OPTIMAL REGION COLORING BY DOING IMAGE CHUNKING

by

Paul Andrew Wagner

B.S., Physics
Georgetown University
(1985)

SUBMITTED TO THE DEPARTMENT OF
BRAIN AND COGNITIVE SCIENCES
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF

MASTER OF SCIENCE
IN BRAIN AND COGNITIVE SCIENCES

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
February 1990

Signature of author _____
Department of Brain and Cognitive Sciences
February 2, 1990

Certified by _____
Ellen Hildreth
Assistant Professor of Vision Science
Thesis Supervisor

Accepted by _____
William G. Quinn, Jr.
Chairman, Departmental Graduate Committee

Optimal Region Coloring by Doing Image Chunking

by

Paul A. Wagner

Submitted on February 2, 1990 in partial fulfillment
of the requirements for the degree of
Master of Science
in Brain and Cognitive Sciences

Abstract

Region coloring is one of a set of elemental operations that can be assembled into visual routines for the perception of spatial properties and relations [Ullman 83]. This operation is investigated in light of some basic biological constraints, and some fundamental computational limitations are established. A set of coloring algorithms based on the idea of region chunking are designed, implemented and evaluated. Applying the principles learned yields an optimal algorithm, using square region chunks, that requires a minimal number of time steps for coloring completion. The flexible, optimal scheme is then compacted substantially in terms of space requirements, and finally, is extended to handle cases of broken and moving contours.

Thesis Supervisor: Professor Ellen Hildreth
Department of Brain and Cognitive Sciences

Acknowledgements

I thank Ellen Hildreth for stepping in to be my insightful thesis supervisor; Shimon Ullman for getting me into this and for his continuing guidance; Whitman Richards for his thoughtful advice; and Jan Ellertsen for her tireless support.

I am grateful to Jim Ballas for introducing me to Vision and AI; to Jim Mahoney and Eric Saund for helpful discussions regarding this research; to Mike Villalba, Waqar Qureshi and Walter Gillett for sharing the graduate experience with me; and to Jonathan Koch, Ron Chaney, Peter Graham and Jin Oh for their good humor.

I salute Mike Baird, Christian Fortunel, Don Christian and all my friends at the FMC Corporate Technology Center for their generous attention.

Finally, I thank my family for their love, friendship and typing.

Contents

1 Introduction to the Coloring Operation

The purpose of vision is to tell us what is where by seeing, according to Marr [Marr 82]. Our interaction with the world is strongly guided by visual judgements about the properties and spatial relations of objects. Many spatial reasoning tasks require the explicit representation of regions derived from the image which may be treated as units during the analysis. The computation of such a representation can be achieved through a process called "bounded activation" or "region coloring". Through an iterative process, connected components of a retinotopic map are labeled with a unique identifier, and then the set of these components is used as a part of an even more complex descriptive entity.

Recently, Ullman [Ullman 83] has proposed that the computation of a variety of complex shape properties and spatial relations can be achieved by assembling a set of elemental operations into specialized visual routines. Region coloring is one example of such elemental operations; others include shifting of processing focus, indexing, marking, and boundary tracing. Ullman's visual routine paradigm provides a framework for exploring the nature of basic operations, such as region coloring, and their role in tasks such as spatial reasoning and recognition.

1.1 The Problem and the Approach

The primary goal of this thesis is to develop efficient and flexible algorithms for performing the operation of region coloring in light of its computational role. In doing so, we observe a wide range of biological, psychophysical and computational constraints which impact the design of the algorithm. Most significant are those which assume simple processors, limited and largely local connections, and short messages between processors. In addition, human performance at coloring tasks sets an impressive standard, with speed appearing to be invariant with respect to figure scale, position, and orientation. Finally, we discover some fundamental computational properties pertaining to the broad class of models being considered, which limit the best case performance one could expect from these. With these constraints in mind, we seek algorithms that are fast, require limited hardware, and yield sensible solutions for a wide variety of problems.

The contributions of this thesis are as follows. First, we establish the computational function and limitations of the coloring operation, and show why the representation referred to as image chunking is well-chosen. Second, we develop an algorithm that embodies this representation, which we show is optimal in terms of number of steps. Third, we show that

this algorithm can be compacted considerably in terms of its spatial requirements, while still remaining optimal to within a constant factor. Fourth, we extend these algorithms to handle broken as well as shifting contours. In each case, we demonstrate the performance of the algorithm for a variety of input data, such as noisy edge data from stereo-based range images and sequences of edges in motion. Finally, we motivate the computational work with results from biological and psychophysical studies of natural vision systems.

1.2 Outline of Thesis

The thesis flows through three levels of investigation (see [Marr 82], p. 24). First, Chapter 1 introduces the problem and outlines the general approach taken. Chapter 2 develops the theory of coloring at an abstract computational level. It establishes the role of coloring in vision, and demonstrates its appropriateness and adequacy as part of this information processing task. Coloring is portrayed as a mapping from one kind of information to another, and important properties that constrain this mapping are shown.

Next, Chapter 3 takes up the issue of representation. It presents formally and motivates the choice of representation for the input and the output (image chunking), and determines some important properties associated with the transformation from one to the other. Chapter 4 is a background section which deepens the scope of the investigation. It proposes some important constraints suggested by psychophysical results. The chapter continues the discussion at the same level of understanding as in Chapter 3, for it reviews earlier approaches that employed image chunking, and then describes other representations and algorithms.

Chapters 5 through 7 complement the representation question by introducing several kinds of algorithms to do the transformation. In Chapter 5, we examine a class of algorithms that embody the image chunking representation and analyze empirically the interaction between scale and propagation. In Chapter 6, we derive the optimal coloring algorithm using parallel square modules, and prove its correctness and optimality. We also place the optimal algorithm in the context of an asynchronous mode and discuss stability. Then in Chapter 7 we show that the algorithm can be compacted substantially in terms of space and still be optimal. In addition, each of these chapters concludes with an analytic description of the related implementation and its results.

Further still, Chapter 8 presents compact optimal coloring inside broken contours and Chapter 9 presents the same inside moving contours. Each by itself represents a fairly complete unit of thought from a methodological point of view, by presenting theoretical,

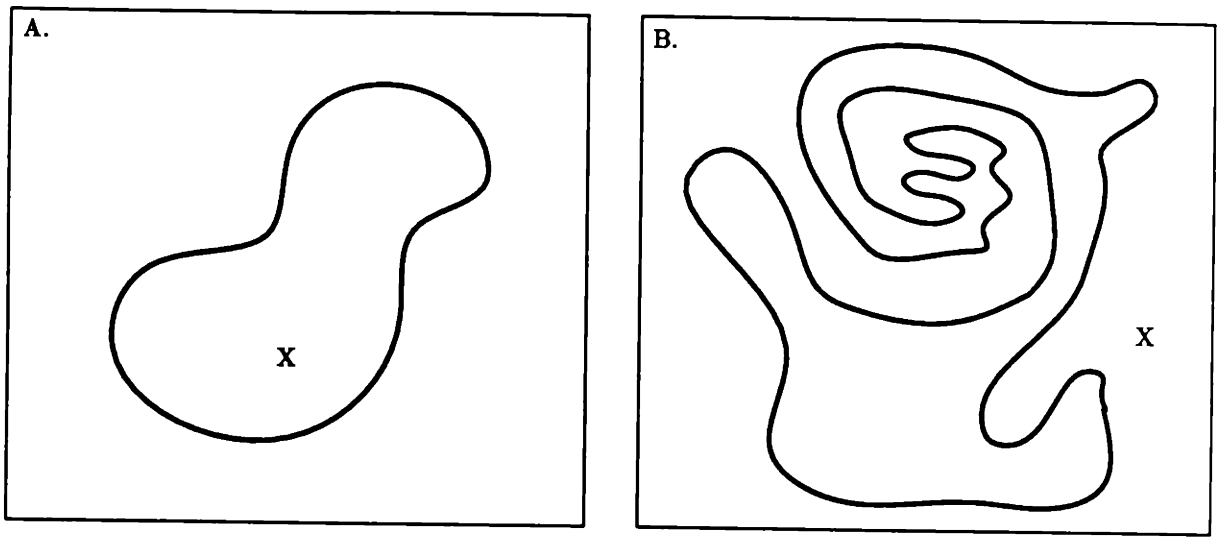


Figure 1: The "XIO Problem": Is X inside any closed figure, or is it outside? A. X is inside. B. X is outside.

algorithmic and implementational descriptions of its respective topic. Finally, Chapter 10 summarizes the logical flow and emphasizes the important points of the thesis.

2 Computational Theory of Coloring

2.1 Theoretical Requirements of Spatial Analysis

An intelligent system seeks to make judgements about the structure and the nature of the world around it. To do so by seeing, its vision system must infer abstract shape properties and spatial relations from the scene. For example, in Figures 1A or B, one might be required to decide whether the "X" is inside or outside of the closed contour. To recognize the snowman in Figure 2A, one needs to notice that the three oblong circles are positioned one above the other. To read the beginning of Beethoven's Fifth Symphony in Figure 2B, one must solve for both relative position and inside/outside relations. Finally, recognition of a face in Figure 2C and not in 2D again involves both position and inside/outside computations. Region coloring can be used to represent subsets of the image as a single unit, prior to position estimation. It can also be used to solve inside/outside relations, as will be shown below.

Solving tasks requiring spatial analysis often seems simple and immediate, but the

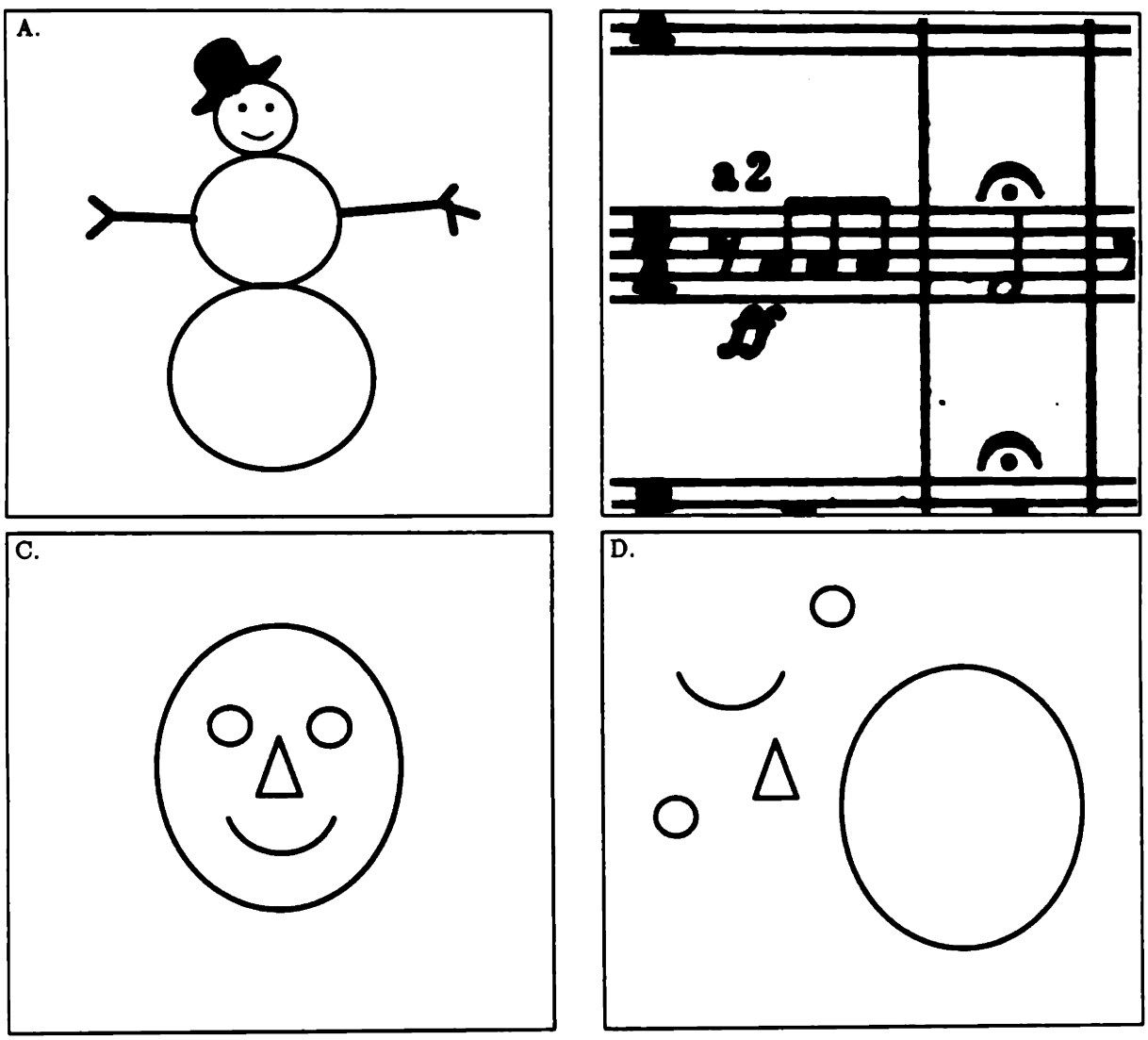


Figure 2: Visual scenes posing spatial analysis tasks. A. A snowman. B. The first few measures of Beethoven's Fifth Symphony. C. A face. D. A jumbled collection of parts of a face – not a face.

underlying computations are found to be quite complex. Fortunately, they can often be decomposed into a set of simpler subtasks. Ullman [Ullman 83] proposed a framework for computing abstract shape properties and spatial relations, called "visual routines". Initially, the vision system has available only early representations of the world, which are created in a bottom-up fashion, retinotopically. Ullman observed that there is a need to transform these base representations into incremental representations that meet specific computational goals. He proposed doing this by using visual routines, which are composed of sequences of elemental operations. Like computer programs, visual routines could draw from a fixed set of operations, and nonetheless solve an unbounded variety of tasks. Some operations which might be useful include:

- shift of processing focus
- indexing
- marking
- boundary tracing
- *bounded activation*

This framework allows a vision system to satisfy three general requirements: (i) *abstractness* — the capacity to establish abstract spatial properties and relations, (2) *open-endedness* — the capacity to establish a large and unbounded variety of properties and relations, and (iii) *complexity* — the capacity to deal efficiently with the complexity involved [Ullman 83].

Coloring a region means uniquely labeling every element in the set of connected region elements. The "Inside/Outside problem" [Shafrir 85] provides a useful application of coloring to investigate. One is given a binary image, consisting of free-space and contour pixels, with one pixel that is initially labeled (call this pixel the "X"). The task is to determine whether or not the X is inside a closed contour. There are many methods to solve this problem. One simple procedure is the *ray-intersection method.* A ray is drawn from the point in question (i.e., X) to some peripheral area in the visual field (the periphery is said to consist of "infinity points"). Then the number of intersections the ray makes with any contour is counted. If this number is odd, X is inside the closed contour. If the number is even, X is outside the closed contour. Figures 3A and B illustrate this method. Unfortunately, the ray-intersection method encounters difficulty in a number of simple cases. Some typical problems are illustrated in Figures 4A through D [Ullman 83]. Figure 4A and B

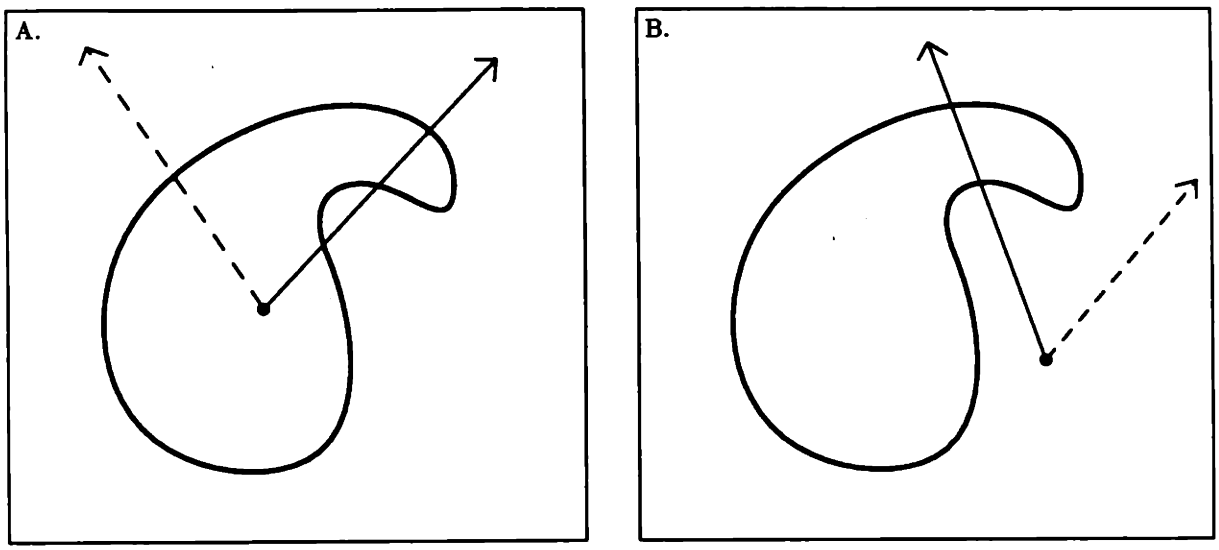


Figure 3: Two examples of the ray intersection method at work. A. When the starting point is inside a closed contour, under simple conditions, a ray drawn from that point to the periphery crosses the contour an odd number of times. B. When the starting point is outside, one can expect a even number of intersections (possibly 0).

depict situations where the number of contours crossed does not correspond correctly to the inside/outside status of the point in question. In Figure 4C, one would have difficulty using ray-intersection in order to locate a point that is inside all the closed curves. Finally, in Figure 4D, one would have trouble using the same method to determine whether the two points are within the same closed curve.

An alternative method that is free of many of the limitations of the ray-intersection method is that involving region coloring. Starting at X, the X and the adjoining area is "colored", and this activation is spread in all directions until a contour boundary or an infinity point is reached. After all possible activation has occured, some simple test is performed, according to the task at hand, to complete the determination. For the case of the "Inside/Outside problem", the test would be to check whether any of the infinity points had been activated. If so, X is found to be on the outside of the closed curve. If not, X is found to be on the inside. Figures 5A and B illustrate the application of this procedure for cases when X is inside and outside, respectively. In the pages that follow, the inside/outside task will provide the context within which we employ the region coloring operation.

Coloring has other more direct uses as well. Coloring can be utilized to *represent an*

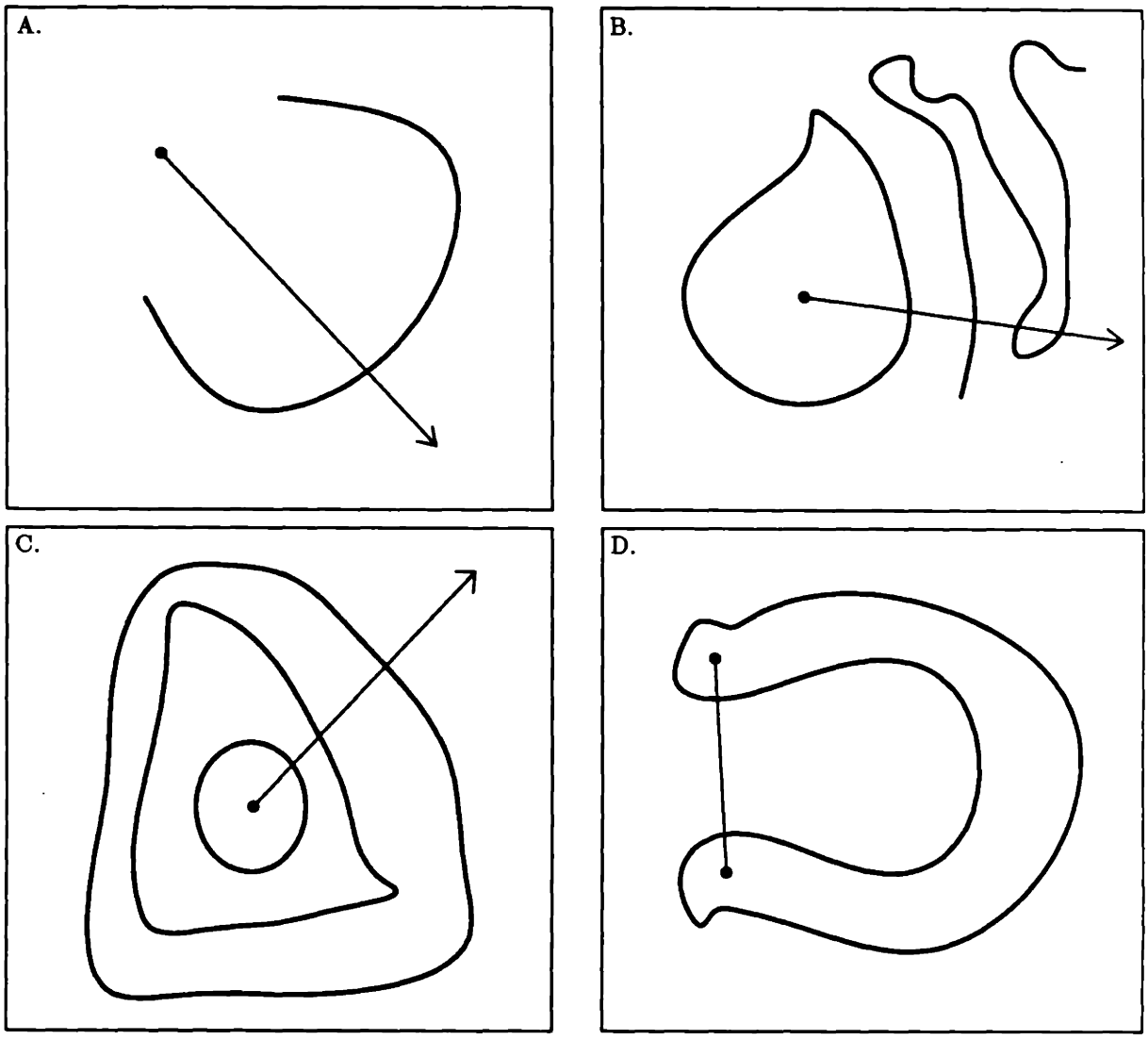


Figure 4: Four exemplary difficulties with the ray intersection method [Ullman 83]. A. & B. One or more extraneous unclosed contours can confound the intersection-counting method. C. A slightly more complicated context, concentric closed contours, poses additional obstacles. D. A slightly more complicated task, the determination of co-occupancy, is not amenable to this method.

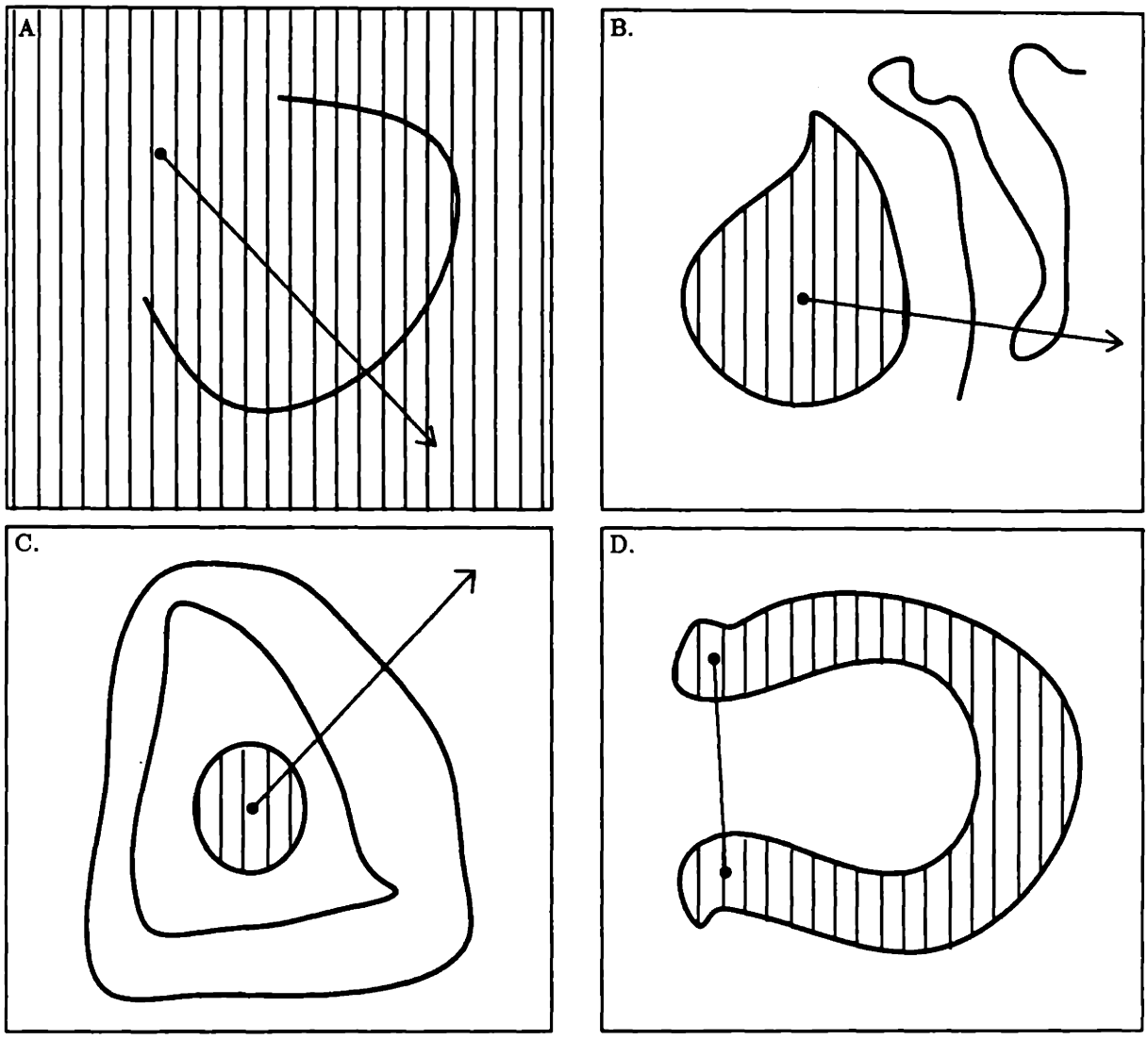


Figure 5: Four examples of the coloring method at work, where the ray intersection method failed. A. - C. If and only if color reaches the periphery, the point in question is outside. D. If and only if color from one point reaches the other, the points occupy the same region.

object as just one or a few labeled units, with the specific spatial relations that are of interest still intact. In some manner this is what happens in the transformation from a retinotopic description of a scene to a symbolic description. For example, one might gaze at a white styrofoam cup and conclude that there is one in view, yet that cup is presumably represented by a multitude of retinotopic elements in the early levels of one's vision system. Furthermore, the contracted description would be sufficient in many cases for making coordinate transformations, such as from a viewer-centered to an object-centered reference frame. In addition, coloring serves to define and separate regions, to which subsequent routines can be applied selectively. The ability to identify and selectively process a region would be a useful step for recognition or for solving other spatial reasoning tasks.

2.2 Givens, Goals

The abstract computational problem can be stated in brief form. We are given a base representation consisting of a set of elements arranged on a retinotopic surface, some of which are deemed "colorable" and the rest deemed "uncolorable." In addition, one of these elements is indicated to be the "point of interest". The goal is to color all those, and only those, colorable elements that are connected, in a 4-connected sense, either directly or via other colorable elements, to the point of interest.

The set of elements to be used for coloring can be represented by a graph, where the elements are represented by nodes and their adjacencies correspond to links. Having defined the general problem, we can already state the fundamental criteria which govern the coloring operation, in one concise statement.

THEOREM 1 *Each of a set of elements M may be colored if and only if (1) each element is colorable, and (2) each element is either adjacent to, or connected via a path of other members of M to, a colored element.* [1]

Proof: This theorem is of the form $A \Longleftrightarrow B$. We shall prove that $B \Rightarrow A$ and $-B \Rightarrow -A$. These imply that $A \Rightarrow B$ and $-A \Rightarrow -B$.

The result $B \Rightarrow A$ follows directly from our stated goal. At any step in time, all colored elements are colorable (only colorable elements may ever be colored), and all are connected to the point of interest via colorable elements. Therefore, we may color any colorable

[1] An already colored element is considered to be connected to a colored element. This theorem applies when M consists of only a single element, too, of course.

element that is either adjacent to, or connected via other colorable elements to, a colored element (and therefore to the point of interest).

To prove that $-B \Rightarrow -A$, we suppose the opposite and show that it would lead to a contradiction. Namely, we shall first assume that there is an uncolored element of M that colors itself when condition (1) is not met, and show that this is illegal. Then we shall assume that there is an uncolored element of M that colors itself when condition (2) is not met, and show that this is contradictory. The case in which both conditions do not hold true can be ruled out, too, because the first two cases are completely general.

If condition (1) did not apply, then this would mean that an element which covers a contour becomes colored. This is illegal, because such a result runs counter to our stated goal.

If condition (2) did not apply, then this would mean that there exists an uncolored element, $e \in M$, that is neither adjacent to a colored element, nor connected by a path of elements of M to a colored element, but that becomes colored. Let us call the set of all currently colored elements C. Now, without violating our assumption, it could additionally be the case that our set of elements happened to be such that the complement of elements $M \cup C$ (i.e., the set $\overline{M \cup C}$) consisted entirely of uncolorable elements. In this case, all of the colorable elements on the retinotopic surface are contained entirely within either M or C. Yet e is neither adjacent to C, nor connected to C by any members of M. There can be no path of colorable elements connecting the two. Coloring e would violate our stated goal. Thus, for any M, there exists at least one configuration of elements (and usually many) which would force an *unconnected* set of colored elements — hence, a contradiction. Coloring the elements of M under such conditions would be inappropriate. •

While this result is basic, it is interesting because it embodies a recursive constraint. It sets up a recursion relation, and as such, implies that any type of coloring algorithm will have to propagate this constraint along, in some iterative fashion. The nature of the coloring task imposes the property of essential sequentiality on any type of computation done for that operation [Ullman 83]. (A number of related tasks are essentially sequential; in fact, order-limited perceptrons cannot compute some spatial relations in parallel — see [Minsky & Papert 88].)

Given that our goal is to color a set of colorable elements, what is it that might constitute such a region in vision? In order to certify that this computation is appropriate for vision, we must show how the base representations are linked to the early stage of vision. That is, what is represented by the activated region? Later, we shall precompute each

element's colorable/uncolorable state, and represent it as the binary quantity, **contour**, where **contour**= 0 means the element is colorable, and **contour**= 1 means the element is uncolorable. But we need to know, how is the activation to be bounded?

We consider three specific questions: what criteria is used, at which stage is it computed, and what is the form of the result that is used as input to the coloring operation? For the sake of this work, we choose to use contours precomputed from earlier processes as the base representation. The input is distributed on a single surface. We shall motivate this particular choice in light of alternative ones in the discussion below.

The first two questions are intertwined. To understand how the boundaries are imposed, it is important to review the goals behind them. Although there are a multitude of uses for the result of the coloring operation, as discussed above, an invariable objective is to identify and label those regions that share some common property. The first question concerns the composition of the input for coloring: should the coloring process receive a binary contour map (whose values mean either "contour is present" or "contour is not present") which may be propagated up the network to various scales with logical combination, or a continuous-valued property field whose values might be combined quantitatively? The former scenario would use *boundaries* extracted from the image by some other (earlier) process, whereas the latter, which had directly detected the relevant property, would still need to compute region boundaries, perhaps using some local *difference* measurement for comparison. The related second issue is whether the homogeneous regions should be computed *prior* to activation (i.e., prior to the labeling phase) [2], or *during* activation, instead. This ties in with the first issue: rather than precompute boundaries, it could be the case that the colorable components each have some similar property, such as color, intensity, depth, surface orientation, texture, motion, or perhaps something more complex; the bounding process could be based on computing local property differences dynamically, "on the fly."

However, a strong case has been made for the detection of boundaries prior to the inference of other properties from the image [Mahoney 87], [Horn 86], [Marr 82], [Ballard & Brown 82], [Riley 81]. Many object properties simply cannot reliably be extracted directly from the image. Due to numerous confounding factors in the image formation as well as subsequent processes, objects having spatially-uniform properties generally do not yield uniform regions at the early levels of vision. Another argument in favor of precomputing is that one must compute these boundaries anyway, for other purposes, such as for recognition; it would seem quite wasteful to have a second set of all those resources dedicated just for the

[2] This would be analogous to the "smart" planning / "dumb" execution phases of Planning in AI research.

coloring operation. In addition, since coloring is computationally so expensive, having an aspect of sequentially, it would not make much sense to expend additional time computing discontinuity boundaries in sequence across a given region if it could have been computed in parallel for those same locations ahead of time. We would like to do as much of the computation as possible in parallel first, before the sequential processing.

The third point to consider, when considering region coloring in the paradigm of human vision, is what is the form of the input, in principle? Is it fundamentally a *single 2D field* (this would include multi-scale descriptions of which each level is derivable from one common surface); or is it given by *multiple surfaces*, each containing complementary information about possibly distinct phenomena obtained through different scales or even different modes? This last possibility is important, because different boundary information can exist at different scales. Particularly for subjective contours, boundaries found at one scale may not correspond to those at another, and vice versa. This, in conjunction with any boundary information captured via divergent physical modes, creates the possibility of conflicting indicators that would require a complex coloring process.

So then, fundamentally, do we have a monolithic description on one retinotopic surface, or a composite description on multiple surfaces? The operation may construct an elaborate representation for its own use, of course. But how "concise" is the information, in essence, before it is used? The single level idea (as well as multiple-levels, mapped to a single surface) is appealing because it implies a simpler, and therefore cheaper, coloring operation. On the other hand, the heterogeneous multi-level input could be useful when used in either of two ways. First, results could be computed in *separate* groups of surfaces. For example, sometimes only large scale results may be of interest, because they are computed faster. It might be worthwhile to locate the approximate orange and black pattern of a tiger right away, and resolve its stripes at some later occasion. Second, the *conjunctive interaction* of levels could provide a boundary for useful coloring. This would be the case when two sets of regions, whose boundaries are found at specific large and small scales, respectively, happen to form an interesting super-region when linked. (Subjective contours can give rise to such boundaries.[3]) One may then want to color this super-region. This example gives some support to the compute-as-you-go theory. It is possible that some boundaries cannot be inferred until some coloring has already occurred. There may exist subjective contours that consist of colored regions. The viewer may then make an inference that would require coloring this figure (all this might happen pre-attentively, without intention). This means

[3]For example, Mahoney distinguishes between "abrupt-change boundaries" and "alignment contours" in [Mahoney 87].

that output of coloring defines the boundaries, and thereby serves as input, for coloring. So one would need either a smart coloring process, which can change its criteria, or, more likely, feedback to the preprocessing of the base representations.

Hence, we choose, for our base representation, contours precomputed from earlier processes, because it appears to be a much stronger model. Also, our input is, when boiled down, distributed on a single surface, because the issues raised by a heterogenous or multi-scale input are much less tractable. However, it is obvious that some questions regarding biological plausibility remain open.

2.3 General Constraints

Many natural constraints motivate this work. Some are widely known, while others are more subtle. For instance, psychophysical results suggest difficult standards for both performance and competence, that we will try to match. A more detailed survey will come later. For now, we note that some of the more important constraints from psychophysics are that the coloring operation in humans appears to be very fast, and exhibits approximate scale invariance with respect to figure size. Other approximate invariances include those of figure orientation, position, and so on. Also, it is quite robust as well as adaptable (e.g., for broken contours). These will be factors in our choice of representation and algorithm, below.

Two important, general constraints are to minimize time and minimize hardware. This could be considered the vision analog to the "least effort" principle in articulated movement. Time is measured ultimately in some physical sense, although in most representations this could be replaced by the number of discrete steps. Given that our goal is to label a constrained subset of the base representations, we would ideally want to color all of them in one time step, or at least many of them each time step. Labeling one or fewer per time step would be less desirable. The approach we shall ultimately take, of course, is to color connected "patches" at a time, using the maximum scale possible. As for hardware, we could assume that at the algorithmic level, the information processing task would be handled by a set of processors. "Processor" is used in a completely general sense here, to mean any information processing unit occupying space. We would like to limit the number of processors, the power of the processors, and the number of connections between them.

Indeed, a number of biological limitations strongly constrain the algorithms employed by the human visual system. The following is a list of some facts and assumptions that we would like to capture (see [Shafrir 85], [Kandel & Schartz 85]). The input consists of roughly $N \times N$ elements, where $N = 1000$. The number of connections from each processor

varies, but a sensible limit might be 1000 to 10,000 local connections each. This suggests that an interprocessor connectivity degree of $O(N)$. We also assume that the time for a processor to react to a message (a chain of pulses) is on the order of 10 milliseconds. As we will see later, humans respond to simple Inside/Outside problems in about 500 msecs. If we suppose that the basic operation requires 100 msecs of that time, then that would permit computations of only 10 steps or so for simple cases ([Shafrir 85]). Processors should be simple (and usually similar), and messages should be short. Finally, since the response time and other quantities vary between processors, an asynchronous communications mode would be indicated. These kinds of constraints impact the representations and algorithms that can be used to solve the coloring task.

A number of algorithms perform the coloring operation, but do not satisfy constraints outlined above. Standard graph-contracting algorithms on Exclusive-Read Exclusive-Write Parallel Random Access Memory (EREW PRAM) machines can color regions in $O(\log \mathcal{N})$ time. Unfortunately the PRAM model requires the capability of constructing pointers or addressing messages to arbitrary locations. That is, any processor is able to communicate with any other in one time step, and addresses may be passed and utilized. This conflicts with the properties of neurons that are currently known. Neurons are believed not to have addresses, nor the capability of sending addressable messages. Our assumptions of rather simple processors, short messages and largely local connections for biological systems would seem to rule out the powerful class of algorithms based on the PRAM model. Furthermore, the biologically feasible model is less costly and simpler to build than shared memory models [Mahoney 87]. [4]

In addition, we can take advantage of an important computational principle to reduce potential connectivity from $O(N^3)$ to $O(N)$. The systolic conversion theorem [Leighton et al. 88] enables one to formulate solutions to many fundamental graph problems in terms of a systolic algorithm. It is described and illustrated in the Appendix. This final constraint stems from a property of the real world — not from the realm of physics or geometry, but from that of parallel computation — which can be exploited to achieve efficient region coloring with a legitimate representation.

[4] See also *T. Poggio. "Routing Thoughts," AI Working Paper No. 258, May 1984.* S. Ullman contributes to this internal memo, in part, by proposing that the type of routing scheme employed is an important issue, and could serve as a basis for useful classification of different types of parallel machines.

3 Representational Issues

3.1 The Image Chunking Representation

Image chunking is the result of a set of fast parallel processes which construct the particular representation from pointwise properties of the image. These are treated as units during spatial analysis. Image chunking ultimately permits a concise, efficient description of the image. Its general efficacy can be conveyed with a couple of illustrative analogies. Mahoney offers the analogy of efficient house-building: if one wanted to build a house quickly, one would use prefabricated parts that could be preassembled in parallel [Mahoney 87]. A more operational analogy might be that of the "One-Minute Manager Delegation Paradigm" – if one can reliably delegate tasks to subordinate processes, being assured that the task can and will be performed, one can accomplish tasks at a much higher rate.

With this vague motivation, we also need to accommodate some specific concerns. For one, higher vision must be *spatially focused* and *goal driven*. This is partly due to the complex and transitory nature of the incremental representations, given limited computational resources. It implies that a general ability to decompose the retinotopic representations into meaningful expressive entities that are in terms of the visual goals would be indispensable. For instance, an indexing task (shift of processing focus) would generally be given in terms of extended scene entities, so image chunks would be a more appropriate basis of location than pixels. [Mahoney 87] In addition, we recall some relational tasks are *essentially* sequential. Examples include coloring, in particular, contour tracing, and even some aspects of house-building (laying the foundation precedes the walls, which precede the roof [Mahoney 87]). It is therefore important to utilize chunks which would minimize the number of steps to perform these inherently serial operations. Finally, chunks should be relatively simple, and easily combined. One needs to be able to construct a variety of useful incremental representations with them.

We shall develop and investigate several coloring algorithms which are based on the image chunking representation. Figure 6 gives a simple example of region coloring by this means. We employ a set of modules, each of which describes the property state of a particular square-shaped area of retinotopic elements from the base representation. Where each module is able to get its information from varies from algorithm to algorithm. In an intuitive view of the complete operation, coloring information flows up from the pixel-scale level to higher-scale modules, where it is passed around at a high speed (i.e., large image distance per step of time) during each cycle. Eventually the pixel-scale modules are

themselves updated with color information.

In Figure 6, for instance, the image chunking representation is deployed in the form of a quadtree. Each module in the lowest level describes the state of its respective base element, extracted from the image, in terms of (1) presence of a contour and (2) whether or not that module is colored at the present moment. Modules labeled as X are done so for illustrative purposes only. They are considered synonymous to a colored module, one that was colored at "time 0." In this example, we have passed up contour and X information during a preprocessing stage, defining that as "time 0." Each module in a given scale level covers a unique square-shaped area, which is the union of the areas of four modules beneath it in the next-lower scale level. During the upward propagation process, each higher-scale module eventually receives messages from its four "children." If *any* sends a message indicating the presence of a contour, then the module records this presence, and then also sends such a message during the next step. The higher-scale modules pass up X information in a similar fashion, although each records and then sends a colored value subject to the *non-presence* of the contour. Thus, with repeated steps, all modules come to describe the contour and starting point properties of the base representation in terms of those from its subordinate modules.

During the main stage, color information is passed up, over, and down in a similar iterative way. At each step, each module records and sends color information to all other modules to which it is connected, but only if it has not recorded the presence of a contour. We reserve further describing the particulars of the process until the algorithmic sections, starting with Section 5. This operation description serves to illustrate how, as a general rule, the image chunking representation yields a multi-scale description of the base representation that is often based on a hierarchy of components. As we shall see, using such a representation can substantially reduce the number of steps required to label connected components.

3.2 Formal Definition of Coloring Problem

After giving the descriptive overview of the image chunking representation, we now state formally the coloring problem. We present the task, the general representational model, and some useful definitions. Then we derive a Corollary to Theorem 1, which is in a simpler form, and which states the fundamental coloring criteria for any specific algorithm that employs the image chunking representation.

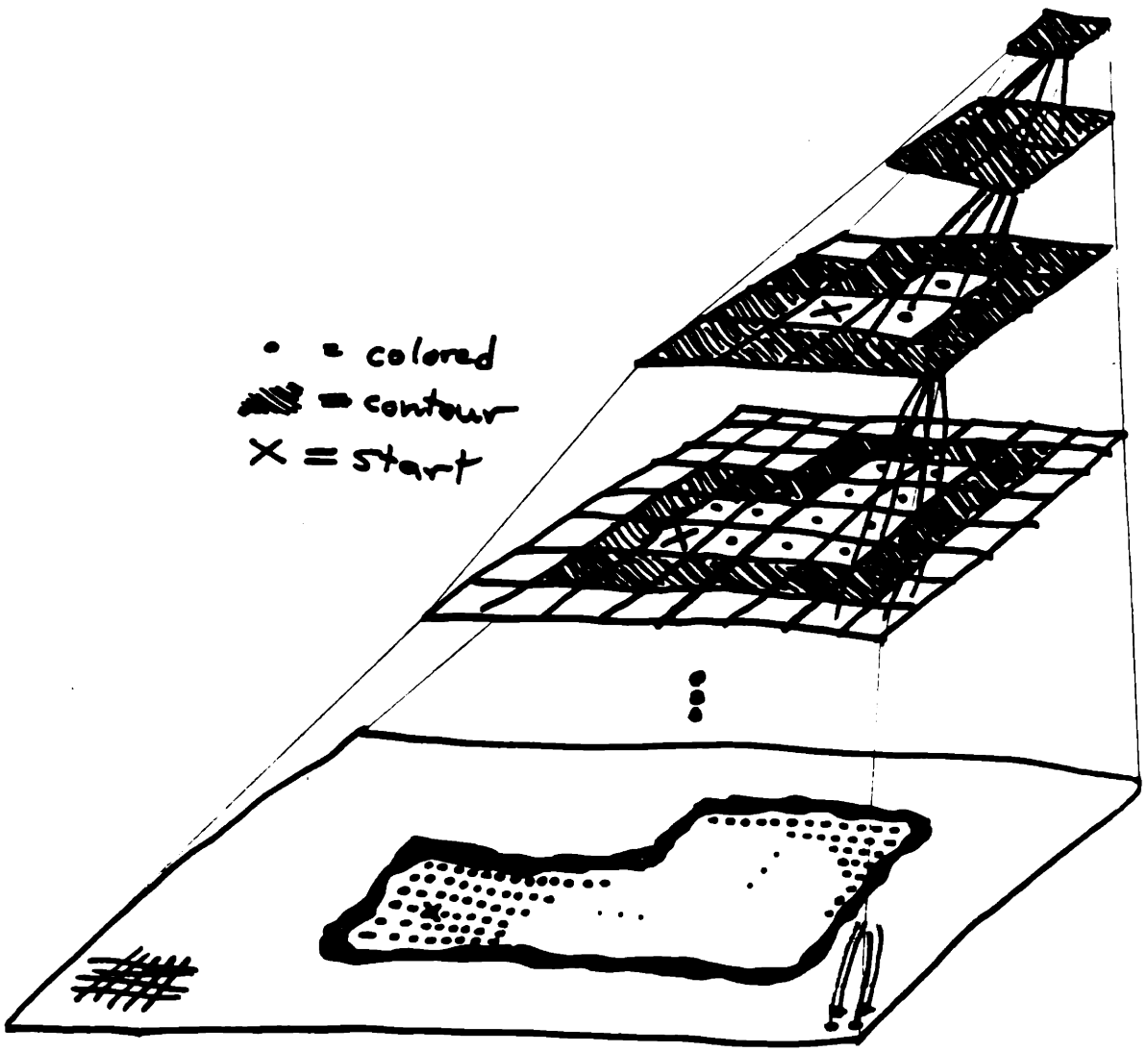


Figure 6: The quadtree is an example of the region chunking representation.

Given: We are given an $N \times N$-pixel binary image, where each pixel indicates either "free-space" or "contour." In addition, one of the pixels is labeled so as to indicate that it is colored at the start.

Initial Assumptions: For simplicity, yet without loss of generality, we assume that the generic region-chunking model consists of

1. an $N \times N$ array of "pixel-scale" (i.e., unit-size) modules, each mapped to a unique image pixel, plus
2. a number of "higher-scale" modules, each mapped to a unique subset of 4-connected image pixels.

We see that one pixel-scale module corresponds with the image pixel that was colored at the start; let us denote this module as X. Thus it is sufficient, and we shall require, that coloring proceed only among the modules, a uniform class of region chunks.

Furthermore, at any step in time during the coloring process, each module has information only about

1. whether or not any adjacent modules have been activated, and
2. whether or not any of its constituent pixel(s) is a contour or X. [5]

Based on this alone, the module makes decisions about whether it should activate (or deactivate) itself. It does not know, for example, the status of non-constituent pixels.

Goal: Our task is to "color the region containing X." Let us define this region to be the maximal set of pixel-scale modules $\mathcal{P}$, such that, for every module $p \in \mathcal{P}$, there is a 4-connected path from p to X via the pixel-scale modules, none of which is mapped to a contour pixel.

We define here a few terms that will facilitate later discussion of chunk-at-a-time region coloring algorithms.

Definitions: Let P_1 and P_2 denote the respective sets of pixels to which modules m_1 and m_2 map uniquely.

[5] Later, when we extend the algorithm to handle broken and shifting contours, additional information will need to be available to the modules.

m_1 **covers** m_2 iff $P_2 \subseteq P_1$.

m_1 **overlaps** m_2 iff there exists a pixel p such that $p \in P_1$, $p \in P_2$, and neither module covers the other. ("m_1 overlaps m_2" $\Longleftrightarrow$ "m_2 overlaps m_1.")

m_1 **borders** m_2 iff there exists no pixel p such that $p \in P_1$ and $p \in P_2$, and there exist two pixels p_1 and p_2 such that $p_1 \in P_1$, $p_2 \in P_2$, and p_1 and p_2 are 4-connected neighbors.

m_1 is **proximate** to m_2 if and only if m_1 covers, overlaps or borders m_2. (A module is proximate to itself.)

Before making the claims below, we impose some final requirements.

Restrictions on the Model: We narrow the class of canonical models to those which are

- capable of coloring any 4-connected region of pixels, and
- comprised of modules, each of which are mapped to a convex set of pixels.

The first restriction is a trivial one that assures generality. The second restriction limits our discussion to those modules that have a convex shape; that is, each module corresponds to a set of pixels which would entirely fill the convex hull of that set.

Now that our general framework is defined, we can state the fundamental criteria which govern any specific coloring algorithm based on image chunking, in one concise statement. It is simply a corollary of Theorem 1, simplified for the special case of our chosen representation (i.e., image chunking).

COROLLARY 1 *An unactivated module may activate itself only if (1) it does not cover a contour, and (2) it is proximate to an activated module.*

Proof: Like Theorem 1, this corollary is of the form $A \Longleftrightarrow B$. We shall prove that $B \Rightarrow A$ and $-B \Rightarrow -A$. These imply that $A \Rightarrow B$ and $-A \Rightarrow -B$.

The result $B \Rightarrow A$ follows directly from our defined goal. Since the module in question is mapped to a set of pixel-scale modules, if premise B is true for the module, then B holds true for each of its pixel-scale modules too. At any step in time, all colored pixel-scale modules are colorable (only colorable pixel-scale modules may ever be colored), and all are

connected to the point of interest via colorable pixel-scale modules. Therefore, we may color any colorable pixel-scale module that is either adjacent to, or connected via other colorable pixel-scale modules to, a colored pixel-scale module (and therefore to the point of interest).

To prove that $-B \Rightarrow -A$, we suppose the opposite and show that it would lead to a contradiction. Namely, we shall first assume that there is an unactivated module that activates itself when condition (1) is not met, and show that this is illegal. Then we shall assume that there is an unactivated module which activates itself when condition (2) is not met, and show that this is contradictory. The case in which both conditions do not hold true can be ruled out, too, because the first two cases are completely general.

If condition (1) did not apply, then this would mean that a module which covers a contour becomes activated. This is illegal, by our definition of coloring.

If condition (2) did not apply, then this would mean that a module that neither covers, nor overlaps, nor is adjacent to an activated module becomes activated. Let us call this hypothetical module h and the set of all currently activated modules M. Let P_h and P_M denote the sets of pixels to which are mapped h and M, respectively. The negation of condition (2) implies that there exists no pixel p such that $p \in P_h$ and $p \in P_M$, and there exist no two pixels p_1 and p_2 such that $p_1 \in P_h$, $p_2 \in P_M$, and p_1 and p_2 are 4-connected neighbors. That is, P_h and P_M are unconnected sets of pixels.

Now, without violating our assumption, it could additionally be the case that our contour image happened to be such that the complement of pixels $P_h \cup P_M$ (i.e., the set $\overline{P_h \cup P_M}$) consisted entirely of contour pixels. In this case, all of the free-space pixels in the image are contained entirely within either P_h or P_M. P_h and P_M are disjoint, non-null sets. There can be no path of free-space pixels connecting the two. Activating h would violate our stated goal. Thus, for any such h, there exists at least one contour configuration (and usually many) which would force an *unconnected* set of colored modules — hence, a contradiction. Coloring h under such conditions would be inappropriate. •

3.3 Region Coloring with Convex Chunks: Not Scale-Invariant

An important issue that motivates the development of the algorithms here is scale invariance. In this subsection, we present a theorem that formalizes an important effect of scaling figures for images. As a figure is reduced in size in one or more dimensions, contour pieces that were previously represented by several pixels become coalesced into a single pixel representation. The figure is effectively smoothed, detail is lost, and the figure becomes

simpler. Therefore, coloring at two different scales can actually be two different tasks. If so, the larger scale typically requires more time. The following theorem shows that, for a convex image chunking scheme, scale invariance is generally unrealizable. The proof is based upon image geometry and theoretical requirements of image chunking, and does not depend upon any particular coloring algorithm.

The $O(N)$ connectivity constraint pushes a valuable goal, *scale invariance*, beyond reach:

THEOREM 2 *Given a bounded activation algorithm that satisfies the criteria defined above, there exists a closed contour for which one can specify two sizes and positions with respect to the image such that coloring the larger region requires more time steps than coloring the smaller region.*

Proof: The proof examines a particular shape at two different scales, and shows that the coloring must take more time for the larger region, unless the network has high connectivity.

Assume that the image consists of $N \times N$ pixels, where $N = 2M$ and M is divisible by 4. [6] The shape is drawn in Figure 8A, for the larger choices of size and position. Notice that it is made up of two parts, each of which can be inscribed in a square. One part of the region consists of an $M \times M$ area of free-space pixels. X is somewhere inside this part. The second part contains a snake-shaped area of free space of width 1. The rest of the image pixels are contours. The region's horizontal position in the image is arbitrary.

To construct the image with the smaller region, we first position the shape such that the square-shaped part is centered under one of the network's pixel-scale modules. Then we simply shrink the shape's size until its square part is just detectable to the module. The snake-shaped part is therefore entirely positioned under an adjacent pixel-scale module which, of course, detects either free-space or contour areas. For appropriate discretization support functions (see [Fleck 88], Ch. 2), the snake-shaped part, at this scale, is treated like a contour pixel. None of the network's modules see it as free space, since there are contour values inside each of their fields of view. That is, the second image, sketched in Figure 8B, will consist of 1 free-space pixel and $N^2 - 1$ contour pixels. [7] Since the X was

[6] This proof can be adjusted trivially to cover cases where N is some other value. Note also that we are not strictly interested in solving the Inside/Outside problem here; to do so for our simple example in Figure 8A would be boring. Rather, we are interested in the problem of coloring an Xed region.

[7] One can start with a smaller snake part — $\frac{N}{4} \times \frac{N}{4}$ (still 1 pixel wide) for instance, joined to the same $\frac{N}{2} \times \frac{N}{2}$ square part, before shrinking the entire region for the second image. The proof still holds. Now

inside the square part, the single free-space pixel is the X. The pixel-scale module that covers the X activates itself in the first time step. Our canonical network, then, can color the smaller figure in one step.

To color the larger region in one time step is impossible. Notice that at least $(\frac{M}{2} + 1) \times \lfloor \frac{M}{3} \rfloor$ or $O(N^2)$ convex modules are required to cover completely the snake-shaped subregion. So there are $O(N^2)$ connected modules that are not proximate to an activated module or (equivalently) the X at the starting time, and which must be colored sometime. By Corollary 1, these modules may not activate themselves in the first time step. Hence, our network will use more time for this larger figure. •

We presented the rather elaborate counter-example above so that we could continue here with some additional important observations. First, the set of modules that cover the snake-shaped free-space subregion in Figure 8A can be represented by a graph with $O(N^2)$ elements and connectivity degree of only one; coloring this region takes $O(N^2)$ time. Second, suppose we relax some of the assumptions about what information is available to each module, so that Corollary 1 no longer applies. To color the larger region in one time step, if some way were possible (using the systolic model of computation, as is standard), we would need $O(N^2)$ connections at pixel X. To see why, note that during that single step, $O(N^2)$ modules must decide whether or not to activate themselves. The modules that cover the snake-shaped subregion, plus other modules (since, indeed, we can draw many other shapes within the snake's square bounds and elsewhere, which would require $O(N^2)$ modules to become completely covered) must each know whether X is in its region. Since each of $O(N^2)$ modules must have access to X during that step, X and hence, the network, must have connectivity degree of at least $O(N^2)$ for the network to be scale-invariant.

We have therefore shown that no $O(N)$-connectivity coloring network with convex modules is scale-invariant. Strictly speaking, they all have $O(N^2)$ runtime in the worst case. Of course, one generally can obtain faster speed with increased connections and/or processors. Nonetheless, the result is somewhat daunting. True scale invariance under the conditions established at the beginning is impossible. It is true that humans exhibit approximate scale-invariance in coloring a broad variety of *simple* closed contours. Our best algorithms perform this way for simple and particularly convex shapes. However, like our canonical algorithms, humans require more time for coloring more *complex* closed contours like that in Figure 8A. These tasks can vary in complexity as their scale varies. In such cases humans seem to suffer the same fate as our theorem dictates. Hence, a worthwhile goal would be

the snake-part covers only $\frac{1}{4}$ of a pixel's "field of view," while $\frac{3}{4}$ of the area is undisputed contour, so the pixel would be likely to respond as if it sees a *contour*.

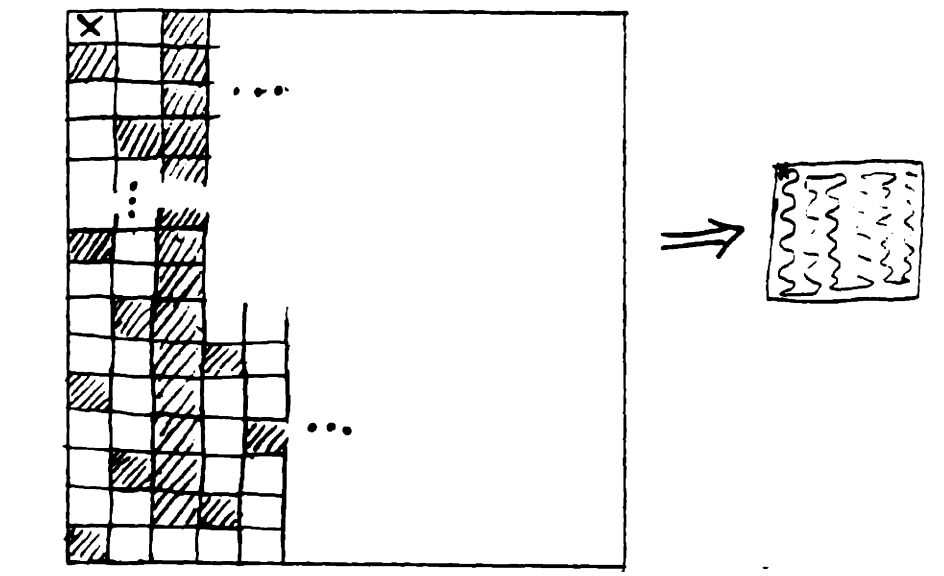


Figure 7: A snake-like figure.

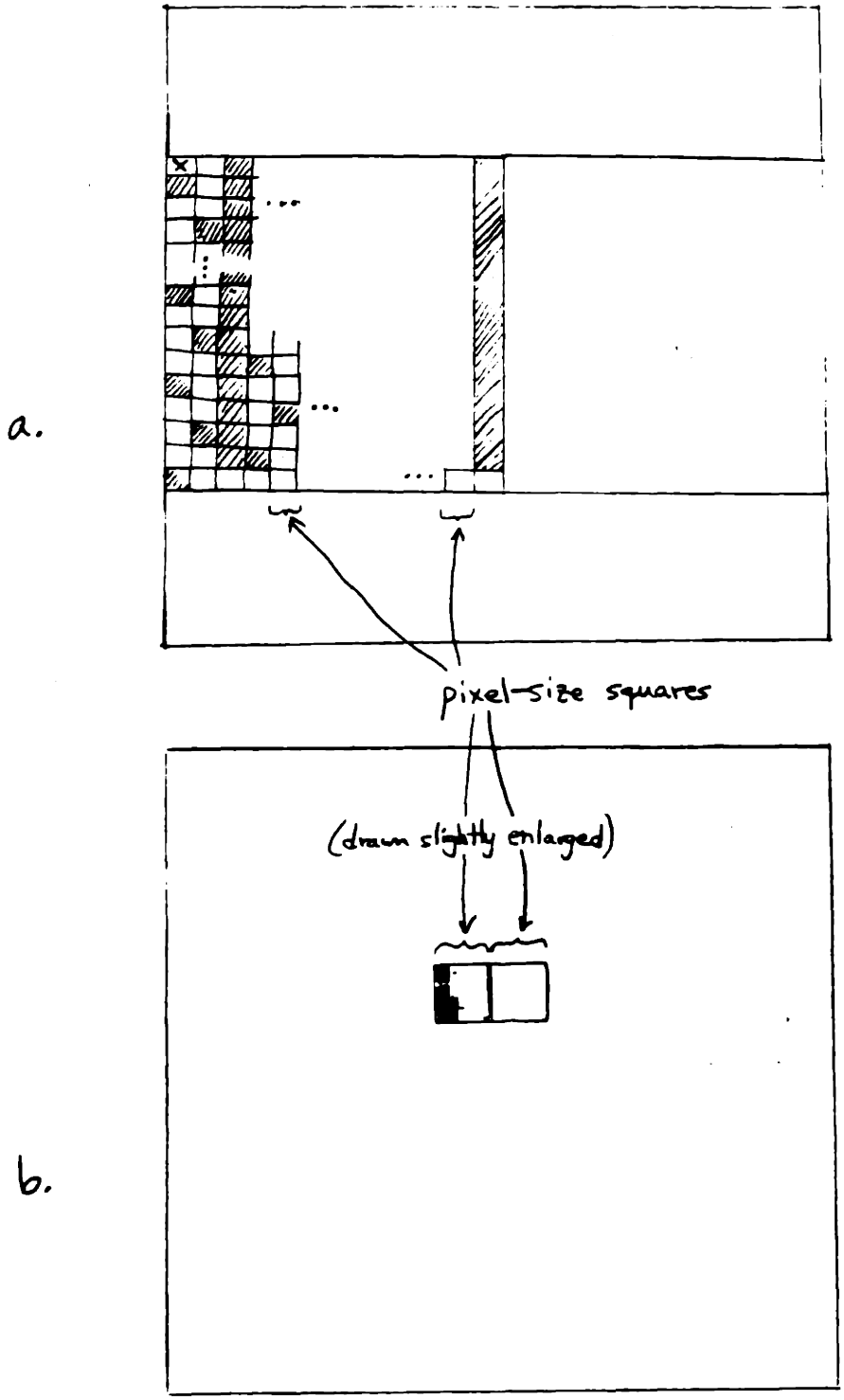


Figure 8: A shape whose coloring time varies with its size, given a coloring network with convex chunks.

to design an algorithm whose *average* runtime for many simpler problems is not only fast, but matches human performance in terms of near-invariance to scale, position, orientation, and so on.

3.4 Use of Parallel Square Region Chunks

The initial motivation to investigate hierarchical square-chunk coloring schemes came from two ideas: regions can be colored faster with larger chunks, and it would be advantageous to view a region at both coarse and fine scales. The ability to view a region at a particular scale is inherent to the hierarchical representation. A module is directly connected to other modules of the same scale, as well as to parent and children modules of the next larger and smaller scales, while being indirectly connected to all other modules. The intuition is that a network should view and color a region primarily at coarser, *simpler* scales, and use the finer modules to handle the complex details near contours.

The use of parallel *square* modules is just a special case of the image chunking representation described earlier that we choose to pursue. Only the module *shape* is specified further. (The specific *algorithms*, of course, introduce some complexity later on.) We have presented the use of image chunks in Section 3.1. Recall that the example given (see Figure 6) was the based on the quadtree model. It uses a set of modules, each of which describes the state of a particular square-shaped area of retinotopic elements from the base representation. We choose squares in part because they yield simple, area-based descriptions. They can be easily combined to give meaningful descriptions in terms of extension in space of a property. Since squares are roughly symmetric, they need be deployed in only one orientation to perform well. As far as algorithm design goes, there is nothing inherently special about the shape – the algorithms below could be modified to account for any kind of area coverage, whether it is some other convex-shaped module or not.

Other properties we strive to capture include usefulness and generalizability. It would be advantageous if the network happened to generate a rich representation of the shape as it colored it. We shall see that each of the square-chunk coloring algorithms below essentially views the image at many different scales at the same time, and yields an ordered set of planar descriptions that can be used by other operations. This feature allows the networks to be easily extended to be more flexible. In particular, they facilitate the generalization of our best network to handle moving contours, and broken contours, with only a constant cost in speed.

4 Background

4.1 Impressive Human Performance at Coloring

In this section we review various results which demonstrate impressive human abilities relating to coloring, and which suggest why the chunking representation is a valid candidate. In general, the studies support the idea that coloring proceeds slowly for complex figures, but is very fast for simpler shapes. The traversal time from one point to another for different scales varies slightly for the relatively simple figures used. There is persuasive evidence for a chunk-based operation in human vision, but it would have to be a very fast routine. So far, there has been little psychophysical inquiry into the nature of the region coloring operation per se. Usually the focus is on some effect to which coloring might have contributed.

One revealing study [Jolicoeur et al. 86] looks into the behavior of a related type of coloring – contour coloring, or curve tracing. In one experiment, subjects were shown several curves in a visual display with two Xs placed on them, and were asked to determine as quickly as possible whether the two Xs were on the same curve. In the other experiment, the task was to judge whether or not a curve joining two Xs had a gap in between the Xs. In both cases, the decision times increased as the curves between the Xs was lengthened. The investigators infer from this finding that human tracing often moves at some constant maximum speed (a substantial average rate of 40° of visual angle per second). However, this is not a strict gauge of scale invariance (and is not claimed to be): the curves remained the same size, while the X's position along the curve varied. So the inter-X distance was scaled along the 1-dimensional space of the curve itself, but not in regular 2D image space. Given that the task called for discrimination which possibly involved another curve not in the space of the first curve, the scaling would not be strictly linear in any relevant dimension. The experiments demonstrate the relatively high speed of operation in humans, and provides evidence that the computation of some spatial relations involves a contour coloring routine.

Some important parallels can be drawn between the underlying processes for region coloring and mental scanning. Considerable literature can be found describing the properties of mental scanning, both along contours and across regions. For example, there is evidence that suggests that mental images preserve metric distance information. ([Finke & Pinker 82]. [Jolicoeur & Kosslyn 85].) It is an open question whether the scanning mechanism resembles or is possibly even synonymous with the mechanism for coloring. In [Reed et al. 83].

the time needed for scanning does not vary strictly with distance under some circumstances. In [Jolicoeur & Kosslyn 85], more time is required to scan longer distances. Experiments typically require a human subject to imagine a small black dot moving as quickly as possibly from one point to another in a mental image. This is problematic: since the process is a conscious response – the subject's idea of "moving at high speed" may not depend on the distance traveled. That is, the subject's "high speed" may be a roughly constant rate, leading to possibly unfounded conclusions of scale dependency.

One interesting result in particular ([Reed et al. 83]) lends at least some support to the psychophysical validity of a chunking-based algorithm. In one experiment, subjects were directed to scan along various shapes. They found that scanning along square spiral paths required the most time, while straight-line paths were fastest. That is, the scanning is constrained to travel between the two contours on either side of the path. This is just the result we see obtained by coloring: a snake or spiral-shaped figure presents extraneous contours which force coloring to occur at a small scale in order to avoid them, hindering its performance. On the other hand, straight-line paths across open areas would present no obstacles to most higher-scale modules, enabling activation to occur at a large scale and rate.

Other experiments, on the shifting of attention, affirm the power of contour coloring in humans. This stems in part from the results of [Shulman et al. 79] and others that the orienting of attention requires a certain amount of time, and that its locus moves through intermediate locations ([Shulman et al. 79]). In addition, there is some empirical support (e.g., in [Remington & Pierce 84]) that the time required to perform a shift of attention does not vary significantly with the distance traversed during the shift. (There is also evidence for a time-dependency [Tsal 83]. Much is unknown or in dispute in this area.) [Jolicoeur et al. 86] postulate that such evidence points to the possibility of a common mechanism between the processes of attention shifting and curve tracing. To carry the speculation one step further, Mahoney describes how contour coloring and region coloring processes could share common machinery in [Mahoney 87]. (See Section 4.2.) These results would lend additional support to the idea that coloring must be effected by a powerful set of chunking processes.

One final result of interest concerns determining inside/outside relations for broken closed contours. For many such tasks, including variations of the "XIO problem", people are able to make consistent judgements suggesting that boundaries presented for coloring do not have to be continuous in some cases ([Ullman 83]). Varanese observed that human response time for solving XIO-type problems with fragmented boundaries was generally

greater than for those with complete boundaries. The additional time required was modest, with a mean difference of about 20msec and an average response time of about 540 msec. ([Varanese 83], [Ullman 83].) This suggests that extra processing might occur in order to handle broken contours, and that it would be quite fast (its time would not grow enormously for large input).

Hence, we are challenged to find a fast, robust region coloring scheme using image chunks. The discovery of an optimal scheme, which could color within fragmented contours at constant additional cost, would be valuable.

4.2 Earlier Approaches using Image Chunking

Several earlier approaches seek to extract meaningful scene components by doing image chunking. Their goals include labeling boundary elements and labeling region elements. All generate descriptions based on two stages of processing: the first involves generating a description of the input in terms of a specific type of image chunking representation, while the second entails the application of some variation of coloring algorithm.

Edelman's work on fast boundary coloring in [Edelman 85] relates to the investigation into human curve tracing by [Jolicoeur et al. 86]. Edelman presents two similar algorithms for labeling a connected curvilinear component. Both methods consist of a first phase in which regions of the image containing a single connected curvilinear figure ("OK-regions") are detected and propagated up a quadtree-based network. Then in the second phase, the connected curvilinear component is colored by activating one maximal OK-module per step. The schemes differ in terms of how the OK-values are computed for each module during the upward propagation phase (first phase). In one scheme, a module is OK if all its submodules are OK, if the Euler Number for its region equals 1, and if a test for closed loops in the region returns null. The second version simply requires that the Euler number computed over the region be 1 (0 for vacant regions). The former algorithm is provably correct, while the latter is much faster but may occasionally label as OK a region with more than one connected component.

The main difficulty with Edelman's approach as a possible model of the operation in humans is that it calls for counting of contour junctions and terminations inside entire module regions,[8] and then passing these arbitrary numeric values between modules. These

[8]In our algorithm extension for closing broken contours. we detect the presence of *any* contour terminus in the input that is covered by the module, plus *whether there are more than one* contour termini. This is a simple logical operation (AND (XOR ...) (IOR ...)) of the inputs in the region, and is not very costly to

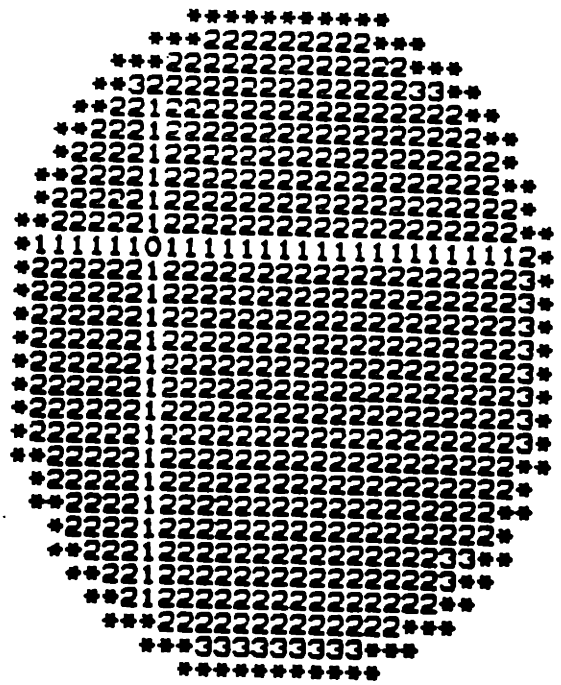


Figure 9: Example of Shafrir's Rectangular Model coloring a circle in three phases.

processes violate our assumption of simple local processors and messages.

Shafrir presents a region coloring algorithm in [Shafrir 85] that is very fast for many tasks. He studies three schemes, each of which have a preprocessing phase in which regions containing no boundaries are detected. Shafrir's most powerful method is based on a network he calls the "Rectangular Model." It consists of modules mapped to a large assortment of needle-shaped region chunks of varying lengths. Each module connects to a number of adjacent pixels in a single row or column. (A simple diagram of this coverage is sketched in 1D in Figure 10.) Each module can be thought of as a "free-space" detector that can take on one of two values — *vacant* or *non-vacant* — depending upon whether or not *all* of its constituent pixels are free of contours. The algorithm proceeds in the following cycle: each colored pixel activates all rectangular modules which cover it, and then these in turn activate all pixels they cover. To constrain the connectivity degree of the processors, modules cannot communicate with each other directly, but rather via their constituent pixels. Even so, Shafrir's algorithm colors many simple closed contours in only a few time steps. (See example in Figure 9.)

However, it becomes obvious that the runtime is at least $O(N^2)$ when the algorithm is

compute.

presented a more complicated region for coloring, such as the snake-like one in Figure 7. Intuitively, this is because coloring advances in overlapping rectangular blocks, and many such steps would be required to fill Figure 7. Humans exhibit considerable slowdown in the completion of such tasks, as well, hypothetically because narrow regions hinder coloring at large scales. [Jolicoeur et al. 86] Furthermore, although the Rectangular Model is one of the fastest algorithms known, it lacks several important abilities that people possess and which we hope to capture with the algorithms below. In addition to meeting a number biological specifications (given above), the algorithms we present below were developed to surpass, in varying degrees, the Rectangular Model in terms of some additional important properties. Like the Rectangular Model, all of these networks use modules to represent subsets of the image elements. Each module maps to a particular square-shaped region of pixels. Each exhibits approximate scale invariance for simple closed curves. (See Figures 14B*ii* and C*ii*, for instance.) Finally, it is interesting to note that the complete, parallel, square-based scheme contains nearly as many modules as the complete, parallel, needle-based scheme. For example, the former requires exactly N^3 modules for a square image of size N, whereas the latter requires $N^3 + N^2$ modules.[9]

The square-based networks developed in this thesis have a number of advantages. First, the connectivity for the complete needle-based scheme is of degree $O(N^2)$ — see Figure 10 — whereas our networks will be shown to have connectivity degree $O(N)$ or $O(1)$. Second, a needle-based scheme that uses only X- and Y-directed modules is drastically sensitive to figure orientation for some cases, while some of our square-based schemes achieve approximate orientation- and scale-invariance for a broad class of problems. Third, an algorithm that scans areas rather than line segment portions of the base representation possesses additional properties which are important for handling some common variations of input in psychophysically plausible ways. Our optimal algorithm can be extended so that its area-covering modules cover either gaps or subregions to be uncolored, and thereby enable broken or shifting contours to handled gracefully. Hence, one challenge faced by the Rectangular Model is robustness: Shafrir's algorithm is fast, but is too specialized to be modified in similar fashion. Likewise, Lim's graph-contraction algorithm ([Lim 86]) would fail for noisy input; it is inherently topological, not geometrical. Our square-chunk coloring scheme possesses such flexibilities. Finally, the intermediate results of the square-based

[9]For square modules, the number of modules at each scale level is $N \times N$. Hence, the formula is given by N^3.

For needle modules, at scale levels $1, 2, 3, ..., N$ we need $N, ..., 3, 2, 1$ modules for any given row or column. For all $2N$ of the rows and columns, we require a total of $(2N) \times (N \cdot \frac{N+1}{2})$ or $N^3 + N^2$ modules.

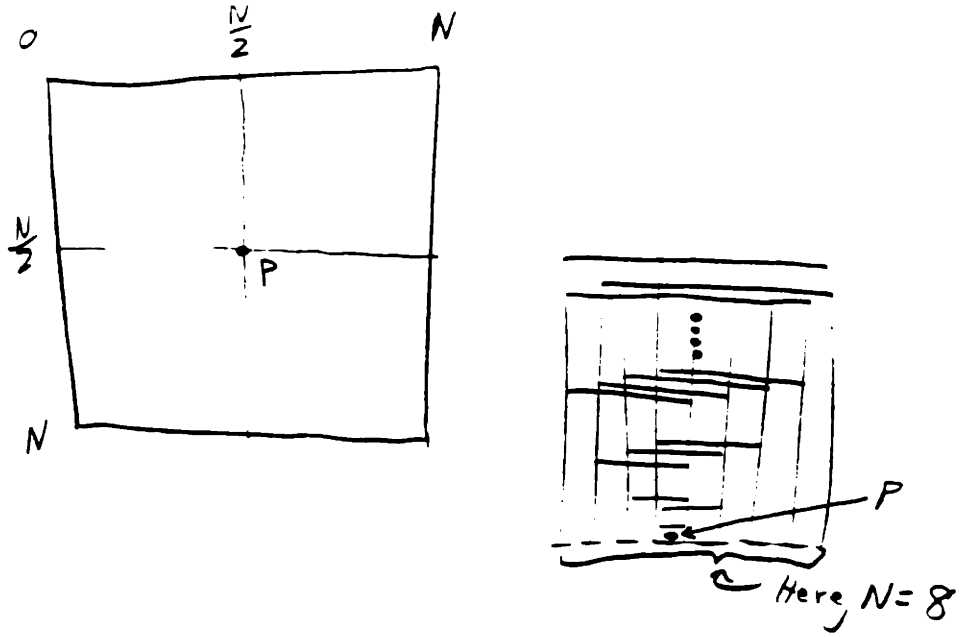


Figure 10: The connectivity for the complete needle-based scheme is of degree $O(N^2)$. Simple Proof: Pixel p is in the $\frac{N}{2}$th row and column of the image. Covering it are one module of length 1-pixel, 2 overlapping modules of length 2-pixels, 3 overlapping modules of length 3-pixels, and so on. This means that $\left|\begin{matrix}\#\text{ of Modules} \\ \text{Connected to } p\end{matrix}\right| =$

$[1+2+3+...+\frac{N}{2}+\frac{N}{2}+...+3+2+1]\times 2 = [2\times(\sum_{i=1}^{\frac{N}{2}} i)]\times 2 = O(N^2)$.
We conclude that there are $O(N^2)$ modules which must communicate with $p \Rightarrow$ needle-based scheme has $O(N^2)$ connectivity. However, this scheme can be retimed.

networks given below are simpler, more stable, and more predictable in the sense that coloring proceeds more like people do consciously. It proceeds radially, like an acoustic wave.

Another important constraint suggested by human biology is that which limits each processor to having $O(N)$ local connections. It has forced Shafrir to implement a weaker, and less elegant, version of the ideal Rectangular Model. First, as mentioned before, higher-scale modules must communicate with each other indirectly, via the pixels. Second, Shafrir is unable to use all of the modules — of all lengths and positions — that would be found in the complete Rectangular Model, for the connectivity degree would be far too great. Instead he constrains the use of modules in such a way that the number of modules which

communicate with any given pixel is $\leq N$. The $O(N)$ connectivity constraint hinders the performance of our best square-based network as well, by a small factor in the worst case. That is, we are able to employ every module called for in the ideal square-chunk model, while suffering only a small constant lag in communication. This is discussed in Section 6.

Mahoney's region chunking proposal in [Mahoney 87] goes beyond Shafrir's work in that he considers the possibility of sharing computational resources between curve and region chunking processes, and he extends region chunking into a general framework. He presents several simple curve and region coloring routines which are based on either fixed, limited extent tesselation or a binary image tree tesselation. He describes how both the curve chunking and the region chunking processes can be decomposed into two processes, that of generating a tesselation, and that of checking region validity. He then describes for several specific cases how curve and region chunking could operationally mesh with one another. He also shows how the region coloring can be expedited by applying the curve coloring routine to the input first. For one thing, this reduces the magnitude of the coloring problem simply in terms of number of elements to color. In addition, it also is equivalent to scaling down the problem geometrically, insofar as the conditioned problem can be described in terms of large-scale chunks only, and is equivalent to a description in terms of small-scale modules of that problem scaled down. So while we later will present the square-chunk coloring algorithm that is optimal for a given problem, a justifiable method for simplifying its task would make it "more optimal." It may be the case that a conjunctive approach to curve/region coloring yields an approach that is superior for meeting the underlying perceptual goals.

4.3 Other Representations and Algorithms

Given that our goal is to label connected components in a base representation consisting of colorable and uncolorable elements, it is natural to ask what is the fastest algorithm known for doing this. Obviously, one could be extreme and imagine a constant-time but high-connectivity algorithm having detectors for regions of every shape, size, position and orientation. More realistically, Leiserson & Phillips have recently presented a linear-processor randomized algorithm that with high probability can contract a bounded-degree planar $\mathcal{N}$-element graph in $O(\log \mathcal{N})$ time for EREW PRAM networks [Lim 86]. (Notice that $\mathcal{N} = N^2$. This is important because an algorithm with time complexity $O(N^2)$ is only $O(\mathcal{N})$.) The contraction algorithm can be applied directly to solve the region coloring problem on suitable parallel machines. Its time complexity is quite modest, and it

is embedded in a fairly weak computational model (although it still depends on shared memory).

Besides this completely general topological solution, one should consider the way in which the solution will be used. There may well be tasks for which the complete coloring solution is not required. In this case, other algorithms and representations may suffice. For instance, there may be a host of geometric approaches, including such heuristics as the ray intersection method, which could be selectively applied depending on the situation and goals at hand.

The underlying motivation for image chunking is to make scene entities distinct quickly so that they can be subjected to further analysis. [Mahoney 87] Mahoney makes an interesting comparison between image chunking and a variety of image analysis techniques which seek to *articulate* a pointwise description of an image, i.e., to generate a new description whose elements are more meaningful. Region-based image articulation methods include those loosely categorized as region segmentation and blob detection. The former are generally non-descriptive, while the latter can be descriptive. Both generally assume that scene entities yield regions that are homogeneous in some way. Such alternative methods are intended to yield bases for solving a number of spatial analysis problems.

5 Learning from Preliminary Algorithms

In this section we investigate variations in algorithm structure to see its impact on performance. A reasonable first step is to examine four algorithms that have each combination of the characteristics of **Module Sizes: every possible size / powers of two** and **Module Positions: every possible position / no intra-level overlap.** We analyze and compare the performance of each scheme through computer simulations. This allows us to gain a critical understanding of how each network behaves for typical input, as well as to test and verify new ideas. This exploration leads to the rejection of two of the algorithms and provides the basis for the design of new, more efficient algorithms.

Most of the following algorithms have been implemented on the Connection Machine; the others were deemed unworkable. Each implemented algorithm was made part of common system, since the implementations have many structures and procedures in common. For example:

Input: The system receives a square image containing contour elements and "pre-colored"

element(s) which could represent X. The image size is $2^N \times 2^N$ for Algorithms #1 and 4, and $N \times N$ for Algorithms #2 and 3, where N is a positive integer limited only by our finite hardware. Specifically, we input a square array of two fields: **contour** and **colored**. Each element can have a value of 0 or 1 for its **contour** field, and for its **colored** field; 0 indicates absence and 1 indicates presence of that property.

Output: The system yields a colored network of modules, with *all* appropriate pixel-scale modules activated. Currently, we require that the coloring not cease until every legal activation has occurred, including the finer scales. The alternative would be to have coloring cease as soon as every pixel in the region containing X is covered by at least one activated module of *any* size, so that the result may be used sooner by other processes.

Each algorithm can employ many general-purpose operations — such as one to propagate values up or down the network according to a specified function. For example, one may pass the "minimum time step when colored" value or "maximum colored module size" value for each module down to their constituent pixel-scale modules. Additionally, the user may view any number of scale levels of the network, at any step in time. Typically, one would view only the pixel-scale level, after the final time step.

The program answers the question regarding whether or not X is inside any closed contour simply by checking whether any border pixel-scale modules have been activated. If so, then "X is outside," and otherwise "X is inside."

Each of the preliminary algorithms uses a network that is a type of tree. A module's position is entirely specified by its scale level and the horizontal position of the "center of mass" of its constituent pixels in the image. Suppose we are given a module m, which has size M. If M is odd, then m's horizontal position is simply the position of the pixel-scale module which is in the center of those covered by M. If M is even, then it's horizontal position is that of the SouthEast pixel-scale module, among the four in the center of the set covered by m. In other words, m covers the NorthWestern-most pixel-scale area which is still roughly centered below m. (This is the case also for the optimal algorithm, although it is not a type of tree, but a 3D mesh. See the explanation in Section 6.2.) We define a few additional terms to facilitate discussion of the implemented algorithms. Let m_1 and m_2 be modules.

> **NEWS** means "north, east, west and/or south," each being a horizontal direction within a given scale level.
>
> m_2 is a **sibling** of m_1 iff m_2 overlaps or borders m_1, and is a NEWS-neighbor.

m_2 is a **child** of m_1, and m_1 is a **parent** of m_2, iff m_1 covers m_2 and is in the scale level just above (just larger than) m_2's scale level. (Every module we discuss in this section — except "root" or "leaf" modules — will have 1 or 4 parent[s], depending on the network, and 4 children.)

m_2 is an **ancestor** of m_1 iff it is a parent of a parent ... of m_1.

m_2 is a **descendant** of m_1 iff it is a child of a child ... of m_1.

The general algorithm has two phases. The first phase is a preprocessing step, in which an image is presented to the pixel-scale modules. Each pixel-scale module inputs the two data fields given above. The values of **contour** are propagated up the network, updating all the higher-scale modules. The **contour** value that a module at scale level M stores for itself is the *OR* of the **contour** values it received from its $M \times M$ children. Again, 1 means it covers a contour, 0 means it covers only free space. This initialization is considered to occur during time step 0. During the second phase, at successive steps in time, each module uses similar criteria for assigning its **colored** value, while also observing Corollary 1. The entire process ends as soon as activation ceases.

A justification for the pre-activation phase follows from the widely-held notion that biological systems extract image contours via parallel channels at multiple scales. We assume that information about contours and the X are available to each scale level at the start. Actually, only in Algorithm #3 can activation proceed faster than one scale level at a time, whereas accurate contour information cannot. Thus, for all other algorithms, including the optimal versions in Sections 6 and 7, contour and color information may be propagated up from the pixel-scale modules continuously during the activation phase. We do the preprocessing with **contour** data because it is faster than propagating up the information each iteration, especially since it does not change with time (although see Section 9 about coloring inside moving contours).

5.1 Various Networks

In this section, we study and implement several varieties of a coloring algorithm based on image chunking. The variation between them consists entirely in the geometry of the networks. The algorithms have different sets of modules with different sets of connections. All connections are bi-directional, although messages are at certain times sent in only one direction. As in Section 3.1, contour and start information is propagated up during the preprocessing stage. From each module, the **contour** and **colored** token values are passed

up every vertical connection available to that module. This iterative stage ends when the highest-scale modules have been updated. Then, color information is propagated about iteratively. As described in Section 3.1, **colored** values pass up, out and then down during each cycle, taking every available path in that direction. These paths and the module configurations are given below.

Modules of Powers of Two Sizes, No Overlapping Positions (Quadtree Network) [Algorithm #1]

This is a simple quadtree network [J. Ullman 84]. Module sizes are powers of two. Modules within any given scale level do not overlap. Each module communicates with its 4 NEWS neighbors and its 4 children, as well as to its single parent. Connectivity is degree 9. (See Figure 6.)

Modules of All Sizes, No Overlapping Positions — Rejected

The modules in this network are of all possible scales (1,2,3,...,N pixels on a side), but no modules within a given scale level overlap each other. This was investigated but not implemented: the network was complicated (there were problems with principled inter-scale module-alignment, messy boundary structures, etc.) and offered few computational advantages (such as speed or hardware efficiency).

Modules of Powers of Two Sizes, All Positions — Rejected

This is a quadtree-like network, whose modules have sizes that are powers of two and have every possible position. Unfortunately, such a network has connectivity of $O(N^2)$, as shown in Figure 11. For fully-overlapping modules to be a worthwhile feature, each must be able to communicate directly with modules above and below it — hence, the high degree of connectivity.

Actually, a justifiable exception to this claim was found later during this work. By retiming, one can adapt this as an optimal algorithm which, in fact, uses a *compact* network. This is presented in Section 7.

Modules of All Sizes, All Positions (Complete Network) [Algorithm #2]

The modules in this network are of all possible sizes (1,2,3,...,N pixels on a side) and positions. Each module of a given size M communicates with its M siblings in each of the NEWS directions, as well as to its 4 children and 4 parents. Each module thereby communicates with *any* overlapping or bordering module of its level in ≤ 2 time steps. (We

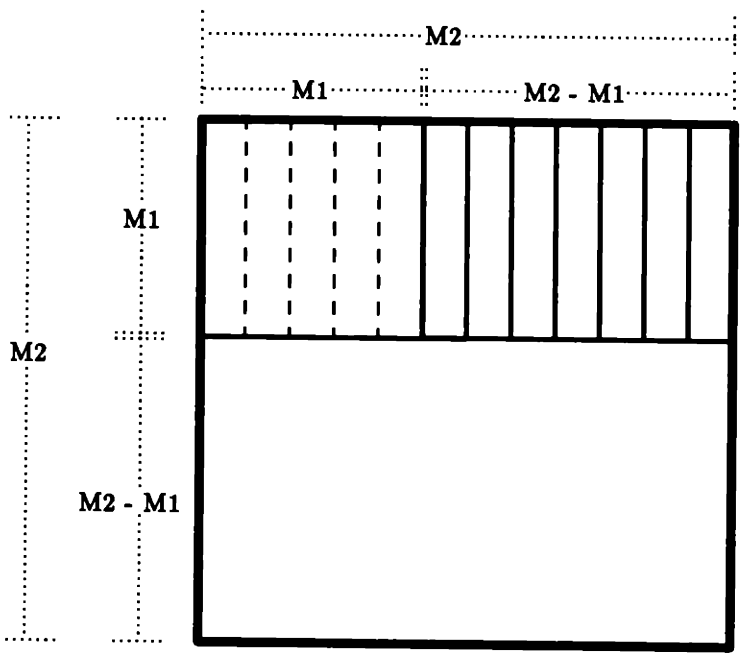


Figure 11: By inspection, any $M_2 \times M_2$ module covers exactly $(M_2 - M_1 + 1)^2$ distinct smaller $M_1 \times M_1$ modules where all amounts of overlap are allowed. So if $M_2 = M_1 + 1$, then M_2 covers 4 modules of size M_1.

Notice that a network with modules having all sizes that are powers of two and all positions would face a challenge: those modules at the highest level, which say cover 1024×1024 pixels, would have connections to $(1024 - 512 + 1)^2 = 263{,}169$ [i.e., $O(N^2)$] distinct 512×512 modules — too many connections!

would need $O(N^2)$ connections to do so in one time step, which is too many.) Connectivity is degree $O(N)$. (See Figure 12.)

Complete Network with Multiple Vertical Connections [Algorithm #3]

The final preliminary algorithm is based on the principle that fully-dynamic local scaling is a critical property for fast chunk-based coloring. This network includes all the modules and connections found in Algorithm #2. It has some additional connections, however. Each module also can communicate with every module above and below it that is centered above the same pixel above which it is centered. More precisely, as shown in Figure 13, a module at scale level M connects directly to the one module at every other level (i.e., levels $..., M-6, M-4, M-2, M+2, M+4, M+6, ...$) that shares its horizontal position; it *also* connects directly to the four most tightly-overlapping modules at every other level (i.e., levels $..., M-5, M-3, M-1, M+1, M+3, M+5, ...$ — such a quadruple of neighboring modules in scale level M', say, would just fit inside a $M'+1 \times M'+1$ square) whose centers vertically project onto the M-scale level and symmetrically (and most closely) surround the center of the said M-scale module.] Refer to Figure 13. Connectivity is degree $O(N)$.

5.2 Analysis of the Implementations

Evaluating the structures and performances of the above algorithms gives us clues for building better ones. Algorithms #1 and #2 each have strengths that are elicited by certain kinds of input. Algorithm #3 incorporates most of the advantages of the first two. It in turn serves to highlight the fundamentals needed to design the optimal coloring algorithm, presented in the next section.

Algorithm #1 has two primary advantages. First, it is comprised of modules with sizes that are powers of two and which do not overlap within scale levels, and therefore uses only $O(N^2)$ modules. In fact, it uses less than twice as many modules as there are pixels. Second, the coloring process is able to pass quickly from one scale level to another one of a vastly different size. This results in considerable speed for regions composed of parts of greatly different diameters. Output for an entire typical run is included in Figure 14A*i*.[10] The input for the network is an image which has an X inside a square-shaped closed contour, and a second X outside of the square in the upper lefthand corner. Algorithm #2 is

[10]In algorithms with powers of two-sized modules only, module sizes are given in terms of the power. So pixel-scale modules are represented as 0's, 2-scale modules as 1's, 4-scale modules as 2's, ..., and N-scale modules as $\log_2 N$.

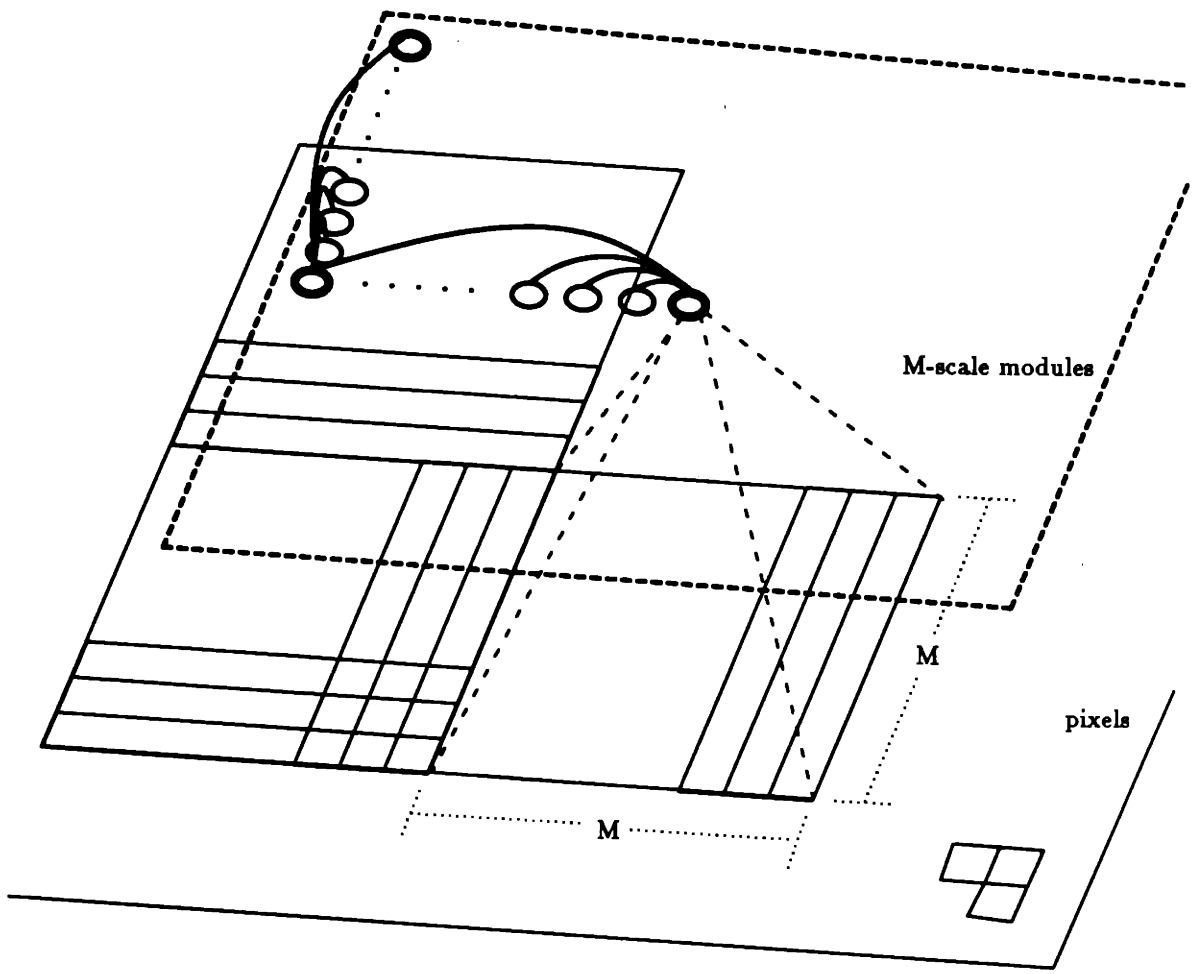


Figure 12: Network for Algorithm #2. Each M-scale module sends messages to its M siblings in each of the NEWS directions, as well as to its 4 children (not shown) and 4 parents (not shown). Each covers an $M \times M$ area of pixels.

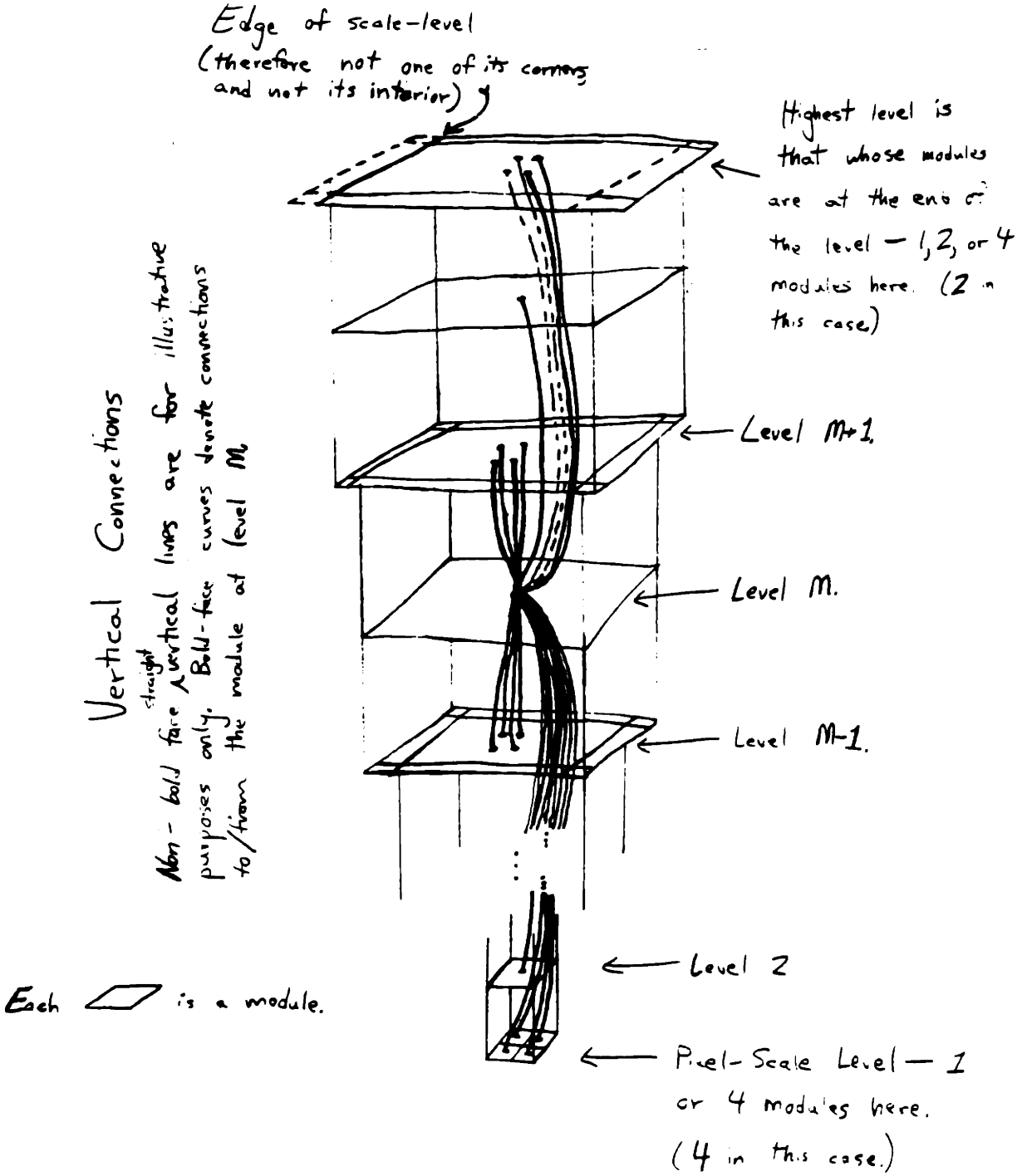


Figure 13: Connections for an arbitrary module in the network for Algorithm #3. Only the vertical ones are drawn — the horizontal connections are exactly like those for Algorithm #2, as shown in Figure 12. Each module is depicted in terms of the pixel area it covers.

hindered by the fact that activation ascends and descends scale levels relatively slowly. Sample results from it are given in Figure 14A*ii*.[11] Its sluggishness relative to Algorithm #1 can be seen by comparing Figures 14A*i* and *ii*. The part of the image that illustrates our point is the region outside the square, particularly where the region changes widths at the square's northeast and southwest corners. Comparing the coloring that occurs around these points for Algorithms #1 and #2, one sees that the coloring of new areas by the latter halts for a while (until activation has passed to a sufficiently low scale level), while coloring by the former proceeds steadily, without any perceptible interruption (because activation descends to a low scale level quickly).

On the other hand, we can see Algorithm #2's advantage *inside* the square: with modules at every position, coloring does not need to descend in scale much to reach the last remaining pixels, at least for reasonably smooth contours. Algorithm #1's coloring is required to descend all the way to the pixel level if none of its higher scale modules happen to be perfectly positioned such that they border the contour. This occurs in Figure 14A*i*. In summary, it would be nice if we could construct a network like that for Algorithm #2, yet which allows the coloring to adjust quickly in scale according to the situation.

Fortunately, we can incorporate single-step vertical activation into Algorithm #2, yielding Algorithm #3, while not increasing network connectivity to greater than $O(N)$. The two algorithms use identical sets of modules, which number $O(N^3)$. The improvement in performance of Algorithm #3 over #2 can be seen by comparing Figure 14A*iii* versus *ii*. Coloring by #3 is much faster than by #2 inside the square, because the coloring can quickly ascend scales. Coloring the region outside the square by #3 is slightly faster, but still suffers a long delay near the corners where the activation must descend scales. This is an important limitation of Algorithm #3: after activation passes downward from a large to a small scale level, it must propagate horizontally from the center of the area covered and already colored by the large module (often taking several time steps) before reaching modules not yet colored. Nevertheless, Algorithm #3 will generally be faster than #2 because its network is actually #2's network plus additional connections.

Algorithm #3 is generally stronger than #1 as well. The coloring of the region outside the square is one case where Algorithm #1 happens to be just as fast as #3. However, #1 is generally slower and has other weaknesses which lead us to choose Algorithm #3 as the best-performing square-chunk coloring algorithm that we have presented so far. We can see that Algorithm #3 is *faster* than #1 for many simple regions by examining

[11] In algorithms with modules of all sizes, module sizes are specified as $SIZE - 1$. So pixel-scale modules are represented as 0's, 2-scale modules as 1's, 3-scale modules as 2's, ..., and N-scale modules as $N - 1$

three pairs of sample runs. Compare the number of steps required by Algorithms #1 and #3 for coloring the insides of the identical squares in Figures 14A*i* and *iii*, and identical circles in Figures 14B*i* and *ii*. Indeed, one might expect #3 to be faster than #1, since it contains all of #1's modules and essentially most of its connections. Another problem with Algorithm #1 is that it is more sensitive to figure *scale* than is #3. This we see exemplified in Figures 14C*i* and *ii*, and Figures 14B*i* and *ii*. Algorithm #3's coloring time for circles of diameter 11 and 44 pixels is 3 and 4 steps, respectively, while Algorithm #1's time for the same input increases from 7 to 12 steps, respectively. Finally, Algorithm #1 can be very sensitive to the *position* of the figure in the image, whereas #3 is inherently invariant to this. Figures 14A*i* and D*i* demonstrate how coloring time can increase markedly with changing contour position for Algorithm #1, while Figures 14A*ii* and D*ii* show #3's position invariance.

Figure 14: This is the first of several pages of output from the preliminary algorithms. It consists of coloring results from Algorithms #1, #2 and #3. Figure 14Ai gives a complete sample run, which includes also the original image and a view of the network itself following coloring completion. The numbering system goes from 0, 1, 2, ..., 9, A, B, C, ..., where the number or letter indicates the step during which the pixel was colored. Boxed numbers indicate starting points for coloring.

```
 BOUNDED ACTIVATION USING QUADTREE NETWORK                    Figure 14Ai

 Current 32-by-32 image:                                      Algorithm #1
00000000000000000000000000000000:0
00000000000000000000000000000000:1
00000000000000000000000000000000:2
00000000000000000000000000000000:3
00000000000000000000000000000000:4
00000000000000000000000000000000:5
00000000000000000000000000000000:6
00000000000000000000000000000000:7
00000000000000000000000000000000:8
00000000000000000000000000000000:9
00000000000000000000000000000000:10
00000000000111111111111111111100:11
00000000000111111111111111111100:12
00000000000110000000000000001100:13
00000000000110000000000000001100:14
00000000000110000000000000001100:15
00000000000110000000000000001100:16
00000000000110000000000000001100:17
00000000000110000000000000001100:18
00000000000110000000000000001100:19
00000000000110000000000000001100:20
00000000000110000000000000001100:21
00000000000110000000000000001100:22
00000000000110000000000000001100:23
00000000000110000000000000001100:24
00000000000110000000000000001100:25
00000000000110000000000000001100:26
00000000000110000000000000001100:27
00000000000111111111111111111100:28
00000000000111111111111111111100:29
00000000000000000000000000000000:30
00000000000000000000000000000000:31
 ................................
 00000000001111111112222222233
 01234567890123456789012345678901

###### Activation completed.  ######

 Wait for final results below...
```

```
TIME-STEP = 18 -- status of modules (bit 0 = contour, bit 1 = colored)

22222222222222222222222222222222222222222222222222222222222111:0
22222222222222222222222222222222222222222222222222222222111110:1
22222222222222222222222222222222222222222222222222222111111121000:2
22222222222222222222222222222222222222222222222222211111112111000:3
22222222222222222222222222222222222222222222222222211122210000000:4
22222222222222222222222222222222222222222222211111111111222112210000000:5
22222222222222222222222222222222222222222211111111111222112210000000:6
22222222222222222222222222222222222222211222222212221111110000000:7
22222222222222222222222222222222222222211222222212000000000000000:8
22222222222222222222222222222222222222211222222212000000000000000:9
22222222222222222222222222222222222222211222222212000000000000000:10
22222222222111111111111111111122222221122222221200000000000000000:11
22222222221111111111111111111222222211222222212000000000000000000:12
22222222221122222222222222211222222211222222212000000000000000000:13
22222222221122222222222222211222222211111111112000000000000000000:14
22222222221122222222222222211222222222222222220000000000000000000:15
22222222221122222222222222211220000000000000000000000000000000000:16
22222222221122222222222222211220000000000000000000000000000000000:17
22222222221122222222222222211220000000000000000000000000000000000:18
22222222221122222222222222211220000000000000000000000000000000000:19
22222222221122222222222222211220000000000000000000000000000000000:20
22222222221122222222222222211220000000000000000000000000000000000:21
22222222221122222222222222211220000000000000000000000000000000000:22
22222222221122222222222222211220000000000000000000000000000000000:23
22222222221122222222222222211220000000000000000000000000000000000:24
22222222221122222222222222211220000000000000000000000000000000000:25
22222222221122222222222222211220000000000000000000000000000000000:26
22222222221122222222222222211220000000000000000000000000000000000:27
22222222221111111111111111111220000000000000000000000000000000000:28
22222222221111111111111111111220000000000000000000000000000000000:29
22222222222222222222222222222220000000000000000000000000000000000:30
22222222222222222222222222222220000000000000000000000000000000000:31

Projected minimum TIME-STEP when colored
11111111222222233333333444444440:0
11111111222222223333333344444444:1
11111111222222223333333344444444:2
11111111222222223333333344444444:3
11111111222222223333333344444444:4
11111111222222223333333344444444:5
11111111222222223333333344444444:6
11111111222222223333333344444444:7
00000000334455556666666677777777:8
00000000334455556666666677777777:9
00000000335677778888888899999988:10
0000000033#####################88:11
0000000033#####################99:12
0000000033##CBBAAAA99998888##99:13
0000000033##B99888877776666##AA:14
0000000033##B99888877776666##AA:15
1111111144##A8855555554444##BB:16
1111111144##A8855555554444##BB:17
1111111144##A8855555554444##CC:18
1111111144##A8855555554444##CC:19
1111111144##977555544443333##DD:20
1111111144##977555544443333##DD:21
1111111144##977555544332233##EE:22
1111111144##977555544321233##EE:23
2222222255##866444433210122##FF:24
2222222255##866444433221122##FF:25
2222222255##866444433332222##GG:26
2222222255##866444433332222##GG:27
2222222255####################HH:28
2222222255####################HH:29
22222222556678899AABBCCDDEEFFGG:30
22222222556678899AABBCCDDEEFFGG:31
```

Projected maximum LEVEL with colored module

```
33333333333333333333333333333333:0
33333333333333333333333333333333:1
33333333333333333333333333333333:2
33333333333333333333333333333333:3
33333333333333333333333333333333:4
33333333333333333333333333333333:5
33333333333333333333333333333333:6
33333333333333333333333333333333:7
33333333111111111111111111111111:8
33333333111111111111111111111111:9
33333333100000000000000000000011:10
33333333110####################11:11
33333333110####################11:12
33333333110##0000000000000000##11:13
33333333110##0111111111111111##11:14
33333333110##0111111111111111##11:15
33333333110##0113333333322222##11:16
33333333110##0113333333322222##11:17
33333333110##0113333333322222##11:18
33333333110##0113333333322222##11:19
33333333110##0113333333322222##11:20
33333333110##0113333333322222##11:21
33333333110##0113333333322222##11:22
33333333110##0113333333322222##11:23
33333333110##0112222222222222##11:24
33333333110##0112222222222222##11:25
33333333110##0112222222222222##11:26
33333333110##0112222222222222##11:27
33333333110####################11:28
33333333110####################11:29
33333333111111111111111111111111:30
33333333111111111111111111111111:31
```

The X is *inside* a closed contour.

Figure 14A*ii*

Algorithm #2

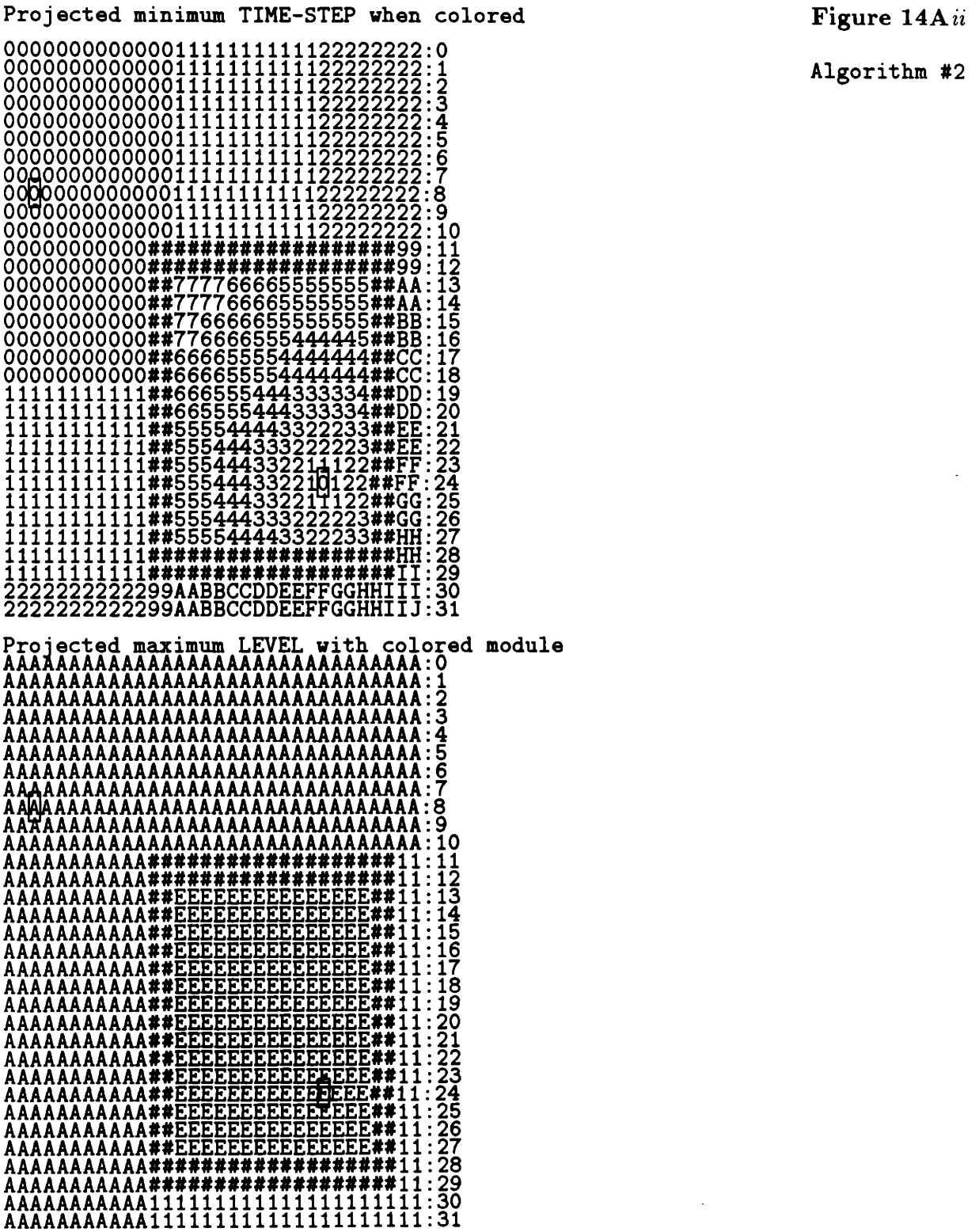
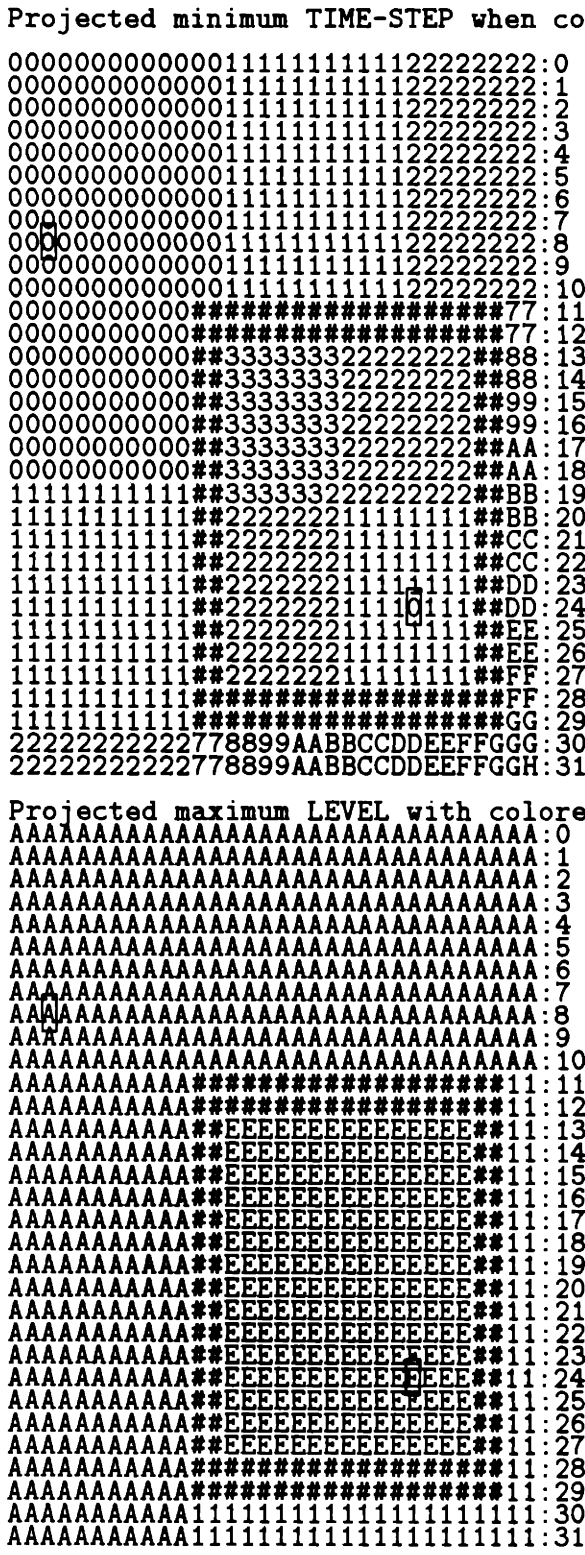


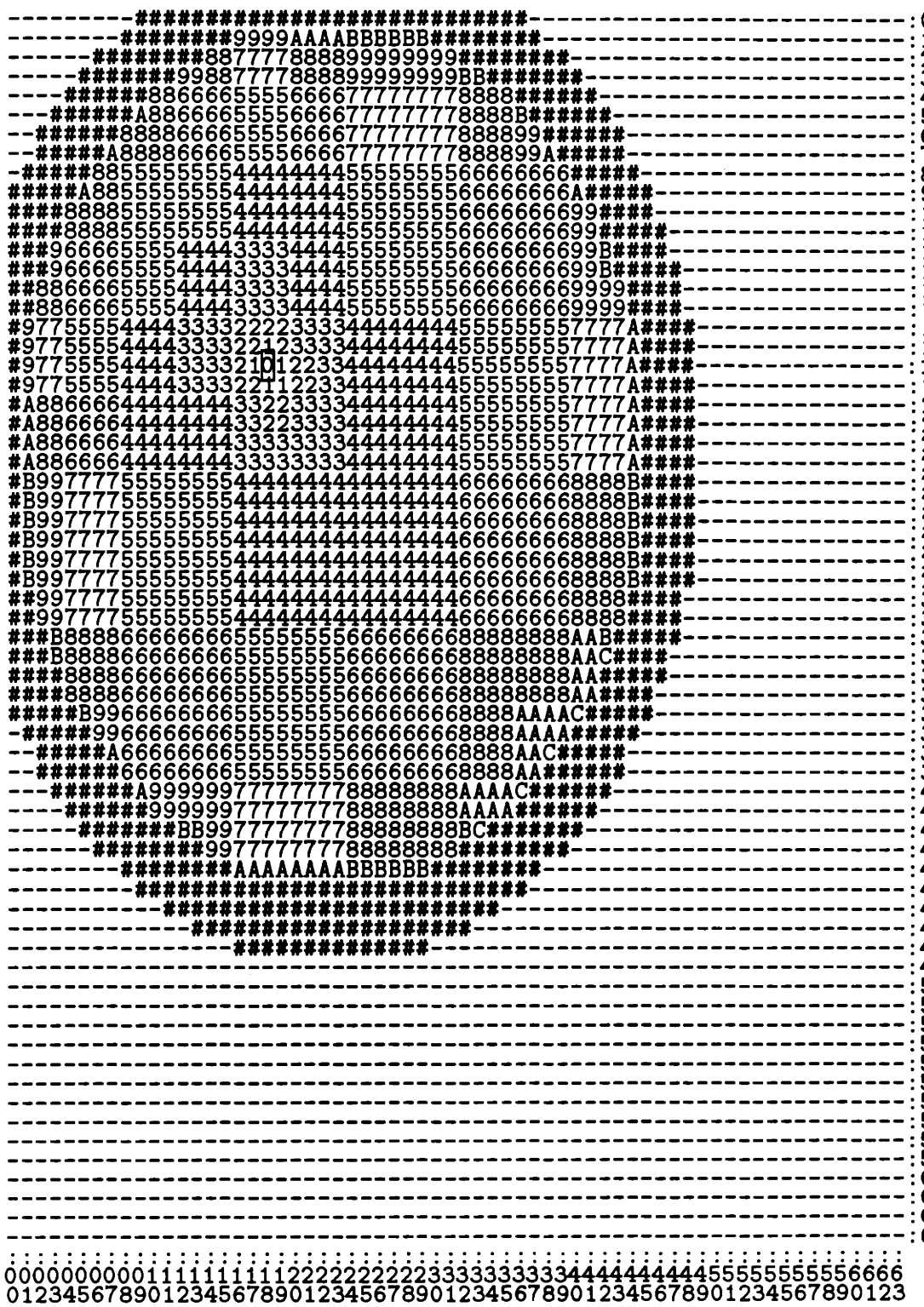
Figure 14A*iii*

Algorithm #3

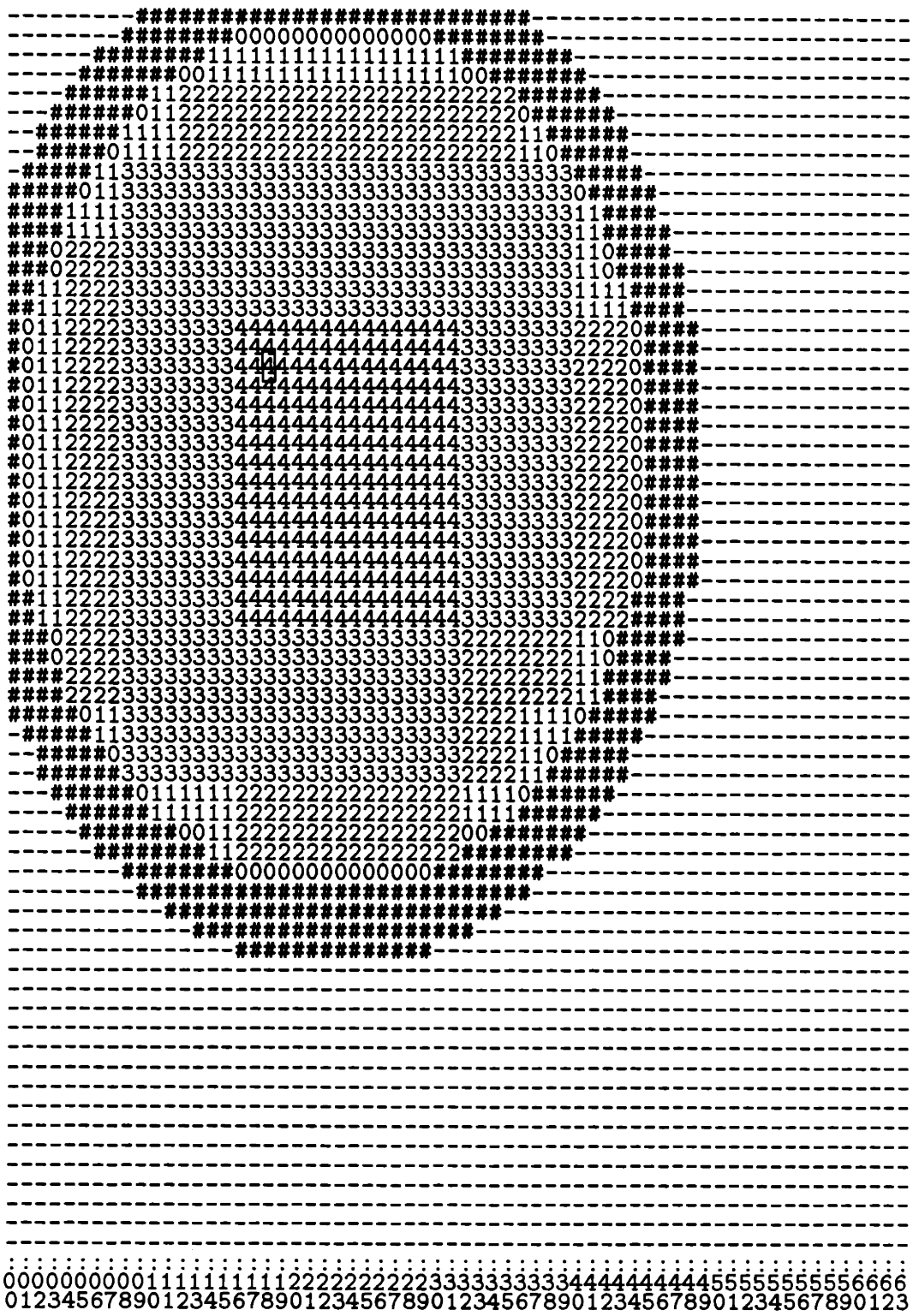


Projected minimum TIME-STEP when colored

Figure 14B*i*



Projected maximum LEVEL with colored module



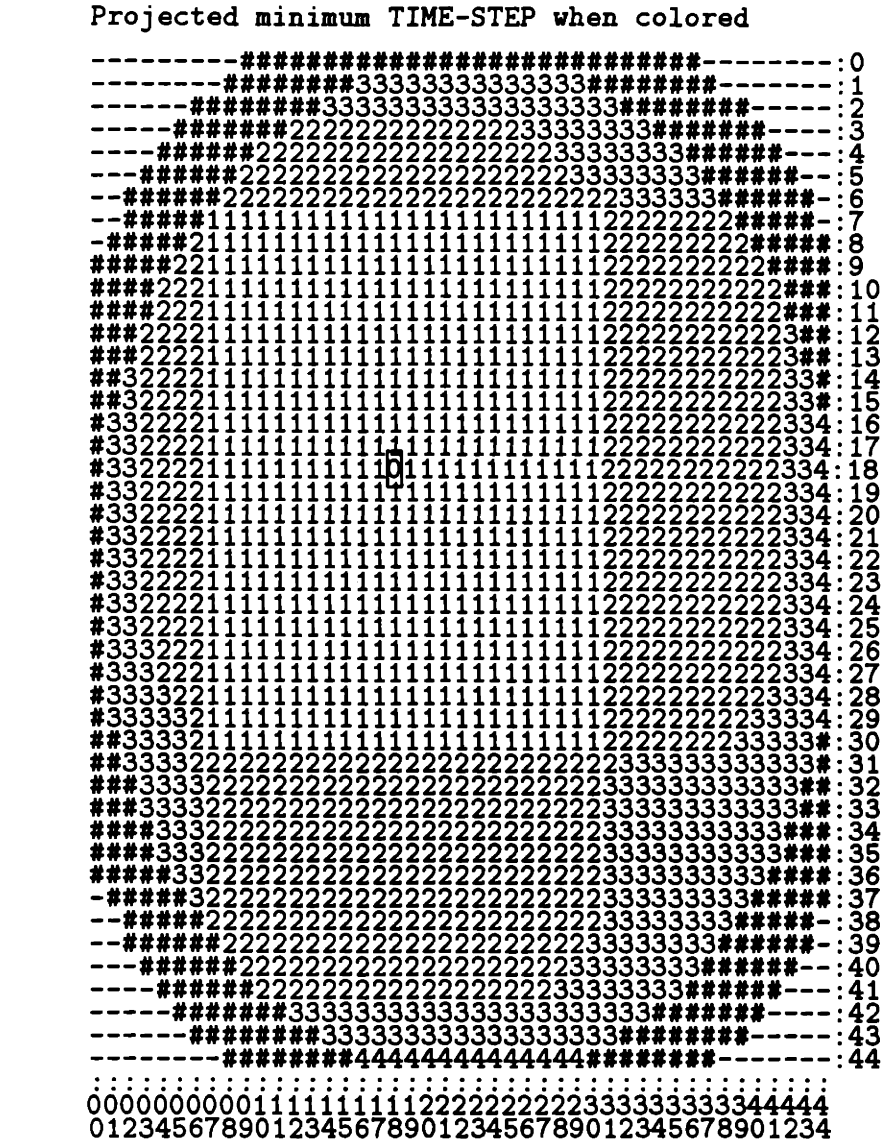
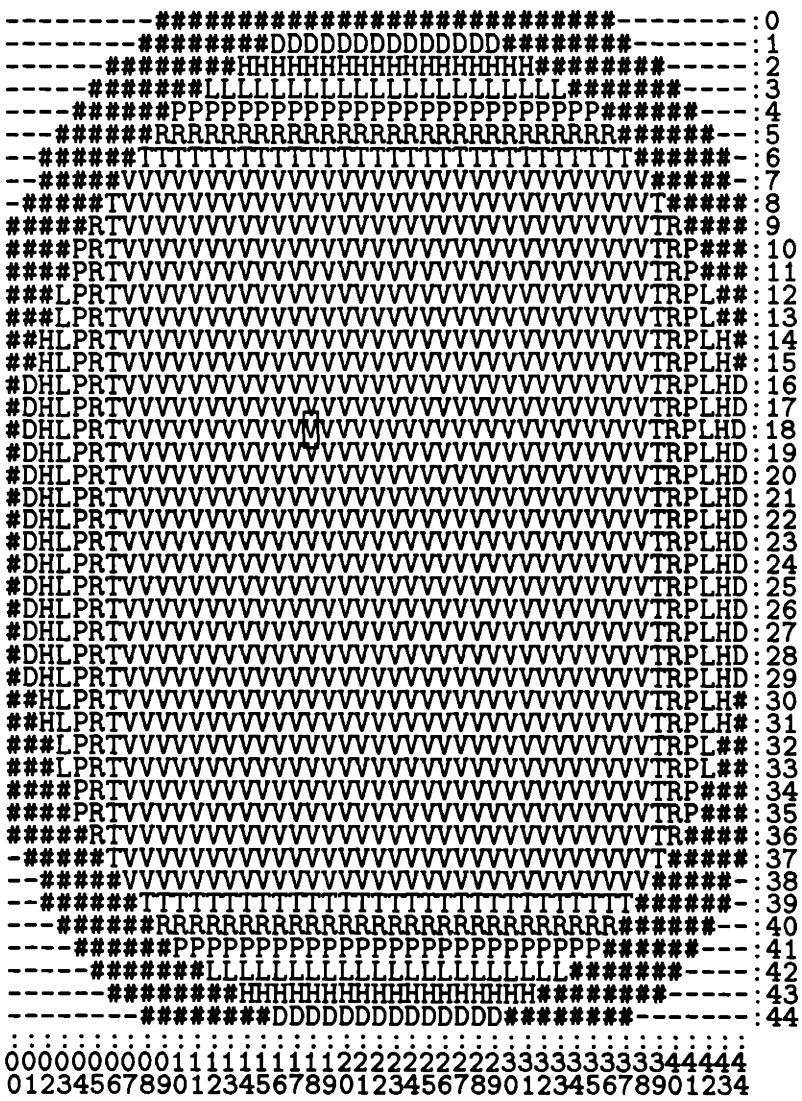


Figure 14B*ii*

Algorithm #3

Projected maximum LEVEL with colored module



Projected minimum TIME-STEP when colored

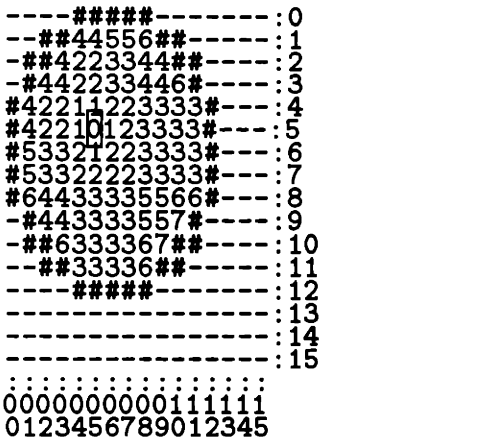
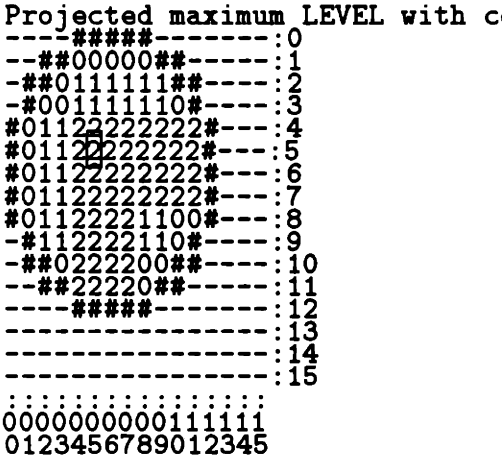


Figure 14C*i*

Algorithm #1

Projected maximum LEVEL with colored module



Projected minimum TIME-STEP when colored

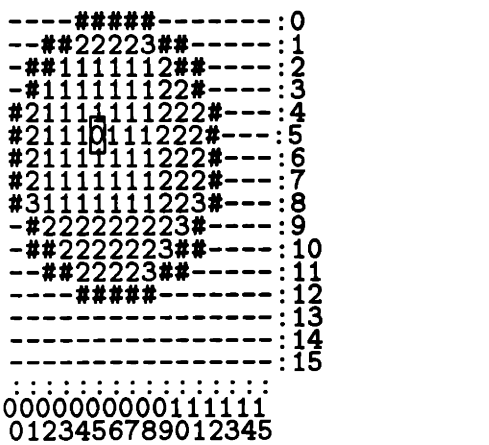


Figure 14C*ii*

Algorithm #3

Projected maximum LEVEL with colored module

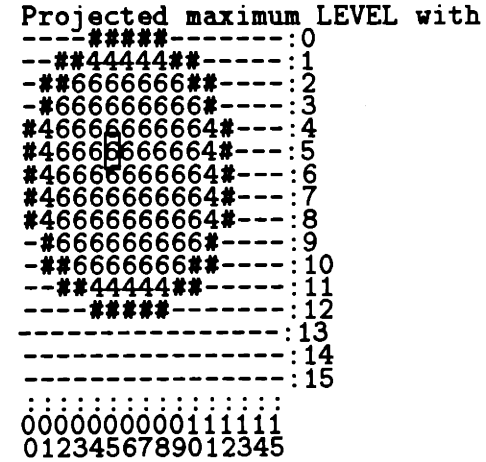


Figure 14D*i*

Algorithm #1

```
Projected minimum TIME-STEP when colored
--------------------------------:0
--------------------------------:1
--------------------------------:2
--------------------------------:3
--------------------------------:4
--------------------------------:5
--------------------------------:6
--------------------------------:7
--------------------------------:8
--------------------------------:9
--------------------------------:10
--------------------------------:11
--------------------------------:12
--------------------------------:13
--------------##################:14
--------------##################:15
--------------##4444444444444444:16
--------------##4444444444444444:17
--------------##4444444444444444:18
--------------##4444444444444444:19
--------------##4444444444444444:20
--------------##4444444444444444:21
--------------##4444444444444444:22
--------------##4444444444444444:23
--------------##4444444433333333:24
--------------##4444444433333333:25
--------------##4444444433332233:26
--------------##4444444433321233:27
--------------##4444444433210122:28
--------------##4444444433221122:29
--------------##4444444433332222:30
--------------##4444444433332222:31

Projected maximum LEVEL with colored module
--------------------------------:0
--------------------------------:1
--------------------------------:2
--------------------------------:3
--------------------------------:4
--------------------------------:5
--------------------------------:6
--------------------------------:7
--------------------------------:8
--------------------------------:9
--------------------------------:10
--------------------------------:11
--------------------------------:12
--------------------------------:13
--------------##################:14
--------------##################:15
--------------##4444444444444444:16
--------------##4444444444444444:17
--------------##4444444444444444:18
--------------##4444444444444444:19
--------------##4444444444444444:20
--------------##4444444444444444:21
--------------##4444444444444444:22
--------------##4444444444444444:23
--------------##4444444444444444:24
--------------##4444444444444444:25
--------------##4444444444444444:26
--------------##4444444444444444:27
--------------##4444444444444444:28
--------------##4444444444444444:29
--------------##4444444444444444:30
--------------##4444444444444444:31
```

Figure 14D*ii*

Algorithm #3

```
Projected minimum TIME-STEP when colored
--------------------------------:0
--------------------------------:1
--------------------------------:2
--------------------------------:3
--------------------------------:4
--------------------------------:5
--------------------------------:6
--------------------------------:7
--------------------------------:8
--------------------------------:9
--------------------------------:10
--------------------------------:11
--------------------------------:12
--------------------------------:13
--------------##################:14
--------------##################:15
--------------##3333333322222222:16
--------------##3333333322222222:17
--------------##3333333322222222:18
--------------##3333333322222222:19
--------------##3333333322222222:20
--------------##3333333322222222:21
--------------##3333333322222222:22
--------------##3333333222222222:23
--------------##2222222211111111:24
--------------##2222222211111111:25
--------------##2222222211111111:26
--------------##2222222211111111:27
--------------##2222222211110111:28
--------------##2222222211111111:29
--------------##2222222211111111:30
--------------##2222222211111111:31

Projected maximum LEVEL with colored module
--------------------------------:0
--------------------------------:1
--------------------------------:2
--------------------------------:3
--------------------------------:4
--------------------------------:5
--------------------------------:6
--------------------------------:7
--------------------------------:8
--------------------------------:9
--------------------------------:10
--------------------------------:11
--------------------------------:12
--------------------------------:13
--------------##################:14
--------------##################:15
--------------##FFFFFFFFFFFFFFFF:16
--------------##FFFFFFFFFFFFFFFF:17
--------------##FFFFFFFFFFFFFFFF:18
--------------##FFFFFFFFFFFFFFFF:19
--------------##FFFFFFFFFFFFFFFF:20
--------------##FFFFFFFFFFFFFFFF:21
--------------##FFFFFFFFFFFFFFFF:22
--------------##FFFFFFFFFFFFFFFF:23
--------------##FFFFFFFFFFFFFFFF:24
--------------##FFFFFFFFFFFFFFFF:25
--------------##FFFFFFFFFFFFFFFF:26
--------------##FFFFFFFFFFFFFFFF:27
--------------##FFFFFFFFFFFFFFFF:28
--------------##FFFFFFFFFFFFFFFF:29
--------------##FFFFFFFFFFFFFFFF:30
--------------##FFFFFFFFFFFFFFFF:31
```

One important observation we can make from comparing Algorithms #1 and #3 is that it is useful for a coloring network to have complete sets of overlapping modules within its scale levels. First, of course, it renders the network invariant to figure position. Second, it enables us to color a region completely in many cases without descending to fine-scale modules. Each module can activate all of its valid siblings (those which do not cover a contour) in one or two steps; groups of sibling modules having relatively large scales frequently are sufficient to cover all free-space pixels that are near particular segments of contours. Thus we can often color entire regions, including areas near contours, very quickly. (We demonstrate this for real edge data in the remaining sections.) Furthermore, this suggests that we may not need every fine scale level, and likewise could do away with some coarser scale levels as well. We are assured that the modules in the remaining scale levels (including the pixel-scale modules, which we leave intact) will completely color the region with a graceful (slight) degradation in performance. We shall further discuss reducing the $O(N^3)$ cardinality requirement in the next two sections.

Algorithm #3 seems to be a good model. Like the Rectangular Model, it colors simple closed figures in only a few steps. Shafrir's version of the Rectangular Model appears to be slightly faster than #3 for some shapes (although not for bent, elongated figures, such as snakes); however, both are fundamentally $O(N^2)$ in time complexity. In the next section, we investigate a way to improve the average performance of Algorithm #3 in a way that enables single-step downward propagation to finer scale modules, especially those located near the periphery of already-colored and yet-to-be-colored modules. Furthermore, there are some other desirable properties that such area-based multi-scale level chunking schemes possess. These include various flexibilities, a few of which we utilize for some useful applications in Sections 8 and 9. Also, as mentioned before, the algorithm constructs a number of planar representations of the image that can be used by other operations. The components of these representations are entirely connected via horizontal connections between modules; these sets (scale levels) in turn are linked via vertical connections. Neither Lim's algorithm nor Shafrir's Rectangular Model generates such a readily-accessible group of descriptions.

6 Optimal Coloring with Parallel Square Modules

We can apply some of the principles that we have investigated before to design the optimal coloring algorithm for parallel square modules, while also preserving $O(N)$ connectivity. The optimal coloring network using parallel square chunks is one which is the fastest

possible, to within a constant factor, which does not violate Corollary 1. It would be sufficient for it to have (1) parallel square modules of every possible size and position, and (2) connections that would allow every unactivated module which does not cover a contour, and which is proximate to an activated module, to activate itself in one time-step (or some constant number of steps). To construct such a network (or the complete Rectangular Model, for that matter) using direct connections would require $O(N^3)$ connectivity. [12] Fortunately, this proposed network can be retimed so that all nodes are reduced to degree $O(N)$ while communication is lagged by a factor of 7.

6.1 Important Natural Constraints

From the preliminary implementations, we gained some important additional constraints that impinge in our design of an final algorithm. First, activation must ascend and descend from fine to coarse scale levels quickly. Second, we observed that overlapping modules are a powerful means of coloring. In fact, as we will see in the next section, it is a valid substitute for numerous scale levels. Densely-overlapping higher-scale modules can color inside of relatively smooth contours in just a few iterations. Where they cannot quite reach to the contour, we would then like a quick descent to smaller-scale modules to finish the operation. These results actually support the psychophysical objectives we have in mind. The full-size models (Algorithm #3 and especially the one below) yield performance which is invariant to figure position, as well as near invariance to orientation due to the module's shape. Of particular interest, they exhibit approximate scale invariance for simple shapes. Being able to conduct the coloring operation at all scales at the same time is a powerful way to achieve this goal.

Given our earlier theory for coloring, we intend to develop an algorithm which embodies as well the biological constraints discussed above. The principle of indirection discussed in the Appendix, and our restrictions and assumptions for our chosen representation, further serve to constrain our work below.

6.2 Algorithm

The overall process of the optimal coloring algorithm is quite simple. At each step, we would like to color every square-shaped region of pixel-scale modules that is proximate to a

[12] For example, the single module that comprises the highest scale level (level N) must communicate with all $\sum_{i=1}^{N-1} i^2$ modules on scale levels $1, 2, ..., N-1$; $\Rightarrow O(N^3)$ connectivity.

colored pixel-scale module. To do this, we first color every module that satisfies Corollary 1 at each of the scale levels. That is, every module at scale level M ($M = 1, ..., N$) which is proximate to a colored pixel-scale module, but does not cover a contour, is colored. Then we update the pixel-scale modules, by activating every pixel-scale (scale level 1) module which is covered by a colored higher-scale module. (It is guaranteed that neither will cover a contour.)

To enable these upward and downward propagation processes, the following network configuration is used. We have a 3D NEWS grid of $N \times N \times N$ modules, and axes z, y and x, whose values increase in the upward, Southern and Eastern directions, respectively. [13] Recall that the earlier algorithms were imbedded in trees, whereas now we employ a 3D mesh. Along the z dimension are scale levels $z = 1, ..., N$, where each scale level consists of $N \times N$ modules. Each pixel-scale module is at scale level 1, and has position $(1, y, x)$. Each module at some higher level M, $M \in [2, ..., N]$, has position (M, y, x), and covers $M \times M$ pixel-scale modules. In addition, each pixel-scale module has exactly one **superior** module in each higher scale level, which has x and y coordinates that are identical to its own x and y coordinates. Conversely, each higher scale module has a single **subordinate** pixel-scale module that has x and y coordinates that are identical to its own. The network configuration is illustrated in Figure 15 below.

The many-to-many mapping between pixel- and higher-scale modules is a key factor that makes the algorithm optimal. Simply stated, each higher-scale module at level M covers a square $M \times M$ region of pixel-scale modules, and its single subordinate pixel-scale module is positioned roughly in the center of that region. Precisely stated,

> any higher-scale module m at position (M, y, x) covers a set of pixel-scale modules $\{(0, b, a) | b \in [y - \lfloor \frac{M}{2} \rfloor, ..., y, ..., y + \lfloor \frac{M-1}{2} \rfloor], a \in [x - \lfloor \frac{M}{2} \rfloor, ..., x, ..., x + \lfloor \frac{M-1}{2} \rfloor]\}$; m has exactly one subordinate module p, which has position $(0, y, x)$.

So p is positioned such that the $M \times M$ region spanned by $\lfloor \frac{M}{2} \rfloor$ pixel-scale modules to the North and to the West of p, and $\lfloor \frac{M-1}{2} \rfloor$ pixel-scale modules to the South and to the East of p are properly covered by m. We see that a neighbor of m, whose position would be that of m shifted by 1 in either the y or x direction, would have a subordinate pixel-scale module and a covered pixel-scale region whose positions correspond to those of m, but shifted by 1 in the same direction. In addition, when M is an odd number, p is perfectly centered in the region covered by m. When M is even, there are four pixels in the $M \times M$ pixel-scale

[13] We use the terms "up," "down," "North," "South," "East," and "West" only for the sake of convenience; the data structure is completely general.

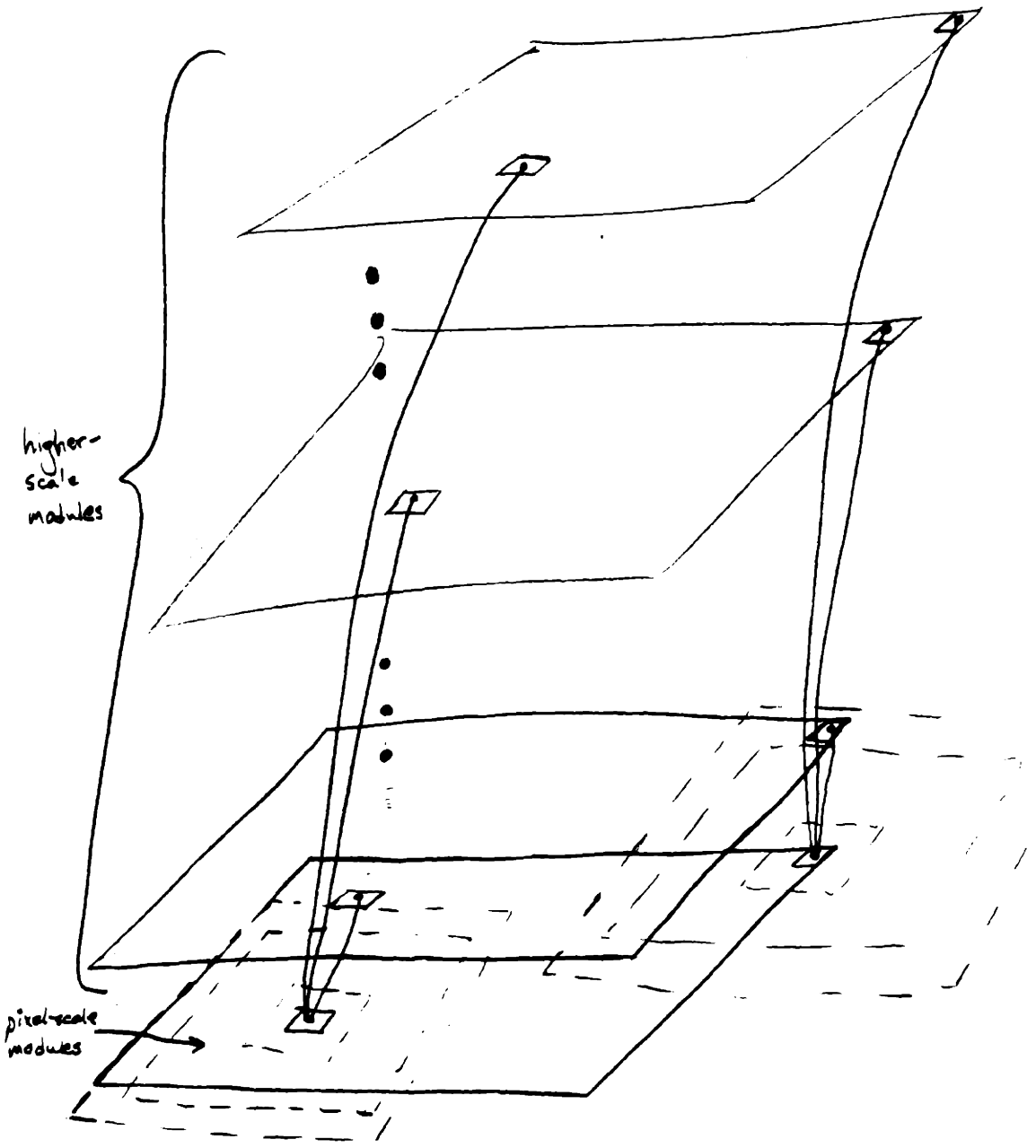


Figure 15: The network configuration for the optimal algorithm. Here the modules are portrayed as squares on a grid, with their vertical connections given as well. The areas covered by the higher-scale modules are sketched in dashed lines. Notice that some of the higher-scale modules only partly cover the pixels.

region covered by m that are equally close to the center of the region. Of the four, m's subordinate pixel-scale module p shall be the one with the SouthEast position (i.e., with the larger y and x values). In addition,

> any pixel-scale module p at position $(0, y, x)$ is covered by a set of M-scale modules $\{(M, b, a) | b \in [y - \lfloor \frac{M-1}{2} \rfloor, ..., y, ..., y + \lfloor \frac{M}{2} \rfloor], a \in [x - \lfloor \frac{M-1}{2} \rfloor, ..., x, ..., x + \lfloor \frac{M}{2} \rfloor]\}$;
> p has exactly one superior module m, which has position (M, y, x).

Finally, we note that some higher-scale modules, having y or x values near 1 or N, do not cover a set of pixel-scale modules that belong entirely to level 1, but which project somewhat beyond the periphery. However, every image element is mapped to a module in level 1, and each of these has superior modules at all higher levels.

Now that the module positions and a few more terms have been defined, it will be easy to describe the connections between the modules. First, each pixel-scale module will have one connection to each of its superior modules. Next, every module, including pixel-scale modules will have one or more horizontal connections to its one or more closest siblings. To be precise, a module m in scale level M (for all values of $M = 1, ..., N$) has individual connections to its $\lfloor \frac{M}{2} \rfloor$ North, West, South and East siblings. Every connection is bi-directional; that is, we need to be able to pass information between two connected modules in both directions during the same time step. Each pixel-scale module sends message(s) in the x- and y-direction, or upward, depending upon from which of these directions it received the last message(s). All other modules send message(s) in the x-direction, y-direction, or downward, depending upon from which of these directions it received the last message(s). The network's local connections are illustrated below in Figure 16. Connectivity degree is $O(N)$.

It is thereby possible for any given pixel-scale module to send a message to every proximate higher-scale module, in four steps. Similarly, any given higher-scale module can send a message to precisely all pixel-scale modules which it covers, in three steps. We shall examine both of these assertions below. First, we present the three main components of the basic algorithm, each in a sequence decomposed into time steps. Then we show the results of applying each process, thereby proving our assertions. Later, we state the algorithm as a whole, make our optimality claim, and in the proof that follows, show that effecting these results in a certain cycle makes for a coloring algorithm which is optimal.

We begin with a simple process. Any given pixel-scale module p can pass a message $\mathcal{M}$ to its 4 neighbors, by doing the following:

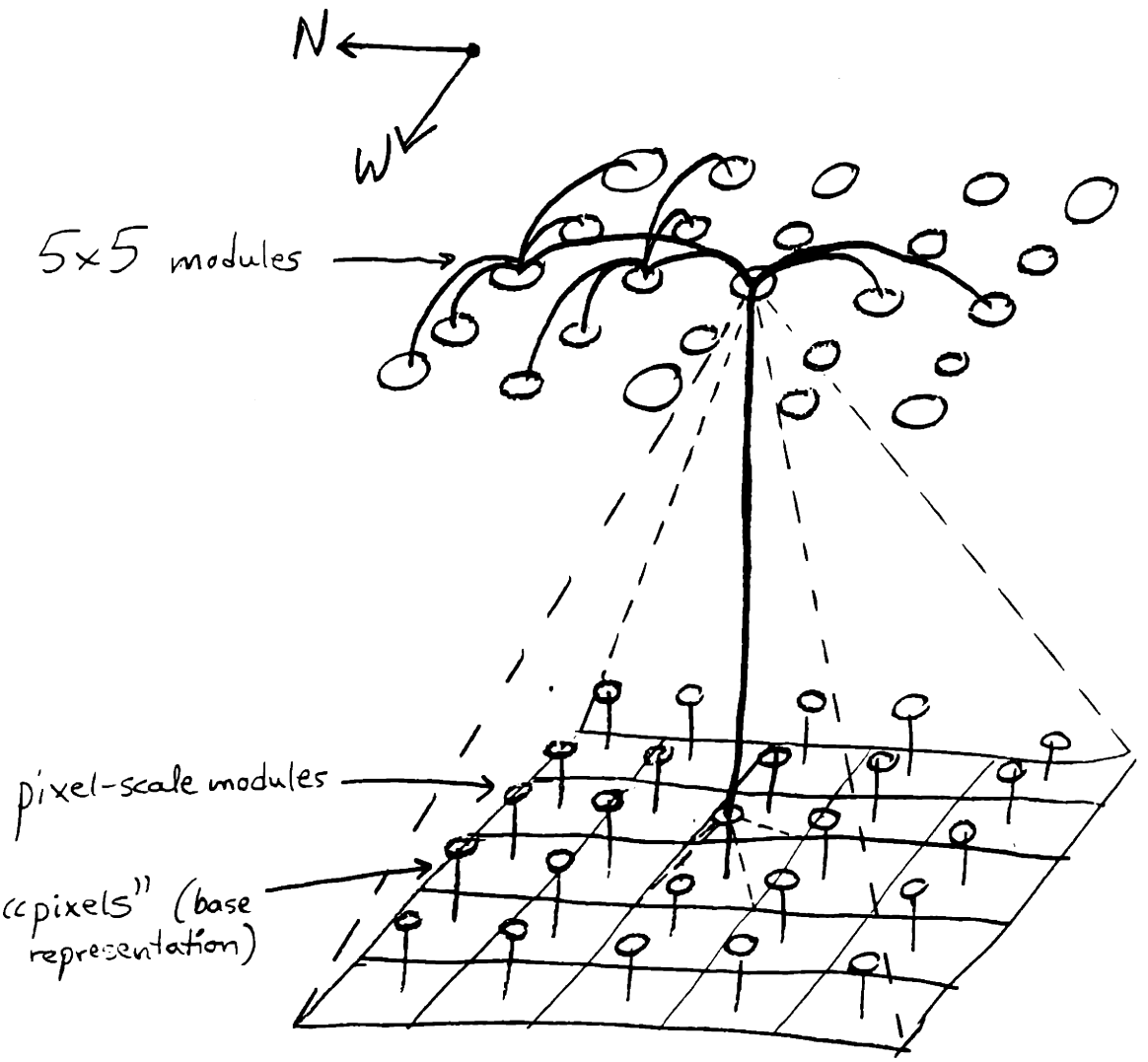


Figure 16: During the Upward Propagation process, each pixel-scale module sends a message $\mathcal{M}$ to its superior module at scale-level 5. Then this 5-scale module passes $\mathcal{M}$ to its two North and two South siblings, all of whom then pass $\mathcal{M}$ to their two West and two East siblings. The pixel-scale module can thereby send $\mathcal{M}$ to each of the $5^2 = 25$ 5-scale modules that cover it, within 3 steps, and via *a unique path*. This minimizes the number of connections needed. The Downward Propagation is essentially this in reverse.

Pixel-Level Propagation

i. p sends $\mathcal{M}$ to the closest pixel-scale module in each of the North, South, West and East directions.

Result of Pixel-Level Propagation: Coming from p, $\mathcal{M}$ arrives at each of p's 4 NEWS neighbors in the network in 1 time step. That this is so can be seen by inspection.

Next, any pixel-scale module p can pass a message $\mathcal{M}$ to precisely the M^2 modules in scale level M that cover it, and would do so in the following way:

Upward Propagation

i. p sends $\mathcal{M}$ to each of its superior modules, one at each scale.

ii. Upon sending up or receiving a message from below, module m passes $\mathcal{M}$ to its $\lfloor \frac{M-1}{2} \rfloor$ North siblings and to its $\lfloor \frac{M}{2} \rfloor$ South siblings.

iii. After sending or receiving a message to/from the North or South, these M M-scale siblings (including m) pass $\mathcal{M}$ to their $\lfloor \frac{M-1}{2} \rfloor$ West siblings and to their $\lfloor \frac{M}{2} \rfloor$ East siblings.

Result of Upward Propagation: Coming from p, $\mathcal{M}$ arrives at each of the M^2 M-scale modules in the network that cover p within 3 time steps. (Generally, we will send a message from some pixel-scale module p to every module in *all* higher-scale levels that cover p.)

To see that this is so, suppose that a message $\mathcal{M}$ is sent up from p, where p has coordinates $(0, y, x)$. Let us examine the results in one arbitrary level, M. At the completion of step **i.**, module (M, y, x) has received $\mathcal{M}$. After step **ii.**, modules $\{(M, b, x) | b \in [y - \lfloor \frac{M-1}{2} \rfloor, ..., y, ..., y + \lfloor \frac{M}{2} \rfloor]\}$ have each received $\mathcal{M}$. (Recall our convention that South and East are each positive directions.) Then, after step **iii.**, modules $\{(M, b, a) | b \in [y - \lfloor \frac{M-1}{2} \rfloor, ..., y, ..., y + \lfloor \frac{M}{2} \rfloor], a \in [x - \lfloor \frac{M-1}{2} \rfloor, ..., x, ..., x + \lfloor \frac{M}{2} \rfloor]\}$ all have received $\mathcal{M}$. From the description of the network structure given above, we know that these are exactly the coordinates of the set of M-scale modules which cover a module at position $(0, y, x)$, namely, p. Hence, every module that covers p receives $\mathcal{M}$.

Result of Pixel-Level and then Upward Propagation: Coming from p, $\mathcal{M}$ arrives at each of the $(M+2)^2 - 4$ M-scale modules in the network that are proximate to p within 3 time steps. (Generally, we will send a message from some pixel-scale module p to every module in *all* higher-scale levels which are proximate to p.)

To prove this, we shall examine this conjunctive operation initially after the Pixel-Level Propagation has occurred but before the Upward Propagation. At this stage, we apply the Upward Propagation process separately to each of the 5 modules which would possess message $\mathcal{M}$ at the conclusion of the Pixel-Level Propagation process. Since messages are coalesced in a "non-destructive" way if they ever collide (typically via an INCLUSIVE-OR), each instance of the Propagation process works independently at the local level. We can thus look at the union of the results of these 5 instances of Upward Propagation when considering the effect of Pixel-Level and then Upward Propagation from p. This effect is illustrated in Figure 17.

The set of all modules that are proximate to p consists of modules that cover, overlap or border p. First of all, since p is a module of size 1, there exist no modules that *overlap* it. (Recall from our definition of "overlap" that "m_1 overlaps m_2" $\Longleftrightarrow$ "m_2 overlaps m_1;" p cannot overlap any module because it has unit-size.) Second, Upward Propagation from p ensures that all modules which *cover* p receive $\mathcal{M}$, as we have just seen above. What remains is to show that all of the M-scale modules that *border* p receive $\mathcal{M}$. If a higher-scale module borders p, then, again from one of our preliminary definitions, it must cover point p_N, p_S, p_W or p_E. Fortunately, Upward Propagation from each of these 4 pixel-scale modules ensures that all of the M-scale modules that cover them, including those which border p, receive $\mathcal{M}$.

In a reverse manner, any given M-scale module m can pass a message $\mathcal{M}$ to precisely the M^2 pixel-scale modules which it covers, and would do so in the following way:

Downward Propagation

i. Module m sends $\mathcal{M}$ to its $\lfloor \frac{M}{2} \rfloor$ North siblings and to its $\lfloor \frac{M-1}{2} \rfloor$ South siblings.

ii. After sending or receiving a message to/from the North or South, these M M-scale siblings (including m) pass $\mathcal{M}$ to their $\lfloor \frac{M}{2} \rfloor$ West siblings and to their $\lfloor \frac{M-1}{2} \rfloor$ East siblings.

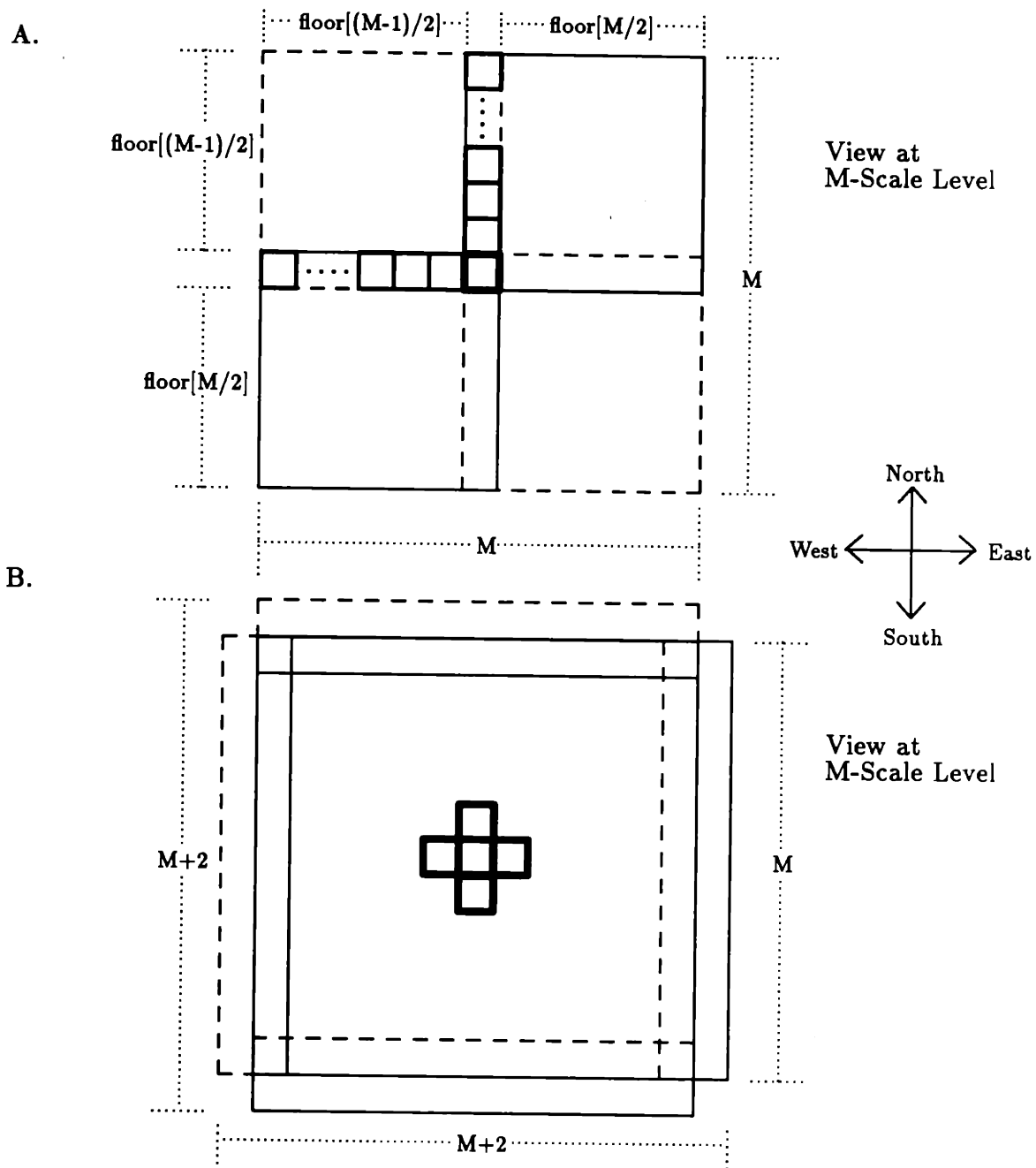


Figure 17: This diagram shows why there are $(M+2)^2 - 4$ M-scale modules that have received $\mathcal{M}$ after a Pixel-Level and then Upward Propagation. Part A. shows the result of an Upward Propagation: one pixel-scale module (in bold outline) sends $\mathcal{M}$ to all modules in each scale level M within a distance of $\lfloor \frac{M-1}{2} \rfloor$ to the North or West or within a distance of $\lfloor \frac{M}{2} \rfloor$ to the South or East. It thereby reaches M^2 modules, including itself. If this is preceded by a Pixel-Level Propagation, illustrated in part B., five pixel-scale modules (each in bold outline) send $\mathcal{M}$ to five over- lapping sets of M^2 modules each. In all, a total of $(M+2)^2 - 4$ M-scale modules receive $\mathcal{M}$.

iii. After sending or receiving a message to/from the East or West, each of these M^2 M-scale modules (including m) pass $\mathcal{M}$ to its respective subordinate pixel-scale module p.

Result of Downward Propagation: Coming from m, $\mathcal{M}$ arrives at the M^2 pixel-scale modules in the network which are covered by m within 3 time steps.

To see that this is so, suppose that a message $\mathcal{M}$ is sent down from m, where m has coordinates (M, y, x). Let us examine the results in level 1. At the completion of step **i.**, modules $\{(M, b, x) | b \in [y - \lfloor \frac{M}{2} \rfloor, ..., y, ..., y + \lfloor \frac{M-1}{2} \rfloor]\}$ have each received $\mathcal{M}$. After step **ii.**, modules $\{(M, b, a) | b \in [y - \lfloor \frac{M}{2} \rfloor, ..., y, ..., y + \lfloor \frac{M-1}{2} \rfloor], a \in [x - \lfloor \frac{M}{2} \rfloor, ..., x, ..., x + \lfloor \frac{M-1}{2} \rfloor]\}$ all have received $\mathcal{M}$. (Recall our convention that South and East are each positive directions.) Then, after step **iii.**, modules $\{(0, b, a) | b \in [y - \lfloor \frac{M}{2} \rfloor, ..., y, ..., y + \lfloor \frac{M-1}{2} \rfloor], a \in [x - \lfloor \frac{M}{2} \rfloor, ..., x, ..., x + \lfloor \frac{M-1}{2} \rfloor]\}$ all have received $\mathcal{M}$. From the description of the network structure given above, we know that these are exactly the coordinates of the set of pixel-scale modules which are covered by a module at position (M, y, x), namely, m. Hence, every pixel-scale module that is covered by m receives $\mathcal{M}$.

Each type of message we shall use is a binary associative type such that any collisions between incoming messages at a module may be resolved by an INCLUSIVE-OR operation. Our two basic processes, then, are (1) to send a message from some set of pixel-scale modules to every module in all higher scale levels which is proximate to at least one element of that set, and (2) to send a message from some set of higher-scale modules to every pixel-scale module which is covered by at least one element of that set. The "some set," in both cases, typically might be "all those that are colored." Again, they require 4 and 3 time steps, respectively.

Finally, we can construct the optimal algorithm, which meets the requirements enumerated in the first paragraph of this section. The procedure starts each time an image frame is presented to the pixel-scale modules. The input contains information about contours and initially-colored areas. The network then sends messages, and each message $\mathcal{M}$ consists of simply one or more of the following tokens:

Message Tokens

Basic Algorithm

- contour
- colored

Each token can have a value of 0 or 1, indicating the *absence* or *presence* of that property, respectively, for the module in question. Modules assign to their internal value the INCLUSIVE-OR of the values received most recently for that token, subject to Corollary 1.

The manner of arrival of a message can cue the recipient module as well as can the message's content. Each module can utilize information based on whether it has received a message token recently (since some other event), which type the token is (**colored**, **contour**, etc.), which value the token has (0 or 1), from which direction the message came (from above, from below, from North or South, and from West or East), and a history of past messages in terms of these items. If the network is globally (externally) synchronized, then all that is needed is the value of the token; the synchronization will specify the context for action. However, if the additional information is available, then a much less expensive model may be used (see Section 6.4). Hence, there are two correctness issues: the first is whether the presented sequence of actions results in coloring at an optimal speed; the second issue, introduced here, is whether the available cues can be used as proposed to guarantee that the presented sequence of actions occurs (as well as fault tolerance — stability) in the absence of global synchronization. We present the optimal algorithm below with the proper cues to use if they are accessible. If not, then the steps are simply performed in the given sequence. In the Connection Machine implementations, we use global synchronization.

Let p denote any pixel-scale module, and m denote any higher-scale module.

The Optimal Algorithm

1. When the p's receive a new image frame, they each perform an Upward Propagation with **contour** to initialize the m's so as to be consistent with the new image. In addition, the p's and m's can optionally reconstruct broken contours and update all modules accordingly ([generates the **in-gap** token] see Section 8) and/or uncolor those previously-colored modules which are no longer enclosed by a contour due to contour motion ([clears some values of the **colored** token] see Section 9).

2. Until the p's receive a new image frame, loop through the following steps: **(a)** after receiving a **color** message from any m during the previous cycle, each p [unless its **in-gap**=1] performs a Pixel-Scale Propagation with the **colored** token; each p that receives a "**colored**=1" message updates its internal value accordingly, unless its internal **contour**=1; **(b)** after receiving a **color** message from one of its 4 neighbors, each p [unless its **in-gap**=1] performs an Upward Propagation with **colored**; each m that receives a "**colored**=1" message updates its internal value accordingly, unless its internal **contour**=1; **(c)** after receiving a **colored** message from an East or West sibling, each m performs a Downward Propagation with **colored**; each p that receives a "**colored**=1" message from above updates its internal value accordingly (none would have **contour**=1 — i.e., none covers a contour).

We see that part **2** captures all of our requirements for an optimal algorithm. At the start of part **2**, the maximum extent of the coloring so far is represented by the pixel-scale level. This is because either a new image frame has just been received, or else the Downward Propagation process had occurred at the end of the last iteration of part **2**. The Downward Propagation process updates the pixel-scale modules by coloring all those which are covered by a colored module; put another way, the result represents the union of all colored square regions of various scales projected onto level 1. Therefore, at the start of the first iteration, and for all subsequent iterations, we must color every uncolored module which is proximate to a colored pixel-scale module, but which does not cover a contour, in some constant number of time steps. We have shown above that the result of the Pixel-Level Propagation process followed by the Upward Propagation process — processes **2.(a)** and **(b)**, respectively — is just this. We perform these processes at a cost of one step for Pixel-Level Propagation, and an additional 3 steps for the Upward Propagation. Again, Downward Propagation — process **2.(c)** — updates the pixel-scale modules for the next iteration, so that they again entirely capture the extent to which the image has been colored so far. Figure 18 shows the maximum possible result of a complete coloring iteration at one scale level, starting with one colored pixel-scale module; it is a maximum result in the sense that no contours impede the coloring process at that scale. As a result, these three processes are sufficient to ensure optimality for all succeeding iterations. Since Downward Propagation takes 3 steps, our algorithm is optimal to within a factor of 7. The derivations needed to support this optimality claim are unified and further developed in the proofs

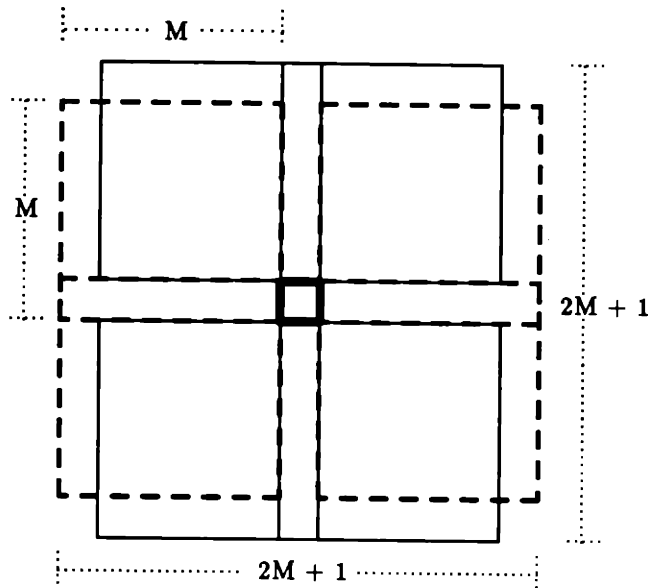


Figure 18: The effect of the M-scale level during a coloring interaction, in the absence of contours, is that the pixel-scale modules in all $M \times M$ areas that cover or border the initially-colored pixel-scale module (in bold outline) become colored. The result is that a total of $(2M+1)^2 - 4 = 4M^2 + 4M - 3$ pixel square modules are colored via that level.

below.

Besides having optimal speed for chunk-based networks of its kind, our algorithm has some additional nice properties. Most importantly, connectivity degree is only $O(N)$. We still have N ordered representations of the image at different scales. Also, as mentioned before, notice that there is a well-defined chain of contingent operations at a local level — for each module — which ensures that events are sequenced correctly within the compound cycles. Furthermore, the retiming technique can be applied to other related networks to minimize hardware or maximize speed. Due to the power of having modules of all positions within scale levels, the hardware of our optimal network can be considerably reduced in other ways with only a slight degradation in performance. One possibility is to use only those modules (and the relevant connections) whose scale is a power of two; this yields the network for Algorithm #5, which requires only $O(N^2 logN)$ modules. Another network that could benefit from this retiming method is the one implemented by Shafrir; Shafrir could use the *complete* Rectangular Model, while limiting network connectivity to $O(N)$.

6.3 Proof of Optimality

We would like to find an algorithm for this model which is the fastest possible (to within a constant factor). At the same time, we would like to preserve $O(N)$ connectivity and contain our use of space. The optimal algorithm developed here preserves $O(N)$ connectivity, but uses $O(N^3)$ space. (A compact version of the optimal algorithm requires only $O(N^2 \log N)$ space, and is presented later. The image by itself occupies $O(N^2)$ space.) The basic principle here is *retiming*: using direct connections would imply $O(N^3)$ connectivity, but the network can be retimed so as to have $O(N)$ connectivity and communication lagged by a factor of 7.

OPTIMALITY CLAIM *For the class of computational models given above (simple, local interconnections, parallel square chunks, and observance of Corollary 1), our algorithm requires the minimum number of time steps, to within a constant factor, to complete any given coloring task.*

Our algorithm colors regions optimally because it can proceed optimally at every location during every cycle. That is, the total number of steps it requires to complete a coloring task is the fewest possible (to within a constant factor) because, in every cycle, the maximal legitimate set of uncolored modules — those that meet the criteria in Corollary 1 — become colored. Each cycle requires 7 time steps; this introduces our lag factor of 7. To prove optimality, we must show that (1) during each cycle, *all* eligible uncolored modules become colored, and that (2) coloring the maximal set of legitimate modules from each location in every cycle yields the shortest possible total coloring time.

LEMMA 1 *During each cycle, every uncolored, free-space module that is proximate to a colored module becomes colored.*

Proof: The main point of this proof is that, given any colored module, then for every proximate module that is uncolored, there exists a colored pixel-scale module such that the uncolored module is proximate to it as well, and which will cause the uncolored module to be colored during that cycle. From our preliminary definitions, we know that if one module m_1 is proximate to another m_2, then there exists at least one pixel-scale module

p_1 covered by m_1 such that m_2 is proximate to p_1. We also know from our more recent discussion that, at the beginning of each cycle, any higher-scale module that is colored covers a set of pixel-scale modules all of which are colored as well. That is, p_1 is colored at the beginning of the cycle. Finally, it was shown that the result of Pixel-Level and then Upward Propagation is that every module (of scale-levels $1, 2, ..., N$) that is proximate to a colored pixel-scale module receives the message $\mathcal{M}$. In this context, $\mathcal{M}$ is "**colored**= 1"; at the conclusion of step **2.(b)**, each recipient of such a message colors itself if it does not cover a contour. Therefore, during that cycle, every uncolored, free-space module that is proximate to a colored module becomes colored. •

This is equivalent to saying that, in each cycle, every square-shaped region of pixel-scale modules of every size that covers or borders a colored pixel-scale module, and that does not cover a contour, becomes colored. Next, we show that doing so for every cycle yields an algorithm which is optimal.

LEMMA 2 *Coloring the maximal set of legitimate uncolored modules in each cycle yields the shortest possible total coloring time.*

Proof: By "legitimate" we mean those that are proximate to a colored module and that do not cover a contour.

We can represent the *complete network* of modules and their *proximity* as an undirected graph. Here we are not concerned with actual connections between modules in any particular algorithm. (Recall that the complete network contains the set of all parallel square modules — every size and position — that have subordinate pixel-scale modules mapped to an image element.) Each *module* can be represented by a vertex; each *association* between a pair of distinct, proximate modules can be represented by an adjacency edge. For an arbitrary contour map, let us define the graph G to be the colorable subset of that graph: let G consist of V and E, such that (1) V is the set of vertices which are mapped to the union of s — the starting module — and all other 4-connected free-space modules, and (2) E is the set of all edges which represent proximity between pairs of these modules. (Every edge has a length of 1.)

We have shown that our algorithm is one in which, for each succeeding cycle, each colored module passes coloring to every uncolored, proximate, free-space module, with arbitrary resolution of collisions. This recursion can be mapped directly onto our proximity graph G. The coloring process then exactly embodies that for finding the breadth-first

spanning tree of G from s. That is, G is searched breadth-first from s, and a breadth-first spanning tree is constructed with s as the root. We can assign to each vertex v a number equal to the distance from root s to vertex v in the spanning tree. Since the spanning tree is breadth-first, the depth number of vertex v indicates the length of the shortest path from s to v in G. Thus, the spanning tree is of *minimum* depth from s, and the number of cycles required for propagation is a minimum. •

Our algorithm is unique because it is able to conduct a breadth-first search on the entire proximity graph G, whereas previous algorithms conduct breadth-first searches on particular subgraphs of G.

6.4 Asynchronicity and Stability

In order for the optimal algorithms to serve as a biologically plausible model, it would be preferable that its computation exhibit asynchronicity and stability. It is not clear whether in nature there could be a clocking mechanism which could coordinate the actions of the processors. Also problematic is how to respond when a processor becomes aberrant, and behaves in a pathologic way, due to common noise or a specific malfuntion. It might be indicated by its sending the wrong type of messages (at the wrong time step during the iteration), or the wrong value (one that always differs from those of its neighbors). In both cases we would like for the coloring process not to degrade completely. This might entail either identifying and ignoring modules when they become "defective" or else continually performing some type of local averaging or smoothing of the incoming values.

The investigation of issues at this level is beyond the scope of this thesis. However, attention was given such that in each of the optimal algorithms and their extensions, the processing units follow a locally defined sequence. This means that from the *type* of message that it receives from each of its neighbors, in conjunction with the *direction* of transmission, the module has enough information to determine at which stage in the computation its neighboring modules are. Given this local information, and a clarified set of goals, one could develop a unified theory as well as an algorithm for tackling this essential modeling problem.

6.5 Optimally Coloring 8-Connected Regions

One can optimally color 8-connected regions by making a couple of simple modifications to the optimal algorithm. First, step **2. (a)** is removed. Then, the **Upward Propagation** process is altered by adding 1 to each of the quantities of siblings to whom messages are sent. For example, "... module m passes $\mathcal{M}$ to its $\lfloor \frac{M-1}{2} \rfloor$ North siblings ..." becomes "... $\lfloor \frac{M-1}{2} \rfloor + 1$ ..." Likewise, 1 is added to the numbers of South, West and East siblings to whom messages are sent.

These modifications simplify the optimal algorithm to some degree, and reduce the number of steps per iteration from 7 to 6. However, the 8-connected algorithm is not as attractive from a practical point of view. Contours are commonly represented as edges of single pixel widths. 8-connected activation unfortunately passes through contours of this kind wherever local sections are oriented diagonally, rather than vertically or horizontally. Nonetheless, the 8-connected version was implemented for completeness' sake, and the results can be seen below in Section 6.6.

6.6 Implementation Results

Two simple examples are given first, and then the optimal algorithm is run using real edge maps. The scheme colors convex areas in just one or a couple of iterations. In Figure 19A, the large modules that are centered near the periphery of the image enable one to color the exterior of regions very quickly, in the absence of other contours. Figures 19A and B can be compared against the corresponding results for the preliminary algorithms, thereby demonstrating the relative power of the optimal algorithm.

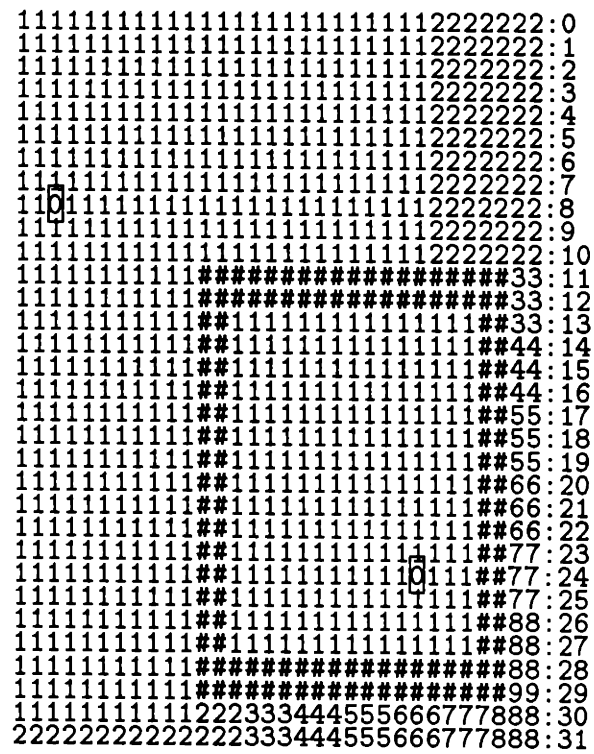
Finally, for real data, we take an image of objects arranged on a desk. We would like to label all elements in the image that correspond to the light-colored spiral notebook in the upper righthand part of the image. See the "warped" righthand image of a stereo pair in Figure 20. (Its unwarped counterpart will be used as well in the section on moving contours.) Applying the Canny edge-finder yields a completely closed contour in which to color, shown in Figure 21. We then choose one pixel to be the starting point, and apply the optimal coloring algorithm. The result is given in Figures 22 and 23.

Figure 19: On the following pages are results from the full-size optimal algorithm.

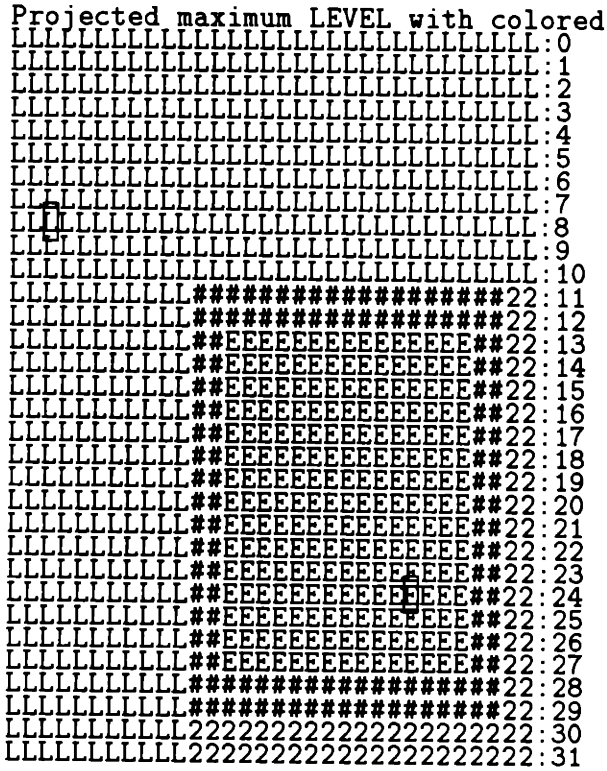
Figure 14A*ii*

Full-Size Optimal Algorithm

Projected minimum TIME-STEP when colored

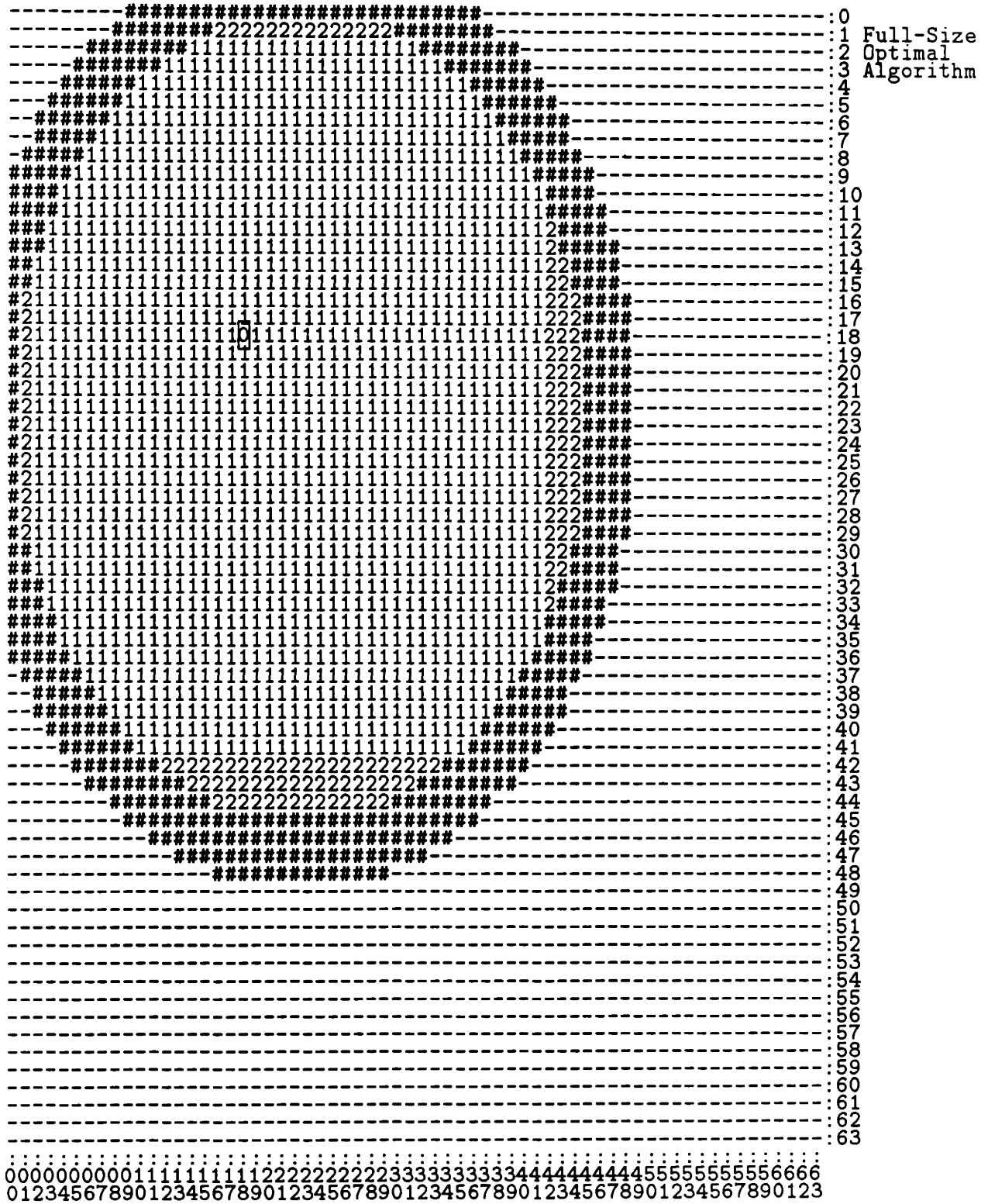


Projected maximum LEVEL with colored module



Projected minimum TIME-STEP when colored

Figure 14B*i*



Finally, we see the problems that the 8-connected version of the optimal algorithm has on Canny edge data in Figure 25. This came from the image in Figure 24. It may offer a solution of interest in other theoretical domains (e.g., with a different type of boundary), but it does not seem applicable here.

7 Compact Optimal Coloring

An important weakness of the approach given above is that, while it is optimal in terms of speed for the defined class of computational models, it requires a lot of space. It uses $O(N^3)$space for an image that occupies $O(N^2)$ space. We can easily show that a similar algorithm employing a more compact network — requiring $O(N^2 \log N)$ space — is still optimal, lagged by an additional factor of less than 2.

By looking at the implementation results, one can see that the optimal algorithm gains considerable power from the fact that its modules overlap very densely in each level. It utilizes image chunks of every size and position: at every scale level M, there is one module centered over each pixel-scale module that covers an $M \times M$ area. This means $N \times N$ modules per level, $\times N$ levels $= N^3$ modules. If we could eliminate all levels except those whose scale is a power of 2, we would have a network with only $(1 + \log_2 N) \times N^2$ modules.

Let us consider, just in principle, what loss in performance any optimal coloring algorithm would suffer if it used modules having all positions but only sizes that are powers of 2. The specific question we would like to answer is, given any colored pixel-scale module and the fact that an arbitrary scale level M causes $(2M + 1)^2 - 4$ pixel-scale modules to be colored (see Figure 18), subject to the presence of contours, how many iterations would be required to match its result using only our power-of-two levels? To answer this, we observe that we cannot count on using a scale level larger than M to achieve M's coloring result, because there may exist a contour just beyond this result that would prevent us from activating such a large module. We *can* use the largest power-of-two level that is $\leq M$, calling it M_2, as long as its area of coverage is within M's coloring result.

Now we have to get into the details. In Figure 26, we have sketched the coloring range of the M_2 level so that it is concentric with that of the M level. Refer also to Figure 18. To achieve the coloring result of the M level, it would be adequate for the M_2 level to color the central pixel plus the next M modules in each direction. We will focus on coloring the Northeastern part of M's region, since our coloring range is symmetric in each of the four directions. It will help to discuss the pixel-scale configuration in terms of integer

Figure 20: A grey-scale image of objects on a desk top.

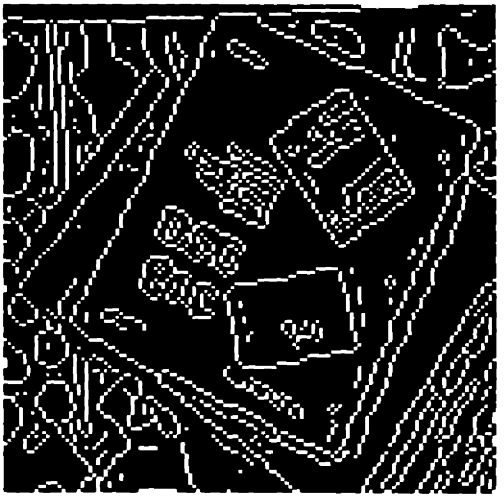


Figure 21: Contours of the desk objects from the Canny edge-finder.

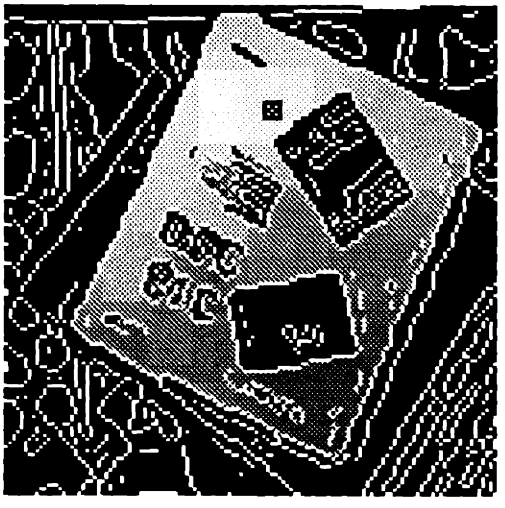


Figure 22: Result of coloring with full-size optimal algorithm. The projected minimum TIME-STEP when colored is indicated by inverse brightness.

In the natural edge maps, an inverse brightness scale indicates the time when colored and maximum module level for each element, rather than the system of numbers we use for smaller synthetic maps. That is, brighter locations mean a smaller numeric value, and vice versa. A small white X indicates starting points for coloring.

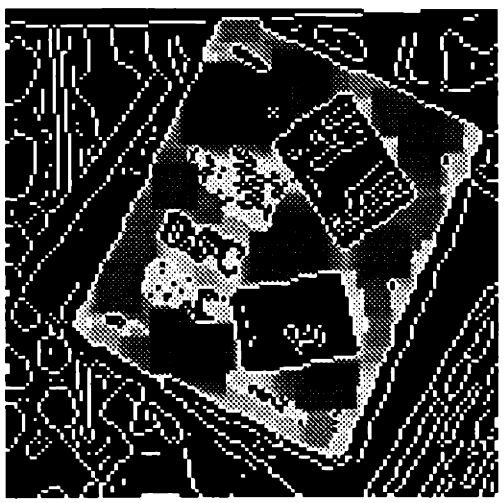


Figure 23: Result of coloring with full-size optimal algorithm. The projected maximum LEVEL with colored module is indicated by inverse brightness.

coordinates, where the central pixel has coordinates $(0,0)$. Our goal then is to color all pixels in the (x,y) range $(0 \ldots M, 0 \ldots M)$ using only the M_2 scale level. Referring again to Figure 18, we recall that from p_1 we can color all modules in the range $(0 \ldots M_2-1, 0 \ldots M_2)$. In the Figure p_2 is at position $(M_2 - 1, M_2)$. We could also color all modules in the range $(0 \ldots M_2, 0 \ldots M_2 - 1)$ from the starting point $(0,0)$. We choose instead to add this latter range to the result of applying the former range to p_1. Thus, by coloring twice successively, we are assured of coloring all modules in the range $(0 \ldots M_2 - 1 + M_2, 0 \ldots M_2 + M_2 - 1)$, plus several others that we are not accounting for here. Note that from p_2 we are able to reach the last row and column of M's coloring result. That is because the corner piece (which actually is missing) has coordinate (M, M); since $M < 2M_2$, this point is within the range $(0 \ldots 2M_2 - 1, 0 \ldots 2M_2 - 1)$. Therefore, every non-power-of-two level in the full-size optimal network can be simulated for coloring purposes by the power-of-two level whose size is less than or equal to it, in at most twice the time. Our compact optimal algorithm is as least half as fast as our full-size version, with substantial savings in space.

The compact optimal approach was implemented on the Connection Machine. We present some implementation results below to show that the scheme works, and in fact almost as quickly as the full-size algorithm. See Figure 27.

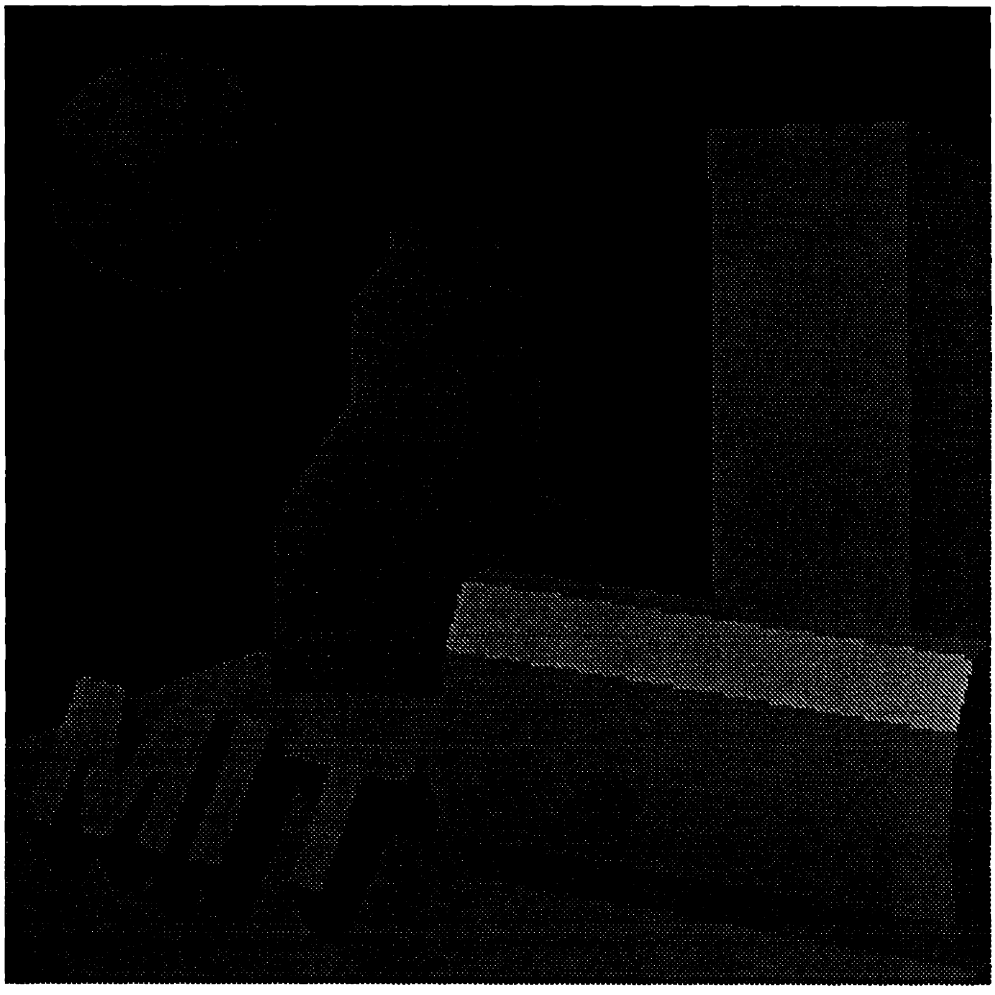


Figure 24: Graphics rendering of a city skyline.

Figure 25: Edges from city image. Unfortunately 8-connected coloring goes through most edges, although not through horizontal or vertical sections of them. The projected minimum TIME-STEP when colored is indicated by inverse brightness. (Edge pixels are in black.)

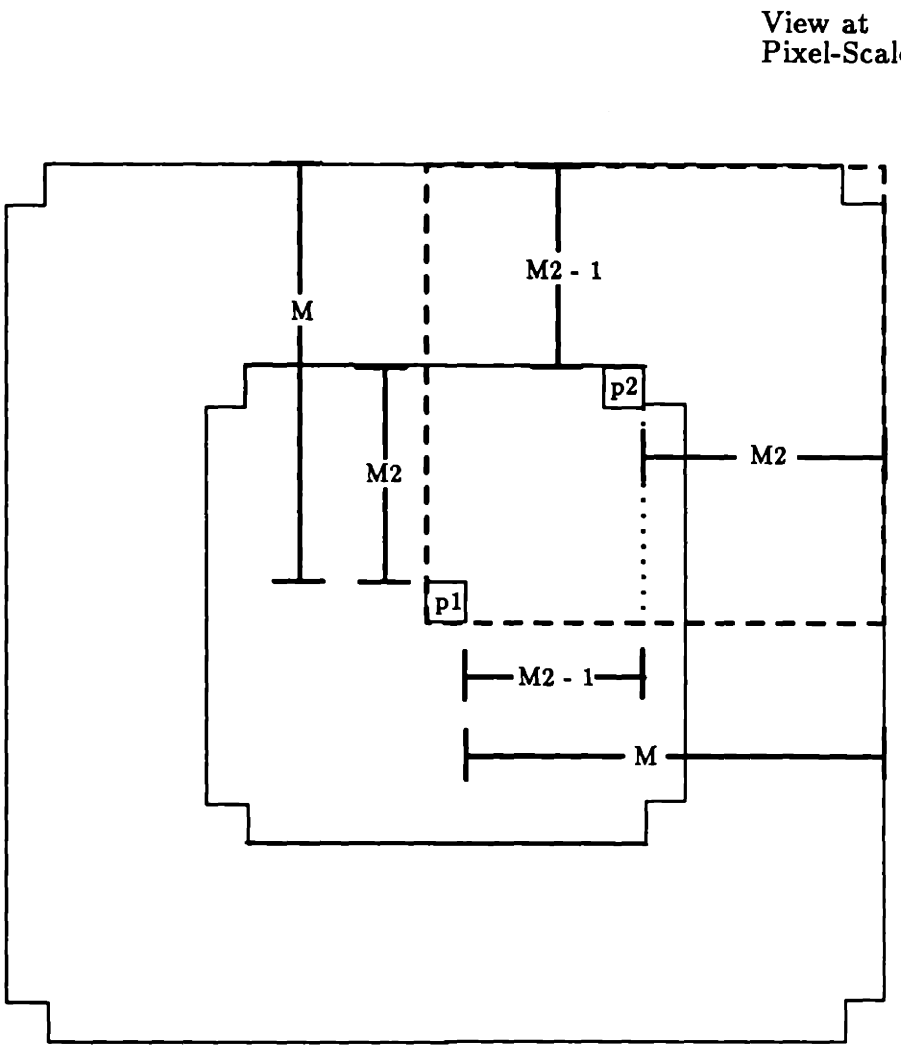
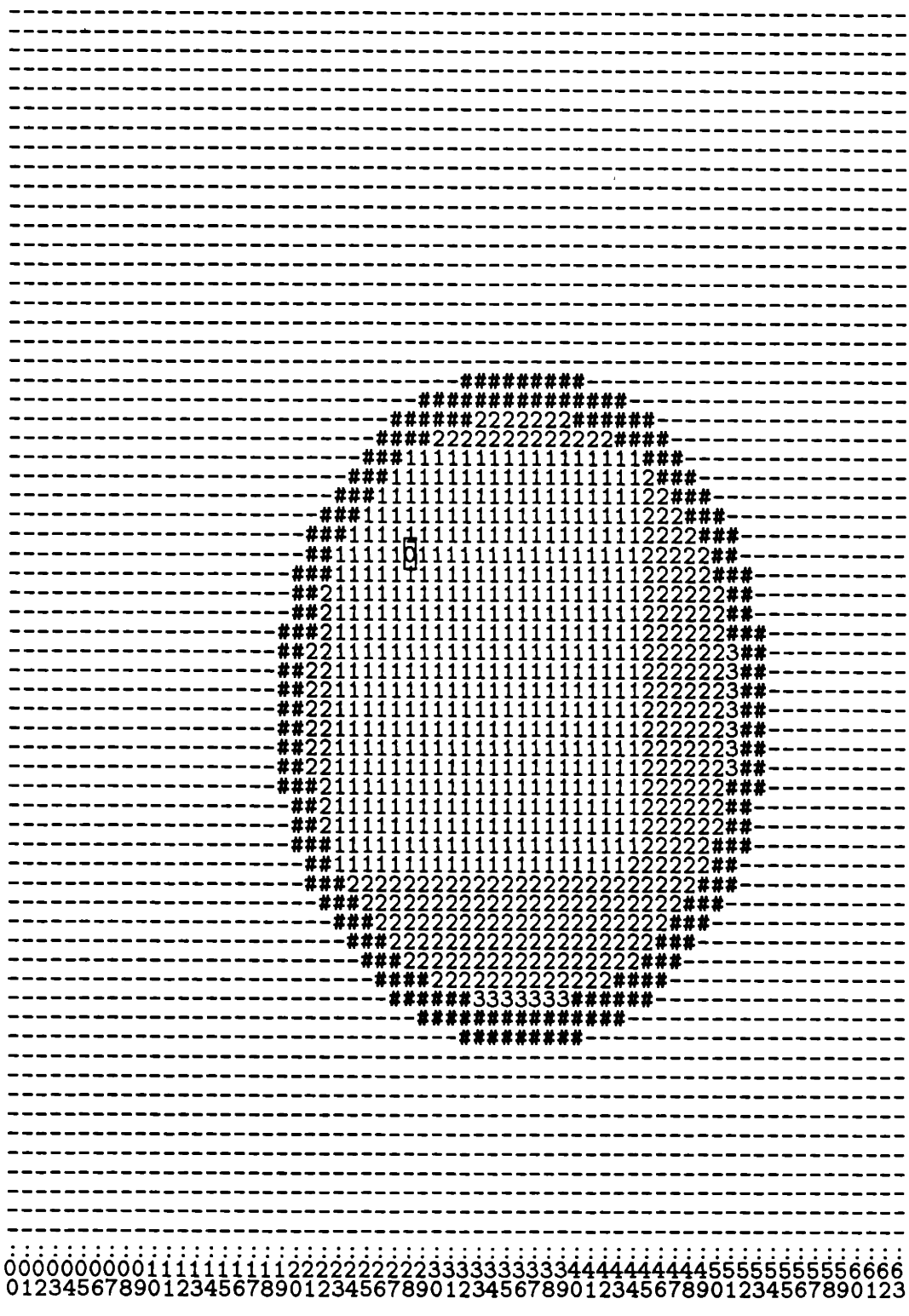


Figure 26: The compact optimal algorithm can color at least half as fast as the full size scheme. That is because scale level M_2, M_2 a power of two, can color any region that an arbitrary scale level M, $M_2 \leq M < 2M_2$, could do in at most twice the time.

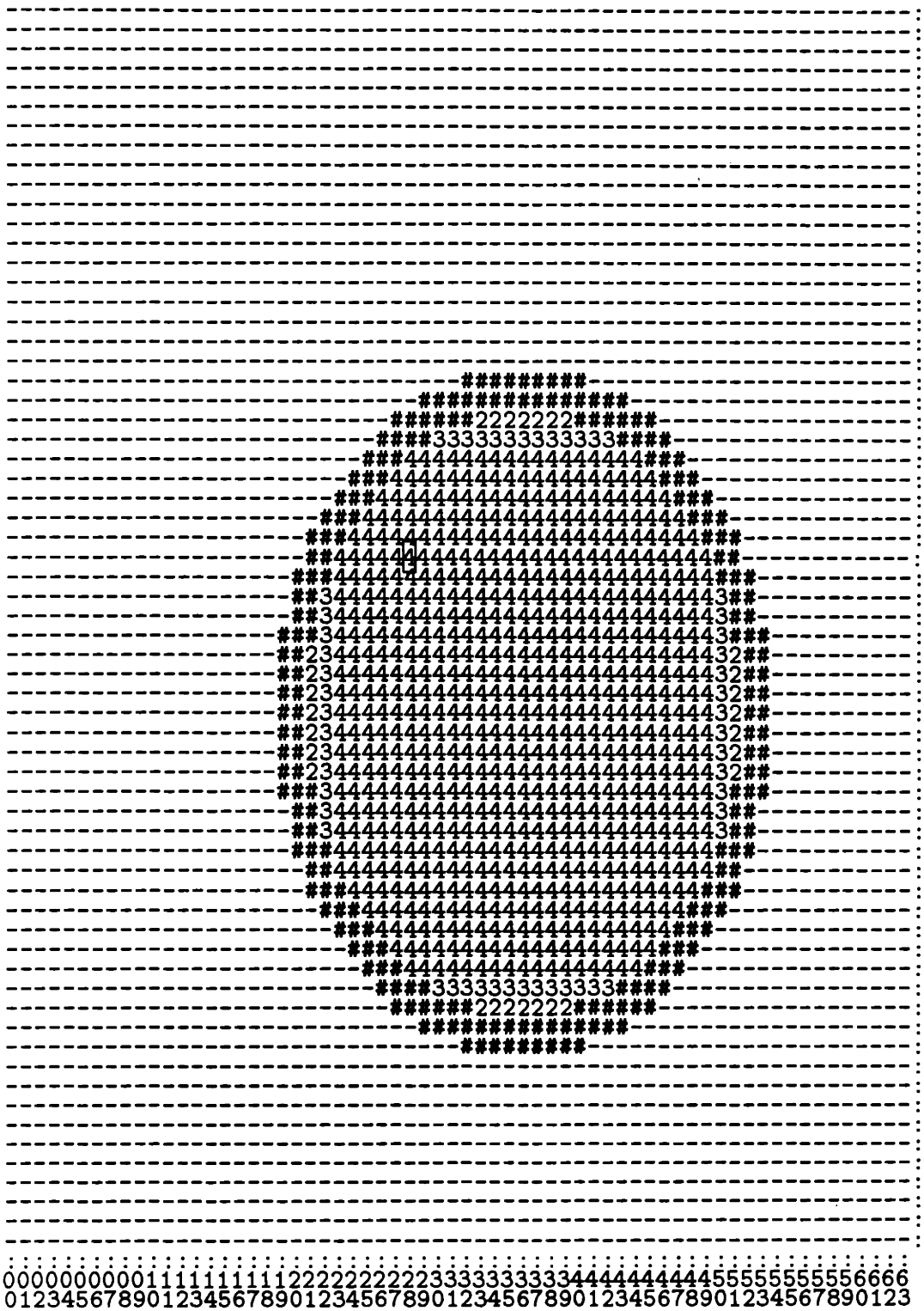
Figure 27: On the following pages are results from the compact optimal algorithm.

Projected minimum TIME-STEP when colored

Figure 27



Projected minimum TIME-STEP when colored



8 Compact Optimal Coloring inside Broken Contours

An important requirement for intermediate visual processes, such as coloring, is that they be robust. The early visual processes encounter a rich variety of input, and according to Marr's proposal [Marr 82], generate from these the primal sketch and the $2\frac{1}{2}$-D sketch. These base representations change with time (often not smoothly), and are generally somewhat degraded in quality. They nonetheless are used as input by our coloring operation. The coloring must yield a solution that is generally stable, and which is abstract enough to be useful for later processes. The linked multi-scale property of the algorithms given above allows for adept handling of real world data with several common difficulties.

We can view the advantages of the compact scheme in the results below. In practice, it does not seem to require much more time than full-size optimal scheme. It does save considerable space. Whereas we were limited to an square image size of 128 with the full-size algorithm with our current Connection Machine configuration of 8192 physical processors, the compact version makes possible use of images of size 1024.

8.1 Theoretical Motivation

One common problem is the appearance of gaps between edges that correspond to a real, physically-defined boundary. Generally such undesired gaps are caused by some distortion in the imaging process; other times, they are the result of an actual phenomenon in the scene, such as a line drawing like Figure 30. Hence, by "broken contour" we mean that the bounding contour has spurious gaps which "ought" to contain a contour segment, but for some reason the segment is missing. In many cases, such as in Figure 30, the "missing contour segments" exist only in a "subjective" sense, since there is no principled method to infer the path of the missing segments *a priori*. Indeed, the hypothetically completed contour is often termed a *subjective* contour. In most natural situations, on the other hand, there is often an underlying physical discontinuity even where the edges are missing, and these must be reconstructed by some inferential computation from the image. In either case, it is often correct to interpret broken edges as meaning there exists a continuous physical contour. So, for instance, humans perceive Figure 30 as some form of closed curve, with an enclosed foreground region and a background as well. Notice that the distance spanning nearest pairs of endpoints varies. An algorithm that is a viable model of the coloring process in human vision should color a representative portion of the semi-enclosed region, without coloring any significant portions of the background.

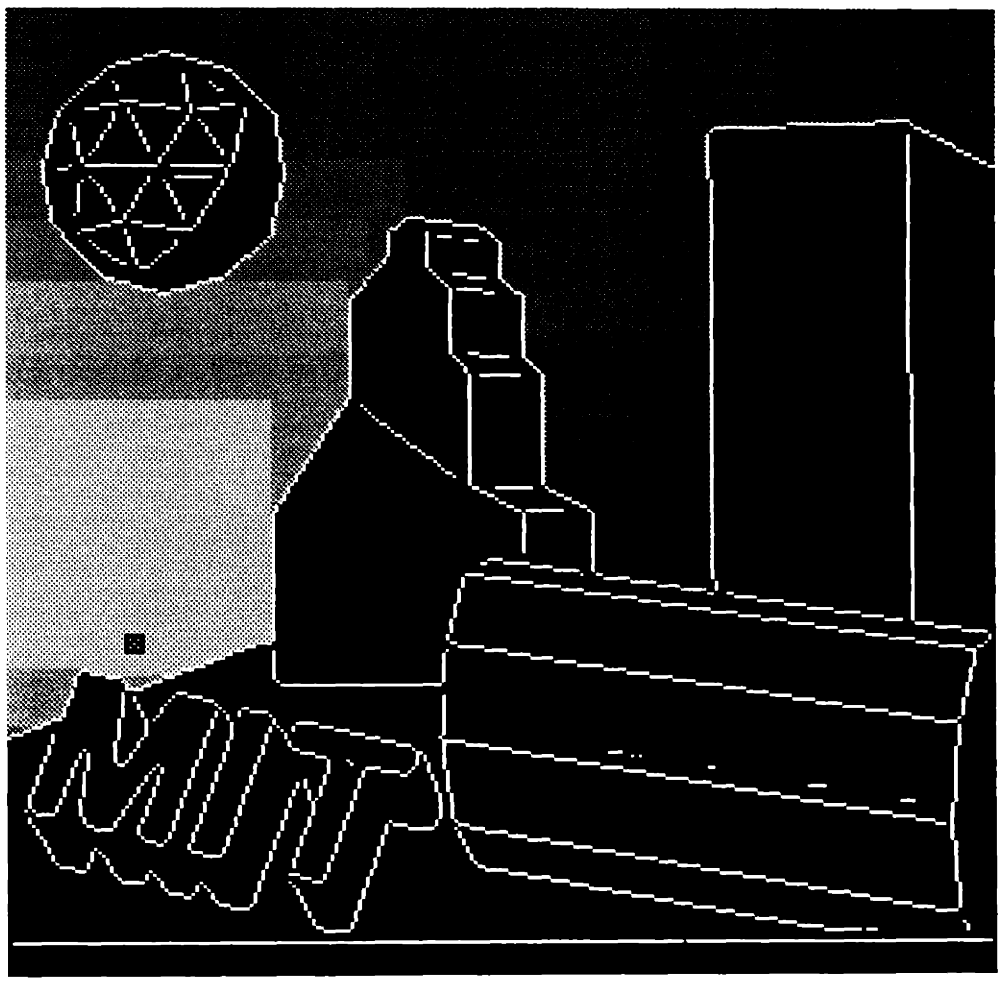


Figure 28: City edges colored with compact optimal algorithm. The projected minimum TIME-STEP when colored is indicated by inverse brightness.

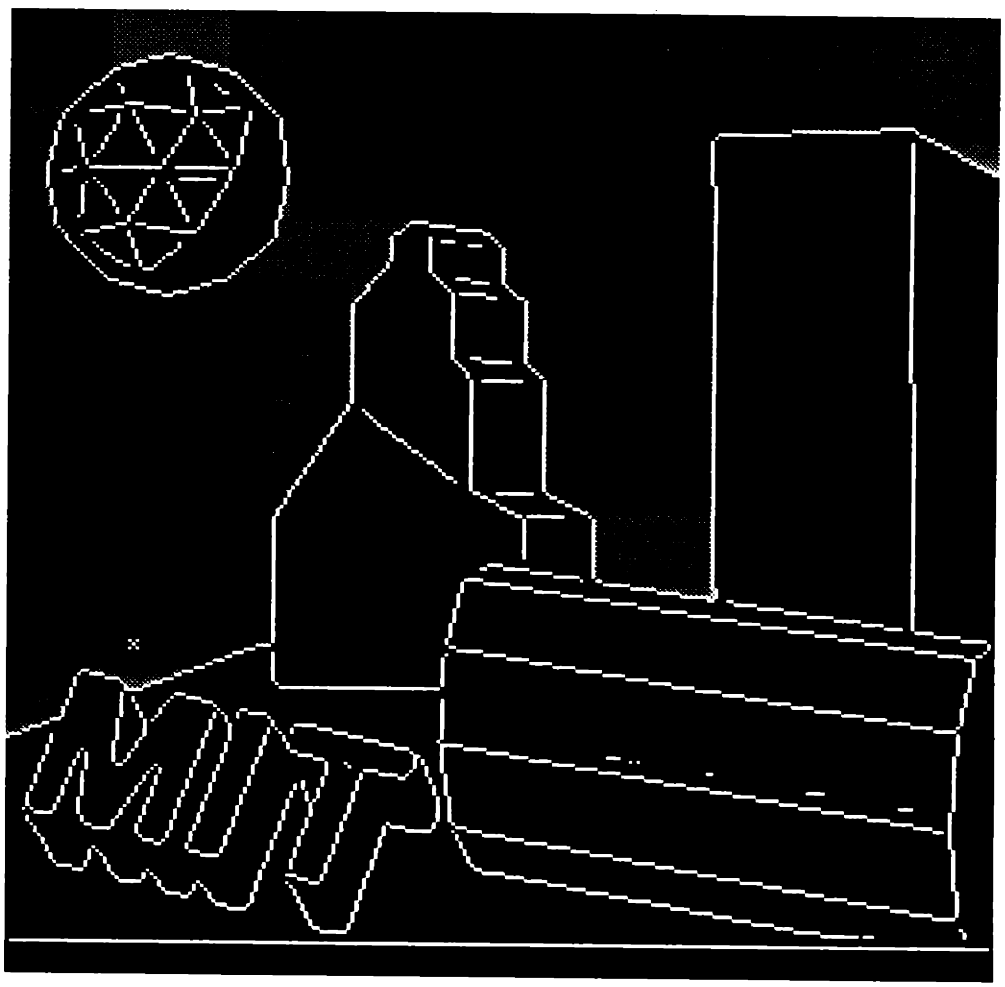


Figure 29: City edges colored by compact optimal algorithm. The projected maximum LEVEL with colored module is indicated by inverse brightness.

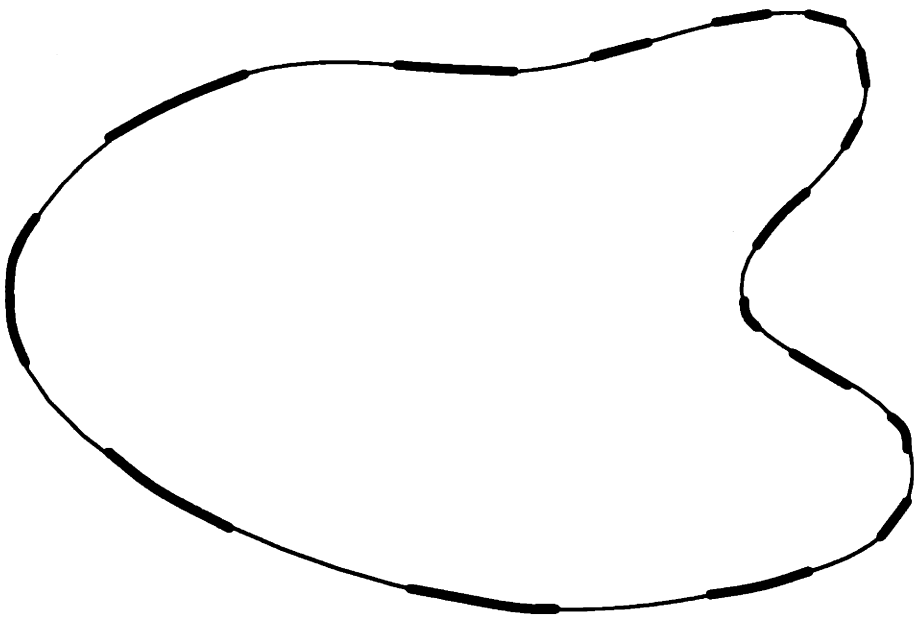


Figure 30: A broken closed contour, drawn in bold print, with gaps of varying widths. Subjective contour segments are sketched in fine print.

How might we color inside *broken* closed contours? We make the simplifying assumption here that each endpoint subjectively "joins" to the endpoint(s) nearest to it. Thus there are numerous figures, such as Figure 31, which we would consider unqualified for coloring purposes. Since the network represents figures at a variety of scales, we will be able to locate those modules whose constituent pixels lie within the gap and block their responses in an appropriate way. Figure 14G illustrates the catastrophic effects of just a single gap for a naive coloring algorithm (i.e., Algorithm #3). In this case, we would like the pixel-scale module (identified as a 0 in the second frame of Figure 14G) found in the gap to cease passing **colored** messages. The linked hierarchical description helps to make this task fast and easy to accomplish.

For our constraint to yield success, some preprocessing steps should occur. One generally would not proceed directly from a raw set of edges. Instead, the appropriate input would be a set of edges selected from the raw edge set. Also, many of the contour endpoints (i.e., termini) can be joined or rendered unimportant by either low-level or goal-driven processes. For instance, a local scale for thresholding closable gap widths could be determined locally, possibly based on contour curvature. Then, applying these to the given termini,

each terminus is joined to one other one, with matching based on greatest proximity. Although we limit ourselves in this paper to a representation which is isotropic, and which does not quantify orientation, location, distance or size, we nevertheless can close gaps for the sake of coloring in a satisfactory way for many situations. Our measured criteria consists only of surface set membership, and yet with this we can determine maximal proximity. Without any preprocessing, the following algorithm extension gives false positives in some cases. Its main weakness is that a coloring boundary may be established where it is not desired.

Certainly, one would need a comprehensive, principled approach to segmentation in order to adequately delineate regions in nearly all cases. Propounding a possible piece to this puzzle, Beymer [Beymer 89] examines junctions of three or more regions in an image, and proposes a method to reconstruct contour junctions in gaps typically left behind by edge finders. He focuses on junctions that are caused by occlusions, called T junctions. (The other type of junction results from vertices, points where surfaces intersect.) Utilizing the gradient of image intensity values, Beymer is able to extend curves despite gradient direction skew and a vanishing gradient. Meticulously extending the contours, and locating the juction, is useful for determining how surfaces occlude or intersect one another, for use in grouping, or as matching tokens. However, for the sake of region segmentation, T junctions are a place where bounding generally should occur anyway. Indeed, three termini localized in the input map are joined by the routine below in one compound filled gap.

Foregoing image intensity information, other than an edge map, Ullman and Sha'ashua [Ullman & Sha'ashua 88] are able to compute a map of structural saliency, and, as a by-product, smooth contours and fill in gaps. The work is based on linking saliency with low curvature (or low curvature variation) and large curve length, with these formulated as an extensible function. One would need to apply a scheme, such as this, to select salient contours from image primitives before proceeding with bounded activation. Of course, this scheme by itself would not suffice for closing all appropriate gaps; there might still be a number of gaps remaining, along sections of contours with high curvature, such as object corners, or with missing segments. Their algorithm is iterative, and generally requires a number of iterations that grows in proportion to the number of points missing from the gap; often this amounts to dozens of iterations, or more. In any case, for present purposes, we apply our extended algorithm directly to edge maps in the discussion below. It actually works as one would like most of the time.

Our present goal, then, is to develop a modified version of the optimal network which will not color beyond contour gaps. The solution can be divided into two phases. First,

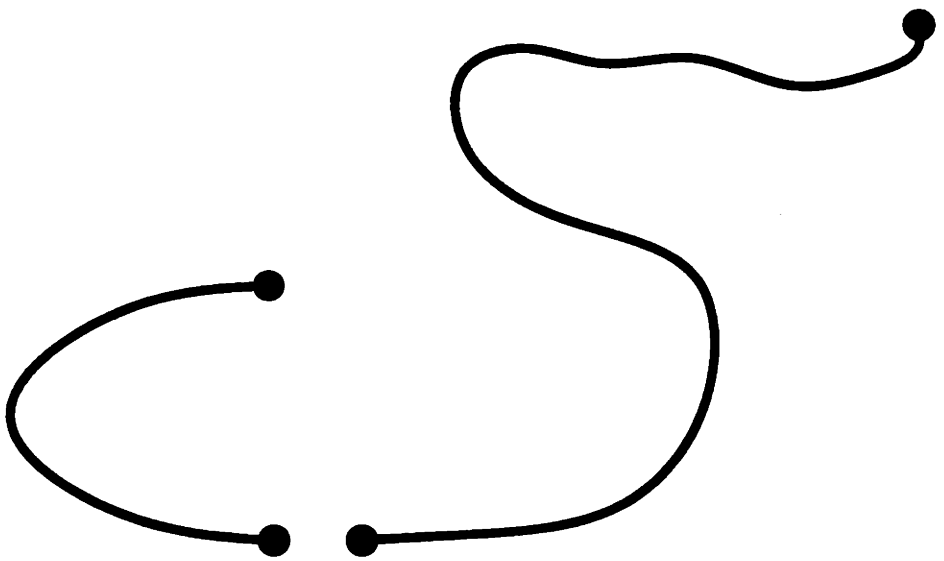


Figure 31: A broken contour which we deem "unclosable" and therefore uninteresting.

we establish the boundary, which would strictly enclose the activation. Although assigning inside/outside relations to gap areas is ambiguous, we choose here to assume that both contour segments and gaps are part of the foreground object and therefore to color the gaps. The algorithms presented below can be trivially altered to treat the gaps as "uncolorable" (just like contours) instead of as "colorable." It will be explained below that all modules which cover a colorable gap are instructed to receive (and record) but not to send activation messages. The coloring process is thereby stopped before it can proceed beyond the gap. The second phase calls for activation to begin, starting at the X as usual. We add the boundary-construction phase to the optimal algorithm (either full-size or optimal) at step 1. The routine takes constant time, so the altered optimal algorithm's time complexity remains the same as before.

Let us define the task more formally. Figure 32 depicts an arbitrary gap that must be closed. Points A and B are a pair of contour segment endpoints to be joined. A rectangle can be drawn, using A and B as opposite corners; the other two corners are positioned so that the four resulting line segments have the same directions as the X- and Y-directions implied by the square modules hovering over the image. Let us define such a rectangle — i.e., rectangle $AA''BB''$ — to "strictly contain the gap." By this, we mean that, as far as

coloring is concerned, the missing contour segment is assumed to lie within the rectangle. Without loss of generality, we view the rectangle in Figure 32 so that its two longer edges appear parallel to the longer sides of the page, and that "figure" is to the left and "ground" is to the right; activation is assumed to start somewhere within the "figure". Our task then is to ensure that activation does not proceed beyond the rectangle into the ground.

8.2 Algorithm

To achieve this effect using the image chunking representation, we make a modest extension to either the full-size or compact optimal algorithm. The following additional hardware is needed. First, every module is connected so that it can send a message to its one superior module at the next higher scale level; this amounts to one more link per module (zero for the highest-scale module). Connectivity degree is again $O(N)$. [14] Second, we require that each module be able to store and pass information concerning whether or not it covers a contour terminus. This token from the primal sketch can be represented by **terminus**. The extended list of tokens is as follows:

Message Tokens

Basic Algorithm

- contour¶
- colored¶

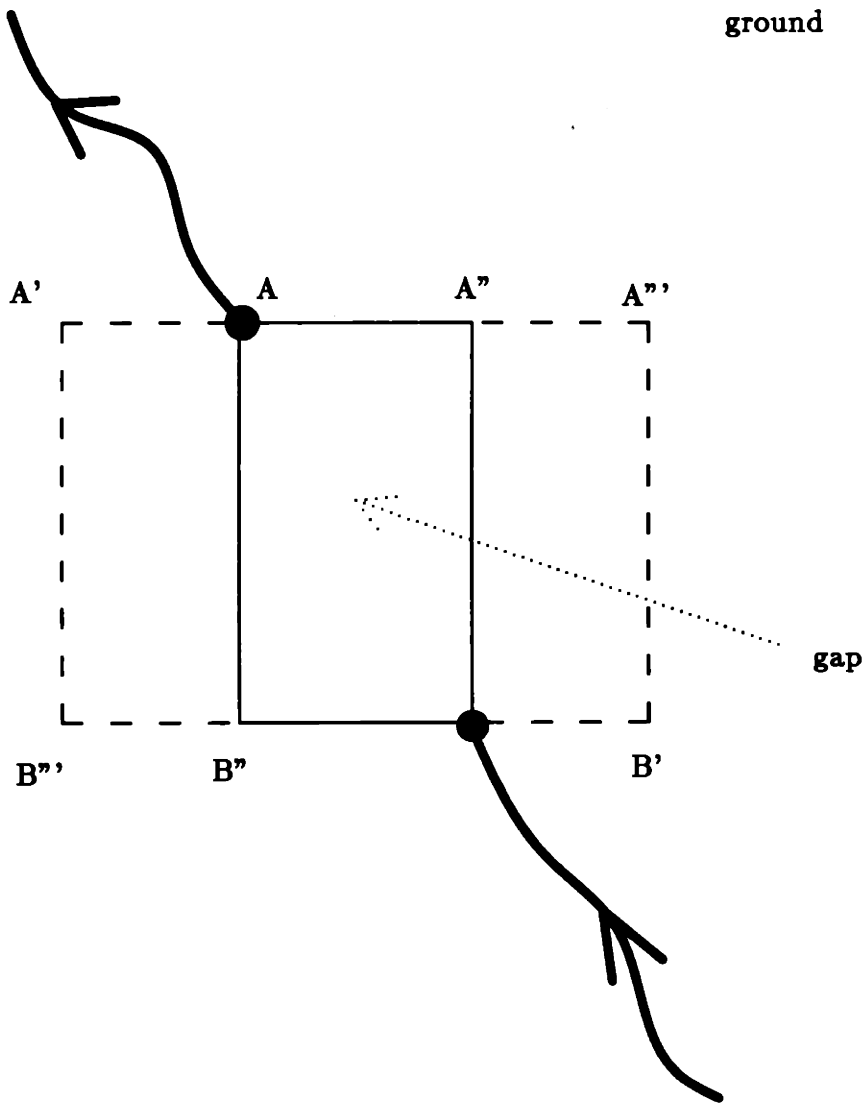
Extension for Broken Contours

- terminus
- terminus-column
- more-than-1-terminus
- paired-below
- near-unpaired-column
- in-gap¶

¶ = global tokens; the rest are temporary

Terminus enters the network at the pixel-scale modules, along with **contour**. (Recall that **contour** and **colored** are required for our basic optimal scheme.) Third, it will

[14] Each module is connected to one module directly above and below it, $< N$ modules directly East or West of it, and $< N$ modules directly North or South of it; $\Rightarrow O(N)$ connectivity.



figure

Figure 32: An arbitrary gap. The areas enclosed by squares $A'A''BB'''$ and $AA'''B'B''$ (including their sides) represent the leftmost and rightmost modules, respectively, which are of minimum size but also cover both A and B.

prove to be useful to have each module know whether or not it covers *two or more* termini. Hence, we introduce **more-than-1-terminus**. Each module must be able to determine whether it received a "**terminus**=1" message from only one, or more than one, connected module during any one of the 3 steps (i.e., during either step **ii.** or **iii.**) of the upward propagation process. If, during any step, two or more such messages are coalesced into one message, then this would mean the destination module covers two or more contour termini.[15] The message is changed at the end of this step from "**terminus**=1; **more-than-1-terminus**=0" to "**terminus**=1; **more-than-1-terminus**=1" in order to indicate this. Next, we use three temporary tokens,[16] which we shall call **terminus-column**, **paired-below** and **near-unpaired-column**. **Terminus-column** will be set to 1 in whichever modules have subordinate pixel-scale modules whose **terminus**=1, forming columns one module wide. **Paired-below** will then be set to 1 in the smallest-scale module in each of these **terminus-column**=1 columns that covers more than one terminus. **Near-unpaired-column** is set to 1 if the nearby terminus-column has not been paired below. Finally, we need another token in order to halt activation inside of gaps. In particular, when **in-gap**=1, that module will not initiate activation. It will not pass along "**colored**=1" messages to other modules, although it may update its own internal **colored** value if it receives such a message. Any module whose **in-gap**=0 will function as usual.

Now we give a precise description of the routine for coloring broken closed contours. It is included in the optimal algorithm inside of step **1.**, prior to the optional "uncolor" routine. Again, let p denote any pixel-scale module, and m denote any higher-scale module.

Reconstruct Broken Contours

1. When the p's receive a new image frame in Process **1.** of the optimal algorithm, they each perform an Upward Propagation with "**terminus**" and "**more-than-1-terminus**" to initialize the m's so as to be consistent with the new image; during this Upward Propagation, we do two special actions:

[15]The coalescence of two or more messages during one of these three steps is a necessary and sufficient condition to indicate that two or more source modules (termini) are covered by the messages' destination module. First, a message sent from a single terminus — a single pixel-scale module — will never pass through or arrive at any module more than once during the entire upward activation process. At each step, messages from this common source move in parallel directions, and there are no redundant connections. Second, messages sent from two nearby termini do activate every higher-scale module which covers them both. They may merge at some other module first, and then share the last connection.

[16]We use three for purposes of clarity only; one token could be used three times.

after the first (upward) step, set **terminus-column**=1 wherever **terminus**=1; after the second and third (outward) steps, set "**more-than-1-terminus**=1" wherever any "**terminus**=1" messages have collided. *[Four steps.]*

2. At each level, each **more-than-1-terminus**=1 module sends a " **more-than-1-terminus**=1" message to all modules in its neighboring span of that same level (using the two outward steps — and not the downward step — of the Downward Propagation routine). Any **terminus-column**=1 module that receives such a message sends up "**paired-below**=1" to its superior module in the next higher scale level (while affecting neither its own internal **more-than-1-terminus** nor **paired-below** values). Any module receiving a "**paired-below**=1" message sets its internal **paired-below**=1. *[Three steps.]*

3. At each level, each module whose **terminus-column**=1 and **paired-below**=0 sends a "**near-unpaired-column**=1" message to all modules in its neighboring span of that same level (using the two outward steps — and not the upward step — of the Upward Propagation routine). Any **more-than-1-terminus**=1 module receiving the "**near-unpaired-column**=1" message sets its **in-gap**=1. *[Two steps.]*

4. After receiving an **in-gap** message, each m performs a Downward Propagation with **in-gap**; each p updates its internal value accordingly. *[Three steps.]*

5. After receiving an **in-gap** message from above, each p performs an Upward Propagation with **NOT(in-gap)**; each m that receives a "**in-gap**" message inverts it again — "**in-gap**" → "**NOT(in-gap)**" — and then updates its internal value accordingly. *[Four steps.]*

The routine works as follows. Step **1.** initializes the modules so as to be consistent with the new image, with **terminus**=1 for each module which covers a terminus, **more-than-1-terminus**=1 for those which cover more than one terminus, and **terminus-column**=1 for those which have subordinate pixel-scale module whose **terminus**=1; all other token values are set to 0.

The result of steps **2.** and **3.** is that, for each terminus in the image, only the module(s) in the minimum scale level which barely cover(s) that terminus and some other terminus will remain with **in-gap**=1 — all others will have **in-gap**=0. For example, suppose the arbitrary gap in Figure 32 is part of an input image. Then, after step **3.**, only the modules

which cover both points A and B and whose constituent pixels fall entirely within (or include) the edges of rectangle $A'A'''B'B'''$ would be labeled as **in-gap**=1. (They would all have size $|\overline{A'B'''}|$.) Since A and B are each the closest terminus to the other, any module m which covers A or B and some third terminus must have scale $\geq |\overline{A'B'''}|$. However, any module covering more than one terminus, one of which is A or B, and which is higher than the $|\overline{A'B'''}|$-scale level, will have received a paired-below message from the module of the next-lower level that is in that A or B terminus-column. Hence, only those modules of minimum scale which join one terminus to a nearest other terminus have set **in-gap**=1; all other modules have **in-gap**=0.

Finally, step **4.** causes all pixel-scale modules covered by those higher-scale modules with **in-gap**=1 to be labeled likewise. Step **5.** results in all higher-scale modules that cover only pixel-scale modules with **in-gap**=1 (none =0) being assigned **in-gap**=1. Hence, after step **5.**, every module that is inside rectangle $A'A'''B'B'''$, including the edges, will not issue activation instructions during Process **2.** of the optimal algorithm.

After executing the reconstruction routine, the network continues with Process **2.** of the optimal algorithm. Now, however, the activation will produce a result different from before. Our extension establishes that no modules which cover any of the contour, line segments $\overline{AA''}$ or $\overline{A''B}$, or pixels to the right of this boundary can be colored. All modules in the vicinity of the gap whose scale are $\geq |\overline{A'B'''}|$ will function as usual. Some may overlap the gap and become colored, but activation at such large scale levels cannot pass beyond the contour endpoints. By Corollary 1, activation would have to be propagated across this boundary via modules which are entirely inside square $A'A''BB'''$. However, all modules whose scale are $< |\overline{A'B'''}|$ and which are entirely inside rectangle $A'A'''B'B'''$ cannot propagate activation. The modules may themselves be activated, but they cannot send "**colored**=1" messages — they cannot extend activation horizontally through the gap, for instance. At all scale levels, activation which approaches or enters the gap ceases there.

The gap-closing routine runs in constant time, requiring a total of 16 time steps. It preserves the network connectivity degree of $O(N)$. Like for the optimal algorithm, an explicit chain of contingent operations for each module ensures that events are sequenced correctly.

8.3 Implementation Results

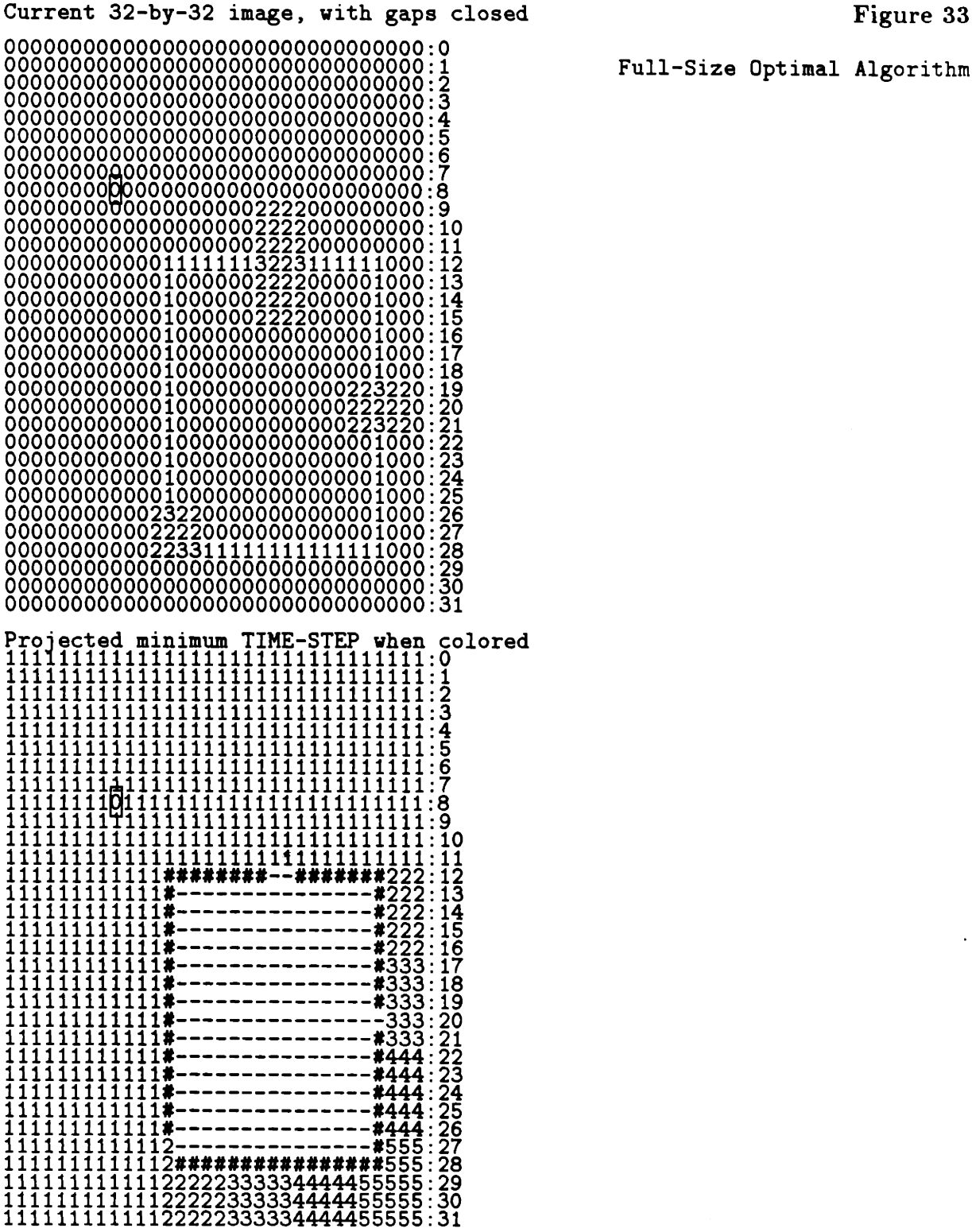
In order to find contour termini, we used a preprocessing routine with a simple criterion: if among a contour pixel's eight 8-connected neighbors, there exist fewer than two contour pixels, or if there are two but they are adjacent, then that pixel is taken to be a terminus. Some example runs are given in Figure 33.

Using square multi-scale region chunks affords us the opportunity to handle broken contours rather elegantly. The coloring boundary that we construct inside the gap becomes increasingly refined as the width of the gap — the ambiguity — decreases. The importance of this scheme is not just that it can handle contrived *subjective* contours consisting of dotted or dashed lines, but that it furnishes our optimal algorithm with some robustness in general. It colors a reasonable-looking area for closed contours with gaps. Generally, the primal sketch will have contours with gaps, and it is required of intermediate processes such as coloring that they gracefully (usefully) handle such raw input.

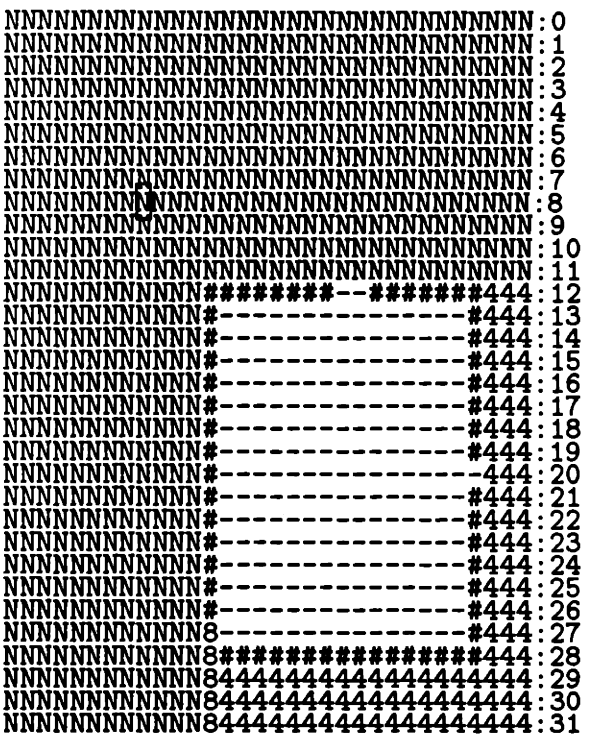
Figure 33: On the following pages are results from both full-size and compact optimal algorithms in which broken contours are closed. In the first map of this Figure, the numbering system is as follows: 1's and 3's are original contour elements, while 2's and 3's are additional boundaries established by closing gaps.

Figure 33

Full-Size Optimal Algorithm



Projected maximum LEVEL with colored module



Full-Size Optimal Algorithm

9 Compact Optimal Coloring inside Moving Contours

9.1 Theoretical Motivation

One desirable property of a coloring network is that its performance be stable. In particular, contours that move are an inevitable part of visual input at some level, and so can pose a difficult challenge to the design of useful algorithms. Contours can translate, rotate and/or flex over time. Figure 40 gives an example: the "Old Contour" changes into the "New Contour" as time advances by some Δt. Since the modules are stationary, only those in the "preserved region" should remain colored at both times. At the later time, the "incoming region" should become colored while the "outgoing region" should become uncolored. We would like an algorithm which can correctly perform this as quickly as possible. It is important to minimize redundant work. For instance, we would hope to avoid generating a whole new description for the latest region, when some of the results of earlier coloring might be used. In addition, we would not want to miss any pixels which should be colored, or leave colored any pixels that are now outside of the contour.

Next, we define our problem domain. We shall assume that the moving contours to be used as input have the following property. Each contour changes position sufficiently slowly between succeeding frames and/or has sufficiently low curvature such that each point in an outgoing region is on a "Maximal Proximity Vector (MPV)." (Elements in incoming regions are not required to be on an MPV. The MPV precondition is precisely stated a few paragraphs below.) Let us again suppose that there exist only two contours, the old one and the new one. An MPV is one which connects a point on one of the contours to the nearest point that is on the other contour. For sufficiently smooth (and isolated) contours, as one moves continuously along a segment of one contour, the nearest point that is on the other contour will move along it continuously. Even if the nearest point on this second contour "doubles back" for some distance, as long as it shifts continuously, an area of MPVs will be swept out. Examples of such areas are shown in Figures 40A and B: "maximal proximity" areas are swept out by moving the point of reference along segments of the Old and New Contour, respectively. We require a pair of contours such that the union of these areas completely fills the regions that need to be uncolored.

This restriction is stricter than necessary but is sufficient to ensure that our network will handle its input correctly. Let us get a precise idea of why it is sufficient and an intuitive idea of why many contour pairs not meeting this condition would still be uncolored correctly.

Figure 34: Walking like an Egyptian for an image.

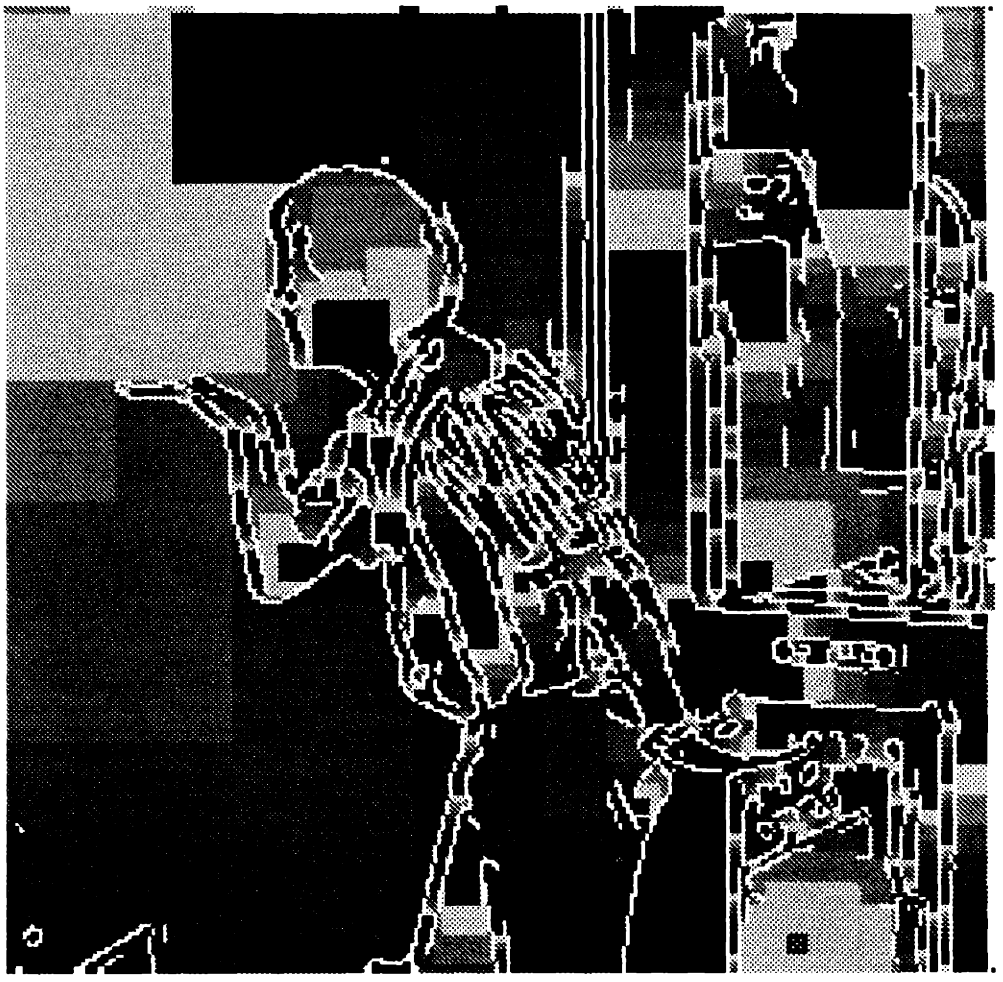


Figure 35: Generally, raw edges from most known edge finding schemes do not completely enclose many meaningful regions for our purposes. Here we see a useless coloring result of the Egyptian edges from the basic compact algorithm. The projected minimum TIME-STEP when colored is indicated by inverse brightness (with wraparound after every N values).

Figure 36: Here the Egyptian edges have been closed by the extended compact algorithm prior to coloring, according to the principles given in the preceding section. This is a more physically meaningful result than before: the elements corresponding to the poster stuck on the door have been labeled. The projected minimum TIME-STEP when colored is indicated by inverse brightness. The small white X's will indicate locations of contour termini below, as well as the starting point for coloring.

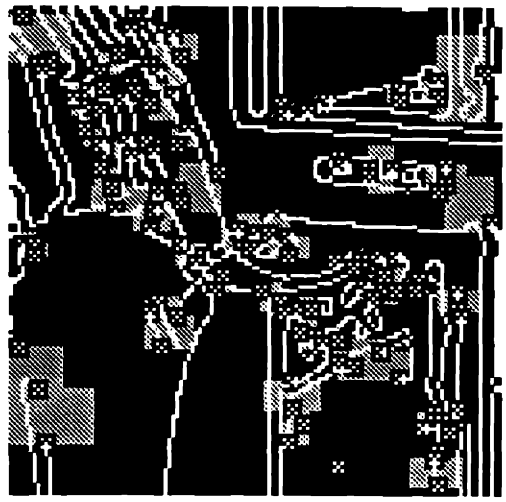


Figure 37: A view of a portion of the Egyptian edges, closed by the extended compact algorithm. The grey areas depict the additional boundaries established by closing gaps. Remember that coloring can procede inside but not past gaps, as long as it does not cross a contour.

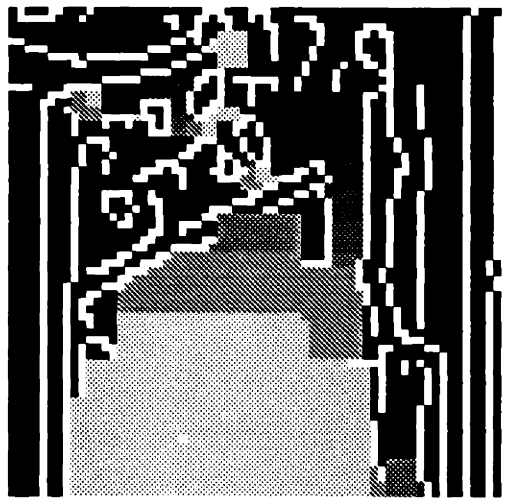


Figure 38: An expanded view of a portion of the Egyptian edges, closed by the extended compact algorithm. The projected minimum TIME-STEP when colored is indicated by inverse brightness.

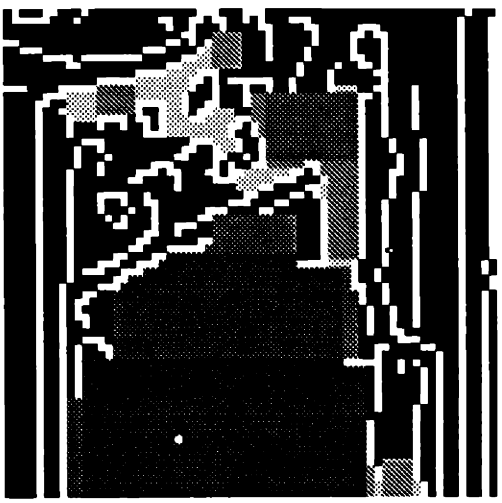


Figure 39: An expanded view of a portion of the Egyptian edges, closed by the extended compact algorithm. The projected maximum LEVEL with colored module is indicated by inverse brightness.

Looking toward Figure 41, we see point P_2, on one contour, joined to the closest point on the other contour, P_1, by vector $\overrightarrow{P_2P_1}$. For the discrete case, imagine also the set of all squares which are parallel to x and y directions, which have sides of length $MAX(d_x, d_y)$, and which inscribe points P_1 and P_2 (including them). The two outermost such squares are drawn in the Figure. If we were able to uncolor all modules that cover any pixels contained by any of these squares (or their boundaries), we would be assured of uncoloring those that lay along $\overrightarrow{P_2P_1}$. Fortunately, the algorithm below does just that: it uncolors all modules of minimum scale that cover P_1 and P_2. Because it uncolors with "excess width," any contour pairs that do not meet our strict precondition are generally uncolored sufficiently in practice. The algorithm extension uncolors all the appropriate modules, plus a few more. We can afford to be a little sloppy (i.e., excessive) when erasing a color, as the activation phase will color all legitimate modules; we cannot be sloppy when coloring.

Let us precisely characterize the property of the edge correspondences that we require, as mentioned above. Figure 42 presents a problematic case of two contour sections whose Maximal Proximity Vectors do not sweep out the entire region in between. As one moves along the bottom contour, the maximal proximity point on the top contour shifts along too, until it reaches the "bump." Here it virtually stops, even as one continues to move along the bottom contour, until it suddenly "jumps" across the bump to the other side

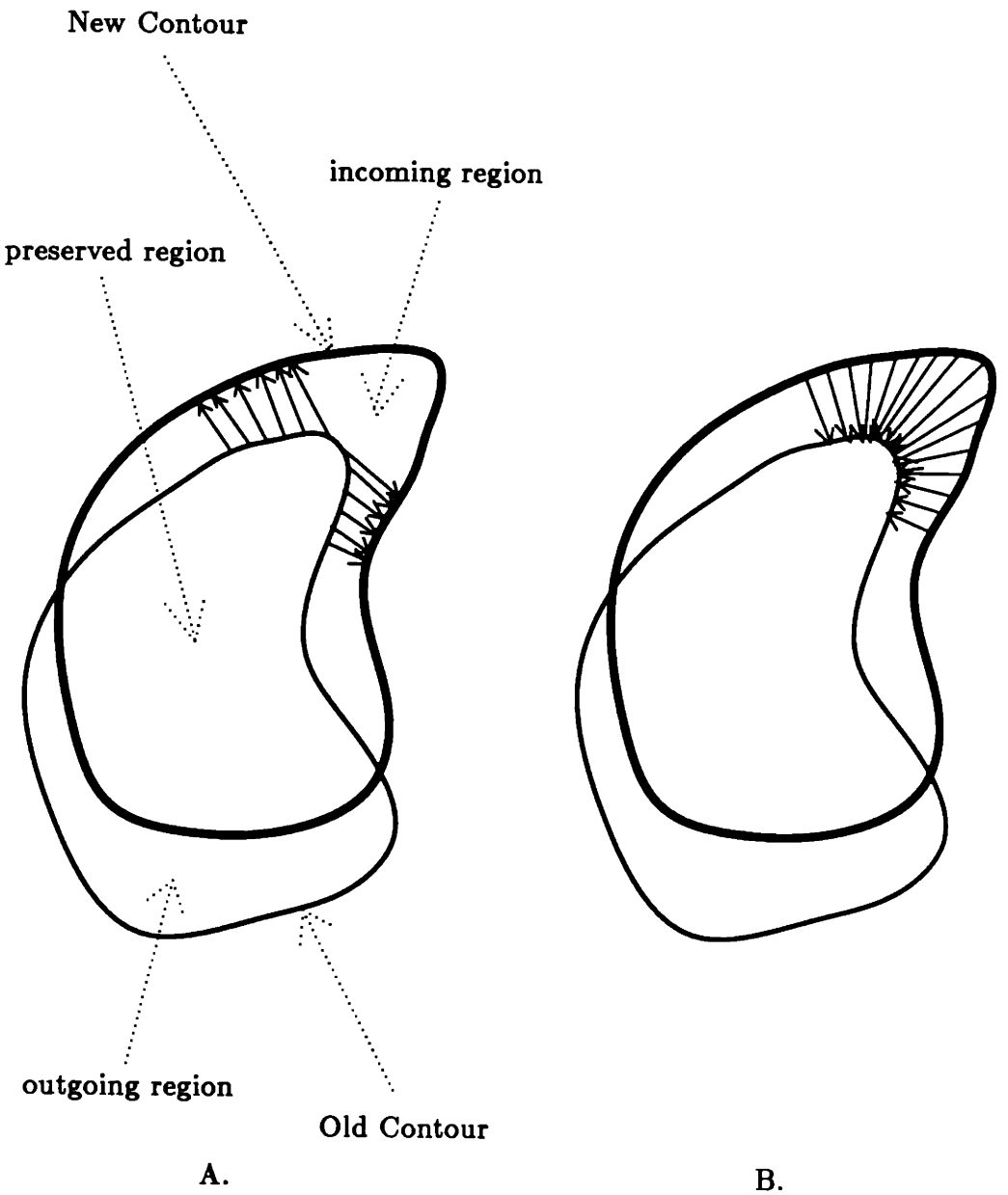


Figure 40: A moving contour. The Old Contour changes into the New Contour during some time period Δt. The preserved region is contained by both contours. The outgoing region is enclosed only by the Old Contour, and the incoming area is enclosed only by the New Contour. The Maximal Proximity Vectors (MPVs) are indicated with rays. In A., the MPVs are from points on the Old Contour to the closest respective points on the New Contour. In B., the MPVs are from the New to the Old Contour.

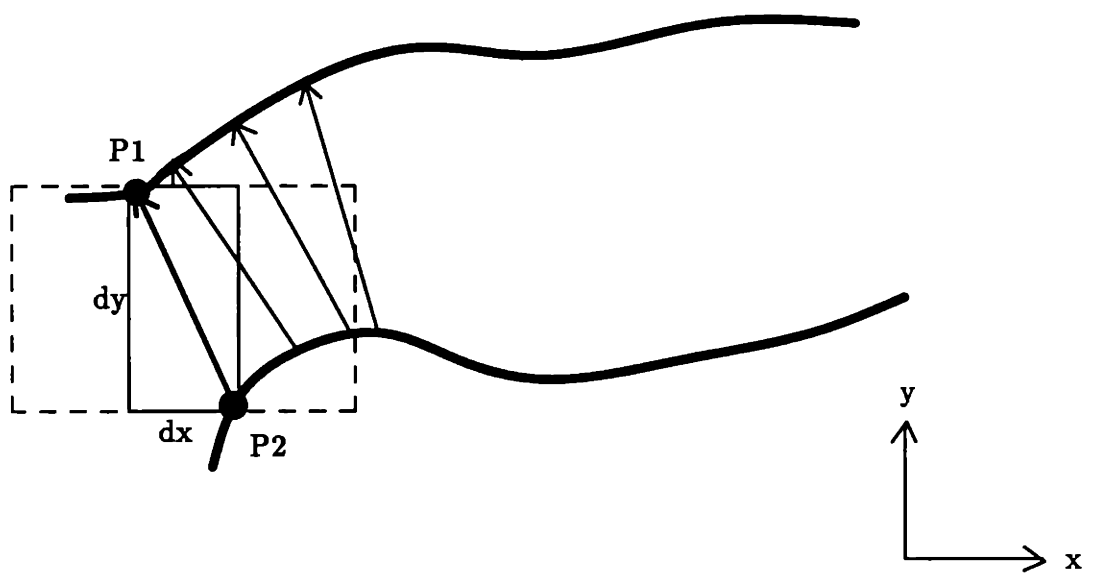


Figure 41: Uncoloring is guaranteed along Maximal Proximity Vectors. This is because any module that covers a pixel contained (or included) by the squares inscribed in P_1 and P_2 as shown will be uncolored by the algorithm extension given below.

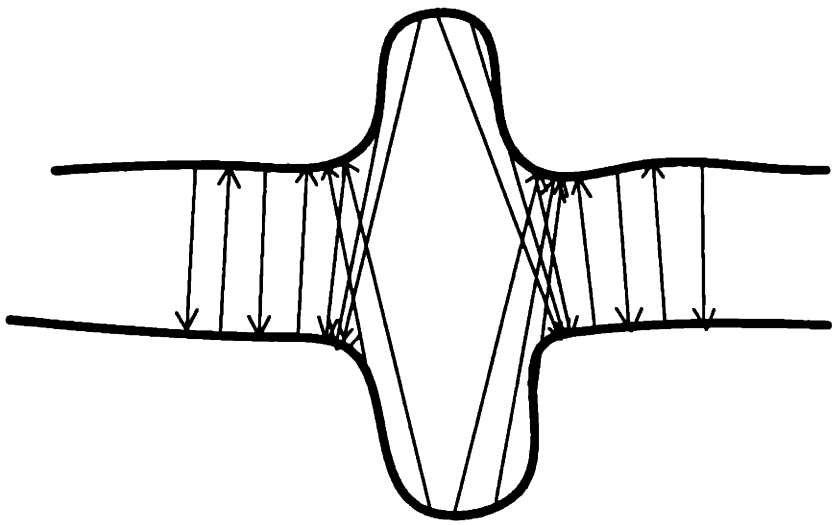


Figure 42: An unerasable pair of contours. There is a discontinuity in the area swept out by both the upward-facing and downward-facing MPVs.

and continues on again in a normal way. The area swept out by the shifting MPV misses a wedge-shaped patch in the middle, and even in union with that area from the shifting MPV of opposite polarity (top to bottom), a diamond shaped patch is missed. In this paragraph and the next, the bottom-to-top versus top-to-bottom configurations are meant to be interchangable, as they are completely symmetric in principle.

We now examine the point on the bottom contour where the MPV shifts discontinuously. In Figure 43, as s_1 moves an infinitesmal distance to the right at d, the MPV's orientation changes abruptly. The question is, under what (hopefully simple) condition will this mysterious event occur? The answer is that the (bottom-to-top) MPV's sweep becomes discontinuous, and a wedge-shaped patch is missed, only if the radius of curvature of the top contour at a point in between the sides of this wedge becomes less than the distance from that point to the origin of the MVP in question (on the bottom contour). Hence,

> We require a pair of contour maps C_1 and C_2 such that the distance from a point on C_1 to the point on C_2 at which the closest preceeding MPV originates never exceeds the radius of curvature of C_1 at that point, and vice versa.

To see why this is so, first note in Figure 43 that nearby points on contour C_2 have MPVs of nearly equal length. If they were significantly different, then the longer MPV would simply point to where the shorter MPV points, and the two would again be nearly the same length. Second, Figure 43 shows an orientation discontinuity between consecutive MPVs that originate near d. No part of C_1 is between $\overrightarrow{de}$ and $\overrightarrow{df}$, and also in the circle, because the two radii are consecutive MPVs. Third, assuming C_1 is continuous between e and f (if they are not, uncoloring this subregion would be irrelevant), then there exists a point $c \in C_1$, between e and f, such that $Curvature[C_1(c)] > \frac{1}{Distance[C_1(c),C_2(d)]}$. We can prove this by circumscribing a "great circle" centered at d about all of C_1 that is between e and f. C_1 may or may not run along this circle tangentially for some distance, but at some point it must turn into the interior of the circle in order to meet at e and f. The radius of curvature of C_1 here must be less than the radius of the great circle. Thus, wherever there is an orientation discontinuity between consecutive MPVs originating from d on C_2, which could possibly result in a patch not being uncolored, there is at least one point c on C_1 between these MPVs whose distance from d exceeds the radius of curvature of C_1 at c.

9.2 Algorithm

The algorithm extension we have derived to do this is essentially the same as that used for closing broken contours, presented above. Briefly stated, we uncolor every module that is due to change its colored/uncolored status (e.g., those inside the Incoming and Outgoing Regions), before continuing with the usual phase of color propagation. As in the last extension, we start with either the full-size or compact optimal algorithm, and employ the following additional hardware. Again, each module (except those at the highest-level) has a single additional connection enabling it to send a message to its one superior module at the next higher scale level. Connectivity degree is again $O(N)$. Fortunately, we can use tokens that are analogous to those in the extension for closing broken contours, although we require two additional tokens. The important reason for this symmetry is that establishing which modules are to be uncolored is a process virtually identical to that of establishing which modules are to be considered in a gap. In this case, of course, there happen to be *many* edge points in one time frame that are to be joined to a nearest edge point in the other time frame, and vice versa.

We use the same tokens as in the last extension, except for three things. First, we substitute "**edge**" in place of "**terminus**" in the token names for clarity. That is, whereas before we had **terminus**, **more-than-1-terminus** and **terminus-column**, we now use

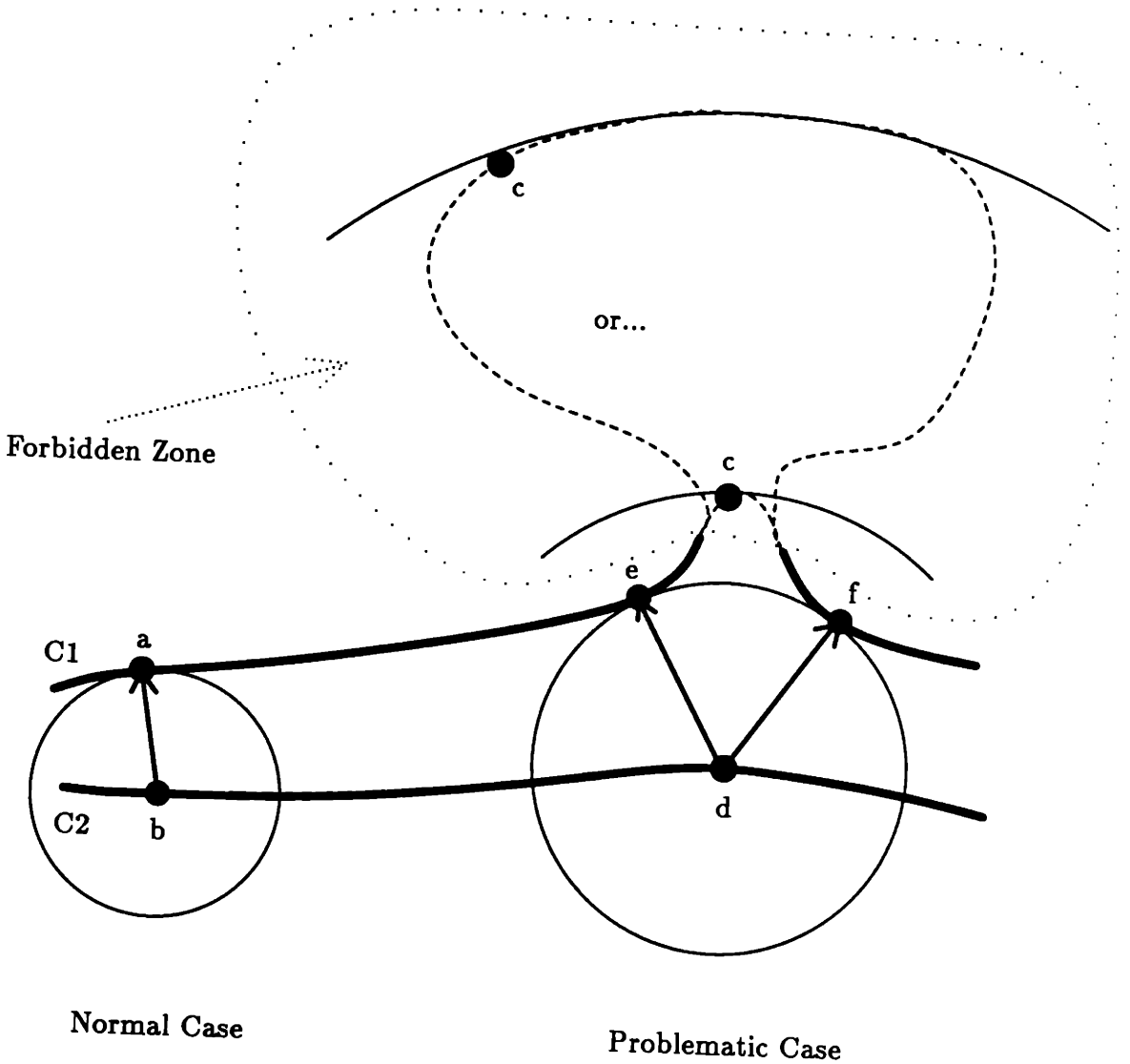


Figure 43: Characterizing the Forbidden Zone. In the Normal Case, $Curvature[C_1(a)] \leq \frac{1}{Distance[C_1(a),C_2(b)]}$. In the Problematic Case, $\exists$ c between e and f (in the Forbidden Zone) such that $Curvature[C_1(c)] > \frac{1}{Distance[C_1(c),C_2(d)]}$.

edge, **more-than-1-edge**[17] and **edge-column**. We also use **paired-below** and **near-unpaired-column**, as before. Second, two more token value are required, **old-edge** and **old-edge-column**, which receive the values of **edge** and **edge-column**, respectively, from the previous run of the extended algorithm. Third, **uncolor-me** will replace **in-gap**. So, the primary variation from the close-gaps routine is that we have **old-edge** and **old-edge-column** tokens, in addition to their current counterparts. However, the two pairs are used conjunctively in a way that is analogous to **terminus** and **terminus-column**, respectively. We use the term **edge** to distinguish it from **contour**, because it receives the OR of **contour** and, optionally, **in-gap**. The extended list of tokens is as follows:

Message Tokens

Basic Algorithm

- contour¶
- colored¶

Extension for Moving Contours

- edge
- old-edge
- edge-column
- old-edge-column
- more-than-1-edge
- paired-below
- near-unpaired-column
- uncolor-me

¶ = global tokens; the rest are temporary

Now we give a precise description of the routine for erasing modules effected by contour motion. It is included in the optimal algorithm inside of step **1.**, after the optional "close gaps" routine. Again, let p denote any pixel-scale module, and m denote any higher-scale module.

Uncolor Modules Which Changed Status During Contour Motion

[17]**More-than-1-edge** means "covers at least one **edge** pixel and at least one **old-edge** pixel."

0. When the p's receive a new image frame in Process **1.** of the optimal algorithm, each module's **old-edge** receives the old values from **edge** before **edge** is updated. **Edge** is then updated with the OR of the latest **contour** and, optionally, **in-gap** from the gap-closing routine. *[One step.]*

1. After the p's **edge** is updated, each performs an Upward Propagation with "**old-edge,**" "**edge**" and "**more-than-1-edge**" to initialize the m's so as to be consistent with the new image; during this Upward Propagation, we do two special actions: after the first (upward) step, set **old-edge-column**=1 wherever **old-edge**=1 and **edge-column**=1 wherever **edge**=1; after the third (outward) step, set "**more-than-1-edge**=1" for any modules in which "**old-edge**=1" and "**edge**=1". *[Four steps.]*

2. At each level, each **more-than-1-edge**=1 module sends a "**more-than-1-edge**=1" message to all modules in its neighboring span of that same level (using the two outward steps — and not the downward step — of the Downward Propagation routine). Any **old-edge-column**=1 module or **edge-column**=1 module that receives such a message sends up "**paired-below**=1" to its superior module in the next higher scale level (while affecting neither its own internal **more-than-1-edge** nor **paired-below** values). Any module receiving a "**paired-below**=1" message sets its internal **paired-below**=1. *[Three steps.]*

3. At each level, each module whose **old-edge-column** or **edge-column**=1 and **paired-below**=0 sends a "**near-unpaired-column**=1" message to all modules in its neighboring span of that same level (using the two outward steps — and not the upward step — of the Upward Propagation routine). Any **more-than-1-edge**=1 module receiving the "**near-unpaired-column**=1" message sets its **uncolor-me**=1. *[Two steps.]*

4. After receiving an **uncolor-me** message, each m performs a Downward Propagation with **uncolor-me**; each p updates its internal value accordingly. *[Three steps.]*

5. After receiving an **uncolor-me** message from above, each p performs an Upward Propagation with **uncolor-me**; each m updates its internal value accordingly. *[Three steps.]*

6. After receiving an **uncolor-me** message from a horizontal direction, each

uncolor-me=1 module sets its **colored**=0. *[One step.]*

The routine works as follows. Steps **0.** and **1.** initialize the modules so as to be consistent with the new image, with **edge**=1 for each module which covers a edge, **more-than-1-edge**=1 for those which cover more than one edge, and **edge-column**=1 for those which have subordinate pixel-scale module whose **edge**=1; all other token values are set to 0. The old values are stored in **old-edge** and **old-edge-column**.

The result of steps **2.** and **3.** is that, for each edge pixel and for each old-edge pixel in the image, only the module(s) in the minimum scale level which barely cover(s) that pixel and an edge/old-edge pixel in the other time frame will remain with **uncolor-me**=1 — all others will have **uncolor-me**=0. For example, suppose Figure 41 is part of an input image. Then, after step **3.**, only the modules which cover both points P_1 and P_2 and whose constituent pixels fall entirely within (or include) the edges of all of the inscribed squares would be labeled as **uncolor-me**=1. (They would all have size $MAX(d_x, d_y)$.) Since P_1 and P_2 are each the closest edge to the other, any module m which covers P_1 or P_2 and any other edge/old-edge of the opposite time frame must have scale $\geq MAX(d_x, d_y)$. However, any module covering more than one edge (i.e., one from each time frame), one of which is P_1 or P_2, and which is higher than the $MAX(d_x, d_y)$-scale level, will have received a paired-below message from the module of the next-lower level that is in that P_1 or P_2 edge-column/old-edge-column. Hence, only those modules of minimum scale which join one edge to a nearest edge of the other time frame have set **uncolor-me**=1; all other modules have **uncolor-me**=0.

Finally, step **4.** causes all pixel-scale modules covered by those higher-scale modules with **uncolor-me**=1 to be labeled likewise. Step **5.** results in all higher-scale modules that cover any pixel-scale modules with **uncolor-me**=1 being assigned **uncolor-me**=1. Hence, after steps **5.** and **6.**, every module that covers any part of the solid rectangle in Figure 41, including the edges, is uncolored prior to entering Process **2.** of the optimal algorithm.

The uncolor routine runs in constant time, requiring a total of 17 time steps. It preserves the network connectivity degree of $O(N)$. Again, an explicit chain of contingent operations for each module ensures that events are sequenced correctly.

9.3 Implementation Results

Figures 44 to 52 below show the result of applying the coloring operation to two slightly different image descriptions. The synthetic edge maps present two identical contour figures.

shifted a small amount in an arbitrary direction. Given this, and with an absence of extraneous contours, we are essentially assured that the MPV precondition imposed on the contour pairs is not violated. The uncoloring extension works as desired here, and preserves a considerable amount of the previous coloring result, most notably in the snake figure. The challenge one must confront when applying the routine to real edge data is getting rid of extraneous nearby contours that are not involved in the physical correspondence. This is usually not a problem if one wishes to link pairs of *strong* edges; by filtering the image using a large mask size, one can eliminate most nearby contours that could possibly confound the mapping. The gap-closing routine can then be applied to the result, prior to uncoloring. Again, recall our disclaimer, which places much of the burden of real world description generation on other powerful low level processes. These should construct the base representation appropriate for coloring without needing much additional computation.

Indeed, in lieu of such a process, we rely on smoothing to remove some of the weaker edges from the desktop image back in Figure 20. We apply the Canny edge finder with a mask size of 14 to the two stereo images. The result of the first frame, after normal coloring, is given in Figures 45 and 46. Then a new edge map is presented, which in this case is the left part of the stereo pair of images. Using the full-size network, the uncoloring extension is performed on these two frames, followed by the regular coloring routine. The result from this should be, and is, that the closed contour of the second frame is completely colored and nowhere else is as well. The colored second frame is shown in Figures 47 and 48. The amount of work saved is given by the black subregion inside the contour in Figure 49. This preserved area would be larger, except for some of the internal contours which map to other to-be internal contours. Perhaps some of those correspondences involving contours that are colored on both sides could be swept away before uncoloring is performed between the remaining contour pairs. Finally, the original lefthand contour map is presented again, uncoloring is performed, and minimum projected TIME-STEP and saved work results are given in Figures 50 and 51. Unfortunately, this time the algorithm fails, illustrating the importance of our precondition. The failure is due to the fact that a few extraneous contours in the lower righthand corner of Figure 50, which paired with some of the "important" contours of the righthand image, caused some colored modules that should have been erased to be left behind. The problem occurs only in a small area where the MPV condition is violated, but the result is over-coloring on a grand scale. In addition, the successful co-occurrence of gap-closing and subregion-uncoloring is demonstrated in Figure 52. The Figures together illustrate the strengths of the work above, and also point to areas where further study is needed.

This extension furnishes our algorithm with some stability in the face of time-varying data. It uncolors a reasonably small yet sufficient area for moving contours. Generally, the primal sketch will have edge maps that shift with time, and it is required of intermediate processes such as coloring that they gracefully (usefully) handle such raw input.

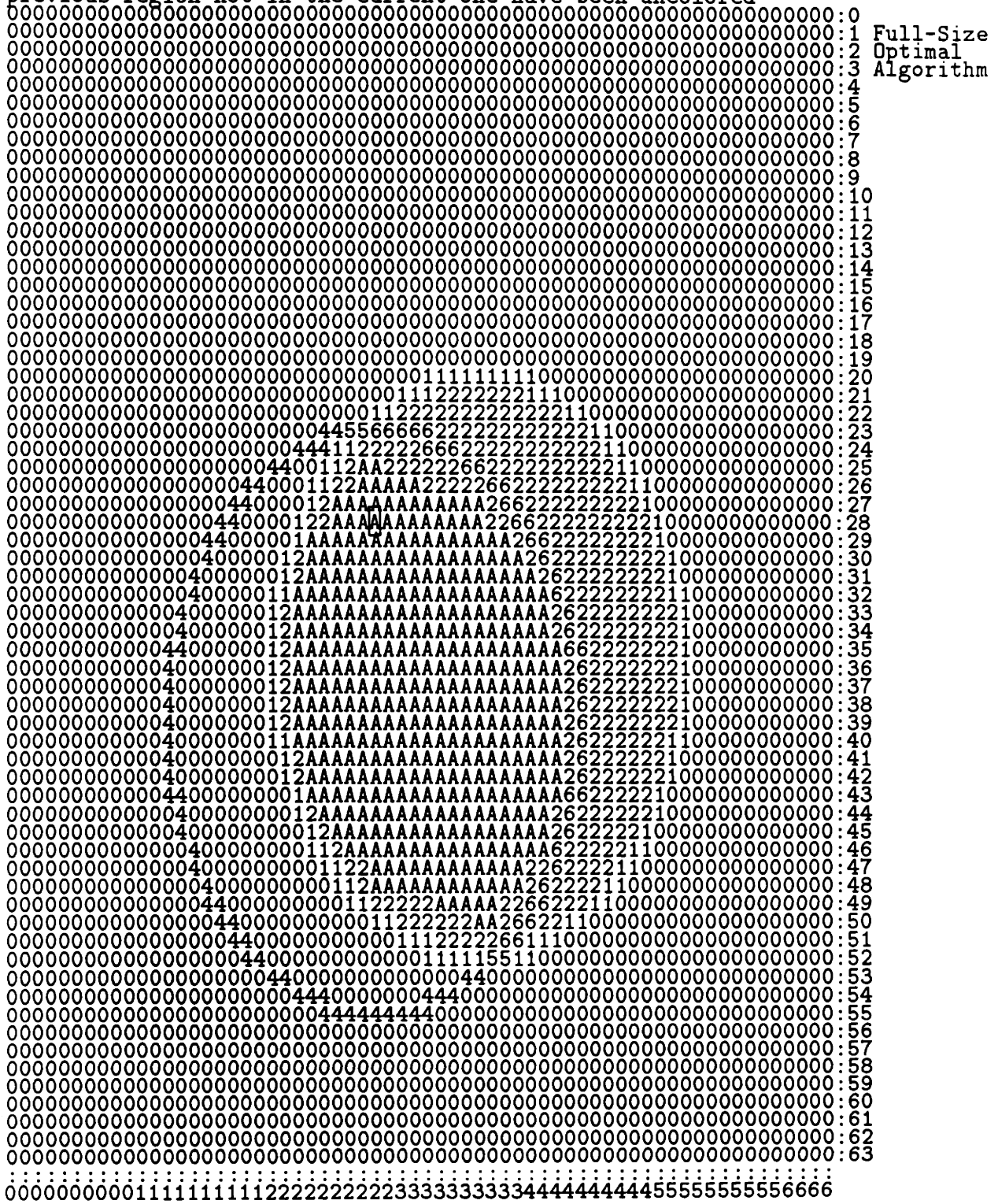
The routine itself can be extended to extract potentially useful information for motion measurement while it runs. Each module with **uncolor-me**=1 — i.e., the minimal-sized modules that cover an edge pixel and an old-edge pixel — can propagate down to its constituent modules this value, which indicates its scale. The pixel-scale modules for which **edge**=1 — i.e., which contain part of the latest contour map (with gaps filled-in) — thereby each receive a value from which it can compute how far away in pixels is the nearest point on the previous contour. This value is the length of the square module which just barely bounds the pair of points. Then, one could do one of two things. Using local contour orientation measurements (relative to the module directions), found by some other means, the values stored in the pixel-scale modules could be used in computing estimates for components of the velocity field, such as $v^{\perp}(s)$, at a cellular level [Hildreth 84], [Horn 86]. On the other hand, if one employed a network having square modules of several orientations, then the **uncolor-me**=1 modules would accurately correspond to the local minimum distance from points on the latest contour map to those on the previous map. Some example runs are given in Figure 44.

Figure 44: On the following pages are results from both full-size and compact optimal algorithms in which areas left behind by moving closed contours are erased prior to the normal coloring routine. In the first map of this Figure, the numbering system is as follows: 1's and 5's are previous contour elements, 4's, 5's and 6's are current contour elements, 2's and 6's are previously colored and now erased, and A's are currently (and were previously) colored, just prior to another coloring cycle.

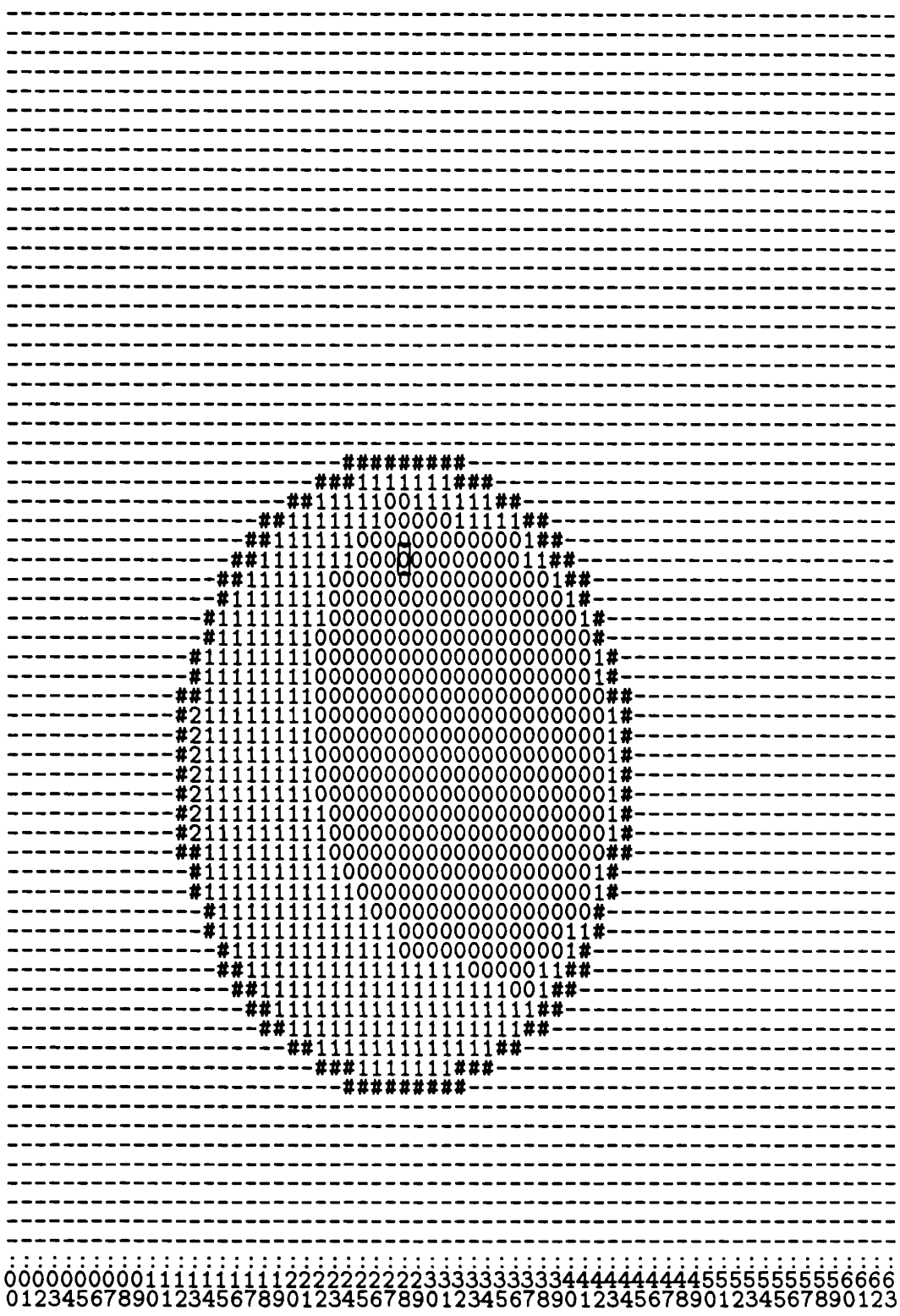
Notice how much previous coloring has been carried over from the earlier frame, as indicated by TIME-STEP=0 in the second and third maps of this Figure. A crooked, elongated region like the snake would be expensive for any chunk-based scheme to have to completely recolor.

Current and previous 64-by-64 images; the parts of the previous region not in the current one have been uncolored

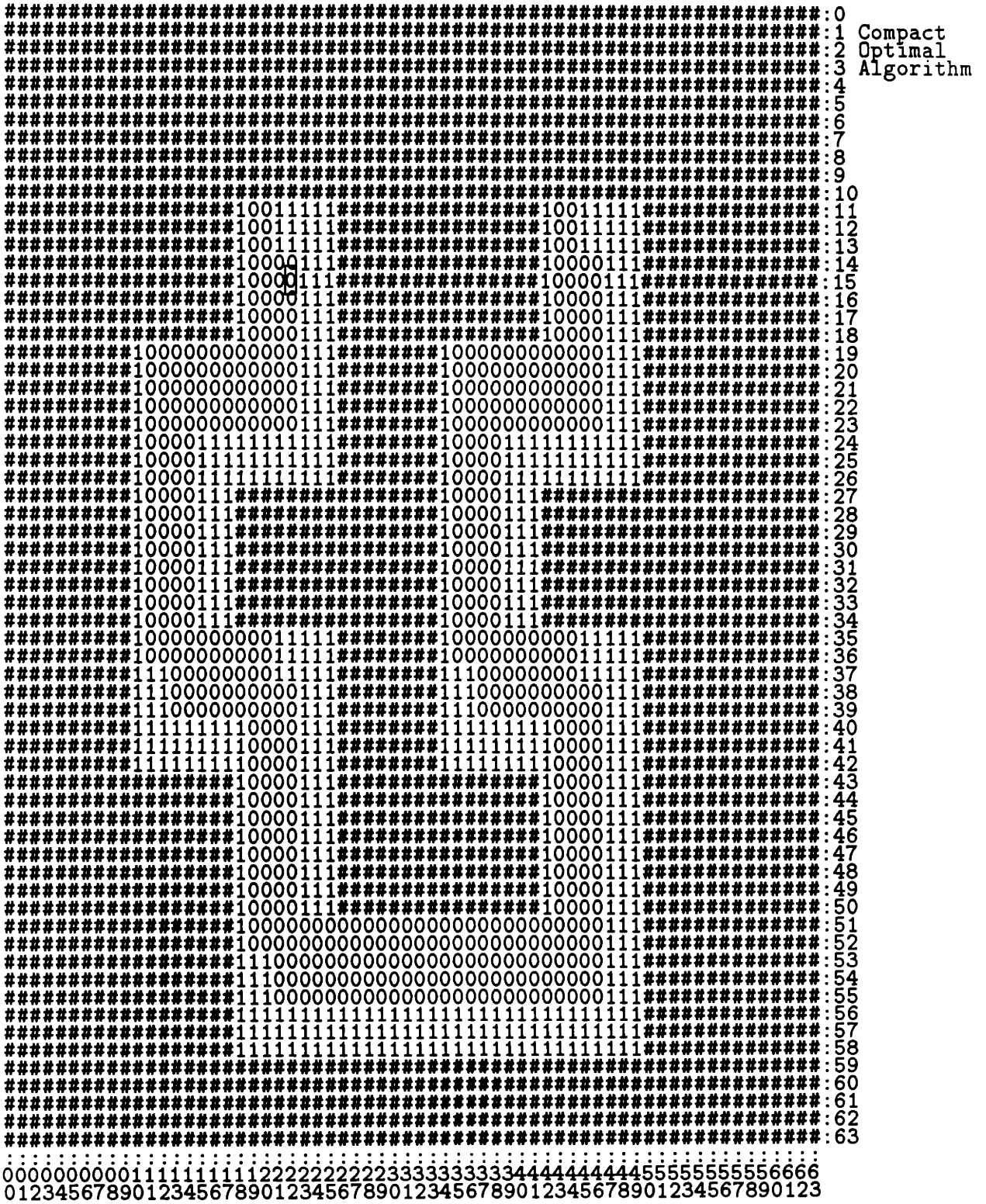
Figure 44



Projected minimum TIME-STEP when colored



Projected minimum TIME-STEP when colored



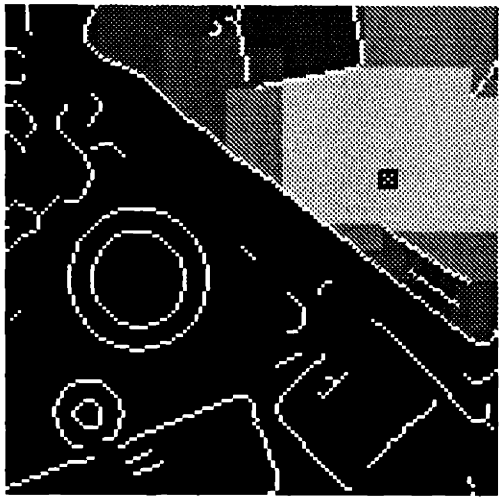


Figure 45: We start by coloring the lefthand edge map. The projected minimum TIME-STEP when colored is indicated by inverse brightness.

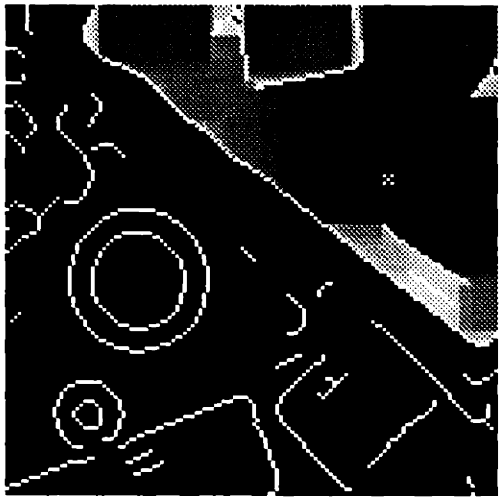


Figure 46: Again, we start by coloring the lefthand edge map. The projected maximum LEVEL with colored module is indicated by inverse brightness.

Figure 47: At the intermediate stage, we color the righthand edge map, after uncoloring obsolete areas in the previously-colored lefthand contour. The projected minimum TIME-STEP when colored is indicated by inverse brightness.

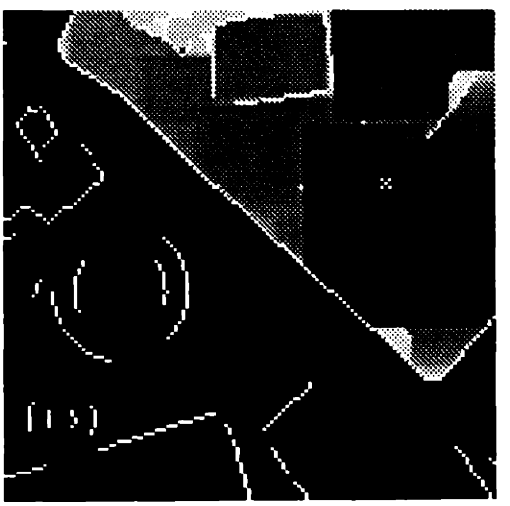


Figure 48: Again, at the intermediate stage, we color the righthand edge map, after uncoloring obsolete areas in the previously-colored lefthand contour. The projected maximum LEVEL with colored module is indicated by inverse brightness.

Figure 49: Black areas inside the closed contour depict the work (time) we have saved in reaching the intermediate stage. The projected minimum TIME-STEP when colored during the second coloring cycle is indicated by inverse brightness.

Figure 50: We finish by coloring again the lefthand edge map, after trying to uncolor all obsolete areas in the previously-colored righthand contour. The projected minimum TIME-STEP when colored is indicated by inverse brightness. This time the color process explodes, starting from the bottom righthand corner of the frame. (Contours in this [lefthand] frame are given in black.)

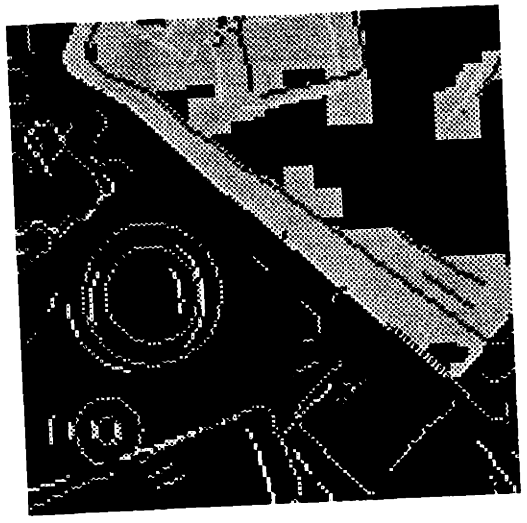


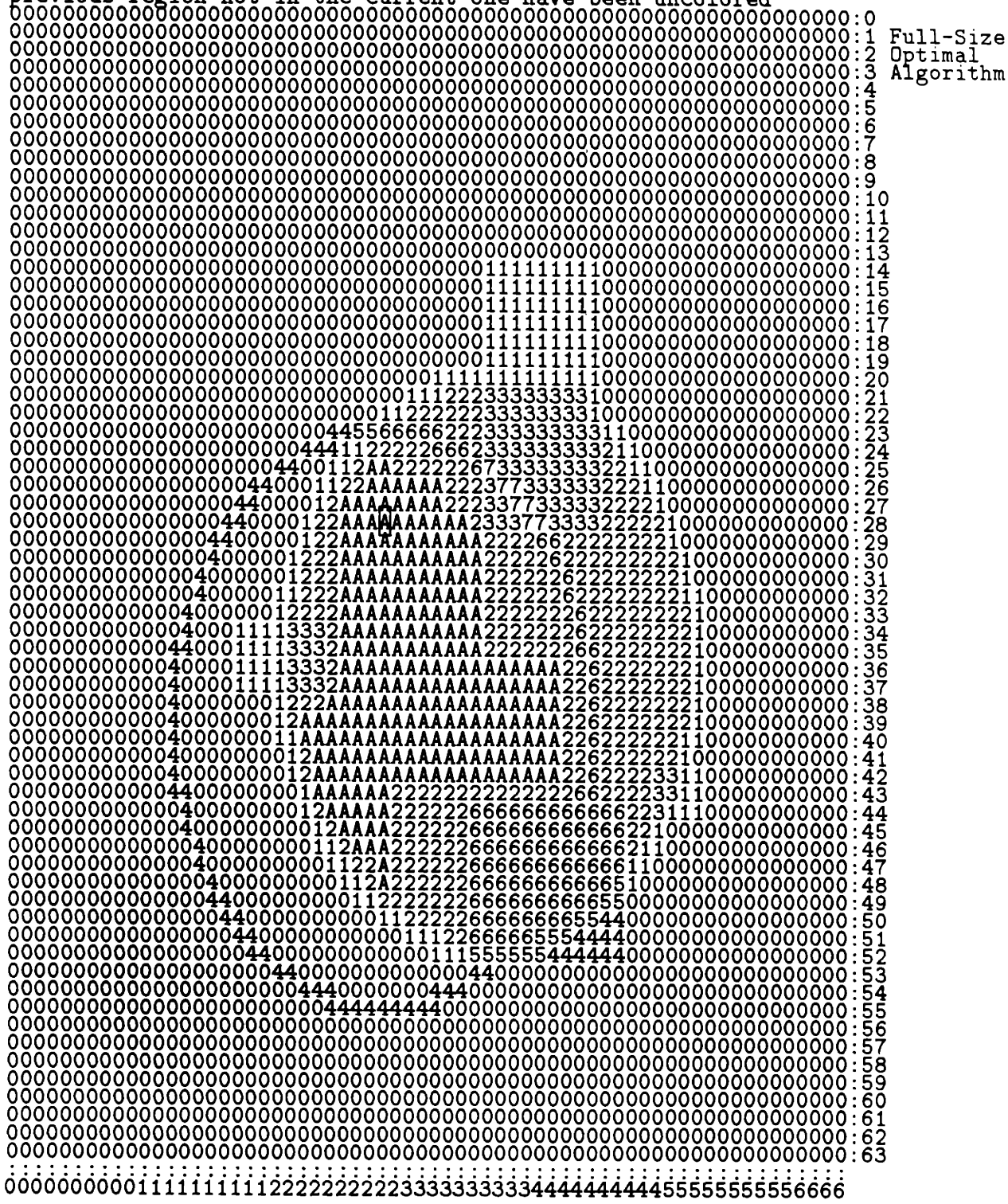
Figure 51: Once again, black areas inside the closed contour depict the work (time) we have saved. Unfortunately, this transition (partly) failed. The projected minimum TIME-STEP when colored during the third coloring cycle is indicated by inverse brightness.

Figure 52: On the following pages are results from the full-size optimal algorithm in which areas left behind by moving contours whose gaps become closed as well are erased prior to the normal coloring routine. In the first map of this Figure, the numbering system is as follows: 1's and 5's are previous contour elements, 4's, 5's, 6's and 7's are current contour elements, 1's, 3's, 4's, 5's and 6's are gaps, 2's, 3's and 6's are previously colored and now erased, and A's are currently (and were previously) colored, just prior to another coloring cycle.

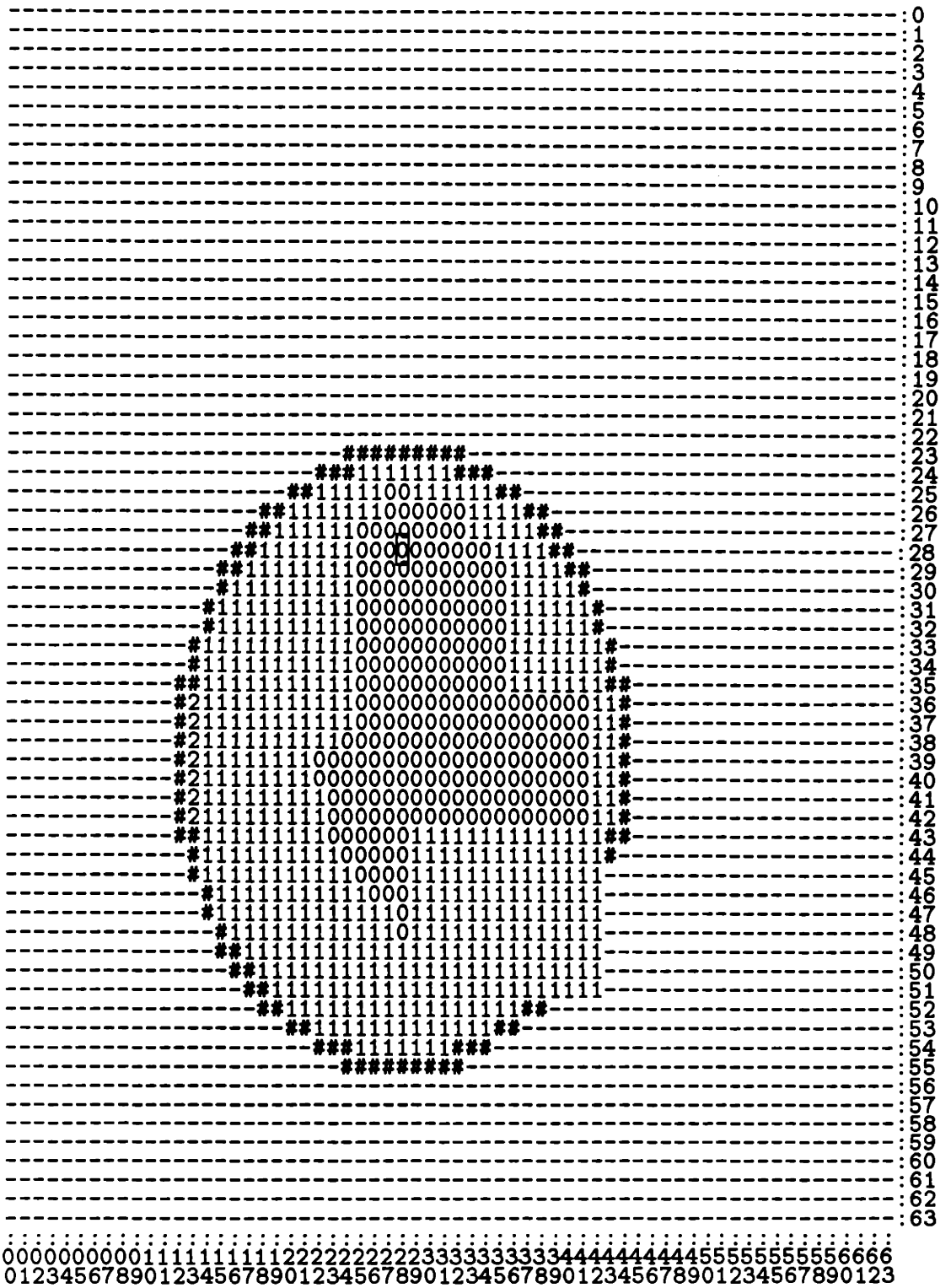
Notice how much previous coloring has been carried over from the earlier frame, as indicated by TIME-STEP=0 in the second map of this Figure.

Current and previous 64-by-64 images; the parts of the previous region not in the current one have been uncolored

Figure 52



Projected minimum TIME-STEP when colored



10 Conclusion

This thesis has sought to develop an understanding of the region coloring operation as an efficient and flexible component of visual routines [Ullman 83]. We provided theoretical motivation for the bounded activation computation, and reviewed its role in the perception of spatial properties and relations. Then, we have investigated it in light of some basic biological constraints and explained why the image chunking representation is appropriate. Next, some fundamental computational limitations for this representation were established. We observed that despite these theoretical bounds the human vision system is able to perform coloring tasks efficiently in many instances.

One potentially fruitful direction for future work would be to try to revise an $O(log\mathcal{N})$-complexity coloring algorithm based on some version of the PRAM model (such as [Lim 86]) — possibly by retiming, and so on — so as to use only local communication and meet other biological constraints as well. Meanwhile, we have designed, implemented and evaluated a set of preliminary coloring algorithms based on the idea of image chunking. Applying the principles learned yielded the optimal region coloring algorithm, using parallel square region chunks. Alteration of this algorithm to optimally employ modules of some other regular given shape, such as "needle-shaped" modules, as well as the proof for its optimality, is straightforward. Then, the flexible, optimal scheme was compacted substantially in terms of space requirements, namely, down to $O(N^2 logN)$ elements, where the base representation is itself $O(N^2)$. Finally, we were able to extend the optimal algorithms to handle cases of broken and moving contours. This affirmed the basic scheme's flexibility and the integratibility of its multi-scale descriptions.

The Connection Machine implementations behave much like human performance. For simple smooth curves, coloring rates for the optimal algorithms are very fast; for complex figures, especially in the vicinity of extraneous contours, it can be considerably slower. Also, there is probably a connection between the effects of motion ambiguity suffered by our uncoloring extension, in matching the wrong pair of contours according to the weak "proximity constraint," and certain psychophysical illusions based on similar causes (e.g., the wagonwheel turning backwards in a movie) [Hildreth 84]. Due to the network's configuration, the coloring times are invariant to figure position, and approximately so with respect to orientation. For many simpler curves, the coloring time does not increase greatly with increase figure scale. These traits generally coincide with what is understood about human performance.

These factors together suggest that the compact optimal region coloring model, and

possibly one or more of its extensions, comprise a sound model for human vision.

A Appendix: Retiming Can Reduce Connectivity

Sometimes bending the rules pays off. For instance, consider the case in which one would like a pixel-scale module to update all modules that cover it and yet do not cover a contour. Rather than passing a message directly to each of these modules and checking for proper conditions in one step (a proposition implying $O(N^3)$-connectivity), one could envision passing data through a few modules, checking for correct conditions along the way, then continuing on to a few more modules from each of these, checking again, and so on, with limited branching. This would appear to contain our connectivity degree. However, we know that we would not be able to perform this stream of actions in one time step (or a constant number of steps).

Fortunately, the systolic conversion theorem [Leighton et al. 88] enables us to convert such a semisystolic array into an equivalent systolic network through a series of local retiming steps. Both types of networks are composed of processors, which in turn consist of combinatorial logic and registers. Refer to Figure 53. The combinatorial logic components perform the calculations, while the registers provide the memory and the timing. At the beginning of each time step, data passes through the network (including zero or more combinatorial logic components, which may alter it) until registers are reached. All data are stored in registers, at the end of the step. The two models of computation differ because the processors differ as follows. For the *systolic processor*, all the data that enters the processor passes through the combinatorial logic component and then is stored in one of the registers within the processor before the end of the time step. For the *semisystolic processor*, the data that enters the processor passes through the combinatorial logic component and then may or may not end up in one of the registers within the processor before leaving the processor. Informally, the major difference is that in a semisystolic array, a processor may broadcast to any number of other processors at any distance in one time step, while in a systolic array, only a limited number of local processors can receive the message. It is easier to design and implement parallel algorithms in terms of semisystolic arrays, partly because broadcasting is allowed.

Based on the systolic conversion theorem, a method called *retiming* mechanically transforms semisystolic networks into systolic networks. First, we represent the semisystolic network as a weighted graph. Each node corresponds to a combinatorial logic component,

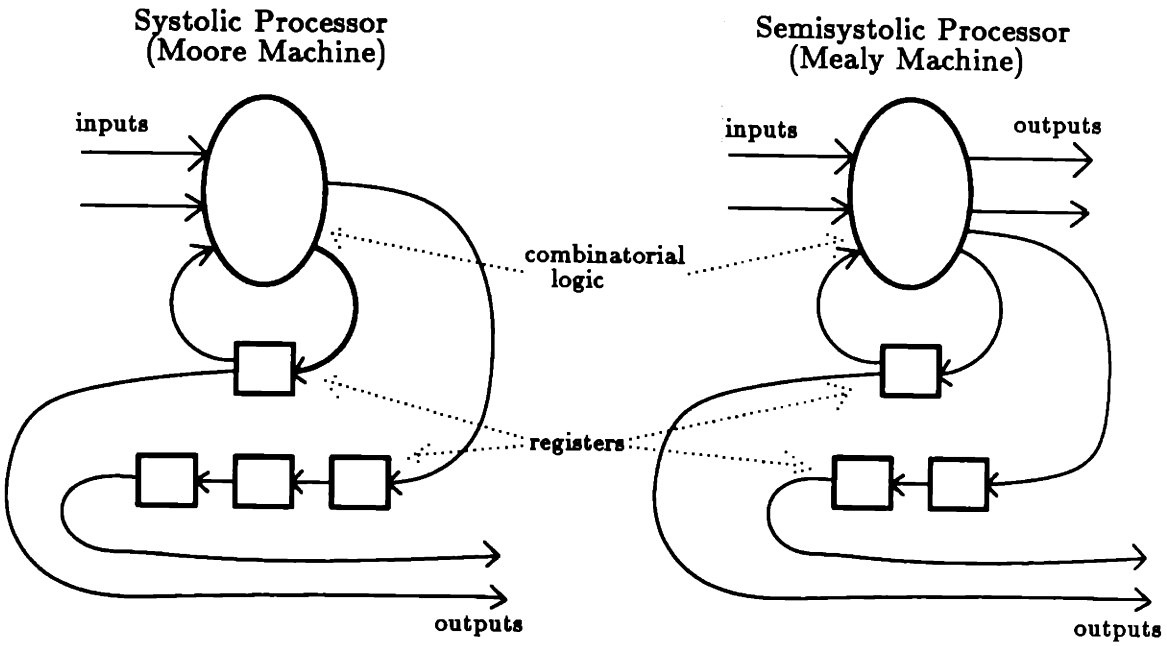


Figure 53: Systolic versus semisystolic processors. (Due to [Leighton et al. 88].)

each edge corresponds to a connection between such components, and the weight of the edge corresponds to the number of registers along that link. The systolic conversion theorem states that *"Given a semisystolic network S with graph G, there is an acceptable global lag function such that the retimed network is systolic* $\Longleftrightarrow$ *the graph G - 1 has no negative-weight cycles."* The global lag function is defined by $lag(v)$ = weight of least weight path from v back to the host in G - 1. Retiming then consists of shifting the registers around so as to eliminate zero-weight edges. Figure 54 shows the two ways of lagging a node.

A rigorous review of semisystolic-to-systolic conversion would be straightforward, but beyond the scope of this thesis (but see [Leighton et al. 88]), especially since we give complete descriptions of our own algorithms above. However, the important points are illustrated in the simple example below: conversion of a network for palindrome recognition. A palindrome is a string of the form $\omega\omega^r$, where ω^r is just the substring ω in reverse order. We would like an algorithm which inputs the string $x_1, x_2, ..., x_N$ one element at a time, and after each element x_i is input ($1 \leq i \leq N$), tells us whether $x_1, x_2, ..., x_i$ is a palindrome. Figure 55 shows the steps in the algorithm of the Palindrome Recognizer. Figure 56 shows the weighted graph for the semisystolic network. Figure 57 shows the semisystolic network. Figure 58 shows the systolic network, resulting from conversion of the semisystolic one. The

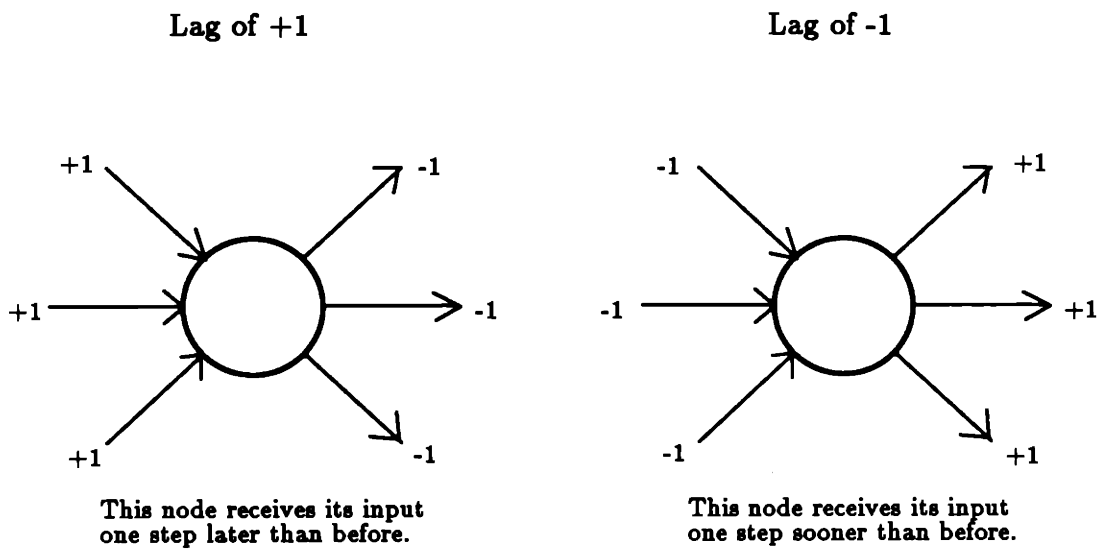


Figure 54: Lagging a node. A. We lag a node by +1 by adding a register to each incoming edge and removing a register from each outgoing edge. B. We lag a node by -1 by removing a register from each incoming edge and adding a register to each outgoing edge.

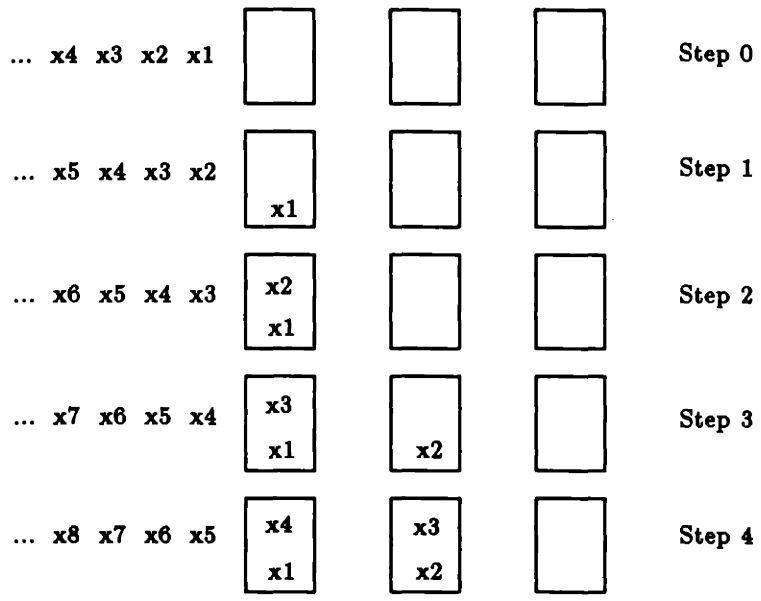


Figure 55: Steps in the recognition of a palindrome.

final network in this case consists of two independent sets of registers. Thus, two problems may be pipelined (interleaved), each running at half speed. Notice what has happened: in the semisytolic form, it would be necessary either to pass an accumulating logical result through several processors in one step, or one would need to read and decode local values from a number of output pads. Instead, retiming has enabled us to use a limited number of input/output pads (one in this case), and still proceed about as quickly as other more obvious algorithms that require more [J. Ullman 84]. In this work, we have formulated an $O(N^3)$-degree algorithm in terms of an $O(N)$-degree semisystolic network, and then, by retiming, converted this into a lagged, systolic network. The additional registers needed coincide exactly with the already existing processors.

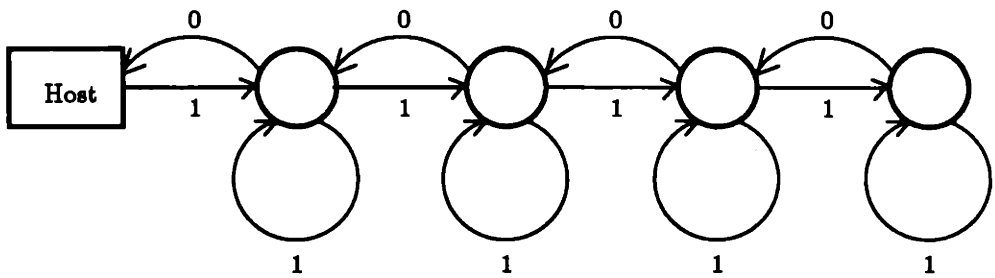


Figure 56: Weighted graph of the semisystolic network of the palindrome recognizer.

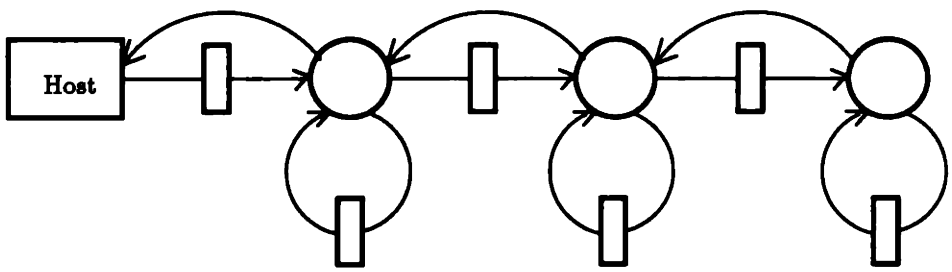


Figure 57: Semisystolic network of the palindrome recognizer.

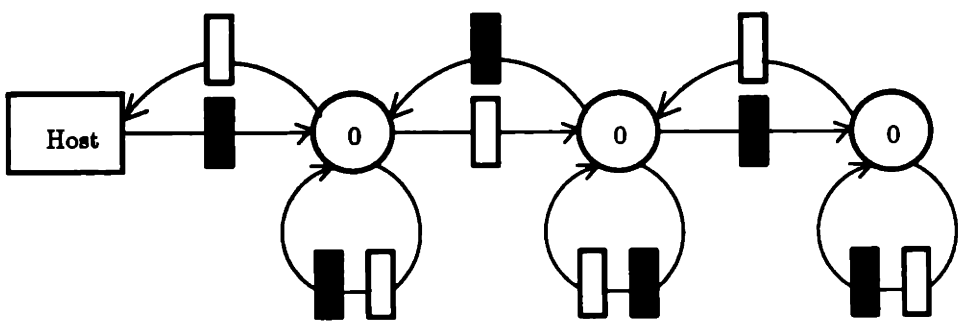


Figure 58: Systolic network of the palindrome recognizer.

References

[Ballard & Brown 82] D.H. Ballard, C.M. Brown. *Computer Vision.* Englewood Cliffs, NJ: Prentice-Hall, 1982.

[Beymer 89] D. Beymer. "Junctions: Their Detection and Use for Grouping in Images," MS Thesis, Department of Electrical Engineering and Computer Science, MIT, May 1989.

[Edelman 85] S. Edelman. "Fast Distributed Boundary Activation," MSc Thesis in Computer Sciences, Weizmann Institute of Science (Rehovot, Israel), June 1985.

[Finke & Pinker 82] R.A. Finke and S. Pinker. "Spontaneous Imagery Scanning in Mental Extrapolation," *Journal of Experimental Psychology: Learning, Memory, & Cognition*, **8**, 142 - 147, 1982.

[Fleck 88] M. Fleck. "Boundaries and Topological Algorithms," PhD Thesis, Department of Electrical Engineering and Computer Science, MIT, September 1988.

[Hildreth 84] E.C. Hildreth. *The Measurement of Visual Motion.* Cambridge, MA and London: The MIT Press, 1984.

[Horn 86] B.K.P. Horn. *Robot Vision.* Cambridge, MA and London: The MIT Press, 1986.

[Jolicoeur 87] P. Jolicoeur. "A Size Congruency Effect in Memory for Visual Shape," *Memory & Cognition*, **15**, 531 - 543, 1987.

[Jolicoeur et al. 86] P. Jolicoeur, S. Ullman and M. Mackay. "Curve Tracing: A Possible Basic Operation in the Perception of Spatial Relations," *Memory & Cognition*, **14**, 129 - 140, 1986.

[Jolicoeur & Kosslyn 85] P. Jolicoeur and S.M. Kosslyn. "Is Time to Scan Visual Images Due to Demand Characteristics?" *Memory & Cognition*, **13**, 320 - 332, 1985.

[Kandel & Schartz 85] E.R. Kandel and J.H. Schwartz. *Principles of Neural Science*, 2d ed. New York, NY: Elsevier Science Publishing Co., Inc., 1985.

[Leighton et al. 88] T. Leighton, C.E. Leiserson, B. Maggs, S. Plotkin, J. Wein. "Theory of Parallel and VLSI Computation," Lecture Notes for 18.435/6.848, Laboratory for Computer Science, MIT, March 1988.

[Lim 86] W. Lim. "Fast Algorithms for Labeling Connected Components in 2-D Arrays," Report No. NA86-1, Thinking Machines Corporation (Cambridge, MA), 1986.

[Mahoney 87] J.V. Mahoney. "Image Chunking: Defining Spatial Building Blocks for Scene Analysis," MS Thesis, Department of Electrical Engineering and Computer Science, MIT, January 1987.

[Marr 82] D. Marr. *Vision: A Computational Investigation into the Human Representation and Processing of Visual Information.* New York: W. H. Freeman and Company, 1982.

[Minsky & Papert 88] M. Minsky and S. Papert. *Perceptrons.* Expanded ed. Cambridge, MA and London: The MIT Press, 1988.

[Reed et al. 83] S.K. Reed, H.S. Hock and G.R. Lockhead. "Tacit Knowledge and the Effect of Pattern Configuration on Mental Scanning," *Memory & Cognition*, **11**, 137 - 143, 1983.

[Remington & Pierce 84] R.W. Remington and L. Pierce. "Moving Attention: Evidence for Time-Invariant Shifts of Visual Selective Attention," *Perception & Psychophysics*, **35**, 393 - 399, 1984.

[Riley 81] M.D. Riley. "The Representation of Image Texture," SM Thesis, Department of Electrical Engineering and Computer Science, MIT, 1981.

[Shafrir 85] A. Shafrir. "Fast Region Coloring and the Computation of Inside/Outside Relations," MS Thesis, Department of Applied Mathematics, Weizmann Institute of Science (Rehovot, Israel), May 1985.

[Shiloach & Vishkin 1982] Y. Shiloach and U. Vishkin. "An $O(logN)$ Parallel Connectivity Algorithm," *Journal of Algorithms.*, **3**, 56 - 57, 1982.

[Shulman et al. 79] G.L. Shulman, R.W. Remington and J.P. McLean. "Moving Attention through Visual Space," *Journal of Experimental Psychology: Human Perception and Performance*, **5**, 522 - 526, 1979.

[Tsal 83] Y. Tsal. "Movements of Attention across the Visual Field," *Journal of Experimental Psychology: Human Perception and Performance*, **9**, 523 - 530, 1983.

[J. Ullman 84] J. Ullman. *Computational Aspects of VLSI.* Rockville, MD: Computer Science Press, Inc., 1984.

[Ullman & Sha'ashua 88] S. Ullman and A. Sha'ashua. "Structural Saliency: The Detection of Globally Salient Structures Using a Locally Connected Network," A.I. Memo No. 1061, Artificial Intelligence Laboratory, MIT, July 1988.

[Ullman 76] S. Ullman. "Filling in the Gaps: The Shape of Subjective Contours and a Model for their Generation," *Biological Cybernetics*, **25**, 1 - 6, 1976.

[Ullman 83] S. Ullman. "Visual Routines," A.I. Memo No. 723, Artificial Intelligence Laboratory, MIT, June 1983.

[Varanese 83] J. Varanese. "Abstracting Spatial Relations from the Visual World," BSc Thesis in Neurobiology and Psychology, Harvard University, 1983.