. This heading is combined with the rest of M (minus the old heading) and saved on a list of new messages. C continues this process for all of its six neighbors, and for all of the messages contained in them to construct a complete list of its new messages.



(a)

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 34 | /// | 0 | 0 | 4 | 3 | 0 | 0 |
| Field: | | 5 | 4 | 3 | 2 | 1 | 0 |

(b)

Figure 12 Propagation Rule Example

An example can serve to illustrate this process. Figure 12a shows a small part (three cells) of figure 7 (method 1) with the cells labelled A, B, and C. We see that a (message with ) heading "B2" in cell C produces a (message with heading) "B4" in cell E and a "B3" in cell D. Now if we look in the production array under 34 (the machine representation of "B2"), we find the entry shown in figure 12b. Notice the 3 and 4 in its second and third fields, respectively. Consider first cell E's point of view. In examining its neighbors' messages, E notices the one in C with the heading B2. It then retrieves entry 34 in the production array and looks in field number three (since C is E's third neighbor). Finding a non-zero entry in that field tells C that it will get a message generated from the cell it is examining. The facts that the entry is equal to 4 and that the heading is of type "B" imply that the new message will have heading "B4". Similarly, cell D looks at the second field (again since C is D's second neighbor), and discovers that it will get a copy of the message with heading "B3".

## 2.4 Layering

"Layering" is a process by which complex image processing tasks can be built from comparatively simple ones. In this process, each cell produces (or decides not to produce) a new initial message based on what it finds in its state. The function to perform this operation is the same for all the cells at any given layer and is applied when the message propagation for that layer is complete (as specified by the updating function). The new layer will in general use a new updating function since both the information carried by the messages and the goal of the message propagation may vary from layer to layer.

As an example, consider the following two-layer process for finding and recognizing squares in the scene. The first layer of message propagation has in effect an updating function which infers axes of symmetry. When the propagation is complete, certain cells in the scene which lie on an axis of symmetry contain that information in their states along with an indication of the symmetry axis' orientation (see figure 13) In the second layer, the cells which are on symmetry axes create new messages encoding that fact. Then the updating function is changed to one which finds midpoints of line segments. A cell recognizes that it is the center of a square when it contains two midpoints of equal length lines oriented ninety degrees apart.

In principle, this process can be reversed using HEXVIS, since the important information about messages and message intersections is retained in each cell. The cells on the midpoints could simply broadcast two messages in the appropriate directions for an amount of time corresponding to the line segment's length to reproduce the symmetry axes. After this, the cells which contain symmetry axis points emit messages again in the appropriate directions; these messages are allowed to propagate for the correct amount of time to reproduce the square. Figure 14 shows this reverse process in action. Note the fuzziness introduced by quantization of

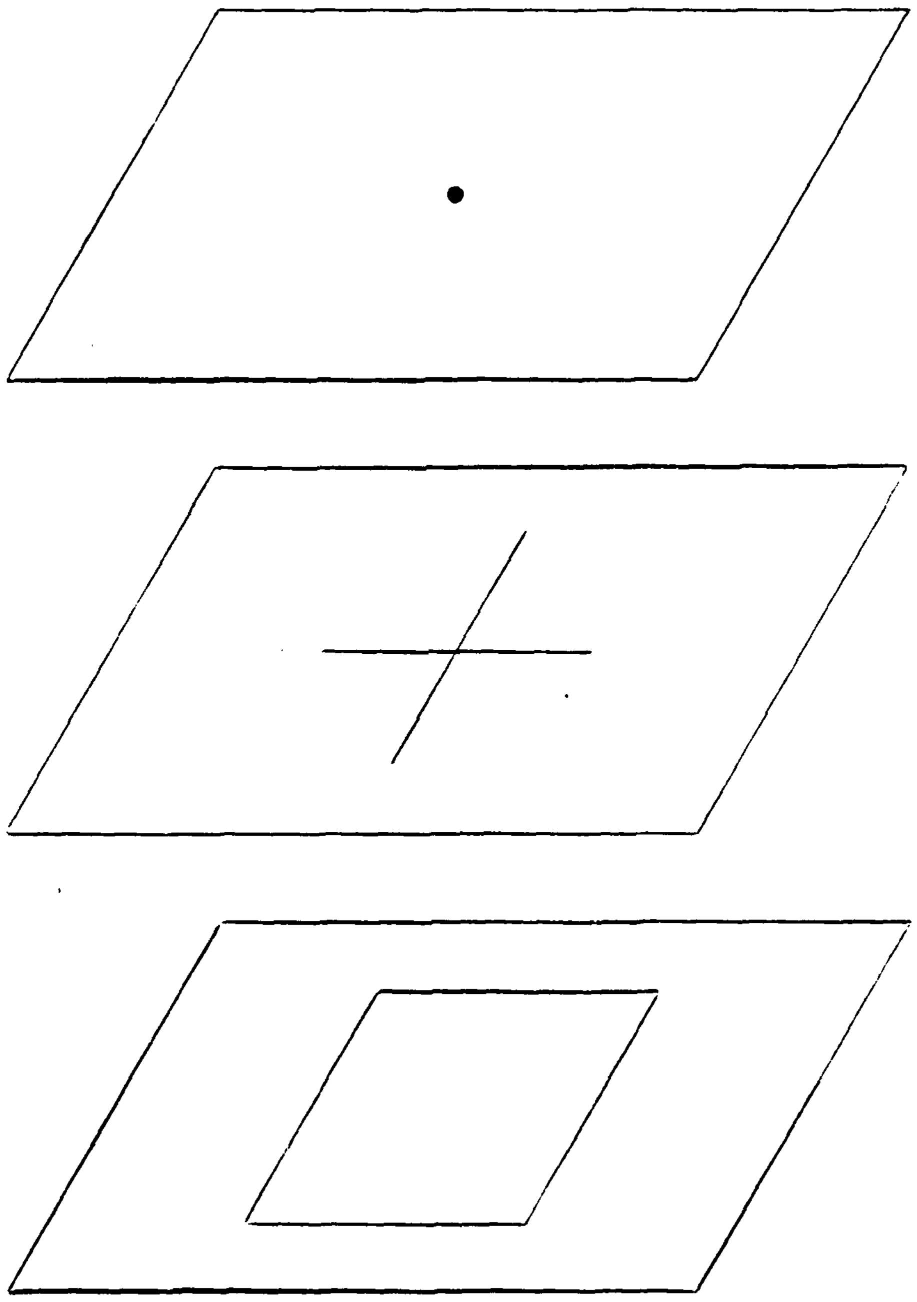


Figure 13 Layering

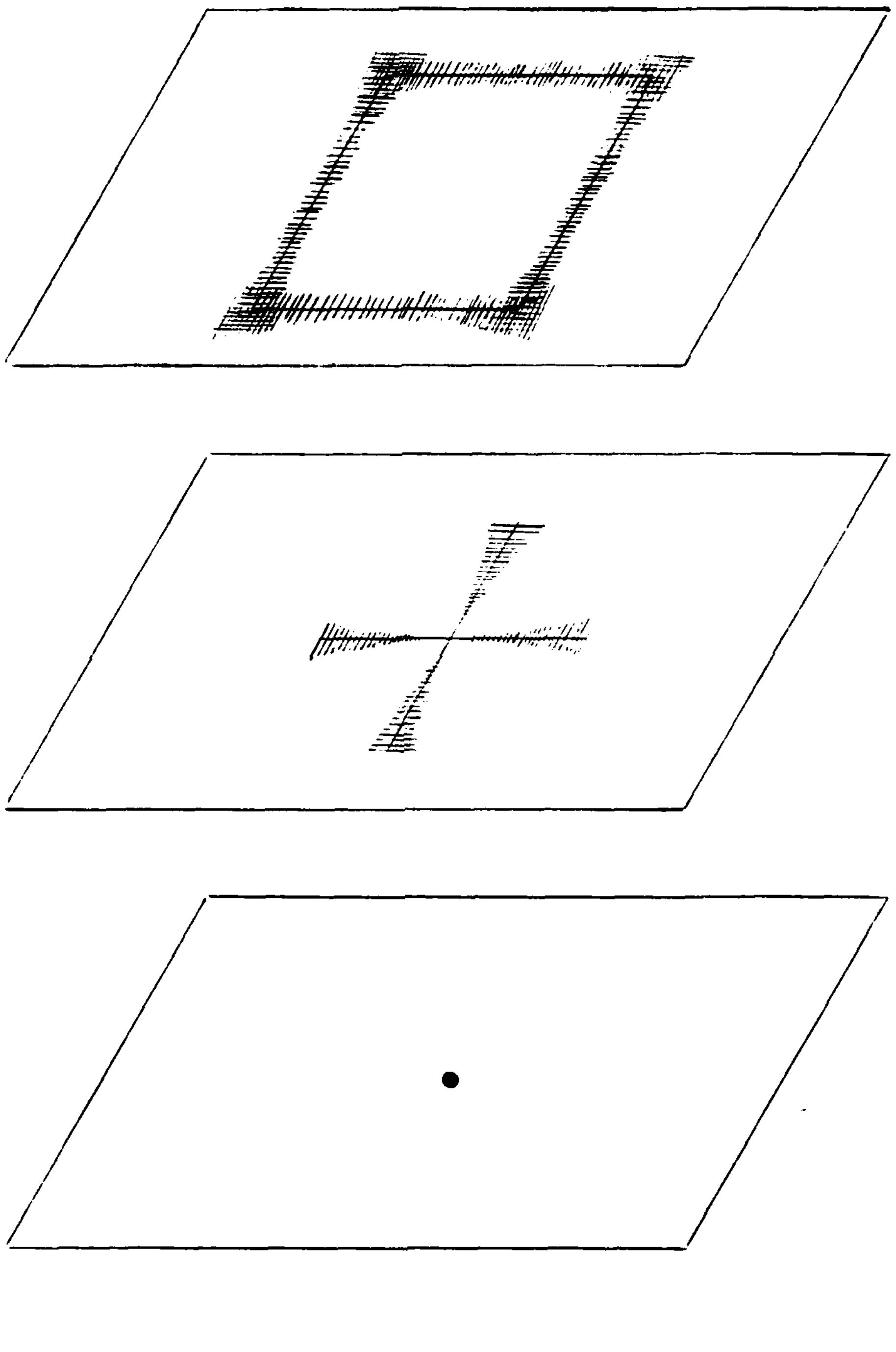


Figure 14  Reversing the Layering Process

the directions.

Notice also that if we were to leave out the criterion that the midpoints be from lines which are the same length, we would generalize the process to recognize rectangles.

The preceding example illustrates another advantage of layering: The information in a scene can be in some sense "chunked". That is, the information that there is a square in the scene, plus its size, location, and orientation, are all compressed into one cell rather that distributed among many. The ability to broadcast all this information in a concise way from a single cell can prove to be quite powerful.

## 2.5 The General Paradigm

The general HEXVIS paradigm is very simple. Each cell has a copy of (or access to) the updating function in effect at the current layer. This function is given the following arguments: 1) a list of the new messages, the messages which will be resident at the next time period; 2) a list of the old messages, the ones resident now; 3) a copy of the cell itself, and 4) and 5) X and Y, the coordinates of the cell in the hexagonal grid. The updating function is executed with the appropriate arguments at the beginning of the time period and returns as its value the new cell, i.e. the cell at time T+1, which replaces the current cell in the array. Notice that this process takes care of propagating the messages as well as updating the cell's state.

As an example of an updating function, consider the following (slightly schematic) one which finds evidence for edges:

```
(LAMBDA (NEWMESSAGES OLDMESSAGES CELL X Y)
        (CONS (APPEND (COMPARE-EDGE NEWMESSAGES NEWMESSAGES)
                      (COMPARE-EDGE NEWMESSAGES OLDMESSAGES)
                      (CAR CELL))
          (CONS NEWMESSAGES (CDR CELL))))
```

COMPARE-EDGE is a function which takes two lists of messages and compares their elements in a pairwise manner, taking one from each list, and keeping track of the number of pairs (and their direction) which form evidence for the presence of an edge. It returns as a value a (possibly empty) list of entries, each of which specifies the direction of the edge and the number of pairs which matched.

The updating function also contains conditions for the cessation of message propagation.

# CHAPTER 3

# RELATED WORK

My work can be compared with other work along a number of dimensions. In particular, I will discuss work in the following areas: 1) hexagonal tessellations, 2) parallel techniques in image processing and computer vision, 3) cellular automata, and 4) message passing.

In addition, Chapter 6 presents a new algorithm for shrinking binary scenes on a hexagonal grid. Some work related to this topic is discussed there. For completeness, I will indicate the references: [Golay, 1969], [Rosenfeld, 1970], [Levialdi, 1972], and [Rao, 1976].

## 3.1 Hexagonal Tessellations

Golay [1969] proposes a model for parallel transformations of sets of points in a hexagonal grid. His choice of the hexagonal tessellation is based mainly on its lack of ambiguity when connectivity relations are studied (as was explained in Chapter 2).

Golay also points out that there is a greater angular resolution available in a hexagonal array. The six nearest neighbors are spaced equally sixty degrees apart while the next nearest neighbors are spaced equally thirty degrees apart. Neighbors further away are not spaced equally. In a square array, the corresponding angles are ninety and forty-five degrees.

Golay had in mind a hardware realization of his system. The pattern transformations he describes take place by sensitizing (under operator control) the machine to a particular set of neighborhoods (Golay calls them "surrounds"). All points whose neighborhoods match the designated neighborhoods either change state or not depending on their current state (again, under operator control).

These pattern transformation are applied in parallel to the scene. However, they are applied cyclically to three subfields which have the characteristic that no two elements from any one subfield are neighbors. The three subfields are shown in figure 15 labelled "A", "B" and "C". The reason given for this division into subfields is that a race condition exists such that the states of a cell's neighbors might be changing at the time the cell needed to examine their contents. It is not clear, though, why in a synchronous machine presumably using J-K flipflops this would be a problem.

```
A B C A B C A B C A B C
 C A B C A B C A B C A B
A B C A B C A B C A B C
 C A B C A B C A B C A B
A B C A B C A B C A B C
 C A B C A B C A B C A B
```

Figure 15 Golay's Three Subfields

If looked at in a different way, however, the division of the cells into three subclasses is an elegant way to avoid the use of dual arrays in simulating a parallel pattern transformation on a serial computer. In the case of HEXVIS, this would not be viable, since the state changes are much more complex involving (among other things) the transfer of (relatively) large amounts of information from cell to cell in the form of messages.

Preston ([1971] and [1972]) has developed a special purpose computer to realize Golay's system. The machine is called GLOPR (Golay logic processor) and interfaces with a minicomputer which in turn interfaces with other peripherals. An interactive language called Glol (Golay logic language) was also written by Preston which allows the user to manipulate the system in real time. Both papers show several examples both of artificial scenes and of real scenes of blood cell nuclei and chromosomes.

Deutsch [1972] reports on thinning algorithms developed for the three regular tessellations of the plane: triangular, square, and hexagonal. The purpose of the thinning algorithms is to reduce thick lines to thin lines. A typical application is to character recognition where the thinning produces a skeleton of the character which can be more easily dealt with than the original. Deutsch concludes that a hexagonal array offers a better choice than the square or triangular arrays in that the thinning algorithm takes much less time to run in the hexagonal array than in either the square (a factor of two) or the triangular (a factor of almost four) arrays and is less sensitive to noise in the hexagonal array than in the others.

In another (earlier) paper [1970], Deutsch comments on Golay's work. He makes clear the connectivity ambiguity involved in using square arrays and mentions further that if the object uses four-cell connectivity then the background must use eight-cell connectivity and vice versa. Deutsch points out that hexagonal arrays do not suffer from either of these problems. He also presents a modification to Golay's thinning algorithm. There are some objects for which Golay's thinning procedure does not work. It instead causes them to disappear. Golay proposed a solution which involved considering the three subfields in random order rather than cyclically. Deutsch proposed a counter-solution which keeps the original cyclic order but destroys the isotropic characteristics of the system. In developing my parallel shrinking algorithm (Chapter 6), I encountered a problem quite similar to this one which probably has the same basis. This basis has to do with a fundamental difference between the hexagonal and square arrangements. I describe the problem in detail in Chapter 6.

Horn [1973] uses a hexagonal tessellation in his work on lightness. "Lightness" is a perceptual quality closely related to reflectance. Humans, it seems, are able to factor out variations in intensity which are caused by variations in illumination to perceive the lightness of a surface. Horn develops a model for this process. His reasons for using a hexagonal grid are that there is only one kind of neighbor (as has been pointed out previously) rather than two (square) or three (triangular) and that circular objects pack tightest in a hexagonal pattern. This latter reason apparently refers to the fact that the rods and cones in the eye have a circular cross section and are arranged in a roughly hexagonal array [Lindsay and Norman, 1972].

Burt [1979] discusses hexagonal analogs to quadtrees. A quadtree can be used as a compact way of describing a scene which has large areas of the same color (or whatever characteristic one is concerned with). Consider, for the moment a binary scene (one in which every point is either black or white) on a square grid. Assume also that the shape of the scene is square with an edge length which is a power of two. The quadtree associated with that scene has a root node which corresponds to the entire scene. Each of the four branches of a node corresponds to one quarter of the area of the scene that its parent node corresponds to -- either the upper right, upper left, lower right, or lower left square. If the area of a scene corresponding to one of the branches is all white or all black, the branch is labelled accordingly and is a leaf of the quadtree. Otherwise, the branch is labelled "grey" and goes to a node which itself has four branches as above.

Burt proposes a number of ways this idea might be adapted to a hexagonally sampled scene. The most promising way is a septree, i.e., one whose nodes have seven branches. This method is interesting because the nodes at all levels correspond to areas in the scene which are roughly hexagonal in shape (as all nodes in a quadtree correspond to areas which are square).

Siromoney and Siromoney [1975] have discovered an interesting application of hexagonal arrays. They have invented a grammar for generating isometric views of rectangular parallelepipeds which uses figures drawn on a hexagonal grid as primitives.

Waltz [1978], of course, was the inspiration for this thesis. Many of his ideas are here. In particular, the choice for the hexagonal grid and the idea of message passing are his, as are some of the ideas for symmetry axis, corner, and edge finding described in Chapter 4.

## 3.2 Parallel Techniques for Image Processing and Vision

Gordella, Duff, and Levialdi [1976] develop a detailed parallel algorithm for thresholding an image. They define three steps in the algorithm: 1) forming a histogram of the grey levels in the scene, 2) detecting valleys in the histogram to determine the threshold value, and 3) relabeling elements in the scene as zero or one depending on whether they fall below or above the threshold value. The operations involved are shown to depend either linearly on n, the number of rows (or columns) in the scene or on $\log_2 n^2$. Serial implementation of the same task, on the other hand, executes in time proportional to $n^2$. For a value of n equal to one thousand, the ratio of execution times for the serial implementation to the parallel one is about one hundred.

Rosenfeld and Kak [1976] have much to say about parallel computation in image processing and vision. They present methods of finding skeletons, of shrinking (both preserving connectivity and not), and of cluster detection. An interesting pair of operations which they define for binary scenes are these: "expanding" or "propagating", i.e., at each step, replacing every zero point which is next to a border point of an object with a one, and "shrinking", i.e., at each step, replacing every border point of an object with a zero. (Note that this shrinking does not preserve the connectivity of the scene.) It is shown how these operations can be used for region filling, cluster detection, detecting elongated elements, and thinning.

## 3.3 Cellular Automata

Dyer [1977] introduces a three dimensional cellular automata-like model for image analysis consisting of a number of square arrays. Each array is one-half the linear dimensions (one-quarter of the area) of the array below it and forms a pyramid shape. Every cell in the pyramid has (excluding itself) nine neighbors: the four edge neighbors in its own array, the parent cell in the layer above, and its four offspring in the layer below. Cells in the lowest layer are associated with the pixels and have no offspring neighbors. (Notice the similarity to quadtrees.) The cells operate in a synchronous manner, and each has associated with it a local memory which can be examined by each of its neighbors.

Besides the obvious fact that cellular pyramids and HEXVIS are both derived from cellular automata, there is another similarity between the two systems: both have a layered structure. In HEXVIS the layering is not as explicit in that cells in one layer do not communicate with those in another directly. Instead, the higher layer consists of a new updating function which is applied to the results of the previous computations. On the other hand, cells in cellular pyramids can communicate directly with cells in layers above or below them so that information can flow in both directions.

Levitt and Kautz [1972] suggest a number of parallel algorithms which use cellular arrays for the solution of graph problems. They mention spanning tree, distance, and path problems and discover that in some cases relatively unknown algorithms which are inefficient when applied serially produce surprisingly good results when adapted to be parallel.

Their representation, however, is in terms of the adjacency matrix of the graph. This is a matrix A with members $a_{ij}$ whose rows and columns correspond to the vertices. If there is an edge from vertex i to vertex j, then $a_{ij}$ is equal to one. Otherwise, $a_{ij}$ is equal to zero. Non-directed graphs have symmetric adjacency matrices. The representation of a graph by its adjacency matrix would of course be possible in HEXVIS, however, HEXVIS is organized in a hexagonal array which (by design) lends itself more to the solution of problems in a spatial domain. The solution of problems represented in other ways using the HEXVIS message passing paradigm is, I think, an intriguing area for further study (see Chapter 7).

Banks [1971] describes several cellular automata systems. The foremost of his contributions (in his own judgement) is a two-state, four-neighbor rectangular system. He has shown this system to be computationally complete, i.e., it can perform any computation.

Another significant contribution of Banks thesis is a four-neighbor universal constructor/computer with only four states (reduced from the previous eight states of Codd [1968]). As far as I am aware, this result has not been improved on.

Banks also discusses information transmission in cellular automata systems. However, the techniques described are more classical in the sense that the signal is carried by a "wire" made up of special states. The signal also spans several cells and carries a minimal amount of information, whereas the messages in HEXVIS are completely contained in one cell (at any one time period) and carry an arbitrary amount of information.

## 3.4 Message Passing

Farley [1979] discusses a message passing technique for four-neighbor square cellular arrays called "gossiping". The process of gossiping is described as follows: at discrete units of time each cell can communicate with at most one of its neighbors. An act of communication (called a "call") between cells A and B results in both cells containing, at the end of the time period, the union of the information contained in A with that contained in B. Gossip is complete when every cell knows the contents of every other cell in the array. Results are derived for minimum gossip time and minimum number of calls required.

Notice that this message transmission technique is similar to the one used in HEXVIS in the sense that the amount of information transmitted in one time unit is unbounded. However the paths the information takes in arriving at its many destination by gossiping is much less regular (indeed, unspecified) than the paths in HEXVIS.

Hewitt's [1976] actors are structures which communicate by passing messages. Each actor has two (indivisible) parts: a script, which describes the action it takes when sent a message, and a set of acquaintance, which are the other actors it knows about. An actor can send a message (which is also an actor) to any of its acquaintances. This message passing process is the basis of all computation in the model and is called "actor transmission"; the message is called the "messenger" of the transmission; and the actor to whom the message is sent is called the "target" of the transmission. An important feature of this model is that each actor has its own context, i.e., there is no concept of a global state of all the actors. As an example of a common function cast in the actor mold, consider the computation of the factorial of three. A messenger (an actor) which knows about the message "3" (another actor) and about the actor "C", to whom the reply should be sent, is sent to the target actor "factorial". The factorial actor causes another messenger with the message "6" (also an actor) to be sent to C. The internal workings and structure of the actors are assumed to be unimportant; they are described only by their input and output behaviors. Hewitt shows that actor transmission is a powerful enough model to support these types of actions: calling a procedure, obtaining an element from a data structure, invoking a co-routine, updating a data-structure, returning a value, synchronization of communicating parallel processes.

# CHAPTER 4

# PAIRWISE MESSAGE MATCHING ALGORITHMS

This chapter is concerned with one of the simplest classes of algorithms which can be embedded in HEXVIS. Members of this class compare all of the messages in a cell in a pairwise manner each time unit. As it turns out these algorithms are quite powerful. They are able to discover (for instance) edges, corners, and axes of symmetry from local gradient information.

The general form of these algorithms can be described as follows: each time unit, each of the new messages is compared to every other one. If the pair of messages satisfy the particular relation in effect, a counter in the cell's state is incremented by one. Then a similar process is repeated by comparing each new message to each old one. If this pair satisfies the relation, the counter is incremented by one-half.

A couple of questions arise here. First of all why compare new messages to old messages at all? As messages propagate in the normal manner, they move outward from their origin at the rate of one cell per time unit. The problem is this: if the two messages originate from cells which are an even number of steps away from each other, they will never be in the same cell at the same time (see figure 16). The solution is to (conceptually, at least) propagate a "ghost" message behind the real message. This concept is realized as described above -- by comparing messages which will be resident in the cell during the next time period with messages which were resident during the current time period -- which amounts to the same thing.

| | | | | | | |
|---|---|---|---|---|---|---|
| t = 0 | → | | | | | ← |
| t = 1 | | → | | | ← | |
| t = 2 | | | → | ← | | |
| t = 3 | | | ← | → | | |
| t = 4 | | ← | | | → | |
| cell | 1 | 2 | 3 | 4 | 5 | 6 |

Figure 16 Messages Missing Each Other

The second question to be answered is: why is the new-old match given only one-half the strength of a new-new match? The reason for this is that if a pair of messages satisfies this new-old type of relation in one cell, it will also satisfy it in an adjacent cell. Consider again figure

16. At t = 3, each of the messages is in the same cell that its opposite number was in during the previous time period. Since both of these matches taken together really represent only one matching message pair, they are each given only half the normal weight.

Incidentally, it is also possible and often desirable to modify the increment (either one or one-half) further by causing it to be inversely proportional to the time unit or the square of the time unit. This modification has the effect of producing messages which decay according to distance traveled.

The next sections illustrate three sample algorithms of the type described above. They find edges, corners, and axes of symmetry. For each of the algorithms described, I have included an example of each one's output when applied to the same rectangle.

## 4.1 Edge Finding

Figure 17a shows a rectangle embedded in the cellular array. Each "O" in the figure represents a cell which has a message with significant local gradient information. Notice that two sides of the rectangle are missing some information. This lack proves to be of little consequence in the processes described here. These algorithms are relatively insensitive to missing information.

```
  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
                               0
  0                           0
                               0
  0                           0
                               0
  0                           0
                               0
  0                           0
                               0
  0                           0
                               0
  0   0   0   0   0   0   0   0
```

a) Partly sparse rectangle

```
           1 3 3 3 3
    7 8 8 8 8 8 8 8 8 8 8 8 7
             3 3 3 3 1
  5                           5
 4 4                         4 6
  5                           7 4
 4 4                         5 7 1
2 5 2                       2 7 5
 4 4                         5 7
  5                           7 4
 4 4                         4 6
  5                           5
           1 1 2 1 2
    7 5 7 5 7 5 7 5 7 5 7 5 7
             2 1 2 1 1
```

b) Edges found

Figure 17  Edge Finding

In order for a cell to consider itself an edge point, it must accumulate evidence from the cells around it. The pairs of messages which are considered evidence must have the following characteristics: 1) they must be traveling in opposite directions, 2) both of the local gradient directions which are carried by the messages must be equal to each other and perpendicular to the direction of travel, and 3) the magnitude of the local gradients must exceed a certain threshold. If these conditions are met, the appropriate increment is added to the counter corresponding to that edge direction in the state of the cell. When enough pairs are accumulated (again exceeding a threshold value), the cell is labeled an edge point. Of course, a cell can be an edge point on two (or more) intersecting edges.

Figure 17b shows the result of the edge finding process for the rectangle in figure 17a. The second thresholding process has been suspended here so that we can see the number of edge matches at each cell (minus one). Notice that the cells which had no local gradients initially are now labeled as edge points. The spreading of the edges of the rectangle is caused by the quantization of message direction. A cell receiving a message can determine its origin only to an arc of about thirty degrees, so cells which are off the true edge can still receive messages which appear to be traveling in (in this case) horizontal or vertical directions. Of course, these spurious matches will be fewer than matches on the true edge, so thresholding can clean things up.

## 4.2 Corner Finding

Figure 18a shows the same sparse-edged rectangle as before. This time, though, we are looking for right angled corners (the extension to corners with angles other than ninety degrees will be obvious). As in the case of edge finding, the message pairs must satisfy certain criteria: 1) the two messages must be traveling in directions which are perpendicular to each other (or at whatever is the angle of the corners we are interested in finding), 2) the local gradient direction carried by the messages must be perpendicular to the messages' directions of travel and must both be pointed either inward or outward relative to the angle formed by the directions of travel of the two messages, and 3) the magnitude of the gradients must exceed a threshold.

The result of applying this process to figure 18a is shown in figure 18b. The wide area covered by the "corner" (which is due to the same effect that caused the edges to spread out) at first glance looks like a real drawback. However, the greatest number of matches occurs in the cells which are precisely on the corners of the rectangle. (The ">" sign means "more than ten matches".) A second pass of thresholding or, better yet, hill-climbing can quickly zero in on the true corner.

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
                             0
0                           0
                             0
0                           0
                             0
0                           0
                             0
0                           0
                             0
0                           0
                             0
0   0   0   0   0   0   0   0
```

a) Partly sparse rectangle

```
 6 6 6 6 6                 4 6 6 6 6 6 6
5 7 7 7 7 6                 5 8 9 9 8 7
 5 8 > 7 6                 2 6 9 > 9 7 6
4 8 8 8 7 6                 4 7 8 8 8 7
 4 4 4 4 4                   5 7 7 7 7 5



 4 5 4 5 4                   3 4 3 4 3
4 6 5 6 5 3                 5 6 8 8 8 4
 5 6 9 7 4 2                 6 7 > 9 7 4
5 6 7 7 7 4                 5 6 7 7 7 5
 4 5 4 5 4 3                 6 6 6 6 6 5
4 3 4 3 4 3 2
```

b) Corners found

Figure 18  Corner Finding

## 4.3 Axes of Symmetry

My final and perhaps most useful example from the pairwise message matching class of algorithms is a process to find axes of symmetry.

Blum [1967] has discussed a need to characterize figures in a scene in some schematic way. His method, called the "symmetric axis transform", or more colloquially the "prairie fire technique", can be visualized in the following way: Imagine the object in the scene to be made of flammable prairie grass. Then imagine starting a fire along the boundary of the object and allowing it to burn inward. The set of points where the fire meets itself and dies out forms a representation of the object. An equivalent and more formal description is to define the resulting skeleton as the set of centers of all circles which will fit inside the figure but are not contained wholly in any of the other circles in the set.

As Waltz [1978] and others point out, the main problem with the prairie fire technique is its high sensitivity to small variations in the original figure. For instance, figure 19 shows two rectangles, one of which has a notch, and their prairie fire skeletons. Notice that the notch has caused a rather large variation in the corresponding skeleton.

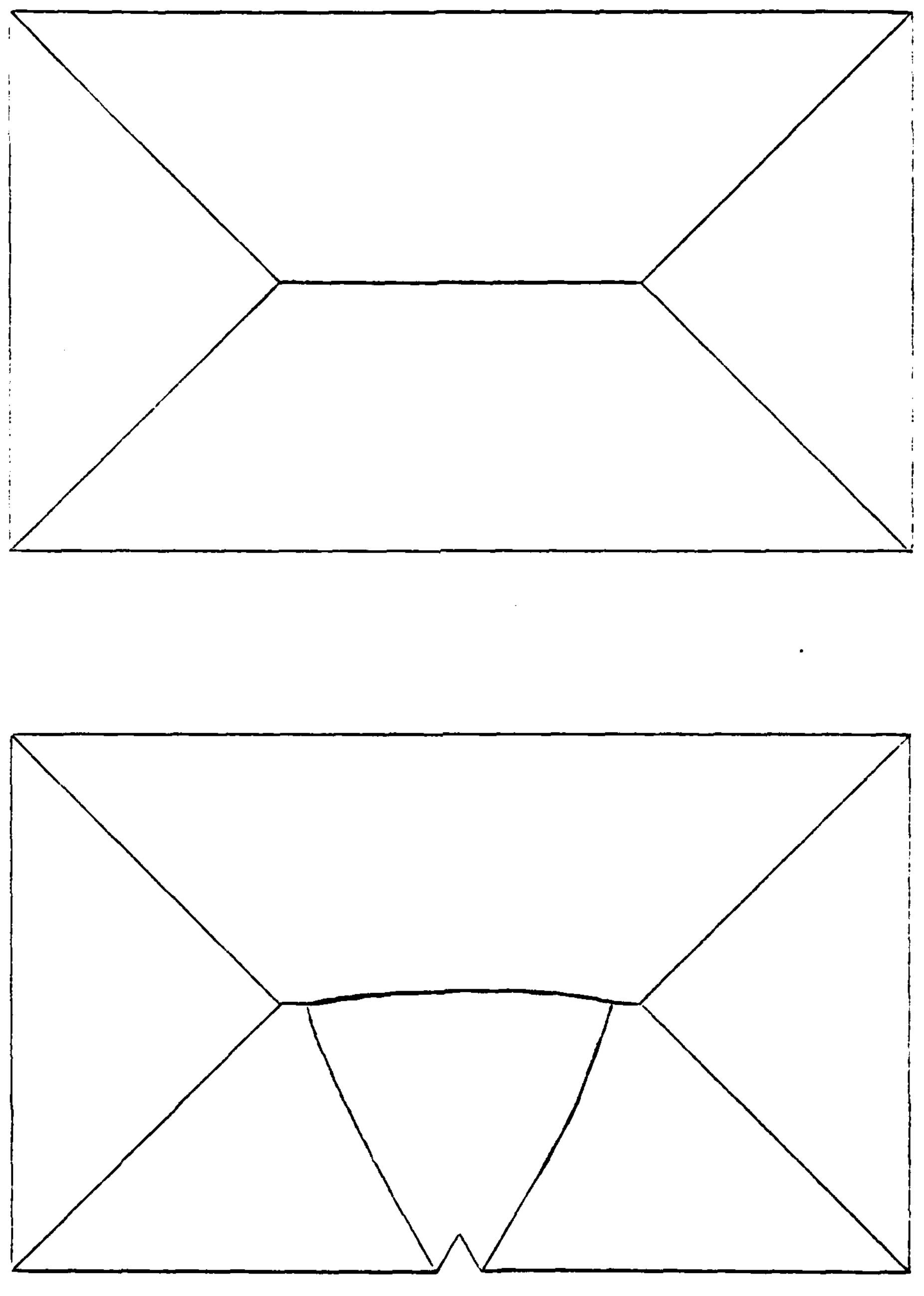


Figure 19 Prairie Fire Skeletons of Rectangle and Notched Rectangle

The technique proposed here lacks this problem; similar figures map into similar skeletons. As in the other cases, the symmetry axis finder operates with the messages' direction of travel and the local gradient information which is carried as part of the message. The criteria in this case are as follows: 1) the messages must be traveling in opposite directions, 2) the gradient direction carried by the messages must be either parallel or anti-parallel to the messages' direction of travel, and 3) once again, the gradient strength must exceed a predetermined threshold.

Figure 20 shows the sparse-edged rectangle and the result of applying the symmetry axis technique to it. Here, the spreading of features is confined to the vertical axis of symmetry. The reason the horizontal symmetry axis escaped this fate is that the wavefronts of the messages met in such a way that their flat parts contacted. In forming the vertical symmetry axis, however, the wavefronts met corner to corner, increasing the uncertainty of the messages' origin.

The algorithm as described above is tuned to symmetry axes which are between parallel edges. However by modifying the second rule, the technique can easily discover symmetry axes between edges oriented in other directions.

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
                             0
0                           0
                             0
0                           0
                             0
0                           0
                             0
0                           0
                             0
0                           0
                             0
0   0   0   0   0   0   0   0
```

a) Partly sparse rectangle

```
               0 1
                3 0
               4 6
                7 1
               4 6
                7 1
               4 6
2 4 > 6 > 6 > 6 > 7 > 6 > 6 > 4 2
               4 6
                7 1
               4 6
                7 1
               4 4
                4 0
               0 0
```

b) Axes of symmetry found

Figure 20  Finding Axes of Symmetry

## 4.4 Discussion

Although these techniques are useful, there is a need for more powerful ones. The next chapter describes several examples of embedded HEXVIS algorithms to compute more complex characteristics of scenes.

# CHAPTER 5

# HIGHER ORDER TECHNIQUES

In this chapter (as in Chapter 4), I discuss various algorithms which can be embedded in HEXVIS. The algorithms discussed here, however, are in most cases more complex than those that examine messages in a pairwise manner. Such techniques as constructing histograms for each cell based on some feature of the messages which pass through it, creating new messages headed in specific directions, destroying messages which satisfy certain criteria, region growing and so on, are considered. Some of these algorithms have been programmed in HEXVIS, and some have not. For the ones which have not been programmed, I describe the techniques necessary. The techniques described here are by no means exhaustive, but are meant to illustrate the wide range of capabilities of the HEXVIS paradigm.

## 5.1 Voronoi Tessellation

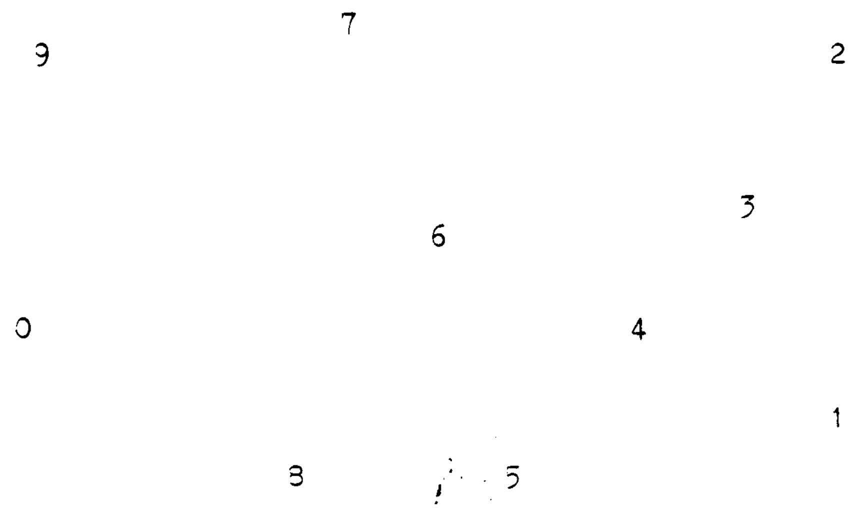
Given a dot scene (a scene consisting only of isolated vertices), the Voronoi polygon associated with a dot consists of all the points in the scene which are closer to that dot than to any other dot. The borders of the Voronoi polygons collectively form a Voronoi tessellation. (This should not be confused with the underlying regular hexagonal tessellation in which HEXVIS is embedded.) Ahuja [1980] points out that the Voronoi tessellation gives rise to a quite natural notion of neighbors of the dots in the scene, namely, a dot v' is a neighbor of v if and only if their Voronoi polygons share a border. He contrasts this with several other notions of dot neighbors such as considering all dots less than a given distance R to be neighbors, picking the K nearest neighbors, and minimal spanning trees.

HEXVIS can be programmed to find Voronoi tessellations in at least two different ways. The first method propagates messages only from those cells which contain the dots. The idea is that when the messages collide, the cell in which the collision takes place changes its state (indicating that the cell is on the Voronoi tessellation), and the messages in the cell are killed. Unfortunately, implementation of this idea proved to be highly non-intuitive. The first implementation had the problem that occasionally, the cells representing the Voronoi tessellation were not connected, i.e., the Voronoi tessellation contained gaps. This effect is due to the fact that HEXVIS uses a discrete array of cells rather than a continuous medium. If messages are destroyed when they are not alone in a cell, all their descendents will be aborted as well. If, for instance, a message would have produced two offspring at the next time period, one of those offspring might be needed to meet with a message coming from another direction in order to form part of the Voronoi tessellation. If the message is not there, a gap forms. The effect is especially pronounced when two neighboring (by sharing a border of their Voronoi polygons) dots are on the convex hull of the set of dots. Their common border should extend to the edge of the cellular array, but since no message copies which originated from these cells occupy any cell in common at great distances (they are traveling in roughly straight lines away from the midpoint of the two originating cells), no border is formed. Another related problem is that the message which would have met the offspring of the message which was killed, itself lives on. It can travel through the gap and possibly meet with another message in a similar situation to form spurious Voronoi tessellation points.

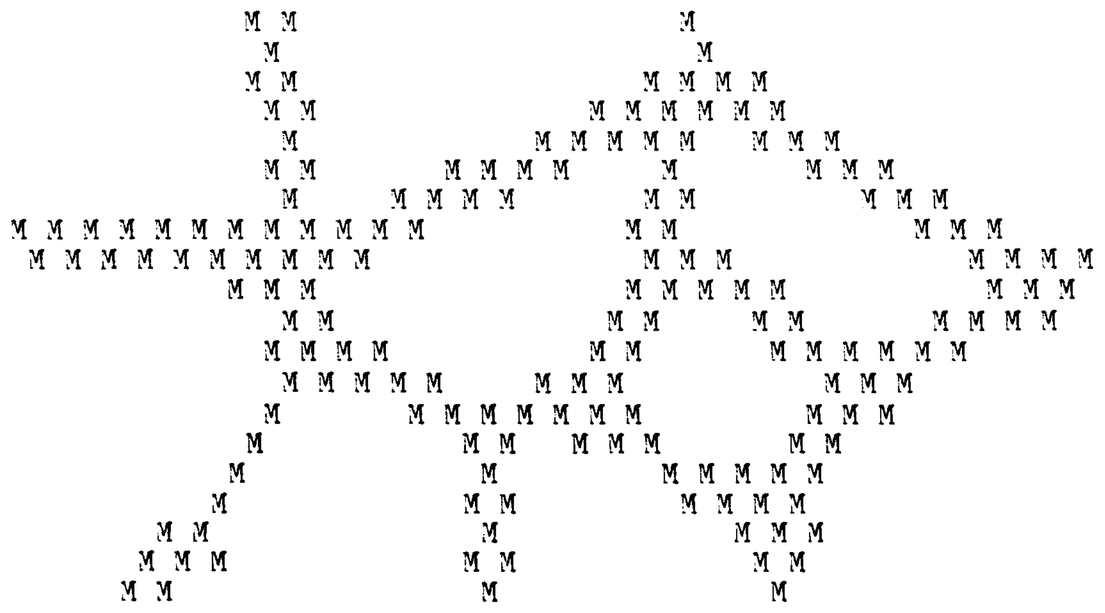
There is a partial solution to this problem. That is to not immediately kill of the messages which meet, but rather mark them and let them propagate for one more time unit, at which time they are killed (truly "marked men"). This method produces Voronoi tessellations with far fewer gaps than before.

The other way in which HEXVIS can be programmed to find the Voronoi tessellation of a set of dots is a sort of region growing technique. Again the messages are propagated only from the dots, which are given a unique identifier (e.g., their address in the cellular array). The messages produced in the cells carry with them their originating cell's identifier and leave it in the state of cells through which they pass. If the cell already has an identifier in its state, the new one is not left, but the messages are not killed. If a cell has two messages at the same time unit and has not already found an identifier, it chooses the identifier of the first message on its list, and, again, the messages are not killed. This method seems to be a better choice than the former method because gaps are not formed. However if the borders of the regions rather than the polygons themselves are desired, another layer must be added to derive them from the regions. This is easily done.

Figures 21a and 22a show an arrangement of dots in which both methods do a fairly good job of deriving the Voronoi tessellations. Figure 21b shows the Voronoi tessellation derived by the first method (collision) described above. The "M"'s in the figure are the points at which the wavefronts collide. Figure 22b shows the Voronoi tessellation for the same arrangement of dots but derived by the second method (region growing) described above. The numbers in both parts of figure 22 establish the correspondence between dots and voronoi polygons. I have circled each cell in figure 22b

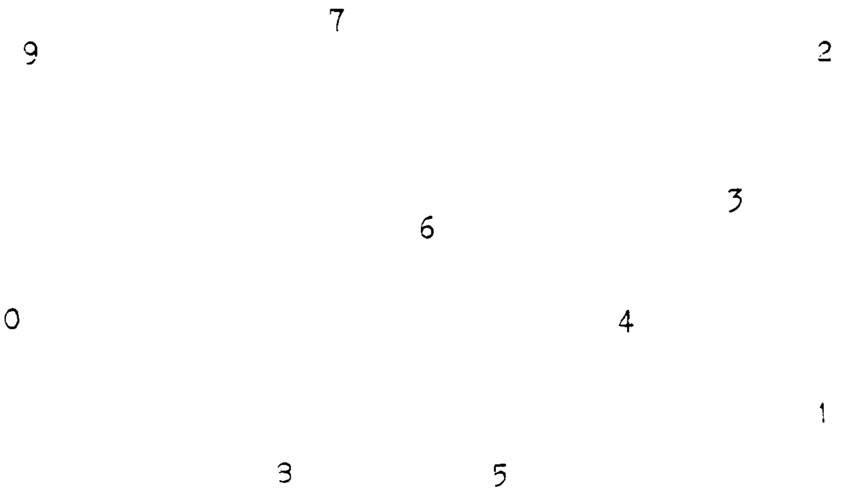


(a) Original Points

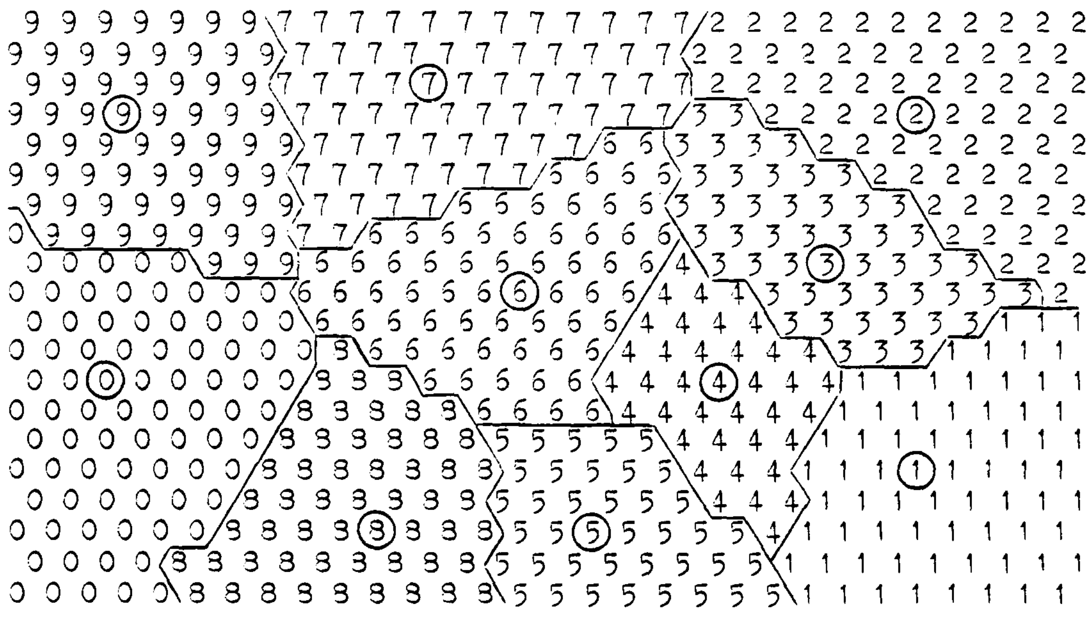


(b) Voronoi Tessellation

Figure 21 Voronoi Tessellation by Collision Method



(a) Original Points



(b) Voronoi Tessellation

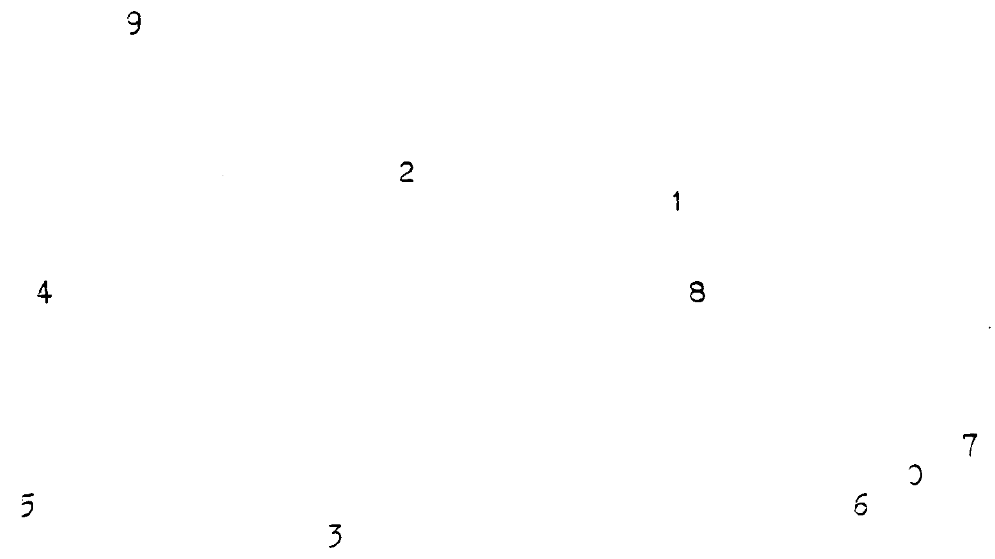
Figure 22 Voronoi Tessellation by Region Growing Method

which corresponds to one of the original dots and have drawn lines between the voronoi polygons.

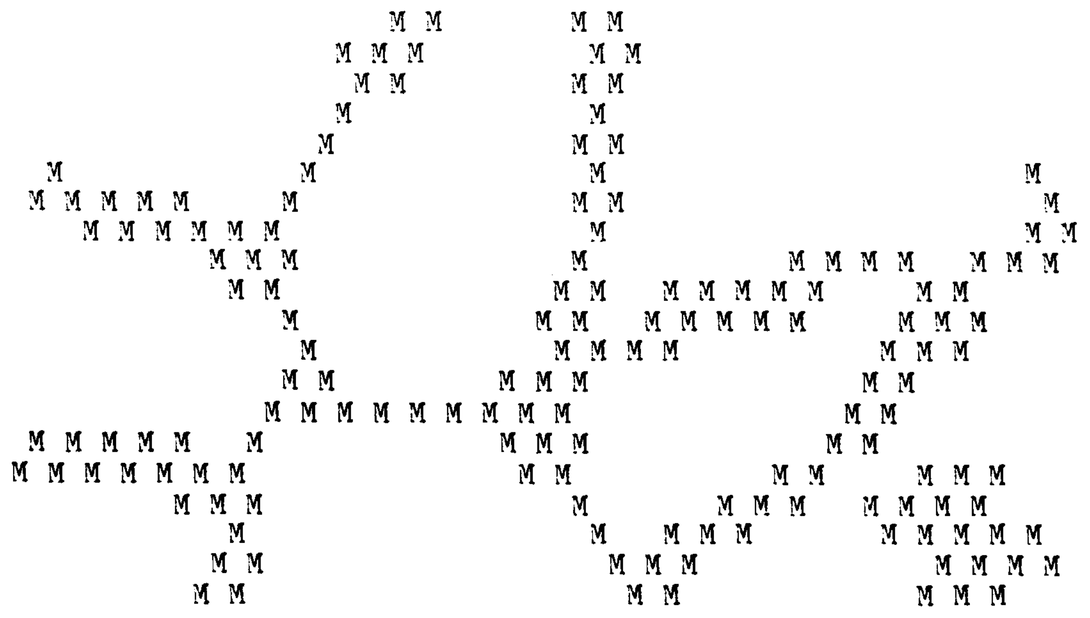
Figures 23a and 24a show an arrangement of dots in which the collision method does not do such a good job, but the region growing method does. Notice that in the lower right corner of figure 23b that there should be three separate regions. Instead, there is just a blob of cells. Furthermore, the cells do not even connect with the rest of the tessellation. Figure 24b shows that the region growing method can perform much better.

## 5.2 Prairie Fire Techniques

Blum ([1967] and [1978]) describes a function of an object in a scene called the symmetric axis transform. I will repeat the description given in the last chapter: Formally, the transform can be described as the set of the centers of all the circles which are wholly contained in the object but not wholly contained in any other circle of the set. The transform has a more intuitive description. If the object is imagined to be made of flammable prairie grass, and surrounded by concrete, a fire started on the border of the object will (if it burns at the same rate everywhere) burn itself out on just those points which are elements of the symmetric axis transform of the object. This (more picturesque) description, of course, gives rise to the more colloquial "prairie fire" label.

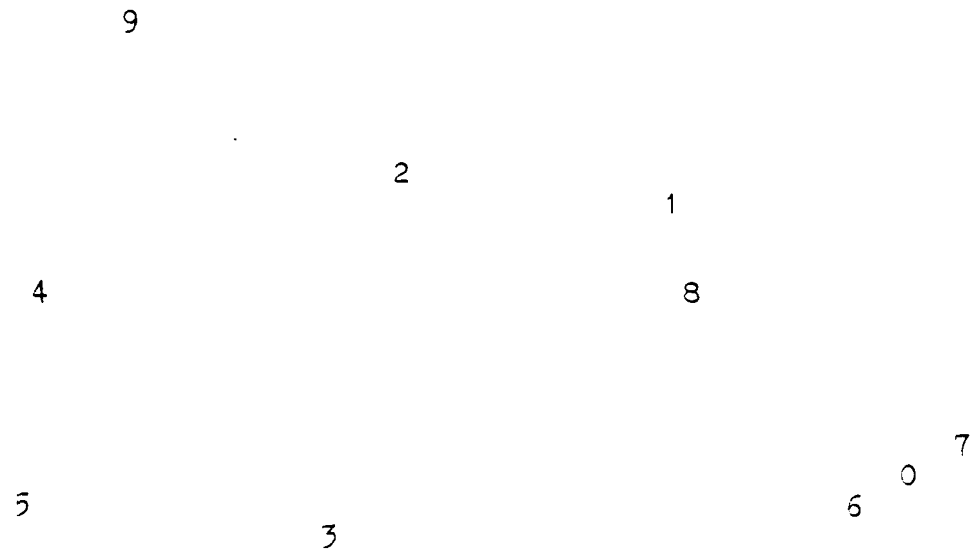


(a) Original Points

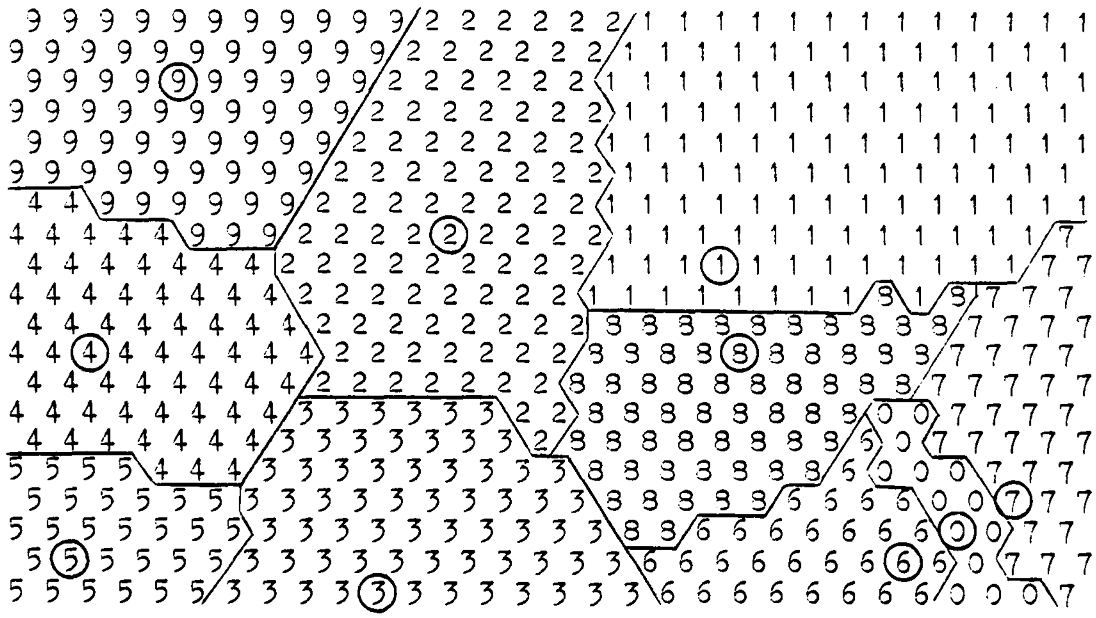


(b) Voronoi Tessellation

Figure 23 Voronoi Tessellation by Collision Method



(a) Original Points



(b) Voronoi Tessellation

Figure 24 Voronoi Tessellation by Region Growing Method

Implementation of this technique in HEXVIS is quite similar to the first method for implementing the Voronoi tessellation. Here, messages are propagated from each point on the border of the object (obtained by a lower layer). Messages are propagated only toward the inside of the object and are not allowed to produce a point (of the symmetric axis transform) if they are carrying local gradients which point in the same direction. This assures that messages from cells on the border do not produce points everywhere inside the object.

## 5.3 Area

The area of an object in the hexagonal array is defined as the number of cells which compose the object. In the case of convex objects, an algorithm can be described which will determine, at each cell in the object, the area of the entire object. Consider binary scenes. We assume for the purpose of this discussion that a binary scene can be obtained from another scene by the use of some sort of filtering or thresholding technique. I will speak of "1-messages" and "0-messages" to indicate that they originate respectively from a cell which is part of an object and one which is not.

The method of determining the area of a convex figure is for each cell to count the 1-messages as they pass through. If the cell at some time period sees that all the messages coming from a particular direction carry zeros (implying that their originating cells are not parts of any object), then the cell should stop counting 1-messages in that direction. If it were to continue, the cell would run the risk of counting 1-messages from another object in the scene.

This technique is not error-free. One problem with it is that in some cases a cell could receive two 1-messages at the same time from the same direction but from cells which are part of separate objects. This could happen if (for instance) both objects had elongated sections which were close together (but not touching). Then a cell in one of the objects in the direction of the elongation would see no break in the 1-messages. One way to avoid this problem is to propagate messages only through areas which are similar to their originating cells.

In the case of concave objects, the problem is more difficult because some cells in the object will not be in the "line of sight" of other cells in the object. These cells might therefore experience breaks in the 1-messages from one or more directions causing them to discontinue counting in that direction even though subsequent 1-messages may originate from the same object. Even in this case, there is a possibility that the area can still be determined. That is, if there is a cell (call it "C") in the object from which all other cells in the object can be "seen". C then will determine the correct number for the area of the object. Of course, the number it determines will be larger or equal to the numbers obtained by any of the other cells in the object. This suggests a second layer to be applied in which the numbers in each cell representing the (possibly incorrect) area of the object are propagated as messages. If a message passes through a cell containing an area greater than the one in the cell, the new area is substituted. In this way all cells in the object will eventually contain the correct area because the messages originating in C which does have the correct area eventually reach all the cells in the object (for the same reason that the messages from all cells in the object eventually reached C).

In the case of the class of non-convex objects without a cell C as described above, different techniques must be applied to determine the area. It may be though that in many applications, one is guaranteed that the objects of interest do not fall into such a class.

## 5.4 Texture

There are many ways to represent textural information. See, for example [Ahuja, 1979], [Rosenfeld, 1976], and [Haralick, 1978]. The technique described here assumes a model of texture which is characterized by the distribution of gradient directions and magnitudes.

My first attempt at texture identification and discrimination employed a histogram technique. Each cell contains its own twelve-element histogram which is stored in the cell's state. Each message in this technique carries the local gradient direction and magnitude present in its originating cell. As a message passes through a cell, the histogram entry corresponding to the gradient direction carried by the message is incremented. (Of course, thresholding techniques would be useful here, too, to sensitize the cells to a particular magnitude.) The hope was that, given similar enough propagation time, similar textures would have similar histograms. As a test of this technique, I filled the cells in the array with random gradient directions. These were formed into messages and propagated in the usual way as the cells built up histograms. The messages were allowed to propagate for six time units which means that ninety-nine messages passed through each cell (except the cells close to the edge, of course). Figure 25 shows several examples from the set of histograms generated. These are typical of the histograms generated by this technique. The features which one might expect to be similar in such histograms are the number, location, and values of maxima and minima, total number of messages which contribute (although in this case, all cells generated messages), average (again, in this case the same) and standard deviation of the histogram entries, and so on. It is difficult to find any features of the group in the figure which are similar. If the messages were allowed to propagate for a longer time, the histograms would give a more representative view of the texture, but then there are the problems of losing resolution

```
 1.  XXXXX                       1.  XXXXXXXXXX
 2.  XXXXXX                      2.  XXX
 3.  XXXXXXX                     3.  XXXXXXX
 4.  XXXXXXXXXXXXXXX             4.  XXXXXXXX
 5.  XXXXXX                      5.  XXXXXX
 6.  XXXXXXXXX                   6.  XXXXXXX
 7.  XXXXXXXXXXX                 7.  XXXXXXXXXXX
 8.  XXXXXXXXXXX                 8.  XXXXXXXXXXXXXXXXXX
 9.  XXXXXXXXXX                  9.  XXXXXXXX
10.  XX                         10.  XXXXXXXX
11.  XXXXX                      11.  XXX
12.  XXXXXXXXXXXX               12.  XXXXXXXXXX

 1.  XXXXXXXXX                   1.  XXXXXXX
 2.  XXXX                        2.  XXXXXXXXXXXX
 3.  XXXXXXX                     3.  XXXXXX
 4.  XXXXXXX                     4.  XXXXXXXXXXX
 5.  XXXXXXXXX                   5.  XXXXXXXXXX
 6.  XXXXXXXXXXX                 6.  XXXX
 7.  XXXXXX                      7.  XXXXXXX
 8.  XXXXXXXXXXXXXX              8.  XXXXXXXXXX
 9.  XXXXXXX                     9.  XXXXXXX
10.  XXXXXXXX                   10.  XXXXXXX
11.  XXXXXXXX                   11.  XXXXXXXXX
12.  XXXXXXXXX                  12.  XXXXXXXXX

 1.  XXXXXXXX                    1.  XXXXXXXXX
 2.  XXXXXXXXX                   2.  XXXXXX
 3.  XXXXX                       3.  XXXXXXXX
 4.  XXXX                        4.  XXXXXXXXXX
 5.  XXXXXXXXXX                  5.  XXXXXXXXXX
 6.  XXXXXX                      6.  XXXXXXXXXXX
 7.  XXXXXXXXXXXX                7.  XXXXXXXX
 8.  XXXXXXXXXXXXXXX             8.  XXXXXXX
 9.  XXXXXXX                     9.  X
10.  XXX                        10.  XXXXXXXXXXXXXX
11.  XXXXXXXXX                  11.  XXXXXXXX
12.  XXXXXXXXXXX                12.  XXXXXXX
```

Figure 25  Histograms of Local Gradient Directions

and errors at texture boundaries. As in the case of area, it again might be possible to constrain the input so that some feature does predominate.

There is another inherent limit to this technique, however -- at least as compared to human ability. That is, it can only discriminate textures that differ in their first-order statistics (of local gradient direction, in this case). Julesz [1965] has pointed out that humans can differentiate textures which have the same first-order statistics but differ in their second-order statistics (i.e., how groups of texture elements occur as opposed to how they occur singly).

A promising alternative to the previous method of texture analysis is to use generalized cooccurrence matrices ([Dyer, 1979], [Davis, 1979]). The use of this tool presupposes a different model of texture than assumed in the former technique. In this case a textured area is considered an arrangement of primitive texture elements. This model is not always accurate, for example a rippled surface on a lake is not easily describable in terms of primitive texture elements. In this case generalized cooccurrence matrices may not be as useful a tool. A generalized cooccurrence matrix (GCM) describes the texture of scene in terms of how often a feature is in the neighborhood of another feature. "Neighborhood" is a term whose definition can vary. As an example, consider the feature, gradient direction, quantized to (say) twelve levels (as in HEXVIS). Let the neighborhood be defined as the cells at a distance d. Then the GCM G consists of a twelve by twelve matrix in which an entry $g_{ij}$ is the number of times edge direction i occurs exactly d cells from edge direction j. Notice that when d is approximately equal to the size of the texture element, information about the texture elements can be inferred from the GCM. Other features which have been considered in addition to edge direction are grey level and local edge maxima.

The implementation of GCM's in HEXVIS is straightforward. Information on the feature of interest is propagated in such a way that the cells are able to receive the information from each cell in the neighborhood. If for instance the cell's neighborhood consists of the cells a distance d away (as above), the cells ignore the messages passing through them until time unit d, when the messages from all the cells in the neighborhood of each cell C are resident in C. At this point, C is in a position to make a contribution to row i of the GCM, where i is the gradient direction contained in C originally. A second layer is now applied in which this contribution to GCM is propagated. There are two choices for the second layer process depending on what is desired. If our goal is segmentation based on local texture, the cells propagate the messages for d time units. Then each cell can build its own GCM, which characterizes the texture in its immediate neighborhood (of radius 2d). On the other hand, the goal may be to generate a GCM for the entire scene, in which case, the messages are propagated as far as possible, so that every cell has access to the information in every other cell. Then all the cells can construct a GCM. This is of course somewhat wasteful, since each cell computes the same output. An alternative is to not apply the second layer, but let a global processor examine the cells' contents.

## 5.5 Septrees

Septrees were discussed in Chapter 3 [Burt, 1979]. To implement them in HEXVIS this assumption must be made: The hexagonal array can be reformatted so that there are a smaller number of cells, but the important information can be retained. The goal is that one out of every seven cells will become the site for a new cell containing information from itself and its six neighbors. The new cells' sites are arranged as shown in figure 26. In the figure the first layer cells are all the numbered sites; the second layer cells are all the sites with numbers 2 or greater; the third layer cells are all the sites with number 3 or greater, and so on. In the figure, all of the cells in each fourth layer site are outlined. One of these is further subdivided to show how the third layer and second layer is arranged. The cells which are circled are the ones which form the sites for the new (larger) cells. Notice how every other layer is rotated slightly to the right.

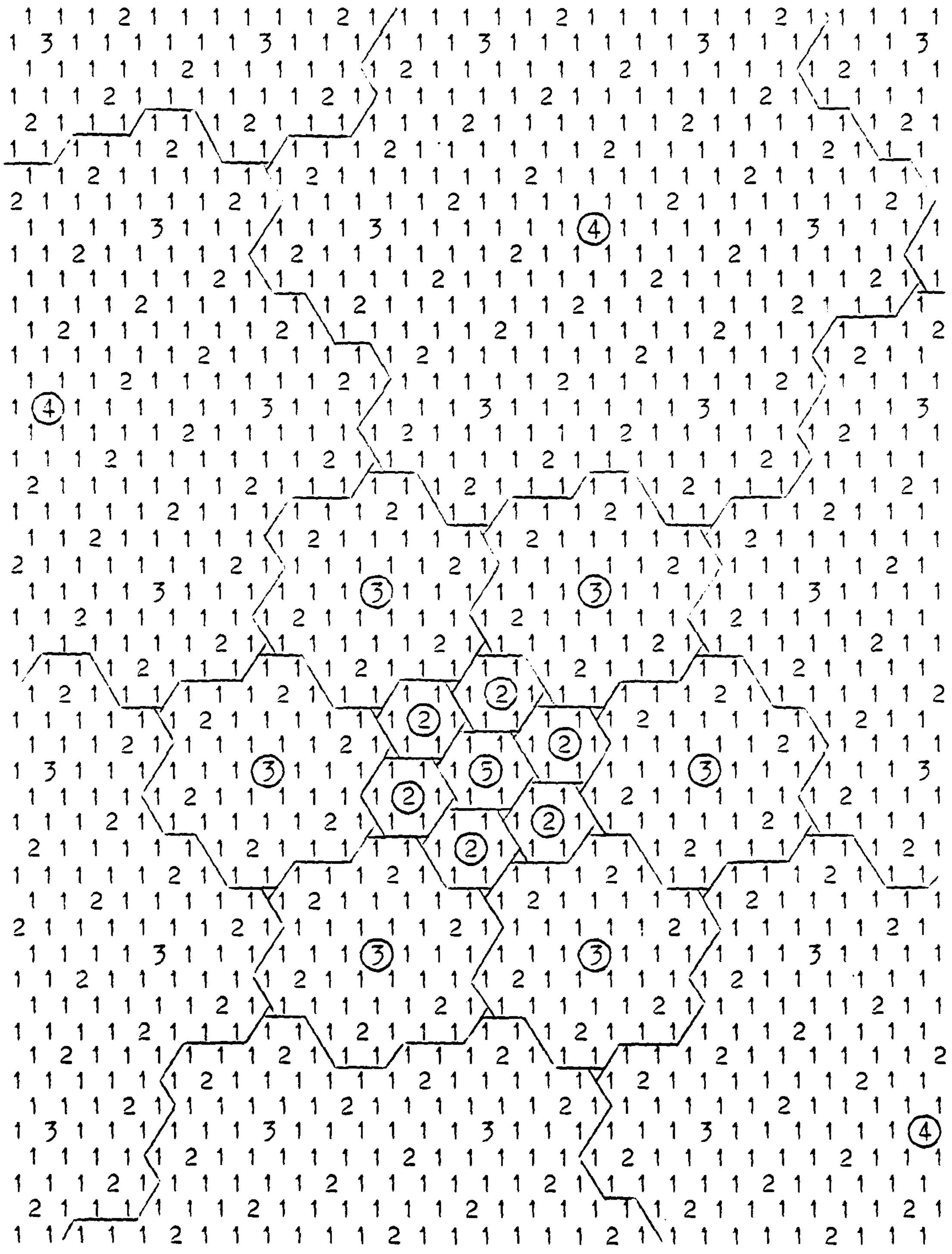


Figure 26  Septree Cell Sites

There are two ways this cell rearrangement could be performed. The first way is simple in a software simulation but not in a hardware realization. This method shifts the desired information to one part of the array and only uses the cells in that part. In this arrangement, each of the cells used corresponds to a larger area than it did before.

A better way is to use higher level rules to influence the behavior of message passing. I will first describe how this can be done in general. Then I show that the full power is not needed for this particular application. However, applications in which the array shrinks at each layer can be imagined.

A sort of virtual message passing scheme can be overlaid on the cells in the array. In this scheme HEXVIS would essentially be simulating itself. Consider first the cells numbered 2 or more in figure 26. To make these cells appear to be neighbors, we add to the messages a pseudo-heading. After three time units, the messages are in the proper cells as well as several other cells. All the cells which are not participating in the layer (all the cells numbered 1 in the figure) destroy their resident messages. The other cells determine which direction their resident messages are headed by looking at the messages' pseudo-headings. These cells then change the <u>real</u> heading of each of their messages to propagate them in the proper direction to meet the correct virtual neighbor. After three more time units, this process repeats itself. In this way HEXVIS behaves as if it has fewer cells, but can retain as much information as necessary from lower layers. For convenience, I will refer to this process as "compression". Of course, the disadvantage to this scheme is that more time is required (a factor of at least three) to propagate the messages.

In the case of generating septrees, as I mentioned, we do not need the full power of this propagation scheme, since the messages are only required to propagate to the immediate neighbors of their originating cells. The septree is generated by first causing a message to be generated with the information "black" or "white" depending on whether the cell is part of an object or not. These messages are propagated for one time unit. Then compression occurs. The cells which are part of the second layer (they know who they are) form new messages. These messages can be one of three types: 1) black, if the cell and all six of its neighbors are black, 2) white, if the cell and all six of its neighbors are white, or 3) grey, if neither of these cases obtains. In the grey case, the cell forms an ordered tree with seven branches (a septree) describing its own label and the labels of each of its neighbors. This septree is included in the message which it forms. There is one cell in the scene which has a number higher than any other (5 in figure 26). As this process is applied recursively, all the information converges there, and this cell forms a message which is the septree for the entire scene.

AD-A096 569 ILLINOIS UNIV AT URBANA COORDINATED SCIENCE LAB F/G 9/4
A CELLULAR AUTOMATA APPROACH TO COMPUTER VISION AND IMAGE PROCE--ETC(U
SEP 80 G D HADDEN N00014-75-C-0612
UNCLASSIFIED T-100 NL

# CHAPTER 6

# A PARALLEL SHRINKING ALGORITHM FOR HEXAGONALLY TESSELLATED BINARY ARRAYS

Many parallel binary shrinking algorithms have been described in the literature ([Golay, 1969], [Levialdi, 1972], [Rao, 1976], [Rosenfeld, 1970]). The purpose of such algorithms is to map a binary scene (that is, a scene whose points are either ones or zeros, ones belonging to the objects and zeros to the background) into a set of points such that each connected part of the scene corresponds to one and only one point. No unconnected parts of the scene are allowed to merge and no connected parts of the scene are allowed to become disconnected. The main use of these algorithms (aside from their purely mathematical interest) is to count objects in a scene, often in a biological setting (e.g., chromosomes, blood cells, etc.).

Rosenfeld proves several results about shrinking algorithms in general -- both sequential and parallel. However, in his discussion of parallel shrinking, he assumes the objects are simply connected, i.e., they have no holes. He also assumes that no zero-points can change to ones.

Levialdi presents a parallel shrinking algorithm which uses a two by two neighborhood and compresses objects toward the upper-right corner. All objects are compressed to a point which is counted before it disappears. Since the point does disappear, objects with other disconnected objects within them may also be shrunk to points. Levialdi proves that his algorithm satisfies three criteria: 1) connected objects will not split, 2) disconnected objects will not merge, and 3) all objects will shrink to isolated points.

Rao, Prasada, and Sarma developed a parallel shrinking algorithm which uses a three by three neighborhood. One feature of this algorithm is its symmetry. Barring interference from nearby objects, each object shrinks to a point at its center. Although a proof is not presented, the authors claim their algorithm satisfies the three criteria mentioned for Levialdi's algorithm for any scene. In fact, it does not. Their algorithm will not shrink objects which contain other disconnected objects. The only ways this can be done is to break the connectivity of the enclosing object, let the single point disappear (as Levialdi does), or let the enclosing object somehow shrink over the enclosed object.

Breaking the connectivity of an object requires a global knowledge of the object's structure. This information is not available locally, so this choice is out. Letting an object shrink over another one can be done by moving the inside object to a parallel array when it becomes a point. This is essentially the same thing as removing it altogether except that positional information is retained.

Golay presents (again without proof) an operator for shrinking patterns on a hexagonal grid. However, the operator will not shrink objects with holes, instead transforming them to a set of connected loops one cell thick, each loop corresponding to a hole in the original object. The operator, like Rao, et. al's, shrinks objects in a symmetrical manner. Golay has an interesting way of assuring that the rules do not interact in strange ways. He partitions the hexagonal array into three sets such that no two points in any set are adjacent, then applies the operator to the points in each set in turn. This technique also allows him to perform parallel operations without using two arrays. Figure 27 shows the subfields which are labeled "A", "B", and "C". (This figure repeats figure 15 in Chapter 3.)

```
A B C A B C A B C A B C
 C A B C A B C A B C A B
A B C A B C A B C A B C
 C A B C A B C A B C A B
A B C A B C A B C A B C
 C A B C A B C A B C A B
```

Figure 27 Golay's Three Subfields

All of the algorithms described above with the exception of Golay's operate on binary scenes which are embedded in a rectangular grid. The algorithm I describe (like Golay's) operates on scenes in a <u>hexagonal</u> grid. As in Rao, Prasada, and Sarma's algorithm, mine shrinks objects symmetrically, collapsing them to a point in the center. The algorithm works on any scene except those in which an object is enclosed by another, although it would be trivially easy to add another parallel array (as described above) and a transition rule to move isolated points to this array.

One aspect of my shrinking algorithm which sets it apart from most of the techniques developed for the HEXVIS paradigm is that this algorithm most closely resembles the classical finite state cellular automata model. Of course it can be (and is) implemented as a program in HEXVIS, but the shrinking algorithm does not require HEXVIS's full message passing power. On the other hand, the algorithm as expressed is not quite a true cellular automata system either. The transition rules cycle through a set of three sets rather than remaining constant from cycle to cycle. This last is a minor point, though, since we can expand the neighborhood, and collapse three cycles into one to produce a finite state cellular automata, albeit with astronomical sets of states and transitions rules. In any case, my description of the process will be in the terms of cellular automata. The reader should keep in mind the fact that this description can easily be embedded in the HEXVIS paradigm.

## 6.1 The Algorithm

Parallel shrinking is accomplished by applying a set of transition rules to each point in the scene in parallel. If the pattern of the point and its neighborhood matches one of the rules, the point is complemented.

Figure 28 shows some typical rules. The significance of the figure is that if a cell and its neighborhood matches the arrangement of ones and zeros shown, the cell changes state.

```
 0 0      1 0      1 1
0 1 1    1 0 1    1 0 1
 0 1      1 1      1 1
```

Figure 28 Some Typical Transition Rules

Notice that zeros can be changed to ones as well as vice versa. This feature is included so that holes in objects can be filled allowing multiply connected objects to be shrunk. Note that the only other way to delete holes in multiply connected objects is to break the connectivity at some point. This act, as I mentioned above, requires a global knowledge of the object's topological structure before it can be performed.

Each cycle of rule application is divided into three subcycles. During each subcycle, a different subset of the rules is applied. Why not apply all the rules at once? Consider the two-cell object in figure 29. Each point in the object has the point of view (judging by its neighborhood) that it is on the end of a line segment. Since a line segment end is a highly convex part of a figure, there exist rules to change such points to zeros in order to contribute to the shrinking of the object. Of course, if both points change to zeros, the object would disappear instead of shrinking to a point. To prevent this situation from occurring there must be more than one subcycle so that the two rules responsible for this shrinking are applied at separate times. The reason I chose three subcycles rather than two is due to a combination of a peculiarity of the hexagonal grid and symmetry considerations (both to be described later).

```
 0 0 0
0 1 1 0
 0 0 0
```

Figure 29 Two-Cell Object

### 6.1.1 Centers of Gravity, Convexity, and Concavity

A concept which I have found quite useful when discussing the shrinking algorithm is "center of gravity". When we change a one to a zero or a zero to a one, the center of gravity of the ones in the cell and its neighborhood shifts in some direction. If we consider only the configurations whose connectivity is preserved by the change, then the shift is restricted to the twelve clock directions (or to no shift at all).

```
  1 0      1 1      1 1      1 1      1 1      1 1
 0 C 0    0 C 0    0 C 1    0 C 1    0 C 1    1 C 1
  0 0      0 0      0 0      0 1      1 1      1 1
```

Figure 30  Cases In Which Connectivity of the Cell's Neighborhood Does Not Depend on the Cell's State

Figure 30 shows the six cases in which the connectivity of the neighborhood of the cell C is not affected by the cell's state. (Rotations of these cases are not shown but the application of this discussion to them should be obvious.) If we consider changing C from a zero to a one (the change from one to zero shifts the center of gravity in the opposite direction), the center of gravity shifts in the five o'clock, six o'clock, seven o'clock, eight o'clock, and nine o'clock directions for the first five configurations respectively. In the last configuration, the center of gravity does not shift when the state of C changes.

With this discussion in mind, the rules can be classified by the direction in which they cause the center of gravity to shift. If a rule causes the center of gravity to move in the four o'clock direction (for instance), it is also said to cause the object to shrink from the ten o'clock direction.

Two other concepts which will be important in later arguments are those of "convex" and "concave" points. Again referring to figure 30, if C is in the "one" state, then the first, second, and third neighborhoods are said to be "convex" and C is said to be a "convex point" and "on a convexity". If C is in the "zero" state, then the third, fourth, fifth, and sixth neighborhoods are "concave", and C is said to be a "concave point" and "in a concavity". These definitions apply in the obvious way to the rotations of the neighborhoods.

### 6.1.2 A Problem

One problem I discovered while writing the program to implement the shrinking algorithm is inherent to the hexagonal grid. As in Rao, Prasada, and Sarma's algorithm, I originally had two subcycles, each of which shrunk from opposite sets of directions. I chose the rules by considering every possible pattern on the six cell neighborhood (except for all zeros) for which the central cell's state did not affect the connectivity. I picked the rules from these patterns such that when applied, they would change a zero in a concavity to a one and a one in a convexity to a zero. In running several test cases, everything went well until I tried a large hollow hexagon. To my chagrin the points just inside the six corners of the hexagon oscillated between zero and one instead of continuing inward and causing the hexagon to shrink.

The reason for this effect is subtle, and I will explain this effect by comparing it to the corresponding situation in the (eight-neighbor) rectangular grid.

```
0 1 0 0 0    0 0 0 1 0
0 1 0 0 0     0 0 1 0 0
0 1 0 0 0    0 0 1 0 0
0 1 1 1 1     0 0 1 0 0
0 0 0 0 0    0 0 0 1 0
```

Figure 31  Corners in Square and Hexagonal Arrays

Figure 31 shows a corner in both a square grid and in a hexagonal grid. Applying the rules as described above to figure 31 yields figure 32.

```
0 1 0 0 0    0 0 0 1 0
0 1 0 0 0     0 0 1 0 0
0 1 1 0 0    0 0 1 1 0
0 0 1 1 1     0 0 1 0 0
0 0 0 0 0    0 0 0 1 0
```

Figure 32  Corners After One Shrinking Step

In both cases, the change has apparently produced the desired effect: the figures appear to be on their way to becoming points. However consider what the change has produced in the hexagonal case. The concave inside corner has become convex. In contrast, in the rectangular case, the convex corner has become a straight diagonal line which is neither convex nor concave. Since the rules as originally implemented for the hexagonal array filled all concavities while eroding all convexities, on the next cycle, a rule was applied which changed the hexagonal scene back to what it was originally.

At first, I thought I would have to be content with a scheme like Levialdi's which compressed the objects asymmetrically. However, I discovered an algorithm which was made symmetric by leaving out certain rules and breaking the cycle into three rather than two subcycles. The transition rules which caused the problem come in six pairs. Figure 33 shows one of them. The other five are obtained by rotation.

```
 1 0        1 0
1 0 0      1 1 0
 1 0        1 0

 (a)        (b)
```

Figure 33 A Pair of Problem Rules

The symmetric solution comes from using only one of each pair of rules. This is done by choosing only the rules from the problem set which move the center of gravity in the three o'clock, seven o'clock, and eleven o'clock directions. So, from the above pair, (a) would be chosen and (b) discarded. These rules shrink the object toward its center and do not interfere with each other in the manner described above. The complete set of transition rules is shown in figure 34. Each subcycle contains only rules which move the center of gravity in one of four directions (if they move it at all). Subcycle one moves the center of gravity in the two, three, four, and five o'clock directions, subcycle two in the six, seven, eight, and nine o'clock directions, and subcycle three in the ten, eleven, twelve, and one o'clock directions. The rule

```
 0 0    0 0    0 1    0 0    0 1
0 1 1  0 1 0  0 1 1  0 1 1  0 1 1
 0 0    0 1    0 0    0 1    0 1

 1 1    1 1    1 0    1 1    1 0    1 1
1 0 0  1 0 1  1 0 0  1 0 0  1 0 0  1 0 1
 1 1    1 0    1 1    1 0    1 0    1 1
```

subcycle 1

```
 0 0    0 0    0 0    0 0    0 0
0 1 0  1 1 0  0 1 0  1 1 0  1 1 0
 1 0    0 0    1 1    1 0    1 1

 1 1    1 1    1 1    1 1    1 1    1 1
1 0 1  0 0 1  1 0 1  0 0 1  0 0 1  1 0 1
 0 1    1 1    0 0    0 1    0 0    1 1
```

subcycle 2

```
 1 0    0 1    1 0    1 1    1 1
0 1 0  0 1 0  1 1 0  0 1 0  1 1 0
 0 0    0 0    0 0    0 0    0 0

 0 1    1 0    0 1    0 0    0 0    1 1
1 0 1  1 0 1  0 0 1  1 0 1  0 0 1  1 0 1
 1 1    1 1    1 1    1 1    1 1    1 1
```

subcycle 3

Figure 34 The Transition Rules

```
 0 0      1 0      1 1
0 1 1    1 1 0    0 1 1
 1 1      1 0      0 0

 1 1      0 1      0 0
1 0 0    0 0 1    1 0 0
 0 0      0 1      1 1
```

Figure 35 Rules Which Were Left Out

representations are arranged within each subcycle so that the eroding rules (the ones which change ones to zeros) are on top and the hole-filling rules (the ones which change zeros to ones) are on the bottom. Complementary rules are grouped vertically. One of the rules (shown last in the figure) appears in each subcycle. Notice that it has no dual represented, for its dual would eliminate single points. This rule is the only one which does not alter the center of gravity. Figure 35 shows the rules which were left out. These rules are grouped in a manner similar to figure 34.

There are two important features to notice about these rules. First, in any one subcycle, no two rules can move the center of gravity in opposite directions. Second, none of the rules applied singly alters the connectivity of the scene. These features will be important in the proof of the algorithm.

## 6.2 Proof of the Algorithm

In this section I prove that when the rules shown in figure 34 are applied in a cyclic manner (i.e., subcycle one, followed by subcycle two, followed by subcycle three, followed by subcycle one, etc.) and in parallel to a scene, each object in the scene, with the exception of those objects which completely surround other objects, shrinks to a point.

First, I will show that repeated application of the rules in cyclic order will not destroy the scene's connectivity. I consider the case of two disconnected objects merging, then the case of one object splitting into two disconnected objects.

Theorem 1: two disconnected objects can never merge by application of the rules shown in figure 34 in the cyclic manner described above.

Proof: Notice first that no single rule can disconnect a connected object or merge two disconnected objects. However, since the rules are applied in a parallel manner, it might be possible that two adjacent cells could change simultaneously in such a way that their connectivity properties are altered.

I consider first the case of two disconnected objects merging. We need not consider combinations of more than two rules. Why is this? Suppose that the merging of any two objects (X and Y, say) requires at least three cells (called "A", "B", and "C") to change from zeros to ones to effect the merging. Then one of these three cells (C, say) must be at least two cells away from both X and Y, since if it were adjacent to one of them and also adjacent to either A or B, then A or B would not be required, giving a contradiction. Informally, we imagine a line of cells, (A, C, and B, in that order), running from object X to object Y, all of which must change from zeros to one to connect X and Y. In order for C to change from a zero to a one, it must have had cells in its neighborhood which were ones, since no rule allows for spontaneous generation of ones from all zero neighborhoods. If this is the case, C is adjacent to yet another object Z (not equal to X or Y since neither X nor Y have any cells in C's neighborhood) which will merge with both X and Y but is only two cells away from each of them. Therefore objects X and Z (also Y and Z) merge by cells A and C (B and C) changing to ones. This contradicts our original assumption that the merging of two objects requires at least three cells to change from zeros to ones. This argument holds for any number of rules greater than two. Therefore, if the application of two rules in parallel can not merge objects, neither can the application of three or more.

For the parallel application of two rules to merge two objects, the two cells which change to ones must connect one cell on one object to one cell on the other object. The two possibilities are shown in figure 36. These and their reflections and rotations represent all the ways two objects can be separated by two zero cells.

```
  0 0 1
1 0 0          1 0 0 1
```

Figure 36 The Two Ways Objects Can Be Two cells Apart

For the first case, refer to figure 37. There are three ways for two rules to connect these objects: if y and z change to ones, if w and x change to ones, or if x and y change to ones. Notice that changing w and x to ones is the same as changing y and z to ones modulo a one hundred eighty degree rotation, so we need only consider one of these two cases (w and x) giving a total of two subcases.

```
 a b c          w = x = y = z = 0, initially
d w x 1
 1 y z e
  f g h
```

Figure 37  Case One

First, notice that all rules which change zeros to ones fill concavities. Therefore, if w is to change from a zero to a one, it must be the case that a and d are ones as well (recall that y is assumed to be a zero) in order for there to be a concavity which w could fill. For x to change from a zero to a one, c must be a one, and either b or z must be a one. But z is assumed to be a zero, and if b is a one, we have a contradiction, since the objects would then be initially connected.

```
 a b c          x = y = 0, initially
1 x y 1
 d e f
```

Figure 38  Case Two

Figure 38 shows the second subcase. Here, x and y (initially zero) must both change to ones to connect the objects. For x to change to a one, there must be a concavity for it to fill, so one of these must be true: 1) a and b are ones, 2) a and d are ones, or 3) d and e are ones.

Case 1): If both a and b are ones, then since c must be a zero (if c is a one, the objects are already connected), y's change alone would cause the objects to become connected. This can not happen because no one rule can alter connectivity.

Case 2): If a and d are ones, then for y to change state, it too must be in a concavity. The only way this can happen without the two objects being connected beforehand is for c and f to be ones and b and e zeros. But then, the centers of gravity must move in opposite directions, which is impossible in any one subcycle.

Case 3): The argument parallels case 1) above.

Therefore, two disconnected objects can not merge. Notice that the arguments made here apply also to my original set of rules, i.e., the union of those in figure 34 and those in figure 35 -- even if only two subcycles are used (as long as no two rules in one subcycle move the center of gravity in opposite directions). That is, even though objects may not always shrink with the rules in that set, no two objects will ever merge and (as is shown in the next theorem) no connected object will ever disconnect.

Theorem 2: one object can never be split into two disconnected objects by application of the rules shown in figure 34 in the cyclic manner described above.

Proof: This result is in a fundamental way identical to the previous one, since splitting an object into two by applying rules which change ones to zeros is the exact dual of merging two objects into one by applying rules which change zeros to ones. The duality depends on the fact that the rules, with one exception, form pairs with these properties: 1) one of the pair can be changed to the other by replacing all occurrences of ones with zeros and all occurrences of zeros with ones, 2) one of the pair changes zeros to ones while the other changes ones to zeros, and 3) both rules in the pair shift the center of gravity in the same direction. The exception mentioned above is the one rule which is applied during all the subcycles. This rule changes zeros which are completely surrounded by ones to ones. Notice that the dual of this rule if included would cause isolated one-points to disappear (a change that would not alter the the connectivity properties of the scene).

One might ask why, if the two sets of rules and the two states (zero and one) are duals, any shrinking occurs at all -- why don't the areas of zero state cells shrink? In fact this does occur and is the reason holes in objects get filled. However the symmetry is broken at the edge of the scene, i.e., the neighborhood of a cell on an edge effectively contains zeros in the positions where there are no cells. If the situation were such that these cells were assumed to be ones, and if the rule which fills isolated zero-points were replaced by its dual, the behavior of the system would be reversed -- all disconnected zero areas would shrink to isolated zeros with a "background" of cells in the one state. In other words, objects in the scene would be represented by zeros rather than ones.

Therefore, any result which depends on these rules is simultaneously established for its dual if the proof of the theorem does not take into account the fact that the "virtual edges" are zeros or the rule which fills isolated zeros. In particular the argument that an object will not be split into two does not depend on these asymmetric aspects so the dual result holds that two objects will not merge to form a single object.

I will now show that simply connected objects shrink to points by defining "frontier lines" which always enclose the (shrinking) object and which themselves move toward the center of the object.

Definition: For any object A in the scene, the lines oriented vertically, at sixty degrees to the left of vertical, and at sixty degrees to the right of vertical and positioned such that they pass through at least one border point of A and through no interior points and such that they are to the left, to the upper right, and to the lower right, respectively are called first, second, and third frontier lines (respectively) of A. Collectively, these lines are called the frontier of A. Figure 39 shows an example of a frontier.

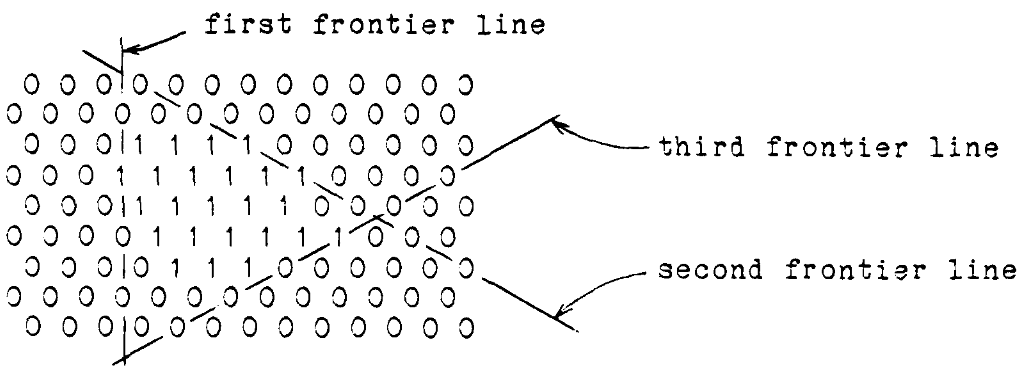


Figure 39 Frontier Lines

In many of the following lemmas I will consider the first frontier line only. It should be understood that this is done without loss of generality and applies equally well to the second and third frontier lines.

Lemma 1: Frontier lines do not move away from the object.

```
 0 1     0 1     0 0     0 0     0 0     0 1     0 0     0 1
0 0 1   0 0 1   0 0 1   0 0 0   0 0 1   0 0 0   0 0 0   0 0 0
 0 1     0 0     0 1     0 1     0 0     0 0     0 0     0 1
```

Figure 40 Possible Neighborhoods of a Zero-Point Next to the First Frontier Line of an Object

Proof: Without loss of generality we consider the first frontier line only. Figure 40 shows the possible neighborhoods of a zero-point which is next to the first frontier line of an object. Since there are no rules to change any of these zeros to ones, they will remain zeros. Notice that this is one case where the removal of the rules in figure 35 has an effect. In particular, the first neighborhood shown above was a rule in the previous version (the version which did not work).

Corollary: There are no "gliders" as in Conway's game of life [Conway, 1970]. That is, objects in the scene do not undergo any translation but remain always within definite predefined boundaries. I would conjecture that this corollary implies the cellular space in which the shrinking algorithm is embedded is not computationally complete, i.e., a computer (or a Turing Machine) can not be embedded in it.

Definition: One-points with neighborhoods shown in figure 41 when they appear respectively on first, second, and third frontier lines are called 1-, 2-, and 3-stragglers.

```
 0 1    0 0    0 0            0 1    0 0    0 1
0 1 0  0 1 0  1 1 0          0 1 0  1 1 0  1 1 0
 0 0    0 1    0 0            0 1    0 1    0 0
```

Figure 41 N-Stragglers          Figure 42 N-Connectors

Definition: One-points with neighborhoods shown in figure 42 when they appear respectively on first, second, and third frontier lines are called 1-, 2-, and 3-connectors.

Any one-point whose deletion would cause an object to split in two is called simply a connector.

Lemma 2: With the exception of N-stragglers and N-connectors, all points on frontier line N change to zeros during subcycle N.

```
  0 1     0 1     0 0     0 0     0 0     0 1     0 1
 0 1 1   0 1 1   0 1 1   0 1 0   0 1 1   0 1 0   0 1 0
  0 1     0 0     0 1     0 1     0 0     0 0     0 1
```

Figure 43 Possible Neighborhoods of a One-Point on the First Frontier Line of an Object

Proof: Figure 43 shows the possible neighborhoods for one-points on frontier line 1. In all but the last two cases (a 1-straggler and a 1-connector), a rule which is applied during subcycle 1 changes ones to zeros (see figure 34). Therefore, we are left with one of two situations: 1) if the frontier line N has no N-stragglers or N-connectors, it moves inward when the rules in subcycle N are applied or 2) if the frontier line N has one or more N-stragglers or N-connectors, then after subcycle N the only one-points on frontier line N are N-stragglers or N-connectors.

Lemma 3: After subcycle N, if the only one-points frontier line N contains are N-stragglers, these one-points will change to zeros during the next three subcycles, and the Nth frontier line will move in.

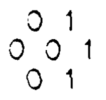


Figure 44 The Only Convex Point Allowed on Frontier Line 1

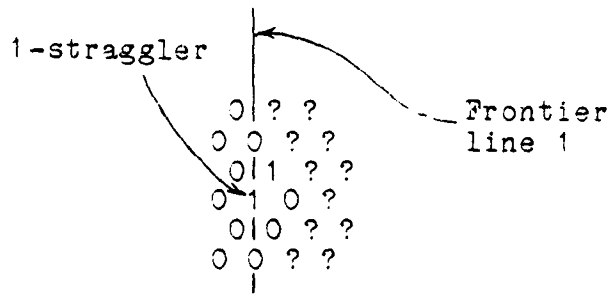


Figure 45 A 1-Straggler

Proof: First notice that since frontier lines do not move outward, no zero-point on frontier line 1 can be in a convexity except as shown in figure 44. There is no rule, however, which will change this to a one-point. Therefore, no new one-points can appear on a frontier line.

At subcycle two if the zero-point to the right of the 1-straggler (shown in figure 45) does not change to a one, then the 1-straggler will change to a zero at subcycle three. So suppose that zero-point does change to a one. In this case, the zero-point to the 1-straggler's lower right can not change to a one because it would have had to do so during subcycle two. So we enter subcycle one again with the situation shown in figure 46, and the ex-1-straggler changes to a zero.

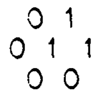


Figure 46 An Ex-1-Straggler

Theorem 3: All simply connected objects eventually shrink to points.

Proof: Combining the above results, we see that for objects without connectors, the frontier must move inward, taking the object with it. However, for objects with connectors or objects which develop connectors, we consider the two parts of the object (which the connector connects) separately. We may have to divide the object in this way several times (e.g., in the case of spirals), but eventually we will arrive at a part of the object which is connected to the rest by a connector but has no connectors itself. Since this sub-object is connected to the rest of the object by only one point, there must be (on the "other side" of the sub-object) at least one line analogous to the frontier lines which serves the same purpose on a local level for this sub-object. That is, this pseudo-frontier line will always move toward the inside of the sub-object. At each cycle, there is the possibility that the active pseudo-frontier line may change, but eventually the sub-object will shrink toward the connector. If the object (the whole object) has more than one connector or the sub-object develops another connector, the process is repeated.

Theorem 4: All holes (i.e., simply connected areas of zero-points) in an object shrink to isolated zero-points and disappear.

Proof: Appealing as before to the duality of the rules and of ones and zeros, we can see that all holes shrink to isolated zero-points just as all simply connected objects shrink to isolated one-points. The additional rule (see figure 47) which is applied during all subcycles changes these isolated zero-points to ones.

```
  1 1
 1 0 1
  1 1
```

Figure 47 Transition Rule Which Changes Isolated Zero-Points to Ones

Theorem 5: All objects (except those which completely surround other disconnected objects) shrink to single isolated points.

Proof: Since no two disconnected objects can be merged nor can any connected object be split in two by any number of applications of the rules, the connectivity of the scene is preserved. Since, for each object, any holes in it are filled and its frontier lines move inward, the object must shrink to a point. These facts imply that every object in a scene will shrink to a point.

## 6.3 Examples

On the following pages are shown examples of the operation of the shrinking algorithm. Figure 48 shows a solid hexagon shrinking. Figure 49 shows the shrinking of a hollow version of the same hexagon. Notice that the hollow hexagon takes only one cycle longer to shrink. It is my conjecture that this is always the case, that is, that the hollow version of an object will take at most one cycle longer to shrink than its solid counterpart. Figure 50 shows a spiral which is the same shape and size as the previous two hexagons. The shrinking of this object takes one and one-third cycles longer than the hollow hexagon. Unfortunately, I do not have any general results to report on a non-trivial upper bound for the number of cycles required to shrink objects. My guess would be that a tightly coiled spiral (as figure 50 represents\ would be among the set of objects with the longest shrinking times. This is an interesting area for further study.

The final example in this section is shown in figure 51. The objects represented are two interlocking spirals. Note that they shrink to two separate points.

(INITIAL OBJECT)

```
    # # # # #
   # # # # # #
  # # # # # # #
 # # # # # # # #
# # # # # # # # #
 # # # # # # # #
  # # # # # # #
   # # # # # #
    # # # # #
```

(CYCLE 1. SUBCYCLE 1.)

```
   # # # # #
  # # # # # #
 # # # # # # #
# # # # # # # #
 # # # # # # # #
# # # # # # # #
 # # # # # # #
  # # # # # #
   # # # # #
```

(CYCLE 1. SUBCYCLE 2.)

```
   # # # #
  # # # # # #
 # # # # # # #
# # # # # # # #
 # # # # # # # #
# # # # # # # #
 # # # # # # #
  # # # # # #
   # # # # #
```

(CYCLE 1. SUBCYCLE 3.)

```
   # # # #
  # # # # # #
 # # # # # # #
# # # # # # # #
 # # # # # # # #
# # # # # # # #
 # # # # # # #
  # # # # # #
   # # # #
```

(CYCLE 2. SUBCYCLE 1.)

```
  # # # #
 # # # # # #
# # # # # # #
 # # # # # # #
# # # # # # # #
 # # # # # # #
# # # # # # #
 # # # # # #
  # # # #
```

(CYCLE 2. SUBCYCLE 2.)

```
  # # #
 # # # # #
# # # # # # #
 # # # # # # #
# # # # # # # #
 # # # # # # #
# # # # # # #
 # # # # # #
  # # # #
```

(CYCLE 2. SUBCYCLE 3.)

```
  # # #
 # # # # #
# # # # # # #
 # # # # # # #
# # # # # # # #
 # # # # # # #
# # # # # # #
 # # # # #
  # # #
```

(CYCLE 3. SUBCYCLE 1.)

```
 # # #
# # # # #
 # # # # # #
# # # # # # #
 # # # # # # #
# # # # # # #
 # # # # # #
# # # # #
 # # #
```

Figure 48  Shrinking of a Solid Hexagon

```
(CYCLE 3. SUBCYCLE 2.)

 # #
# # # #
 # # # # #
# # # # # # #
 # # # # # # #
# # # # # # #
 # # # # # #
# # # # #
 # # #
```

```
(CYCLE 3. SUBCYCLE 3.)

 # #
# # # #
 # # # # #
# # # # # # #
 # # # # # # #
# # # # # # #
 # # # # #
# # # #
 # #
```

```
(CYCLE 4. SUBCYCLE 1.)

# #
 # # #
# # # # #
 # # # # # #
# # # # # # #
 # # # # # #
# # # # #
 # # #
# #
```

```
(CYCLE 4. SUBCYCLE 2.)

#
 # #
# # # #
 # # # # #
# # # # # # #
 # # # # # #
# # # # #
 # # #
# #
```

```
(CYCLE 4. SUBCYCLE 3.)

#
 # #
# # # #
 # # # # #
# # # # # # #
 # # # # #
# # # #
 # #
#
```

```
(CYCLE 5. SUBCYCLE 1.)

 # #
  # # #
 # # # # #
  # # # # # #
 # # # # #
  # # #
 # #
#
```

```
(CYCLE 5. SUBCYCLE 2.)

 #
  # #
 # # # #
  # # # # #
 # # # # #
  # # #
 # #
#
```

```
(CYCLE 5. SUBCYCLE 3.)

#
 # #
# # # #
 # # # # #
# # # #
 # #
#
```

Figure 48 (continued) Shrinking of a Solid Hexagon

(CYCLE 6. SUBCYCLE 1.)

```
 # #
  # # #
 # # # # #
  # # #
 # #
#
```

(CYCLE 6. SUBCYCLE 2.)

```
 #
  # #
 # # # #
  # # #
 # #
#
```

(CYCLE 6. SUBCYCLE 3.)

```
#
 # #
# # # #
 # #
#
```

(CYCLE 7. SUBCYCLE 1.)

```
 # #
  # # #
 # #
#
```

(CYCLE 7. SUBCYCLE 2.)

```
 #
  # #
 # #
#
```

(CYCLE 7. SUBCYCLE 3.)

```
#
 # #
#
```

(CYCLE 8. SUBCYCLE 1.)

```
 # #
#
```

(CYCLE 8. SUBCYCLE 2.)

```
 #
#
```

Figure 48 (continued) Shrinking of a Solid Hexagon

```
(CYCLE 8. SUBCYCLE 3.)          (CYCLE 9. SUBCYCLE 1.)



           #                               #




(CYCLE 9. SUBCYCLE 2.)          (CYCLE 9. SUBCYCLE 3.)



           #                               #
```

Figure 48 (continued)  Shrinking of a Solid Hexagon

```
(INITIAL OBJECT)

    # # # # #
   #         #
  #           #
 #             #
#               #
 #             #
  #           #
   #         #
    # # # # #
```

```
(CYCLE 1. SUBCYCLE 1.)

    # # # # #
   #         #
  #           #
 #             #
# #             #
 #             #
  #           #
   #         #
    # # # # #
```

```
(CYCLE 1. SUBCYCLE 2.)

    # # # # #
   #       # #
  #           #
 #             #
# #             #
 #             #
  #           #
   #         #
    # # # # #
```

```
(CYCLE 1. SUBCYCLE 3.)

    # # # # #
   #       # #
  #           #
 #             #
# #             #
 #             #
  #           #
   #       # #
    # # # # #
```

```
(CYCLE 2. SUBCYCLE 1.)

    # # # # #
   #       # #
  #           #
 # #           #
  #             #
 # #           #
  #           #
   #       # #
    # # # # #
```

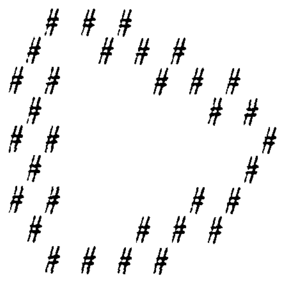
```
(CYCLE 2. SUBCYCLE 2.)

    # # # #
   #     # # #
  #         # #
 # #           #
  #             #
 # #           #
  #           #
   #       # #
    # # # # #
```

```
(CYCLE 2. SUBCYCLE 3.)

    # # # #
   #     # # #
  #         # #
 # #           #
  #             #
 # #           #
  #         # #
   #     # # #
    # # # #
```

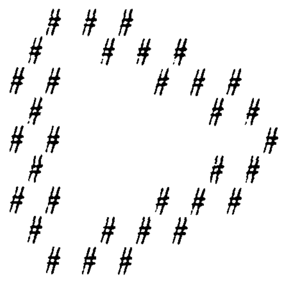
```
(CYCLE 3. SUBCYCLE 1.)

    # # # #
   #     # # #
  # #       # #
   #           #
  # #           #
   #           #
  # #       # #
   #     # # #
    # # # #
```

Figure 49  Srinking of a Hollow Hexagon

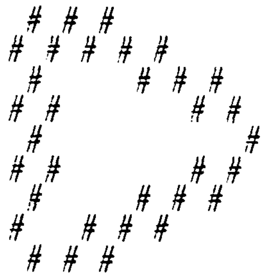
(CYCLE 3. SUBCYCLE 2.)



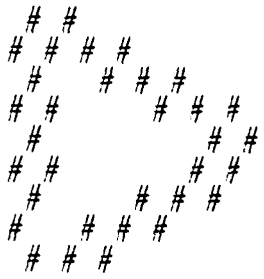
(CYCLE 3. SUBCYCLE 3.)



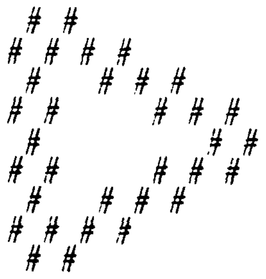
(CYCLE 4. SUBCYCLE 1.)



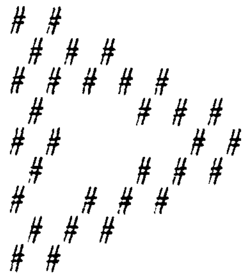
(CYCLE 4. SUBCYCLE 2.)



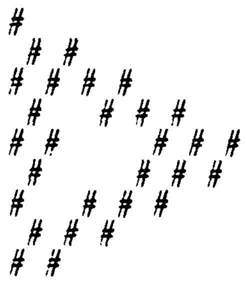
(CYCLE 4. SUBCYCLE 3.)



(CYCLE 5. SUBCYCLE 1.)



(CYCLE 5. SUBCYCLE 2.)



(CYCLE 5. SUBCYCLE 3.)

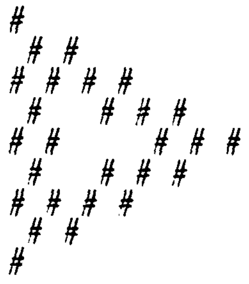


Figure 49 (continued) Shrinking of a Hollow Hexagon

```
(CYCLE 6. SUBCYCLE 1.)

 # #
  # # #
 # # # # #
  #     # # #
 #   # # #
  # # #
 # #
#
```

```
(CYCLE 6. SUBCYCLE 2.)

 #
  # #
 # # # #
  #   # # #
 #   # # #
  # # #
 # #
#
```

```
(CYCLE 6. SUBCYCLE 3.)

#
 # #
# # # #
 #   # # #
# # # #
 # #
#
```

```
(CYCLE 7. SUBCYCLE 1.)

 # #
  # # #
 # # # # #
  # # #
 # #
#
```

```
(CYCLE 7. SUBCYCLE 2.)

 #
  # #
 # # # #
  # # #
 # #
#
```

```
(CYCLE 7. SUBCYCLE 3.)

#
 # #
# # # #
 # #
#
```

```
(CYCLE 8. SUBCYCLE 1.)

 # #
  # # #
 # #
#
```

```
(CYCLE 8. SUBCYCLE 2.)

 #
  # #
 # #
#
```

Figure 49 (continued) Shrinking of a Hollow Hexagon

```
(CYCLE 8.  SUBCYCLE 3.)             (CYCLE 9.  SUBCYCLE 1.)


         #
          # #                                 # #
         #                                   #




(CYCLE 9.  SUBCYCLE 2.)             (CYCLE 9.  SUBCYCLE 3.)


          #
         #                                    #




(CYCLE 10.  SUBCYCLE 1.)            (CYCLE 10.  SUBCYCLE 2.)


          #                                   #




(CYCLE 10.  SUBCYCLE 3.)


          #
```

Figure 49 (continued)  Shrinking of a Hollow Hexagon

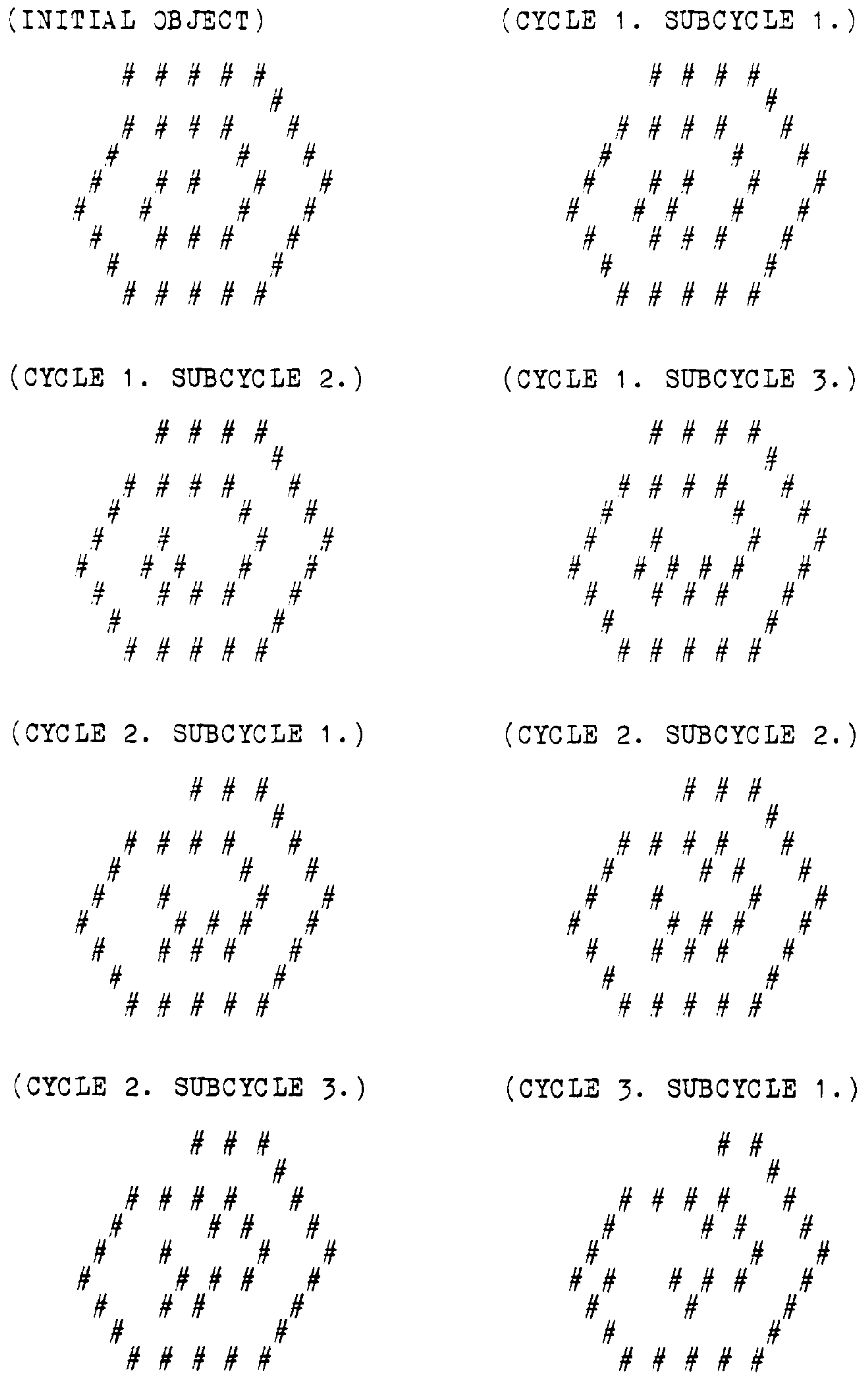
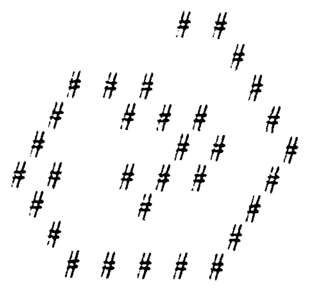
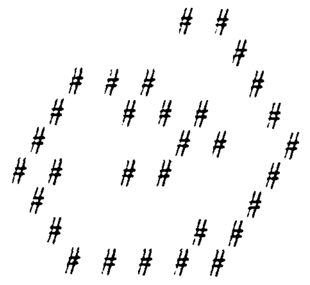


Figure 50 Shrinking of a Hexagonal Spiral

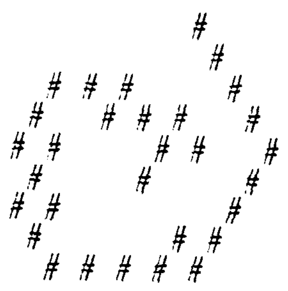
(CYCLE 3. SUBCYCLE 2.)



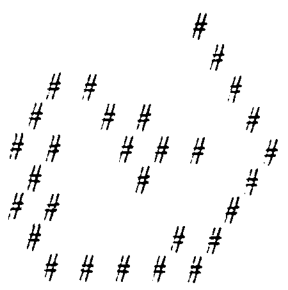
(CYCLE 3. SUBCYCLE 3.)



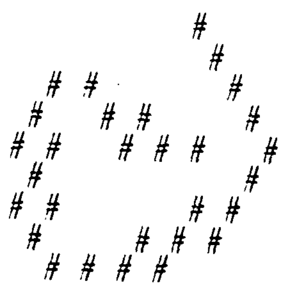
(CYCLE 4. SUBCYCLE 1.)



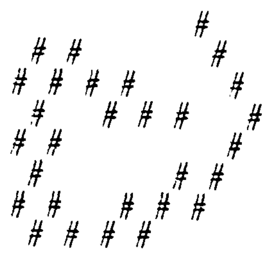
(CYCLE 4. SUBCYCLE 2.)



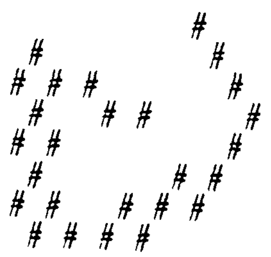
(CYCLE 4. SUBCYCLE 3.)



(CYCLE 5. SUBCYCLE 1.)



(CYCLE 5. SUBCYCLE 2.)



(CYCLE 5. SUBCYCLE 3.)

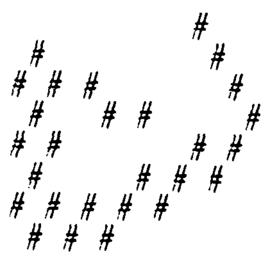
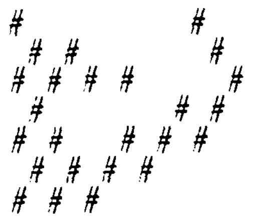
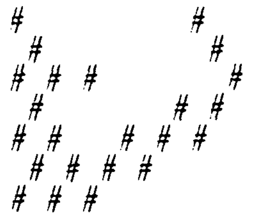


Figure 50 (continued) Shrinking of a Hexagonal Spiral

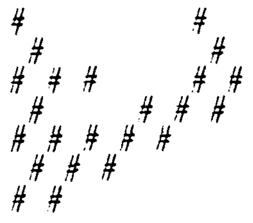
(CYCLE 6. SUBCYCLE 1.)



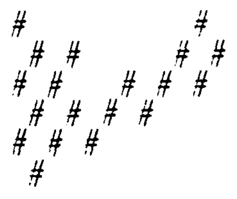
(CYCLE 6. SUBCYCLE 2.)



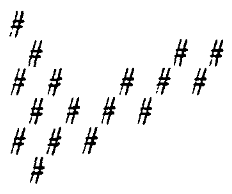
(CYCLE 6. SUBCYCLE 3.)



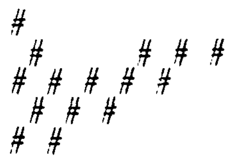
(CYCLE 7. SUBCYCLE 1.)



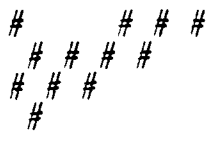
(CYCLE 7. SUBCYCLE 2.)



(CYCLE 7. SUBCYCLE 3.)



(CYCLE 8. SUBCYCLE 1.)



(CYCLE 8. SUBCYCLE 2.)

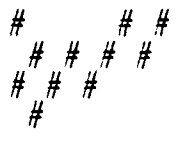


Figure 50 (continued) Shrinking of a Hexagonal Spiral

```
(CYCLE 8. SUBCYCLE 3.)

#   # # #
 # # #
# #
```

```
(CYCLE 9. SUBCYCLE 1.)

   # # #
# # #
 #
```

```
(CYCLE 9. SUBCYCLE 2.)

   # #
# # #
 #
```

```
(CYCLE 9. SUBCYCLE 3.)

 # # #
# #
```

```
(CYCLE 10. SUBCYCLE 1.)

# # #
 #
```

```
(CYCLE 10. SUBCYCLE 2.)

# #
 #
```

```
(CYCLE 10. SUBCYCLE 3.)

# #
```

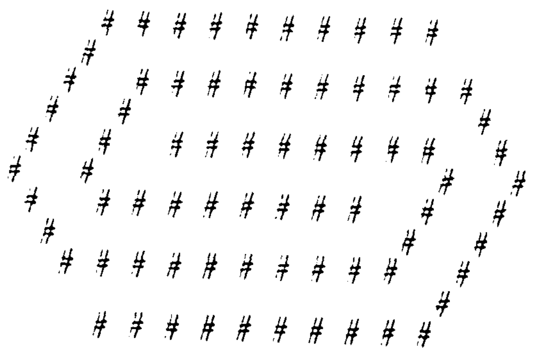
```
(CYCLE 11. SUBCYCLE 1.)

#
```

Figure 50 (continued) Shrinking of a Hexagonal Spiral

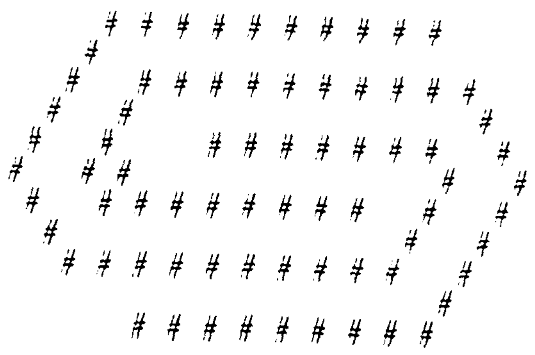
```
(CYCLE 11. SUBCYCLE 2.)        (CYCLE 11. SUBCYCLE 3.)


           #                              #




(CYCLE 12. SUBCYCLE 1.)


           #
```

Figure 50 (continued) Shrinking of a Hexagonal Spiral

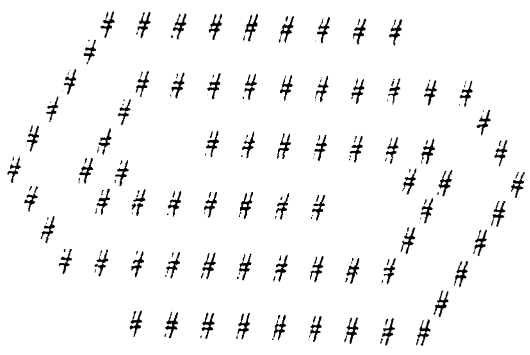
(INITIAL OBJECT)



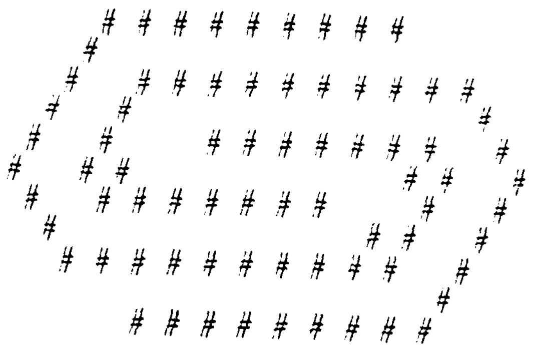
(CYCLE 1. SUBCYCLE 1.)



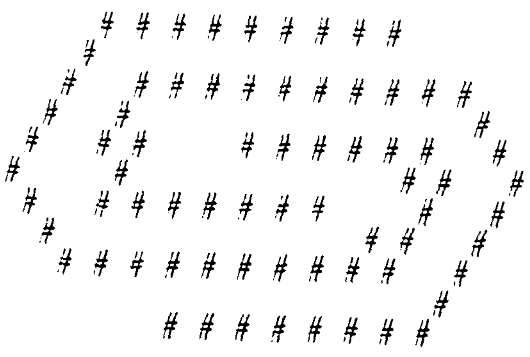
(CYCLE 1. SUBCYCLE 2.)



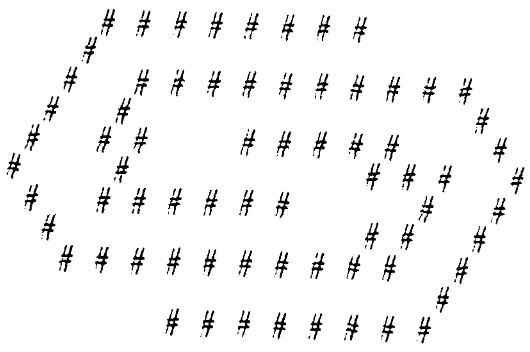
(CYCLE 1. SUBCYCLE 3.)



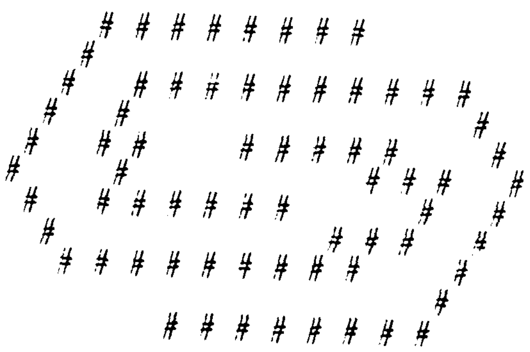
(CYCLE 2. SUBCYCLE 1.)



(CYCLE 2. SUBCYCLE 2.)



(CYCLE 2. SUBCYCLE 3.)



(CYCLE 3. SUBCYCLE 1.)

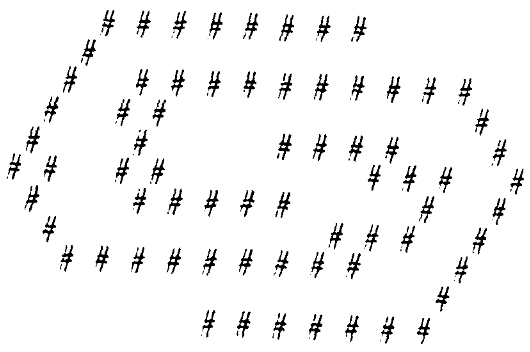


Figure 51 Shrinking of Two Interlocking Spirals

(CYCLE 3. SUBCYCLE 2.)

```
     # # # # # # #
    #
   #   # # # # # # # # # #
  #   # #               # #
 #     #       # # #       #
# #   # #         # # #     #
 #     # # # #       # #   #
  #               # # #   #
   # # # # # # # # #     #
                        #
           # # # # # # #
```

(CYCLE 3. SUBCYCLE 3.)

```
     # # # # # # #
    #
   #   # # # # # # # # # #
  #   # #               # #
 #     #       # # #       #
# #   # #         # # #     #
 #     # # # #       # #   #
  #             # # #     #
   # # # # # # # #       #
                      # #
           # # # # # # #
```

(CYCLE 4. SUBCYCLE 1.)

```
     # # # # # # #
    #
   #   # # # # # # # # # #
  #     #               # #
 # #   # #       # #       #
  #     #         # # #     #
 # #   # # # #       # #   #
  #             # # #     #
   # # # # # # # #       #
                      # #
             # # # # # #
```

(CYCLE 4. SUBCYCLE 2.)

```
     # # # # # #
    #
   #   # # # # # # # # #
  #     #             # # #
 # #   # #       #       # #
  #     #         # #       #
 # #   # # #       # # #   #
  #             # # #     #
   # # # # # # # #       #
                      # #
             # # # # # #
```

(CYCLE 4. SUBCYCLE 3.)

```
     # # # # # #
    #
   #   # # # # # # # # #
  #     #             # # #
 # #   # #       #       # #
  #     #         # #       #
 # #   # # #     # # # #   #
  #           # # #       #
   # # # # # # #       # #
                    # # #
             # # # # #
```

(CYCLE 5. SUBCYCLE 1.)

```
     # # # # # #
    #
   #     # # # # # # # #
  # #   # #           # # #
   #     #               # #
  # #   # #       # #       #
   #     # #     # # # #   #
  # #         # # #       #
   # # # # # # #       # #
                    # # #
               # # # #
```

(CYCLE 5. SUBCYCLE 2.)

```
     # # # # #
    #
   #     # # # # # # #
  # #   # #         # # #
   #     #             # # #
  # #   # #       #       # #
   #     # #     # # #     #
  # #         # # #       #
   # # # # # # #       # #
                    # # #
               # # # #
```

(CYCLE 5. SUBCYCLE 3.)

```
     # # # # #
    #
   #     # # # # # # #
  # #   # #         # # #
   #     #             # # #
  # #   # #       #       # #
   #     #     # # # #     #
  # #         # #       # #
   # # # # # #       # # #
                  # # #
               # # #
```

Figure 51 (continued) Shrinking of Two Interlocking Spirals

(CYCLE 6. SUBCYCLE 1.)

```
  # # # # #
 #
# #   # # # # # # #
 #     #         # # #
# #   # #           # # #
 #     #       #       # #
# #   #     # # # #     #
 #         # #       # #
# # # # # #       # # #
               # # #
              # #
```

(CYCLE 6. SUBCYCLE 2.)

```
  # # # #
 #
# #   # # # # # #
 #     #       # # #
# #   # #         # # #
 #     #             # # #
# #   #     # # #       #
 #         # #       # #
# # # # # #       # # #
               # # #
              # #
```

(CYCLE 6. SUBCYCLE 3.)

```
  # # # #
 #
# #   # # # # # #
 #     #       # # #
# #   # #         # # #
 #     #             # # #
# #         # # #       #
 #       # #       # # #
# # # # # #     # # #
               # #
              #
```

(CYCLE 7. SUBCYCLE 1.)

```
  # # # #
 # #
  #     # # # # #
 # #   # #     # # #
  #     #         # # #
 # #   #             # # #
  #         # # #       #
 # #     # #       # # #
  # # # # #     # # #
               # #
              #
```

(CYCLE 7. SUBCYCLE 2.)

```
  # # #
 # #
  #     # # # #
 # #   # #   # # #
  #     #       # # #
 # #   #           # # #
  #         # #       # #
 # #     # #       # # #
  # # # # #     # # #
               # #
              #
```

(CYCLE 7. SUBCYCLE 3.)

```
  # # #
 # #
  #     # # # #
 # #   # #   # # #
  #     #       # # #
 # #               # # #
  #       # # #       # #
 # #   # # #     # # #
  # # # #       # #
               #
```

(CYCLE 8. SUBCYCLE 1.)

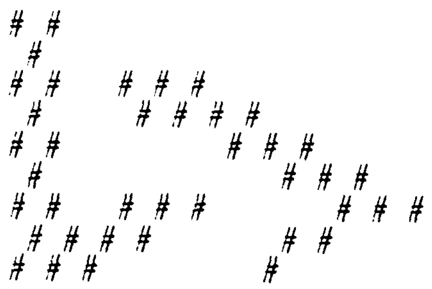
```
  # # #
   #
  # #   # # # #
   #     #   # # #
  # #   #       # # #
   #               # # #
  # #     # # #       # #
   #   # # #     # # #
  # # # #       # #
               #
```

(CYCLE 8. SUBCYCLE 2.)

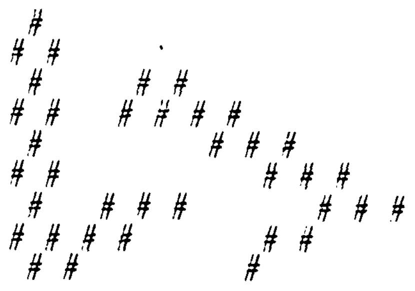
```
  # #
   #
  # #   # # #
   #     # # # #
  # #   #     # # #
   #             # # #
  # #     # #       # # #
   #   # # #     # # #
  # # # #       # #
               #
```

Figure 51 (continued) Shrinking of Two Interlocking Spirals

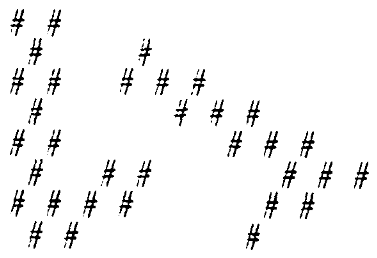
(CYCLE 8. SUBCYCLE 3.)



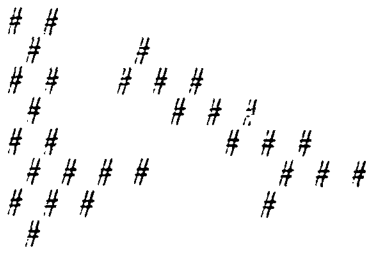
(CYCLE 9. SUBCYCLE 1.)



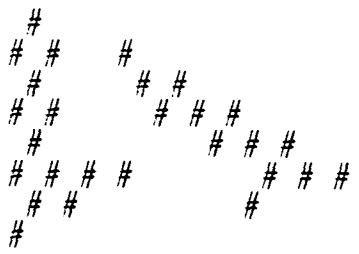
(CYCLE 9. SUBCYCLE 2.)



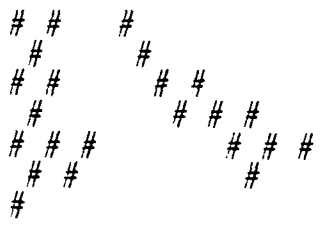
(CYCLE 9. SUBCYCLE 3.)



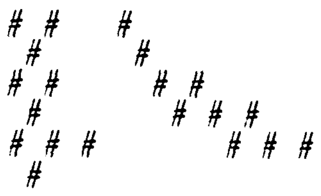
(CYCLE 10. SUBCYCLE 1.)



(CYCLE 10. SUBCYCLE 2.)



(CYCLE 10. SUBCYCLE 3.)



(CYCLE 11. SUBCYCLE 1.)

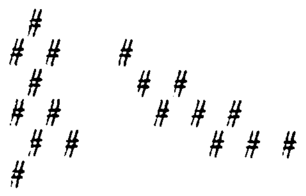


Figure 51 (continued) Shrinking of Two Interlocking Spirals

```
(CYCLE 11. SUBCYCLE 2.)           (CYCLE 11. SUBCYCLE 3.)


        # #   #                           # #   #
         #     #                           #     #
        # #     # #                       # #     # #
         # #     # # #                     #       # # #
        #


(CYCLE 12. SUBCYCLE 1.)           (CYCLE 12. SUBCYCLE 2.)


          #
         # #   #                           # #   #
          #     # #                         #     #
         #       # # #                     #       # #


(CYCLE 12. SUBCYCLE 3.)           (CYCLE 13. SUBCYCLE 1.)


         # #   #                               #
          #     #                             #     #
                 # #                                 # #


(CYCLE 13. SUBCYCLE 2.)           (CYCLE 13. SUBCYCLE 3.)


          #     #                             #     #
                 #
```

Figure 51 (continued) Shrinking of Two Interlocking Spirals

```
(CYCLE 14. SUBCYCLE 1.)          (CYCLE 14. SUBCYCLE 2.)



         #     #                          #     #




(CYCLE 14. SUBCYCLE 3.)



         #     #
```

Figure 51 (continued) Shrinking of Two Interlocking Spirals

# CHAPTER 7

# SUMMARY AND SUGGESTIONS FOR FUTURE WORK

## 7.1 Summary

In this dissertation, I have presented a paradigm for parallel processing of computer generated images. In addition, I have presented several examples of algorithms which could be embedded in the paradigm. The system operates as follows: A rectangular array of pixels (from a television camera, say) is embedded in a hexagonal array of processors (cells). Each cell then forms its information into a message and broadcasts the message outward to the rest of the cells in the array. The paths taken by the messages are governed by a set of production rules such that the message's "wavefront" expands in the shape of a hexagon centered on the originating cell. The system is synchronous, so the messages move in discrete steps and all at the same time. Eventually, if the messages are allowed to propagate for a long enough time, each message passes through every cell in the array. A further property of the message propagation is that at the point when there are twelve copies of each message (i.e., after two time units), each message is said to have one of twelve "clock directions" (since the directions correspond to those of the numerals on a clock face). These clock directions are inherited by the messages' offspring so that when a message enters a cell, the cell can tell from which direction the message came.

Since the messages carry information from their cells of origin, the cells they pass through can combine information from many parts of the scene and make interesting deductions about the scene. For example, with very simple algorithms in which the cells examine resident messages in pairs, the system can find edges, corners, and axes of symmetry. The cells are not limited to being passive observers of the message which pass through them. They (the cells) can be programmed to generate new messages and destroy old messages if conditions (under program control) warrant it.

With more complicated algorithms, the system can derive such things as texture information and object areas from the scene. Various transformations of the scene are also possible. Some examples of these are fast fourier transforms, Voronoi tessellations, prairie fire skeletons, and septrees.

The system is recursive in the sense that information derived in the cells can be re-formed into messages and broadcast in parallel from the cells. This process is called "layering".

Another important part of this dissertation is the parallel shrinking algorithm for hexagonal arrays presented in Chapter 6. The algorithm transforms any binary scene (with one class of exceptions) into a set of isolated points, each of which corresponds to a connected object in the original scene. The class of exceptions referred to above contains those scenes in which one object completely surrounds another. This algorithm shrinks objects with any number of holes and is proven never to merge two unconnected objects nor disconnect any connected object. A proof is also presented that all objects (with the exception noted above) shrink to isolated points. By removing and remembering the locations of isolated points as they occur, completely surrounded objects could also be dealt with.

## 7.2 Suggestions for Future Work

There are several directions in which my work could be extended.

One problem with HEXVIS is its method of programmability. At times, the results of directing the system's behavior are somewhat counter-intuitive. An example of this is the programming of Voronoi Tessellations by the collision method (see Chapter 5). I did not expect the gaps in the lines of the tessellation. It seems that an investigation into a special purpose high level language for HEXVIS could be quite fruitful.

An extension to the HEXVIS paradigm which might greatly increase its power is to remove the stipulation that the cells operate synchronously. It is not clear how this change would be accomplished. Possibly allowing messages to propagate at different rates would be a part of it. Allowing cells to in some sense "wake up" when a particular condition occurs might also be required.

Other propagation schemes are certainly possible (e.g., the dodecagonal scheme of Chapter 2). I have mentioned that a wavefront which is close to a circle is desirable. Another possibility is to increase the resolution of the system by increasing the number of directions available. The logical extreme of this idea is for a cell which receives a message to be able to pinpoint the message's origin exactly. In many situations, this is a highly desirable goal. (This can be accomplished with HEXVIS as it is now implemented. The cell simply appends its coordinates to the messages it generates.)

Of course, the development of parallel algorithms to fit into the message passing paradigm of HEXVIS is quite important. The question which arises here is: How much of the work of a computer vision system could HEXVIS be reasonably expected to assume? It is not obvious how high level processes (such as knowledge representation or recognizing real world objects in a scene) could fit into the HEXVIS mold. It is conceivable that HEXVIS could be a vehicle for an ACTOR type system [Hewitt, 1976] which is designed for high level processes. At the other end of the spectrum, it seems clear that HEXVIS is quite useful for lower level processes such as have been illustrated in this dissertation. In this capacity, HEXVIS could certainly form a part of a overall computer vision system.

Along a more theoretical vein, the shrinking algorithm presented in Chapter 6 generates some very exciting possibilities. First of all, the question of how many cycles an object requires to shrink to a point is open. An upper limit can be placed on this time for some objects, i.e., those without connectors and which do not develop connectors during the course of their shrinking. That limit is 2N/3, where N is the length of one side of the equilateral triangle forming the object's frontier. However, I would conjecture that this limit can be lowered to N/3+1. Other questions can be posed, for instance: What class of objects do develop connectors? Can the location of the final isolated point be predicted more precisely than simply inside the object's frontier? Is the cellular space represented computationally complete? (Probably not, since no information can leave the frontier of an object. Perhaps the frontier should have been called the "event horizon".) For a given frontier, what does the object which takes the most time to shrink look like (conjecture: a spiral)? In fact, the field of parallel processing of hexagonal binary arrays in general (of which shrinking is only one part) is quite fascinating.

Finally, the question of implementation of HEXVIS in hardware poses some interesting problems, the main one being how simple a machine is required to perform the functions of one of the cells in the array. Some issues of global communication have not been extensively considered here. It is clear that some tasks are better handled by a system which has a processor distinguished by being global to the others (e.g., loading the initial updating function into HEXVIS). This processor would not necessarily be governed by the message passing process in its communications with the other cells.

## REFERENCES

[Ahuja] Narendra Ahuja, Larry S. Davis, David L. Milgram, and Azriel Rosenfeld, "Piecewise Approximation of Pictures Using Maximal Neighborhoods", IEEE Transactions on Computers, vol. C-27, no. 4, 1978.

[Ahuja] Narendra Ahuja, Mosaic Models for Image Analysis and Synthesis, University of Maryland Ph.D. Thesis, 1979.

[Ahuja] Narendra Ahuja, "Dot Pattern Processing Using Voronoi Polygons as Neighborhoods", unpublished paper, 1980.

[Bajcsy] Ruzena Bajcsy, "Computer Description of Textured Surfaces", Proceedings of the Third International Conference on Artificial Intelligence, 1973.

[Banks] Edwin R. Banks, "Information Processing and Transmission in Cellular Automata", MIT Project MAC TR-81, 1971.

[Blum] Harry Blum, "A Transformation for Extracting New Descriptors of Shape", in Models for the Perception of Speech and Visual Form, Walthen Dunn, ed., MIT Press, 1967.

[Blum] Harry Blum and Roger N. Nagel, "Shape Description Using Weighted Symmetric Axis Features", Pattern Recognition, vol. 10, pp. 167-180, 1978.

[Brigham] E. O. Brigham and R. E. Morrow, "The Fast Fourier Transform", IEEE Spectrum, December, 1967.

[Brodatz] Phil Brodatz, Textures, Dover, New York, 1966.

[Burks] Arthur W. Burks, Essays on Cellular Automata, University of Illinois Press, Urbana, Illinois, 1970.

[Burt] Peter J. Burt, "Tree and Pyramid Structures for Coding Hexagonally Sampled Binary Images", University of Maryland Computer Science Center TR-814, 1979.

[Codd] E. F. Codd, Cellular Automata, Academic Press, New York, 1968.

[Conway] John Conway, "Mathematical Games", (edited by Martin Gardner), Scientific American, October, 1970.

[Davis] Larry S. Davis, M. Clearman, and J. K. Aggarwal, "A Comparative Texture Classification Study Based on Generalized Cooccurrence Matrices", Proceedings of the

Eighteenth IEEE Decision and Control Conference, December, 1979.

[Davis] Randall Davis and Jonathan King, "An Overview of Production Systems", Stanford Artificial Intelligence Laboratory AIM-271, 1975.

[Deutsch] E. S. Deutsch, "On Parallel Operations on Hexagonal Arrays", IEEE Transactions on Computers, vol. C-19, pp. 982-983, 1970.

[Deutsch] E. S. Deutsch, "Thinning Algorithms on Rectangular, Hexagonal, and Triangular Arrays", Communications of the ACM, vol. 15, no. 9, pp. 827-837.

[Duda and Hart] R. O. Duda and P. E. Hart, <u>Pattern Classification and Scene Analysis</u>, Wiley, New York, 1973.

[Dyer] Charles H. Dyer, "Cellular Pyramids for Image Analysis, 2", University of Maryland Computer Science Center TR-596, 1977.

[Dyer] Charles H. Dyer, "Memory-Augmented Cellular Automata for Image Analysis", University of Maryland Computer Science Center TR-710, 1978.

[Dyer] Charles H. Dyer, Tsai-Hong Hong, and Azriel Rosenfeld, "Texture Classification Using Gray Level Cooccurrence Based on Edge Maxima", University of Maryland Computer Science Center TR-738, 1979.

[Farley] Arthur M. Farley and Andrzej Proskurowski, "Gossiping in Grid Graphs", University of Oregon Computer Science Department CS-TR-78-12, 1979.

[Gardener] Martin Gardener, "Mathematical Games", Scientific American, January, 1977, pp. 110-121.

[Golay] Marcell J. E. Golay, "Hexagonal Parallel Pattern Transformations", IEEE Transactions on Computers, vol. C-18, No. 8, pp. 733-740, 1969.

[Gordella] L. Gordella, M. J. B Duff, and S. Levialdi, "Thresholding: A Challenge for Parallel Processing", Computer Graphics and Image Processing, vol. 6, pp. 207-220, 1977.

[Halstead] Robert H. Halstead, Jr., "Multiple-Processor Implementations of Message-Passing Systems", MIT Laboratory for Computer Science TR-198, 1978.

[Haralick] Robert M. Haralick, "Statistical and Structural Approaches to Texture", Proceedings of the Fourth International Joint Conference on Pattern Recognition, 1979.

[Hewitt] Carl Hewitt, "Viewing Control Structures as Patterns of Passing Messages", MIT Artificial Intelligence Laboratory AI Memo 410, 1976.

[Hewitt] Carl Hewitt and Henry Baker, "Actors and Continuous Functionals", MIT Artificial Intelligence Laboratory AI Memo 436, 1977.

[Horn] Berthold K. P. Horn, "On Lightness", MIT Artificial Intelligence Laboratory AI Memo 295, 1973.

[Hueckel] Manfred H. Hueckel, "An Operator Which Locates Edges in Digitized Pictures", Journal of the ACM, vol. 18, no. 1, 1971.

[Ingram] Marylou Ingram and Kendall Preston, Jr., "Automatic Analysis of Blood Cells", Scientific American, November, 1970.

[Julesz] Bela Julesz, "Texture and Visual Perception", Scientific American, February, 1965.

[Klette] Reinhard Klette, "Parallel Operations on Binary Images", University of Maryland Computer Science Center TR-822, 1979.

[Kohler] Wolfgang Kohler, Gestalt Psychology, The New American Library, New York, 1959.

[Levialdi] S. Levialdi, "On Shrinking Binary Picture Patterns", Communications of the ACM, vol. 15, No. 1, pp. 7-10.

[Levitt] K. N. Levitt and W. H. Kautz, "Cellular Arrays for the Solution of Graph Problems", Communications of the ACM, Vol. 15, no. 9, pp. 789-801, 1972.

[Lindsay and Norman] Peter H. Lindsay and Donald A. Norman, Human Information Processing, Academic Press, New York, 1972.

[Marr] D. Marr, "Representing Visual Information", in A. Hanson and E. Riseman (eds.) Computer Vision Systems, Academic Press, New York, 1978.

[Pavlidis] Theodosios Pavlidis, "A Review of Algorithms for Shape Analysis", Computer Graphics and Image Processing, vol. 7, pp. 243-258, 1978.

[Pratt] William K. Pratt, Digital Image Processing, John Wiley and Sons, Inc., New York, 1978.

[Preston] Kendall Preston, Jr., "Feature Extraction by Golay Hexagonal Pattern Transforms", IEEE Transactions on

Computers, vol. C-20, no. 9, pp. 1007-1014, 1971.

[Preston] Kendall Preston, Jr., "Interactive Image Processor Speeds Pattern Recognition by Computer", Electronics, October 23, 1972, pp. 89-98.

[Preparata] F. P. Preparata, "An Optimal Real-Time Algorithm for Planar Convex Hulls", Communications of the ACM, vol. 22, no. 7, 1979.

[Rao] C. V. Kameswara Rao, B. Prasada, and K. R. Sarma, "A Parallel Shrinking Algorithm for Binary Patterns", Computer Graphics and Image Processing, vol. 5, pp. 265-270, 1976.

[Riseman] Edward M. Riseman and Michael A. Arbib, "Computational Techniques in the Visual Segmentation of Static Scenes", Computer Graphics and Image Processing, vol. 6, pp. 221-276, 1977.

[Rosenfeld] Azriel Rosenfeld, "Connectivity in Digital Pictures", Journal of the ACM, vol. 17, no. 1, pp. 146-160, 1970.

[Rosenfeld] Azriel Rosenfeld and Avinash C. Kak, Digital Picture Processing, Academic Press, New York, 1976.

[Schachter] Bruce Schachter and Narendra Ahuja, "Random Pattern Generation Processes", Computer Graphics and Image Processing, vol. 10, pp. 95-114, 1979.

[Singleton] Richard C. Singleton, "On Computing the Fast Fourier Transform", Communications of the ACM, vol. 10, no. 10, 1967.

[Siromoney] Gift Siromoney and Rani Siromoney, "Hexagonal Arrays and Rectangular Blocks", Computer Graphics and Image Processing, vol. 5, pp. 353-381, 1976.

[Waltz] D. L. Waltz, "A Parallel Model for Low-Level Vision", in A. Hanson and E. Riseman (eds.) Computer Vision Systems, Academic Press, New York, 1978.

[Winston] Patrick Henry Winston, Artificial Intelligence, Addison-Wesley, Reading, Mass., 1977.

[Yonezawa] Akinori Yonezawa, "Specification and Verification Techniques for Parallel Programs Based on Message Passing Semantics", MIT Laboratory for Computer Science TR-191, 1977.

[Zucker] Steven W. Zucker, "Toward a Model of Texture", Computer Graphics and Image Processing, vol. 5, pp. 190-202, 1976.

# APPENDIX

## A SHORT HISTORY OF CELLULAR AUTOMATA

John von Neumann established the study of cellular automata in the early forties during his investigation into self-reproducing machines [Burks, 1970]. In order to investigate this subject he needed a rigorously defined description of both the primitive building units and their environment and how they would interact. Von Neumann's first idea was to imagine a sea with an infinite number of parts floating randomly on the surface. The basic parts he had in mind are the following: "and", "or", and "not" switches, a one time unit delay element, a manipulating element, a cutting element, a fusing element, a rigid element, and a sensing element. An automaton composed of these elements is called a "kinematic automaton". A kinematic automaton floats on the surface of the sea, acquiring parts as it needs them to construct whatever its program directs it to.

A Turing machine can be simulated with a kinematic automaton using the switches and delays as the logical elements. For the tape, a chain of rigid elements can be formed with extra rigid elements at some of the intersection points (see figure 52).

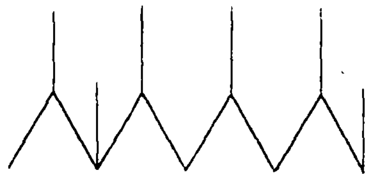


Figure 52 A Kinematic Chain

An intersection with an extra rigid element represents a "1" while one without an extra rigid element represents a "0". Thus, the chain in figure 52 represents the binary number 011101011. The kinematic automaton, through use of its sensing, manipulating, fusing, and cutting elements can read, alter, and extend the tape taking new rigid rods from the sea as needed. Since a machine M can be described totally by lists of its parts and how they are connected, a description of M, D(M), can be stored in binary form on a tape. Imagine now a kinematic automaton $M_{sr}$ which will construct a machine from a taped description, then copy its own tape and attach it to the new automaton. If $M_{sr}$ is given a tape with its own description, it will reproduce itself complete with a self-describing tape.

The problem with the kinematic model is that it is not rigorous enough. The bulk of the material which describes the kinematic automata system (and hence requires formalization) is concerned with the motion of the automaton through the sea, the mechanisms of sensor-manipulator coordination, etc. -- not really the most salient aspects of self-reproduction.

The idea of cellular automata was suggested to von Neumann by S. M. Ulam [Burks, 1970]. It lends itself fairly easily to rigorous formalization while retaining features important to the logic (although not necessarily the physiology) of self-reproduction.

In describing a cellular automata system, it is first necessary to describe the "cellular space". Intuitively, the cellular space is an array of finite state machines called "cells" (to be described shortly). Formally, it consists of a (usually infinite) set of cells and a neighborhood relation on that set. The neighborhood relation gives, for each cell C, a set of cells which are C's neighbors. Notice that this relation could vary from cell to cell although usually, it is quite regular.

The cells themselves are described by giving a finite list of states for each cell, a distinguished state (called the "quiescent" or "blank" state), and a rule which gives the state of the cell at time T + 1 as a function of the states of its neighbors and itself at time T. The quiescent state has the property that if a cell C and its neighbors are all in the quiescent state at time T, then C will remain in the quiescent state at time T + 1.

A particular cellular automaton, then, is described by giving a finite list of cells along with each cell's initial state. The other cells in the system are assumed to be in the quiescent state.

In von Neumann's cellular automata system there are twenty-nine states. He was able to show that a cellular automaton which was computationally equivalent to a Turing machine could be embedded in his system. Thus, the cellular space is "computation universal". Furthermore, von Neumann showed that the same automaton could reproduce any automaton described by a tape, copy its own tape, and attach the tape to the new automaton. Thus, the cellular space is "construction universal". In particular, the automaton is able to reproduce itself.

## VITA

George Daniel Hadden was born in Jackson, Michigan on March 1, 1950. He enrolled as an honors student in Purdue University in West Lafayette, Indiana in 1968. As an undergraduate, he worked as a cooperative engineering student at Burroughs Corporation in Plymouth, Michigan and at General American Research Division in Niles, Illinois. He received the degree of Bachelor Science in Electrical Engineering from Purdue in 1973.

In 1974, he enrolled in the Department of Electrical Engineering at the University of Illinois. At this time, he became a research assistant at the Coordinated Science Laboratory. In 1977, he completed the degree of Master of Science with a thesis entitled "NETEDI: An Augmented Transition Network Editor".

He continued working at the Coordinated Science Laboratory until finishing his doctoral research in 1980. He is the author of several scientific papers in the areas of natural language processing and computer vision and a member of IEEE, ACM, and HKN.

LMED

8