Technical Report 980

Image Chunking: Defining Spatial Building Blocks for Scene Analysis

James V. Mahoney

MIT Artificial Intelligence Laboratory

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AI-TR 980	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Image Chunking: Defining Spatial Building Blocks for Scene Analysis		5. TYPE OF REPORT & PERIOD COVERED technical report
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) James V. Mahoney		8. CONTRACT OR GRANT NUMBER(s) DACA76-85-C-0010 N00014-85-K-0124
9. PERFORMING ORGANIZATION NAME AND ADDRESS Artificial Inteligence Laboratory 545 Technology Square Cambridge, MA 02139		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Advanced Research Projects Agency 1400 Wilson Blvd. Arlington, VA 22209		12. REPORT DATE August 1987
		13. NUMBER OF PAGES 188
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) Office of Naval Research Information Systems Arlington, VA 22217		15. SECURITY CLASS. (of this report)
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Distribution is unlimited.

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

18. SUPPLEMENTARY NOTES

None

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

machine vision
chunking
segmentation
tracing
blob detection
image understanding
visual routines
region growing

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

Rapid judgements about the properties and spatial relations of objects are the crux of visually guided interaction with the world. Vision begins, however, with essentially pointwise representations of the scene, such as arrays of pixels or small edge fragments. For adequate time-performance in recognition, manipulation, navigation, and reasoning, the processes that extract meaningful entities from the pointwise representations must exploit parallelism. This report develops a framework for the fast extraction of

Block 20 cont.

scene entities, based on a simple, local model of parallel computation.

An image chunk is a subset of an image that can act as a unit in the course of spatial analysis. A parallel preprocessing stage constructs a variety of simple chunks uniformly over the visual array. On the basis of these chunks, subsequent serial processes locate relevant scene components and assemble detailed descriptions of them rapidly. This thesis defines image chunks that facilitate the most potentially time-consuming operations of spatial analysis - boundary tracing, area coloring, and the selection of locations at which to apply detailed analysis. Fast parallel processes for computing these chunks from images, and chunk-based formulations of indexing, tracing and coloring, are presented. These processes have been simulated and evaluated on the lisp machine and the connection machine.

Image Chunking:
Defining Spatial Building Blocks for Scene Analysis

James V. Mahoney

Revised version of a thesis submitted to the Department of Electrical Engineering and Computer Science on January 30, 1987 in partial fulfillment of the requirements for the degree of Master of Science.

This report describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the laboratory's artificial intelligence research is provided in part by the Advanced Research Projects Agency of the Department of Defense under Army contract number DACA76-85-C-0010, and in part by DARPA under Office of Naval Research contract N00014-85-K-0124.

Abstract

Rapid judgements about the properties and spatial relations of objects are the crux of visually guided interaction with the world. Vision begins, however, with essentially pointwise representations of the scene, such as arrays of pixels or small edge fragments. For adequate time-performance in recognition, manipulation, navigation, and reasoning, the processes that extract meaningful entities from the pointwise representations must exploit parallelism. This report develops a framework for the fast extraction of scene entities, based on a simple, local model of parallel computation.

An image chunk is a subset of an image that can act as a unit in the course of spatial analysis. A parallel preprocessing stage constructs a variety of simple chunks uniformly over the visual array. On the basis of these chunks, subsequent serial processes locate relevant scene components and assemble detailed descriptions of them rapidly. This thesis defines image chunks that facilitate the most potentially time-consuming operations of spatial analysis—boundary tracing, area coloring, and the selection of locations at which to apply detailed analysis. Fast parallel processes for computing these chunks from images, and chunk-based formulations of indexing, tracing, and coloring, are presented. These processes have been simulated and evaluated on the lisp machine and the connection machine.

Acknowledgements

Shimon Ullman, my thesis supervisor, was an inexhaustible source of ideas, direction and support. I cherish his vision and friendship, without which this thesis would not exist.

Tomaso Poggio was constantly enthusiastic about my work and very generous with his time.

I had many very helpful discussions relating to this research with Harry Voorhees, Eric Saund, David Clemens and Anselm Spoerri. In the process these people have become some of my closest friends.

I am grateful to many other people at the M.I.T. Artificial Intelligence Laboratory for inspiring conversations and/or moral support, especially Ellen Hildreth, Phil Agre, Dan Huttenlocher, Jim Little, David Chapman, Dave Braunegg, and Margaret Fleck.

Thanks to my family at home—Matt BenDaniel, Melanie Meyers, Debbie Field, and Brian and Joo Hooi Albriton—and abroad—Gladys Sumner and Hannah Mahoney.

Without Patrick Winston, Berthold Horn, Charles Fugler, Carlos Perez, and, most of all, the late Victoria Scott-Sawyerr, I wouldn't have had this opportunity.

Contents

1 Image chunking and the analysis of spatial information **8**

1.1 The general requirements of spatial analysis 12

1.2 Visual routines 14

1.3 Basic operations 16

1.4 Parallel models of computation for vision 20

1.5 Prefabrication—a metaphor for chunking 22

1.6 Image chunks 23

1.7 Image chunking for the rapid extraction of scene entities 24

2 Local boundary information—the input to chunking processes **26**

2.1 The need to detect boundaries 29

2.2 Detecting abrupt change boundaries by direct comparisons between neighboring elements 31

2.2.1 Computing a primitive element's neighbors 32

2.2.2 Computing property differences 34

3 Local prominence information **38**

3.1 Redundancy and local prominence information 38

3.1.1 Redundancy; figures and background 38

3.2 Computing local prominence information. 40

3.2.1 Defining boundary segments. 42

3.2.2 Computing boundary segment properties. 43

3.2.3 Computing prominence. 44

3.2.4 Combining prominence information and boundary information. . . . 47

4 Figural chunking **48**

4.1 The need for low-level descriptors of objects 49

4.2 Indexable visual objects 53

4.3 Figural chunks 55

4.4 Detecting figural chunks 58

4.4.1 Measuring fit and relative isolation 58

4.4.2 Spatially uniform, parallel application 59

4.4.3 Performance 61

4.5 Indexing figural chunks 64

4.6 Implications 71

5 Curve chunking **75**

5.1 Curve chunks 76

5.2 Limited-extent curve chunking 85

5.2.1 Serial, limited-extent validity checking 86

5.2.2 Parallel, limited-extent validity checking 88

5.2.3 A limited-extent curve chunking algorithm 91

5.2.4 Performance 93

5.3 Direct, unlimited-extent curve chunking 96

5.3.1 A parallel, unlimited-extent validity check 97

5.3.2 Regions 101

5.3.3 A direct, unlimited-extent curve chunking algorithm 102

5.3.4 Performance 102

5.4 Divide-and-conquer, unlimited-extent curve chunking 105

5.4.1 The pyramid representation 107

5.4.2 The rule for merging regions 109

5.4.3 A divide-and-conquer curve chunking algorithm 112

5.4.4 Performance 115

5.5 Two-label, divide-and-conquer, unlimited-extent curve chunking 128

5.5.1 The two-label divide-and-conquer curve chunking algorithm 131

5.5.2 Performance 137

5.6 Summary of the four curve chunking techniques 143

5.7 Implications 145

6 Region chunking **148**

6.1 Region chunks 149

6.2 Limited-extent region chunking 155

6.3 Direct, unlimited-extent region chunking 156

6.4 Divide-and-conquer, unlimited-extent region chunking 156

7 Comparisons with other work **158**

7.1 Grounds for comparison with other image articulation schemes 159

7.2 The input and output of image articulation 160

7.3 Two-stage image articulation 161

7.4 Region-based image articulation schemes 162

7.4.1 Non-descriptive schemes: region segmentation 163

7.4.2 Descriptive schemes: blob detection 164

7.5 Boundary-based image articulation schemes 166

7.5.1 Non-descriptive schemes: boundary segmentation 166

7.5.2 Descriptive schemes: spine transforms 168

7.6 Sibling studies of fast coloring 170

7.6.1 Line coloring 170

7.6.2 Region coloring 171

8 Conclusion **175**

Chapter 1

Image chunking and the analysis of spatial information

Visual judgements about the properties and spatial relations of objects are the crux of our interactions with our surroundings. We perceive and conceive of the world in terms of objects and configurations, which we recognize, handle, navigate by, and reason about. Our primary source of information about these spatial entities and relations is vision. The visual system makes this information available in a manner that leaves the subjective impression of immediate, complete, effortless awareness. For example, you might look up from this text for a moment, reach for your cup, and take a drink, with hardly a thought. For that matter, vision transparently discerns the words and phrases you are reading, while your conscious thoughts are focused on their meaning.

Visual processing begins with an image array of pointwise measurements of light intensity. The physical entities in terms of which we conceive of our surrounds may have widely varying spatial extent, so they are not, in general, explicitly described in the image array or any other pointwise scene descriptions derived from it. Moreover, meaningful scene entities may appear in a very wide range of shapes and configurations, and what is meaningful may depend on the task at hand. Therefore the problem of making the relevant components of a scene visually distinct is complex from a computational standpoint, and it is quite remarkable that this is normally achieved in human vision in what seems an instant.

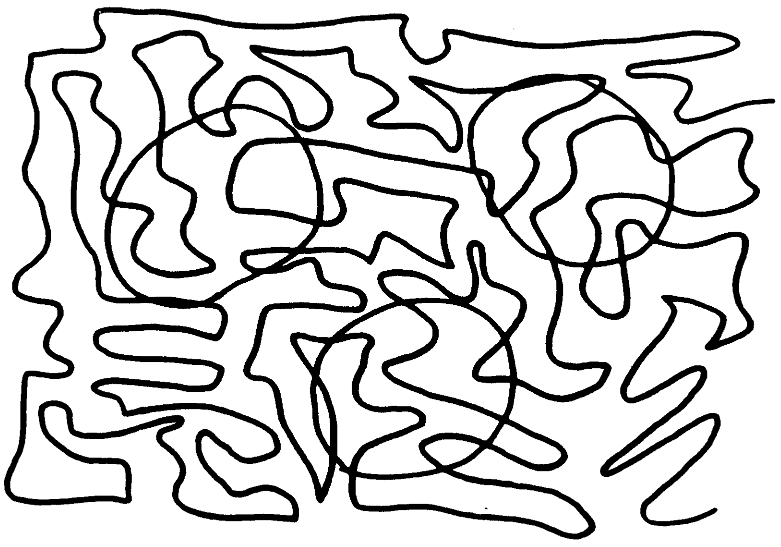


Figure 1.1: Three prominent blobs.

For example, consider Figure 1.1. We immediately perceive three large, striking blob shapes amidst an irregular background of curves. The speed with which the human visual system can locate and describe the outstanding blobs in figures like this does not depend noticeably on the total length of curves in the figure. Consider also the ease with which we can often solve connectivity-related problems, such as those in Figure 1.3. The capacities that these schematic examples illustrate—locating and isolating relevant figures rapidly—serve in all realms of visually guided activity. For example, the task of finding the largest spoon in Figure 1.4 seems to require no effort.

The problems of visually-guided interaction with the physical world impose various requirements on visual processing organization. One of the most crucial requirements is speed. This thesis explores the design of computational processes that could approach the time performance of the human visual system in analyzing visual spatial information, particularly in regard to the extraction of meaningful scene components. The rate of execution of the processes supporting spatial analysis in the human visual system has been tentatively estimated on the basis of the results of psychophysics and current knowledge of the

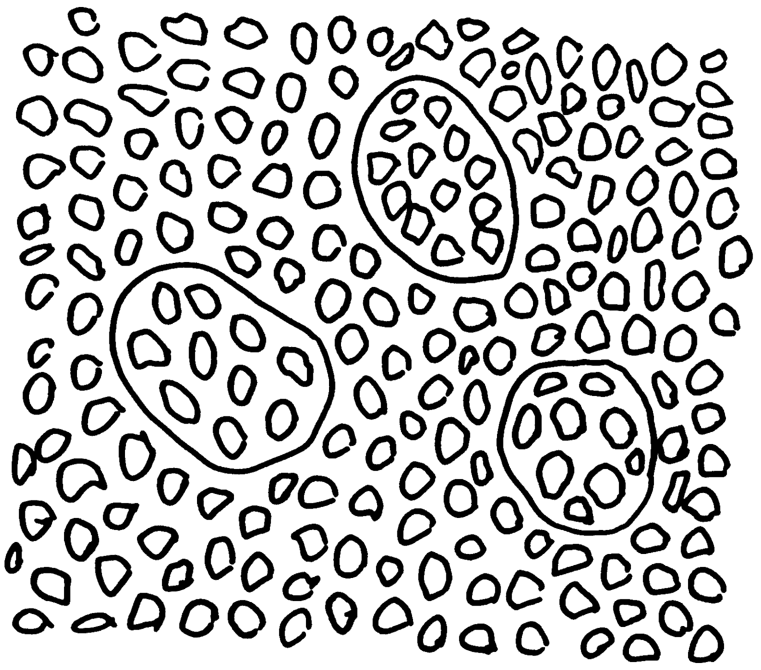


Figure 1.2: Three prominent blobs.

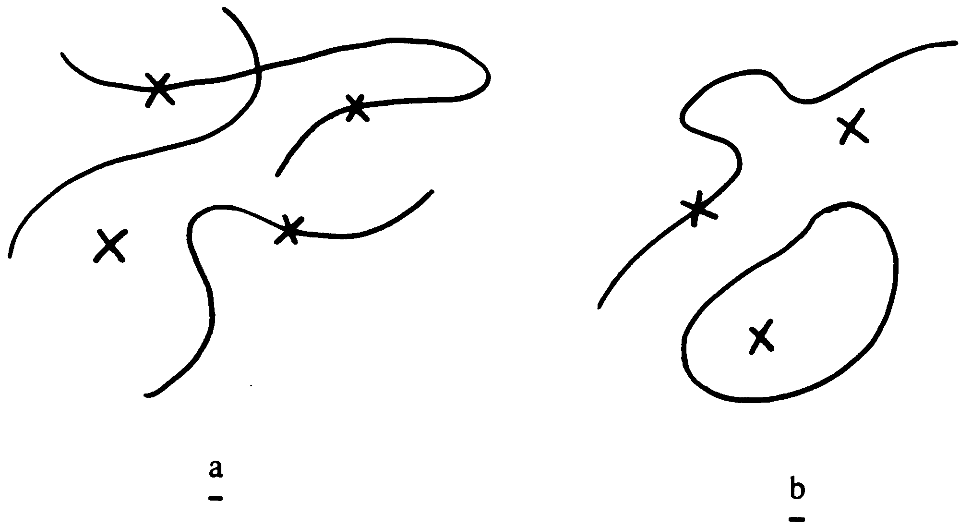


Figure 1.3: (a) Are there two "X"s on the same curve? (b) Are there two "X"s inside the same closed curve?

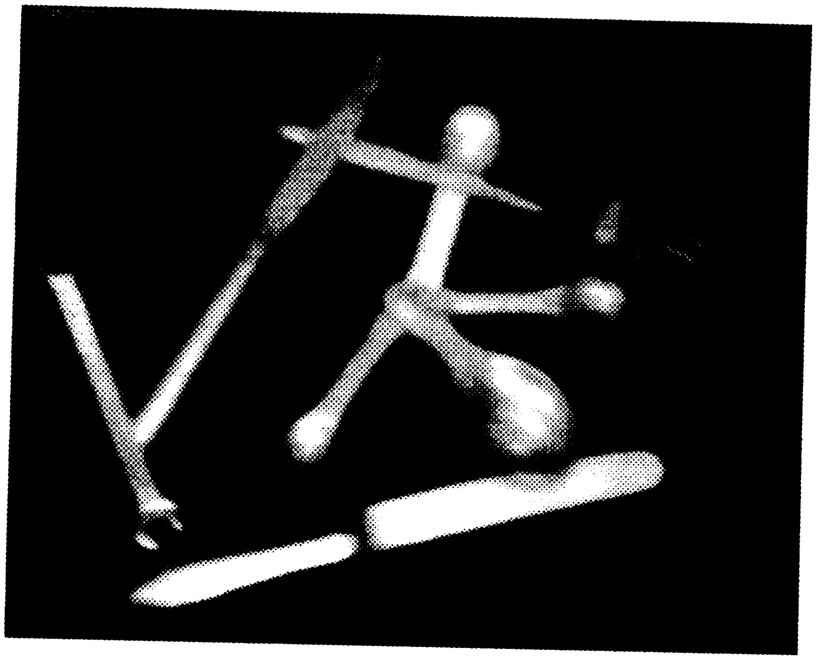


Figure 1.4: Find the largest spoon in this picture.

neurological structures in the visual system. Current estimates suggest that the number of basic computational steps devoted to extracting a scene component is normally in the tens [Edelman 85] [Shafrir 85]. Assuming that these estimates are around the correct magnitude, and ignoring the detailed derivation of them, how can visual processing be organized to be so rapid?

This thesis develops a framework for the fast extraction of scene entities, based on a simple, local model of parallel computation. An image chunk is a subset of an image that can act as a unit in the course of spatial analysis. A parallel preprocessing stage constructs a variety of simple chunks uniformly over the visual array. On the basis of these chunks, subsequent serial processes rapidly locate relevant scene components and rapidly assemble detailed descriptions of them. This chapter expands on and gives support to this framework. The rest of the report presents detailed proposals about the required representations and the means by which they may be computed from images.

The next three sections of this chapter introduce the notion of visual routines, Ullman's proposal for the organization of visual processes leading to the perception of shape properties and spatial relations [Ullman 84]. Included in this discussion are the considerations that suggest a two-stage processing framework, in which the first stage is spatially uniform and the second is spatially focused. The research described in this report is an outgrowth of Ullman's program for research into visual routines. The remaining sections present computational motivations for defining extended spatial primitives in low-level vision.

1.1 The general requirements of spatial analysis

Ullman [Ullman 84] formulated the problem of visually analyzing the spatial properties and relations of scene entities in terms of the following three general requirements: (i) *abstractness*—the capacity to establish computationally abstract properties and relations; (ii) *open-endedness*—the capacity to establish a large and extensible variety of properties and relations; and (iii) *complexity*—the ability to cope efficiently with the computational complexity involved.

A property or relation is said to be abstract if (i) its support is so large that it would be prohibitively expensive to detect the property or relation using a straightforward application of template-matching; and (ii) the set of instances of the property or relation contains *regularities* that can be captured by an efficient computation. [1] Many properties and relations of fundamental importance to vision are abstract in the above sense, and the abstractness requirement implies that a visual system must employ computations for capturing the regularities inherent in these properties and relations. Notable examples of abstract relations are *connectivity* and its close variants, such as "inside/outside" (Figure 1.3), "same-curve" (Figure 1.3), etc. The notion of a "two Xs inside the same closed curve" template-matching detector is implausible – the support of the connectivity relation is the entire input, so a different template would be required for every possible case.

The variety of potentially useful properties and relations is open-ended. It is inconceivable that a detector for every possibly relevant property or configuration could be predefined.

[1] Intuitively, the *support* of a spatial predicate is that subset of an input upon which the predicate "really depends" [Minsky and Papert 69]. For the purposes of this discussion, *template-matching* between plane figures can be defined as the cross-correlation between the figures.

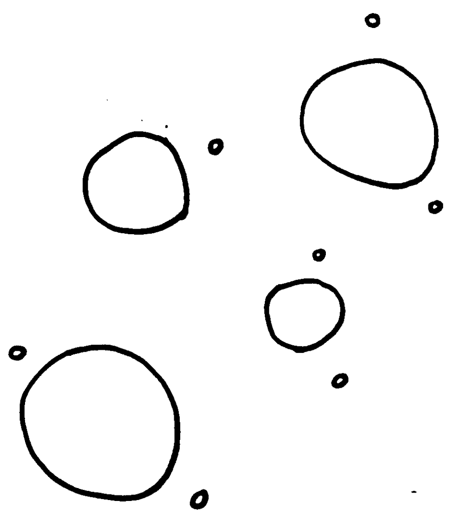


Figure 1.5: Find the small circle nearest the third-largest circle.

For example, the task of Figure 1.5 would require a "small circle nearest the third-largest circle" detector! The open-endedness consideration implies that the processes for detecting different properties or relations must use common machinery.

The complexity requirement addresses the fact that the computations for capturing abstract properties and relations, such as the connectivity relation, may be quite complex from a computational standpoint, and the implementation of them may be expensive. Moreover, the inputs to these computations are not constrained – relevant scene entities may appear with a very wide range of shapes, and they may occur with any spatial extent, at any locations, and in any number across the visual field. The complexity considerations, too, imply that the processes for detecting a property or relation at different locations must use common machinery.

To summarize, all meaningful scene entities, and all their potentially relevant properties and relations, cannot be detected at once, due to combinatorial explosion in required computational resources. Instead, visual processes must be *spatially focused* and *goal driven*, locating and assembling scene entities, and computing their relations and properties, as the need arises. These processes must meet the exacting time-performance requirements imposed by the goal of interacting with an active, changing world.

It is, however, possible and useful to detect certain simple, local, viewer-centered properties and features in a manner that is bottom-up, spatially uniform, and parallel. The very first processes of vision can make this sort of local information available to subsequent focused processes. Research in low-level vision has elaborated the computation of essentially pointwise properties and features, such as intensity changes, depth, surface orientation, texture, and so on. The processes that have been studied create pointwise primitive spatial elements—tokens characterizing depth, orientation, motion, etc., at a point. (For discussions of these low-level vision processes, see [Marr 82], [Horn 86], and [Barrow and Tenenbaum 78], for example.)

1.2 Visual routines

Ullman's proposal for meeting the requirements of spatial analysis is that properties and relations should be established in two stages, by the goal-driven application of visual routines—sequences of basic operations drawn from a fixed set—to a set of base representations that are created in a bottom-up, spatially uniform, parallel manner. New routines are assembled to establish newly specified properties or relations. Routines for establishing different properties and relations, or applications of a routine to different locations, share the machinery implementing the elemental operations they use.

The goals of the study of visual routines are (i) to establish a theory of what spatial properties and relations are useful in the context of spatial reasoning, object recognition, etc.; (ii) to determine a set of basic operations that makes it possible to compute these properties and relations robustly, and to specify the visual routines involved; and (iii) to devise efficient implementions of the basic operations and related machinery.

Ullman proposed a partial set of basic spatial operations, including region coloring; boundary tracing and coloring; location marking with respect to spatial reference frames; shift of a spatial processing focus; and "indexing"—processing shift to a salient location. These

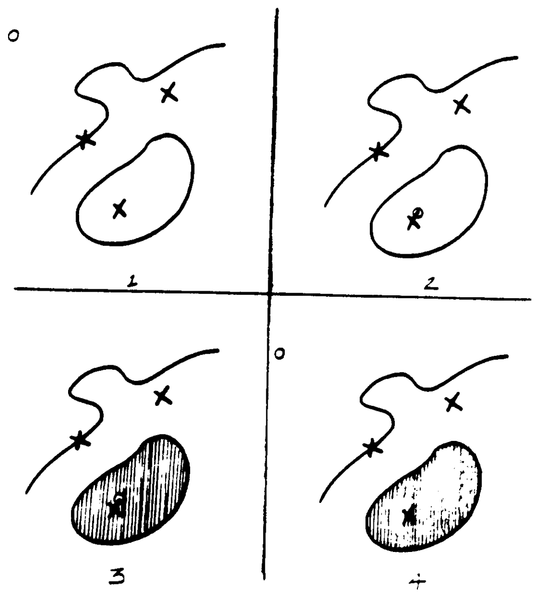


Figure 1.6: Part of the execution of a visual routine to decide if there is an "X" inside a closed curve.

choices were motivated mainly on grounds of potential usefulness in establishing a wide variety of relations.

To make the notion of a visual routine concrete, Figure 1.6 illustrates one possible routine for establishing an instance of the inside/outside relation. The task is to determine whether there is an "X" figure inside a closed curve. The procedure is as follows:

Until an "X" inside a closed curve is found, or Step 1 fails:

1. ***Shift*** the processing focus to the location of an unseen "X".
2. ***Mark*** the location of this "X" seen.
3. ***Color*** the white region that includes the processing focus.
4. ***Shift*** the processing focus to a location at the periphery.

5. If the periphery location is not colored, stop—the most recently visited "X" is inside a closed curve; otherwise "uncolor" colored locations and repeat.

Step 1 involves indexing to certain local features characteristic of an "X" figure, such as terminations. Further processing is required at this step to establish that the figure at the location shifted to is indeed an "X". This routine is based on the simplifying assumption that all boundaries in the input can stop the spread of coloring. For some tasks, it is necessary to first single out relevant boundaries, and allow only these to stop coloring spread. The shift to a periphery location is assumed to be a basic operation.

1.3 Basic operations

This subsection explores the basic operations Ullman proposed in somewhat more detail. It is possible to divide them into the following three main families: *activation* operations; *selection and shift* operations; and *reference frame and marking* operations.

Activation operations

Tracing, coloring, and one form of location marking may be viewed as instances of a fundamental operation in the analysis of spatial information referred to here as *activation.* Activation is the operation of *singling out (uniquely labeling) a set of locations* or primitive spatial elements in the visual array, so that subsequent operations may be applied specifically to the distinguished set. The various activation operations differ according to the criteria that they apply in generating the distinguished set of locations or spatial elements. Region and curve coloring operations activate a set of locations connected to a given starting location. Curve tracing operations activate a set of locations connected to a given starting location, subject to a restriction on the number of neighbors each element of the set may have.

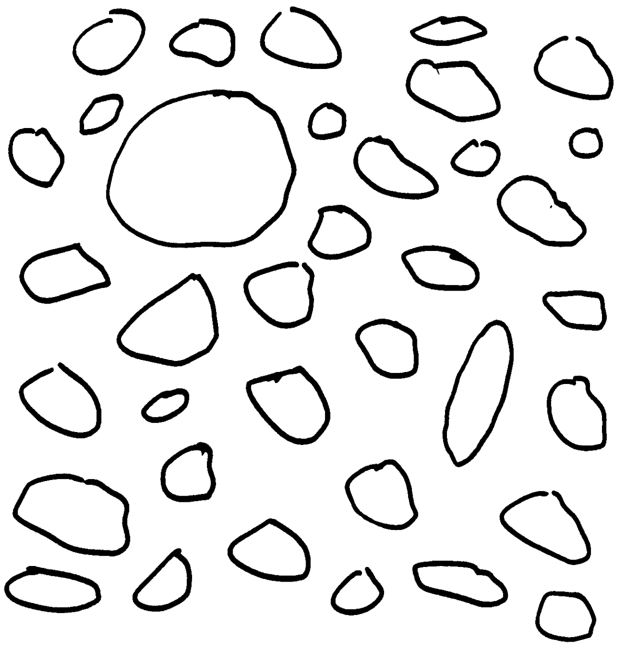


Figure 1.7: Figures are indexable based on prominence in global properties.

Selection and shift operations

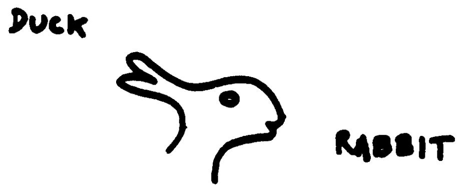
Spatial analysis requires the capacity to apply certain operations at selected locations. To accomplish activation, for example, it is necessary to begin tracing or coloring at one or more locations of the relevant set. In the example routine above to determine whether an "X" is inside a closed curve, coloring begins from the location of the "X" and a decision is based on the knowledge of where coloring started. Therefore, the study of visual routines must account for the processes for (i) selecting the location at which a given operation is to be applied, and (ii) shifting the point of application—the focus of processing—to that location.

Indexing is defined to be a shift of the processing focus to a salient location. The discussion of indexing in this report is mainly concerned with the processes for determining saliency. Various workers have demonstrated pop-out effects in visual search tasks involving arrays of letters and similar stimuli [Treisman and Gelade 80], [Julesz 81]. These demonstrations indicate that local features such as color, curvature, line terminations, etc. can serve for direct indexing, as long as the figure of interest is distinguished from irrelevant figures by a single one of these features. Figure 1.7 illustrates that global properties of an *extended* figure may support direct indexing as well.

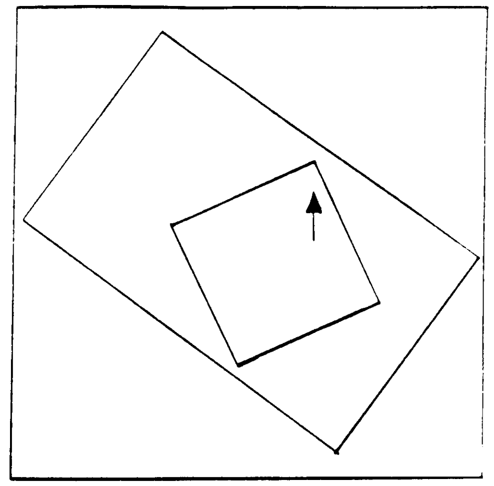
Reference frame and marking operations

Reference frames are implicit in many useful spatial properties and relations, such as "above", "left-of", "vertical", "clockwise", etc. Spatial analysis therefore requires the capacity to employ and manipulate frames of reference. One general application of reference frames is in establishing orientation. For example, the interpretation of a shape may depend on the reference frame with respect to which the shape is described. Figure 1.8 (a) contains a famous example of a figure for which different assignments of the "forward" direction lead to different interpretations (from [Jastrow 00]; see also [Rock 84]). Figure 1.8 (b) illustrates the *frame effect*: the direction attributed to the arrow depends on which of the enclosing rectangles is attended to.

A reference frame also serves as a system for describing and recording location. The location of a spatial entity is not usefully described in absolute retinocentric terms. It is more useful to describe the position of one scene component in relation to others. This requires the capacity to anchor a coordinate system at the image locaton of a given scene component. The operation of marking a location for later reference involves generating and storing a descriptor of a scene location with respect to a particular assignment of an internal reference frame. In Figure 1.9 (a), the point **p** may be described as "immediately below" the "X" or "in the upper-right" of the enclosing square (from [Ullman 84]). In the former case, the internal frame is anchored on the "X" figure; in the latter it is anchored on the enclosing square. The task of Figure 1.9 (b) involves *aligning* an internal frame with the elongated blob.

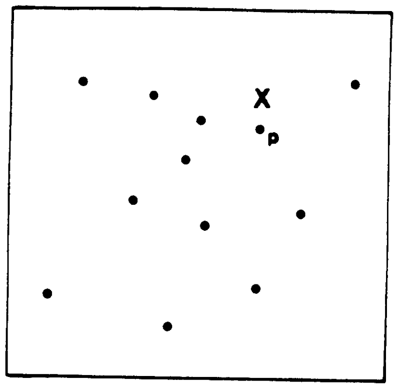


a

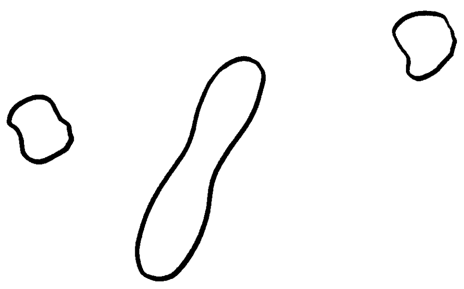


b

Figure 1.8: (a) Interpretation of a shape involves the assigment of a reference frame. (b) An illustration of the frame effect: which way the arrow seems to point depends on which enclosing rectangle one attends to.



a



b

Figure 1.9: (a) Describe the position of the point **p**. (b) Are the two compact blobs on the same side of the elongated blob?

1.4 Parallel models of computation for vision

Visual processing begins with a set of essentially pointwise descriptions of the scene in terms of intensity, color, texture, motion, depth, surface orientation, intensity boundaries, etc. Tracing and coloring operations, which are used to extract meaningful entities from these pointwise spatial representations, can be viewed as specific forms of the graph theoretic operation of *connected component labeling.* Because sequential connectivity algorithms have running time which is linear in the number of elements in the relevant component, a lot of research has gone into parallel connectivity algorithms. A number of connectivity algorithms have been proposed whose complexity is poly-log (a polynomial in the logarithm of the problem size) for a polynomial (and therefore feasible) number of processors. The best known connectivity algorithms have $O(\log N)$ time-complexity [Shiloach and Vishkin 82] [Lim 86] (see also [Cook 83] [Hirschberg 76]). At the completion of a run of such an algorithm, each connected component of elements in the input is, in effect, uniquely labeled, so that it is possible to determine in constant time whether any two given elements are "in the same region", or "on the same curve", for example.

Every poly-log parallel connectivity algorithm I know of assumes a model of computation in which (i) any processor can communicate in a single time step with any other processor, and (ii) it is possible to associate unique identifiers with processors, and a processor may store such an identifier, or transmit it to another processor. I will refer to the class of models providing these two capabilities as *global parallel models.* The global parallel models that have been proposed for connectivity differ mainly with respect to the conventions for arbitrating *read* and *write* conflicts. The $O(\log N)$ algorithm of Shiloach and Vishkin [Shiloach and Vishkin 82] assumes a synchronous computing model in which processors communicate via a common random access memory; concurrent reads from a location and concurrent writes to a location are allowed. Lim [Lim 86] recently described connectivity algorithms with a best case performance of $O(\log N)$, based on a model restricted to exclusive read and exclusive write.

It is costly and difficult to implement global, direct communication between all processors in a large set. For moderately large sets of processors, truly direct communication is physically unrealizable. In practice, global communication is implemented by dynamically routing messages through high-dimensional networks (for example, see [Hillis 85], [BBN 85], and [Gottlieb et al 83]). The duration and/or reliability of this delivery process depends on the pattern of message traffic through the network at a given time. For example, message delivery may be severely hampered when many processors simultaneously attempt to send a message to the same processor. As such, time-complexity figures that treat a message delivery cycle as a primitive operation—as the $O(\log N)$ parallel connectivity figures do—cannot be taken at face value.

Ullman [Ullman 76b] proposed a number of criteria to characterize what he called *simple local processes*, the primary ones being locality and simplicity. The locality criterion is a restriction on the range of the direct communication in the computation. The simplicity criterion is a restriction on the power of the processing elements. It is not feasible to build very large networks of powerful processors, owing to considerations of size and cost. I refer to models of parallel computation observing these restrictions as *simple, local models.* The general properties of this class of models can constrain representations and algorithms, without regard to the details of any particular instance of the class. Importantly, *the capacity to establish and transmit unique processor identifiers is not compatible with the class of simple local computing models.* This, for example, excludes algorithms based on pointer-passing.

A simple, local model of computation suggests itself as a possible basis for exploiting parallelism in connectivity computations on two independent grounds. For low-level applications in machine vision systems, simple, local models may be preferable to global parallel models because they are less costly and simpler to build—for the same price, they can provide more parallelism. For the study of biological visual systems, the restrictions of simplicity and locality are consistent with what is known about biological information processing, so computations observing these restrictions are more likely to be revealing.

The goal of this research is to understand how parallelism can be exploited to make the inherently serial processes of scene analysis sufficiently fast, without relying on global parallel computation models. My thesis is that complex scene entities can be rapidly extracted from pointwise initial representations using only simple, local computations, if a two-stage, parallel-then-serial processing organization is used, in which the first stage assembles simple, extended spatial building blocks. I begin by elaborating on the building-construction metaphor, for it helps to make some important ideas vivid.

1.5 Prefabrication—a metaphor for chunking

If you had to build a house very quickly, you would assemble it from prefabricated sections, not individual bricks and boards. You would use prefabricated components specifically suited to the type of structure you were building; the structural elements required for a tropical bungalow are very different from those required for a bomb-shelter. Moreover, you would select the complement of component sections that entailed the shortest possible number of assembly steps. That is, your main objective would be *optimization* of the assembly process. The first consideration, *specialization*, also helps to optimize the assembly time—it seems natural that components designed with your particular application in mind will make the construction task go faster and more easily.

If you made a living selling prefabricated building supplies, you would offer simple components that could be easily combined into a wide range of more complex structures; they would be regular sections of wall or floor, for example. There are two reasons for this. First of all, it would be easier and more economical to construct such simple components. Moreover, more complex structures would have narrower *scope*, in the sense that they would fit into fewer customers' designs. The components would span a range of sizes and shapes just broad enough that most customers could come reasonably close to their goal of optimizing assembly time. These considerations of *economy*—providing simple components across a sensible range of sizes and shapes—strike a balance between demand and production costs. The optimization criterion must be traded off with those of scope and economy. It would be reasonable to ignore the occasional customer whose needs were outlandish. You would observe the specialization requirement by providing structural components tailored to each of the most common applications.

The power of prefabrication is that it exploits the parallelism inherent in the assembly process. The process of building a house is inherently serial due to the constraints of physics: the roof cannot be put up before the walls (or other supports), and the walls must follow the foundations. As such, if a house is built brick by brick, the first board in the roof must follow the last brick in the walls, and so on. This serialism, however, is not inherent at the level of individual bricks, but only at the higher level of walls, roofs, etc., so *these abstract components may be concurrently pre-assembled.* The main problems of prefabrication, then, are (i) to determine, given a detailed description of a complex structure to be assembled, a decomposition into simpler substructures that can be economically preassembled in parallel; and (ii) to assemble these simpler components.

1.6 Image chunks

An *image chunk* is defined to be any subset of a pointwise representation of the scene that can act as a unit in the sense that applying some spatial operation to (or reading out some spatial properties of) the set of constituent elements requires minimal computational effort. Spatial operations on image chunks are the elements from which more complex spatial operations are composed. Coloring, for example, can generally be expressed as an iteration of the following parallel operation: *color every uncolored image chunk adjacent to a colored image chunk.*

The process of making a scene entity distinct on the basis of a pointwise initial description can be likened to an assembly problem. From this viewpoint, image chunking is closely analogous to prefabrication, and is governed by similar considerations. The motivation for image chunking is to optimize the run-time of the processes that extract scene components. To achieve this, image chunks must be specialized to particular basic spatial operations – the representational requirements of indexing are different from those of tracing, for example. Combinatorics limits the range of distinct instances that a chunking process can generate, so these instances must be carefully chosen to provide the widest possible scope. In this context, scope refers to the class of inputs for which a particular class of image chunks supports efficient processing, and economy pertains to the hardware cost and time performance of the chunking process.

There are three main problems to be solved in the design of image chunks to support a particular basic operation. First, a computational definition of the basic operation must be formulated, in terms of its constituent operations. Second, image chunks must be devised that (i) would lead to the desired performance, and (ii) are economically computable from a pointwise representation of the scene by spatially uniform, parallel processes. Finally, the processes for computing these chunks, and high-performance, chunk-based formulations of the associated basic operation, must be defined, implemented, and tested. All of this must be accomplished in the context of a realizable, affordable model of computation.

1.7 Image chunking for the rapid extraction of scene entities

This report presents the results of a study of image chunking in support of the most potentially time-consuming operations in the extraction of scene components—indexing, tracing, and coloring. Consideration of a wide variety of perceptual phenomena has led to a theory of image chunking that involves several novel low-level vision processes in a novel processing organization. As illustrated in Figure 1.10, the computation of image chunks can be divided into four principal stages: (i) the computation of local image properties from the intensity image; (ii) the computation of local boundary information, which constitutes the primary input to chunking processes; (iii) the computation of local prominence information, which augments the boundary information; and (iv) the application of the chunking processes to this input.

Chunks useful for indexing scene entities may be defined by dividing the boundary data into relatively isolated configurations of boundary locations. Chunks useful for curve tracing may be defined by dividing the boundary data into regions each containing a single segment of a boundary. Chunks useful for region coloring may be defined by dividing the boundary data into regions each containing no boundary locations. All of these classes of chunks are computable by simple, local processes.

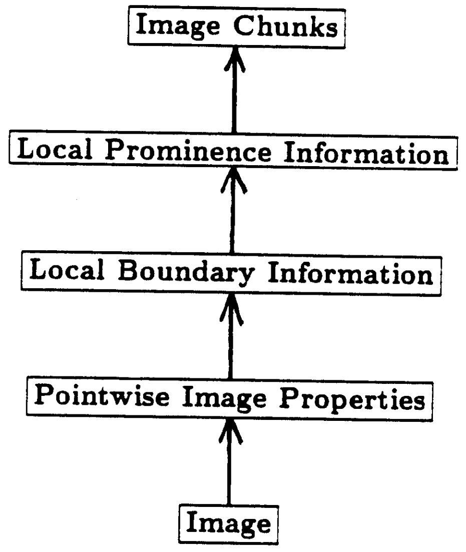


Image Chunks
Local Prominence Information
Local Boundary Information
Pointwise Image Properties
Image

Figure 1.10: Image chunking and the organization of low-level vision.

Guide to the reader. Chapters 2 and 3 are devoted to specifying the input to image chunking processes. Chapter 2 argues that boundary information is required as the basis for chunking. Chapter 3 argues that the boundary information can be more useful if augmented with prominence information. Chapters 4, 5, and 6 are devoted to the design of chunking processes proper. Chapter 4 is concerned with indexing, Chapter 5 with tracing, and Chapter 6 with coloring. Chapter 7 makes comparisons between the two-stage framework for the extraction of scene components (of which image chunking is the first stage) and a wide range of other proposals for achieving the same goal. Chapter 8 presents conclusions from this study and suggestions for future work on image chunking. With the exception of Chapter 6, which relies on the details of the discussion in Chapter 5, each chapter of this report may be read on its own.

Chapter 2

Local boundary information—the input to chunking processes

The first step toward designing image chunking processes is to understand how, in general, the physical components of scenes are manifested in images. The manner in which objects are described in images determines the nature of the input representations upon which chunking processes must operate, and hence the nature of the chunks themselves.

The main argument of this chapter is that scene entities, or chunks of them, cannot be detected directly, because objects do not in general give rise to uniform regions in the image; they must be detected on the basis of *boundaries* detected in the image. The blobs in Figure 2.1, for example, constitute non-uniform regions that are quite similar to the background; the information defining them lies predominantly at their boundaries. In other words, chunk-detection is inherently a two-stage process. There are two broad categories of useful boundaries: *abrupt-change boundaries* are defined by local differences in local properties; *alignment contours* are defined by colinear arrangements of similar features. The blobs in Figure 2.1 are defined by abrupt-change boundaries; those in Figures 2.2 and 2.3 are defined by alignment contours.

The detection of boundaries in certain local image properties, such as intensity, color, motion, texture, etc., has been extensively studied in computational vision. In this chapter, I formulate and demonstrate simple, local computations for detecting abrupt-change bound-

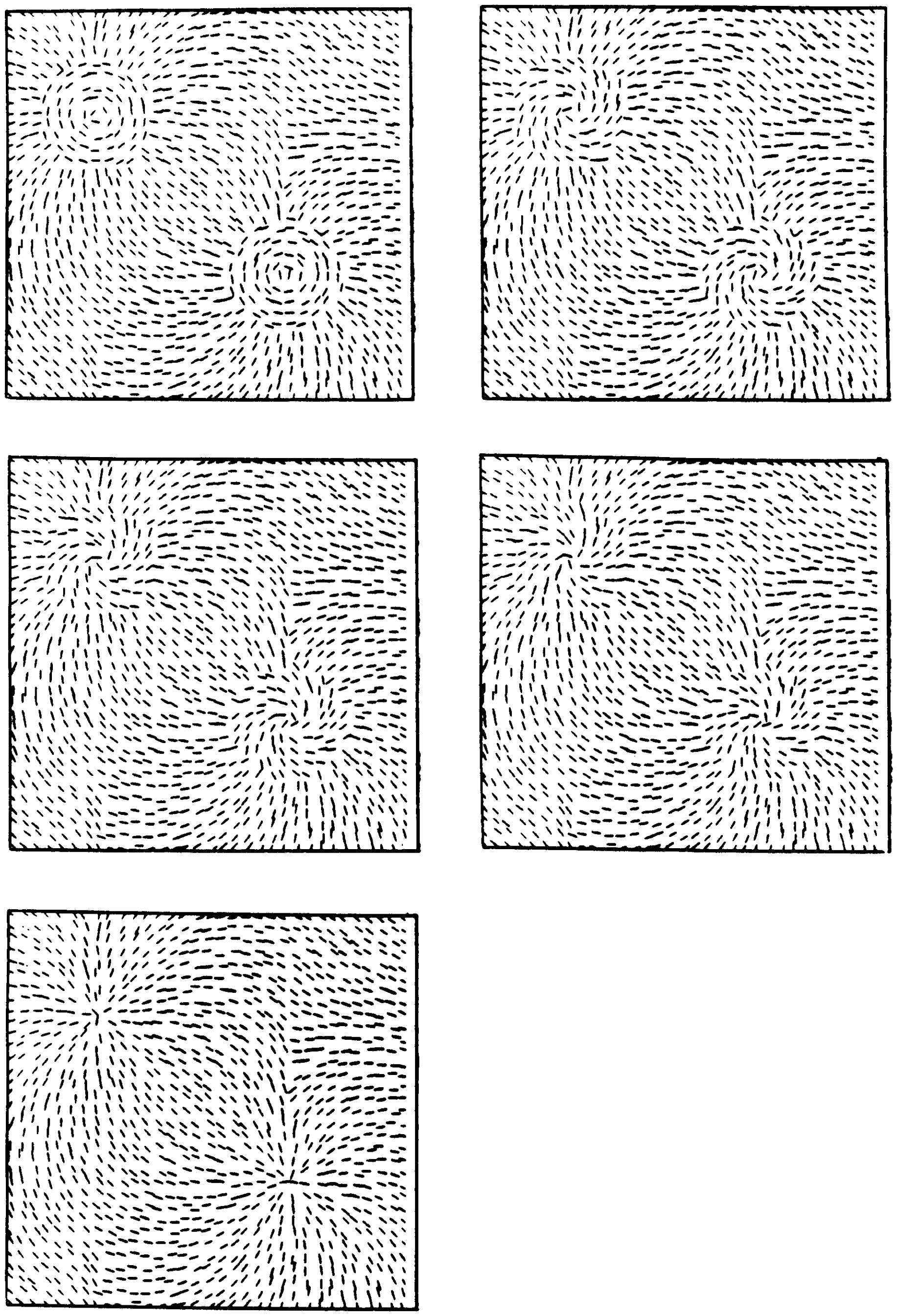


Figure 2.1: Objects defined primarily by abrupt changes at their boundaries, not by uniform region properties—more abrupt changes lead to more definite perception.

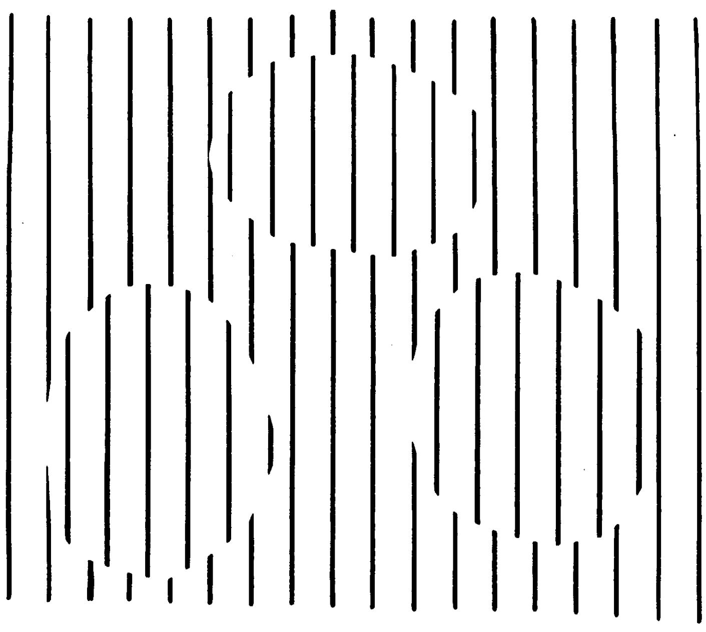


Figure 2.2: Objects defined primarily by bounding alignment contours, not by uniform region properties.

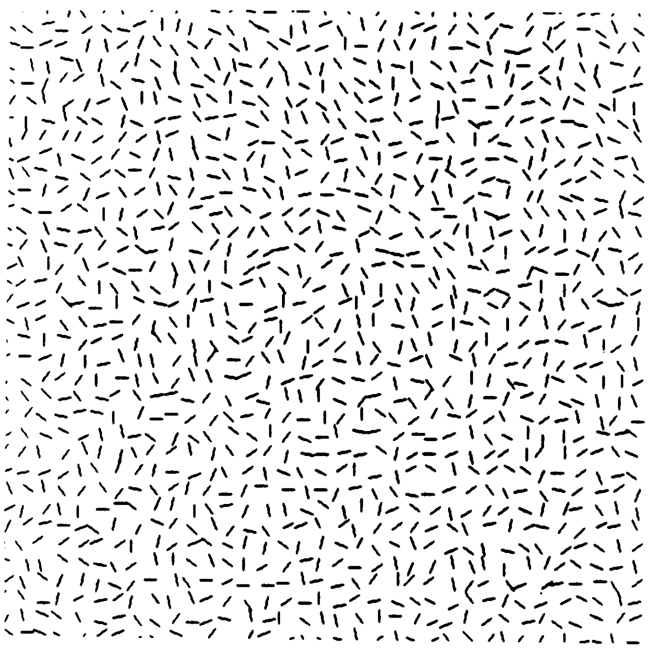


Figure 2.3: An object defined by another kind of alignment contour.

aries in a single sparsely distributed, spatially varying property. The purpose of this chapter is not, however, to present a complete theory of boundary detection, but simply to establish that the detection of boundaries is a requirement. In fact, the study of chunking processes has suggested some challenging new requirements on the boundary computations, which it is beyond the scope of this report to meet. These requirements are exposed in Chapters 4, 5, and 6.

2.1 The need to detect boundaries

Region segmentation is a major scene analysis tool in computer vision. Region segmentation processes are generally designed to extract *image regions that are essentially uniform in some property*, on the assumption that such regions correspond to meaningful scene entities. For example, a red object in the scene would be expected to give rise to a uniformly red area in the image. Region segmentation methods differ according to the particular techniques used to delineate the uniform regions, but their success depends on the degree to which the uniformity assumption is satisfied in the domain of application [Horn 86]. (Region segmentation methods are discussed in Chapter 7. For more detailed reviews, see [Ballard and Brown 82], [Horn 86]).

In fact, scene entities often do not give rise to image regions that are nearly uniform in any properties, for several reasons. First of all, the processes that often generate surface markings, e.g., growth and accretion, generally do not operate in a globally uniform manner, and so do not give rise to globally uniform markings. Furthermore, even when properties like orientation, curvature, elongation, and size are uniform over the entire object surface, perspective projection does not preserve this uniformity. Similarly, changes in distance and orientation of the surface with respect to the viewer do not generally preserve uniformity.

A variety of perceptual examples demonstrate that distinct objects are compellingly perceived in human vision where no image property is uniform over the areas of the objects (see Figure 2.1 and [Muller 86]), or when surface properties are not distinguishable from those of the background (Figure 2.2). The Craik/O'Brien/Cornsweet brightness illusion is an example of a similar phenomenon in the intensity domain [Cornsweet 70].

A more general characterization of the areas corresponding to scene entities in images is that they *vary only slowly* in some properties. That is, for example, that each hair in a coat of fur is similar in orientation to neighboring hairs, not to every other hair. Visible physical boundaries—which include occlusion boundaries, sharp changes in surface orientation, and abrupt changes in surface material —almost always give rise to abrupt changes in image properties.

If the uniformity assumption were correct, processes amounting to template matching could be used, in principle, to detect and describe objects directly. Such a process would integrate (or "pool") local image properties over extended areas. For example, compact red regions could be detected by applying circular red-center/non-red-surround detectors at a variety of sizes, all over the image. The failure of this assumption implies at least two stages in the extraction of objects.

Scene entities do not give rise exclusively to *areas* in the image. Some scene entities have curvilinear geometry and give rise to curvilinear structures in the image. It is also possible for non-curvilinear scene entities to give rise, through projection, to image curves. In a context in which scene entities are expected to give rise to uniform image regions, region-based detection processes are often adopted. As such, additional boundary-based processes are required for the detection of curvilinear structures. In a context in which scene entities are detected on the basis of their boundaries, curvilinear and compact image structures may be detected by the same process—it is possible to define boundary-based detection schemes that work equally well applied to curves as to region boundaries.

The attributes of the locations on the surface of a scene entity include intensity, color, motion relative to the viewer, distance from the viewer, orientation with respect to the viewer, and surface marking structure. The term "surface marking structure" refers to the arrangement of surface marking elements that themselves have attributes of color, motion, extent, elongation, orientation, etc. This notion is related to the broader notion of "texture". For the most part, these properties can be detected in the image by local computations. Previous studies in early vision have addressed the problem of detecting local properties [Marr 82] [Horn 86].

Considerable research effort has been directed to the detection of intensity boundaries. Intensity boundary information is certainly useful, but it is not sufficient, for two main reasons. First, scene entities can be perceptually salient without any intensity changes being present at their boundaries (as seen in Figure 2.1). Second, even when scene entities are bounded by intensity changes in an image, it is often hard to distinguish these boundaries from the many other intensity boundaries present due to surface structure, shadows, highlights, and noise (see [Riley 81] for further discussion of these points.) Because the circumstances of imaging are potentially so varied, no particular property can be expected to provide adequate information on its own, or even to provide a majority of the required information – information must be combined from all available visual properties. Each one of the previously listed image properties can give rise to boundaries useful for the definition of image chunks.

2.2 Detecting abrupt change boundaries by direct comparisons between neighboring elements

The local image property elements computed as input to early boundary detection can have dense or sparse spatial distribution. Dense distribution of properties such as intensity, color, and motion arises over areas in the image corresponding to scene entities whose surfaces are uniform, without local markings. Sparse distribution of a property can arise whenever there are many small entities in the scene (each giving rise to an image property token), or when the extended entities have many local surface markings. This section examines the detection of boundaries in sparsely distributed, spatially-varying image properties.

Most proposals for the detection of texture boundaries rely on measures of the distribution of the values of a property over a neighborhood. Sparseness imposes a lower limit on the neighborhood size that will support a meaningful and sufficiently accurate distribution measure. Spatial variation of properties imposes an upper limit on the neighborhood size over which the measure will distinguish (i) the interior of one region from the interior of another; or (ii) the interior of a region from the boundary between two regions.

A natural way of coping with sparseness and spatial variation is to make direct comparisons between neighboring property elements—boundary elements are defined between sufficiently different neighboring elements. Distribution measures do not seem to be necessary at all.

2.2.1 Computing a primitive element's neighbors

Due to the possibility of sparse distribution, it is necessary to compute an explicit representation of the ***neighbors*** of each image property element. This section introduces a general proximity representation useful for this purpose, termed the ***directional nearest neighbor graph (DNNG).***

Let $d(a, b)$ denote the Euclidean distance between points a and b. q is defined to be a directional nearest-neighbor of p with respect to θ and D (referred to as the orientation and range parameters, respectively) if

1. $d(q, p) < D$ and
2. there exits no other point r such that
   (a) $d(r, p) < d(q, p)$, and
   (b) $qpr < \theta$.

The DNNG of a set of points in the plane is the set of directed links (i.e., edges) between points and their directional nearest neighbors. (Directional nearest neighborhood is not a symmetric relationship, so the links (p, q) and (q, p) are not identical.)

For certain values of θ, the DNNG is somewhat reminiscent of the dual of the Voronoi diagram (see Figures 2.4 (b) and 2.5 (b)). The *Voronoi diagram* of a point set P is the union of the Voronoi polygons of each point p in P. The *Voronoi polygon* of a point p is

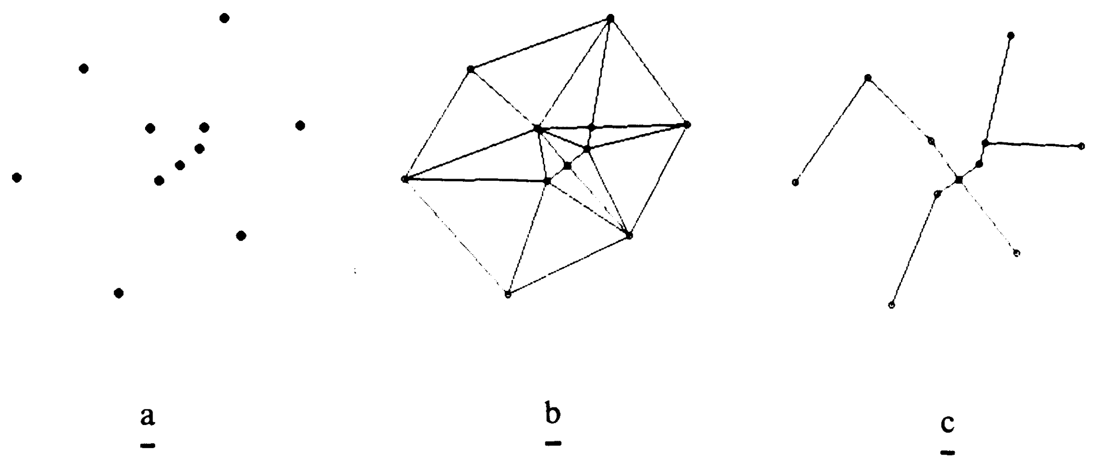


Figure 2.4: Directional nearest neighbors: (a) the input; (b) DNNG for $\theta = 45°$; (c) DNNG for $\theta = 90°$.

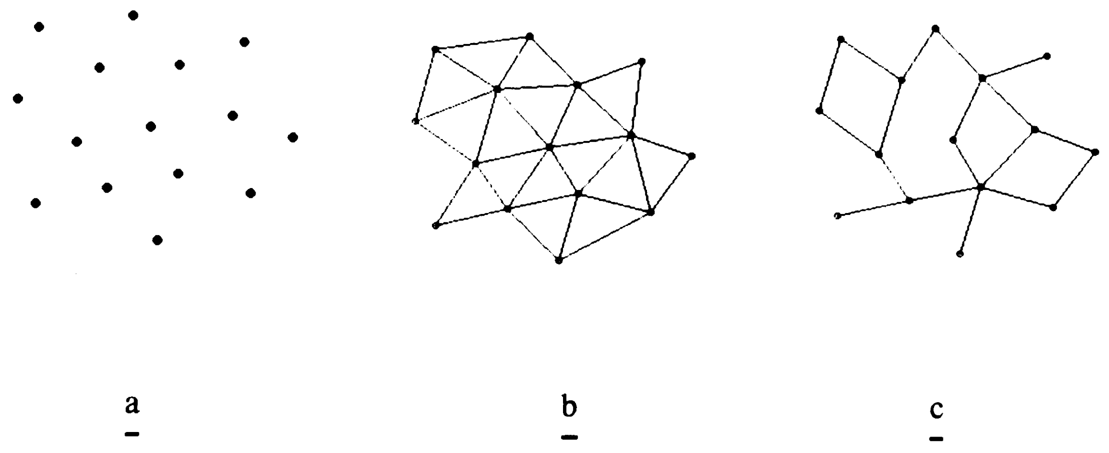


Figure 2.5: Directional nearest neighbors: (a) the input; (b) DNNG for $\theta = 45°$; (c) DNNG for $\theta = 90°$.

defined to be the set of all points in the plane closer to p than to any other point in P. The *Voronoi dual* of P is a graph in which nodes correspond to points in P, and there is an edge between two nodes if the corresponding Voronoi polygons are adjacent. The Voronoi dual is preferable as a proximity representation to the option of simply computing the k nearest neighbors of each point, because it avoids an *a priori* limit on the number of possible neighbors a point can have. As such, the Voronoi dual has been used to represent proximity in a variety of contexts ([Sedgewick 83] discusses a number of these), and it suggests itself for the problem of detecting early boundaries. The DNNG has several advantages over the Voronoi dual, however, as a proximity representation:

1. Computation of the DNNG is local, whereas the Voronoi dual computation is non-local.
2. Computation of the DNNG is simpler to implement.
3. The angle parameter of the DNNG computation provides control over the angular distribution of a point's neighbors; the Voronoi dual computation provides no control.
4. The range parameter of the DNNG computation provides control over the spatial extent over which a point's neighbors are defined.

2.2.2 Computing property differences

Given the DNNG representation of neighborhood between property elements, there are two roughly equivalent alternatives regarding the representation of differences between neighbors, one node-based, the other link-based. In a node-based computation of differences, at each node in the DNNG of property elements, some function of the differences between the value at a node and the values at each of its neighbors would be associated with the node. Possible functions are the average and the maximum. In an edge-based computation, the difference between values at the neighboring nodes defined by a link is associated with the link; the location of the difference measure may be defined to be the midpoint between the locations of the two nodes. The link-based scheme is preferable because it gives a

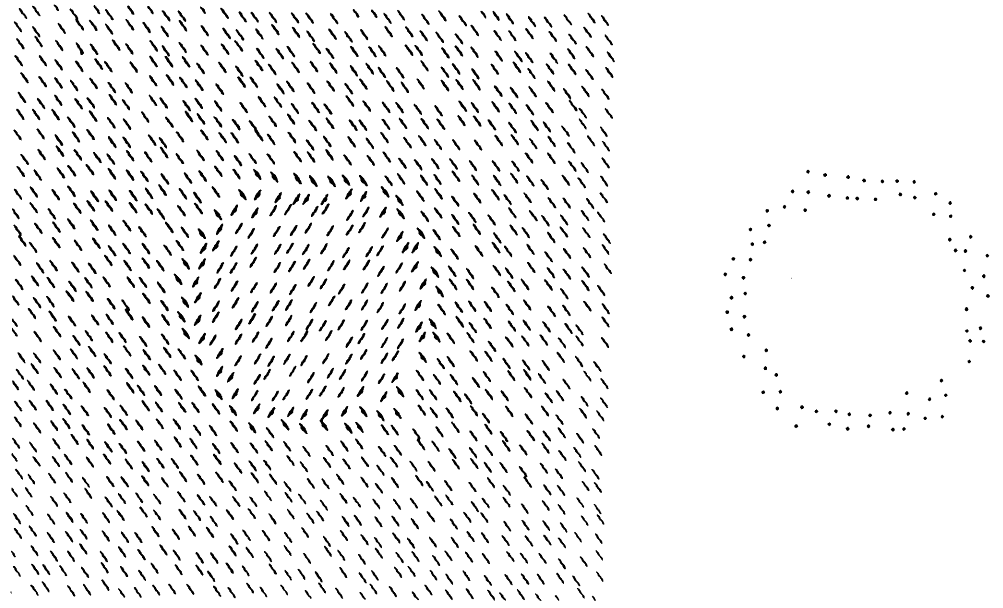


Figure 2.6: Node-based difference computations give rise to parallel pairs of boundaries—boundary locations (right) have been superimposed on the input (left).

single, well-localized boundary between two regions, whereas the node-based method gives "double" boundaries (Figure 2.6). For that reason, in the implementation of abrupt-change boundary detection, the link-based scheme was adopted.

The result of the abrupt-change boundary computation on the example of Figure 2.7 is shown in Figure 2.9. (The DNNG for this example is shown in Figure 2.8.) A circle is displayed at the center location of each edge in the DNNG; the circle's radius is proportional to the difference measured at that location. Object boundaries coincide with significant local maxima in the difference measures. It is not necessary to define computations to distinguish boundary values from non-boundary values at this stage—chunking processes may be defined to respond preferably to the locally maximal values.

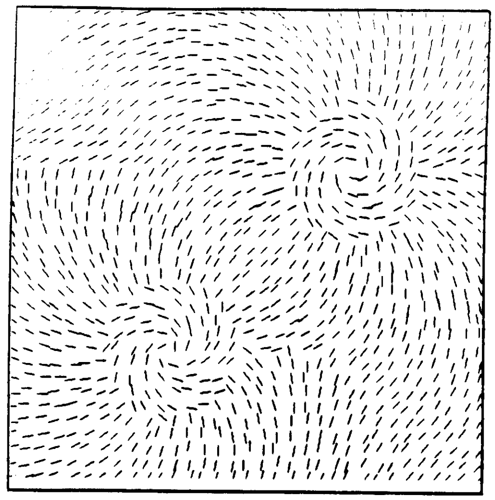


Figure 2.7: An example abrupt-change boundary computation: the input.

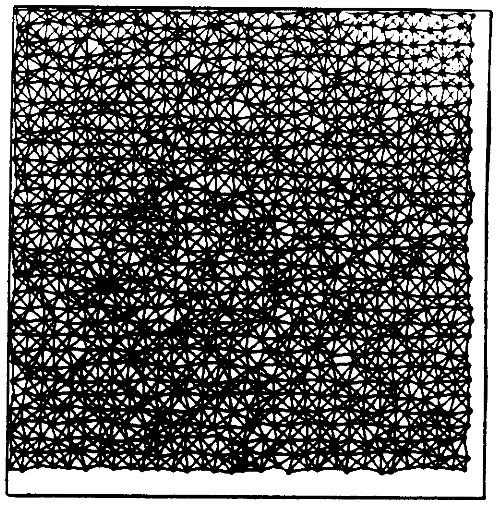


Figure 2.8: An example abrupt-change boundary computation: DNNG defined with $\theta = 30°$.

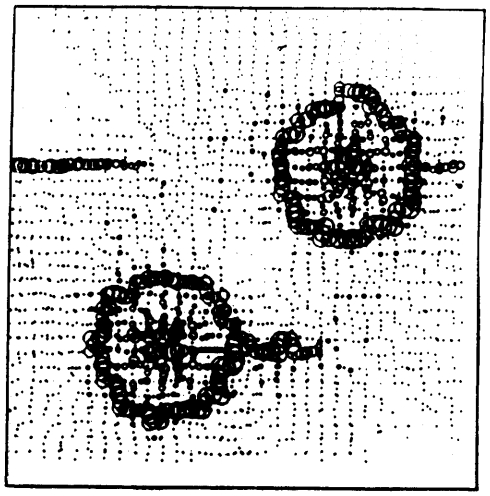


Figure 2.9: An example abrupt-change boundary computation: link-based differences.

Chapter 3

Local prominence information

This chapter extends the bottom-up specification of the input to chunking processes. Chapter 2 argued that since objects do not necessarily give rise to uniform image regions, chunks must be computed from a representation of image boundaries, such as sharp local differences. The main observation of this chapter is that, in general, boundaries alone do not support efficient, spatially parallel detection of visual chunks—*local prominence information* about *local boundary segments*, is required in addition. The local prominence information — the measure of how different each boundary segment is from its neighbors—reflects a type of image redundancy that may be exploited to distinguish *salient* scene structures from "run-of-the-mill" ones. Typically, humans perceive such structures as "figures" standing out against a "background" of other figures, as in Figure 1.2. This chapter formulates and demonstrates computations for describing appropriate boundary segments and measuring their local prominence.

3.1 Redundancy and local prominence information

3.1.1 Redundancy; figures and background

The information in images is highly redundant. Because of redundancy in intensity information, the intensity changes in an image can constitute a nearly complete, but much more compact, description of a scene [Yuille and Poggio 83]. Redundancy can also arise at the

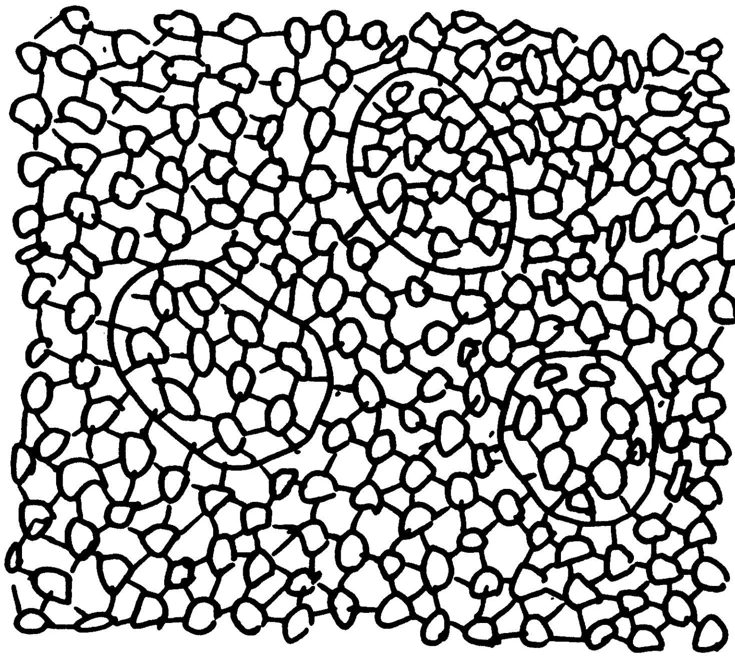


Figure 3.1: Human perception is fast and robust even in the face of dense, connected context.

level of shape or arrangement of the scene's physical components. For example, an image of a uniform, dense arrangement of many small, roughly identical ball bearings and one large bearing is very high in redundancy at this level. It is this *figural redundancy* that makes it possible to describe this hypothetical scene as succinctly, yet as expressively, as I have. The boundary representation of this image would contain the boundaries of every bearing, and this *context* would make it difficult for simple parallel computations to detect and localize the large bearing [Minsky and Papert 69]. A variety of perceptual examples show that human vision copes very effectively with context in certain circumstances (Figures 1.2, 1.1, 3.1)—it seems that this is the case precisely when the scene is high in figural redundancy. In fact, this tendency for objects to "stand out" is so prevalent that it is frequently relied upon to resolve ambiguous references in the course of everyday spatial reasoning and verbal communication about the visual surroundings. In Figure 1.2, for example, one might refer to "the blobs", where there are in fact many entities deserving of the term.

Two observations about the natural world suggest that figural redundancy occurs often enough in scenes to make computations dedicated to exploiting it very beneficial. First, uniform or slowly varying distributions of similar objects or surface markings are quite common. Second, it is common for an object with very different properties to occlude, or be embedded in, such a distribution of objects. I refer to such a situation as a *figure/background situation.* It follows that economical, spatially uniform, parallel computations for exploiting figural redundancy can provide part of the solution to the difficult problem of rapidly extracting meaningful entities from complex scenes. Local prominence information about local segments of boundary may be used to, in effect, *separate* boundaries of salient objects from other boundaries in parallel, so that chunking processes may be applied selectively to the former.

It is shape properties that distinguish foreground figures from background figures in the figure/background situation. The distingushing shape properties may be properties of local portions of the figure or properties of the entire figure. Differences in global properties are often reflected locally. For example, when there is a substantial difference in size between a foreground entity and background entities, local boundary curvature may be substantially lower for the foreground entity than for the background figures, as in Figure 1.2. (In Figure 3.2, the big blobs were camouflaged somewhat by keeping the local curvature of their boundaries similar to that of the background.) The properties with respect to which boundary segments might be prominent include orientation (Figure 1.1), curvature (Figure 1.2), and extent (Figure 3.3). The distribution of the scene entities of the background may vary slowly (Figure 3.4), so differences in shape properties should be detected over local scene neighborhoods.

3.2 Computing local prominence information.

This section outlines computations for measuring local prominence of boundary segments. These computations involve (i) making boundary segments explicit; (ii) measuring their useful properties; (iii) establishing local neighborhoods over these boundary segments; and (iv) measuring local prominence of boundary segments over these neighborhoods.

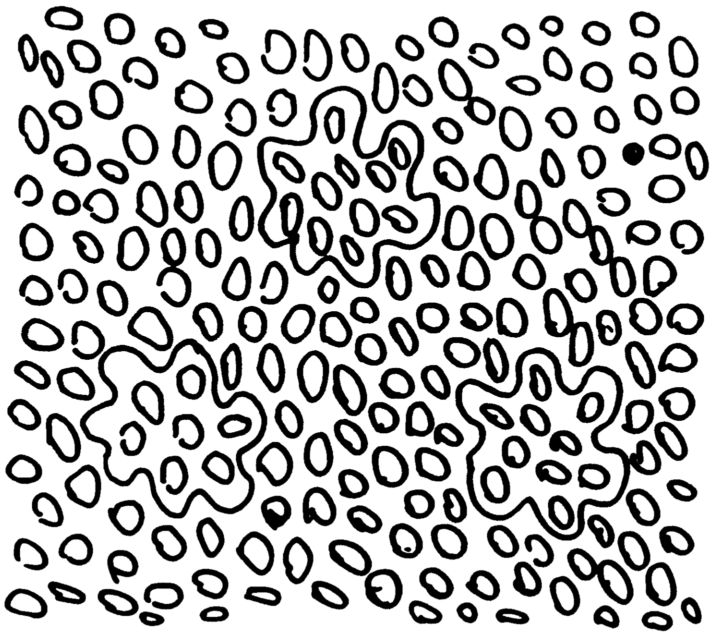


Figure 3.2: The big blobs are not very prominent because local segments of their boundaries are not very different in curvature from neighboring background segments.

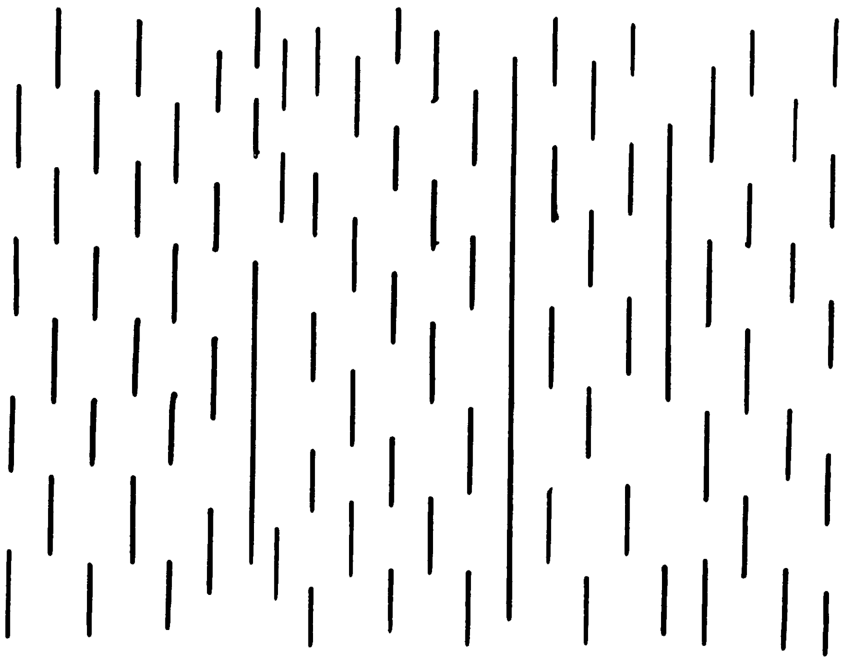


Figure 3.3: Boundary prominence due to extent.

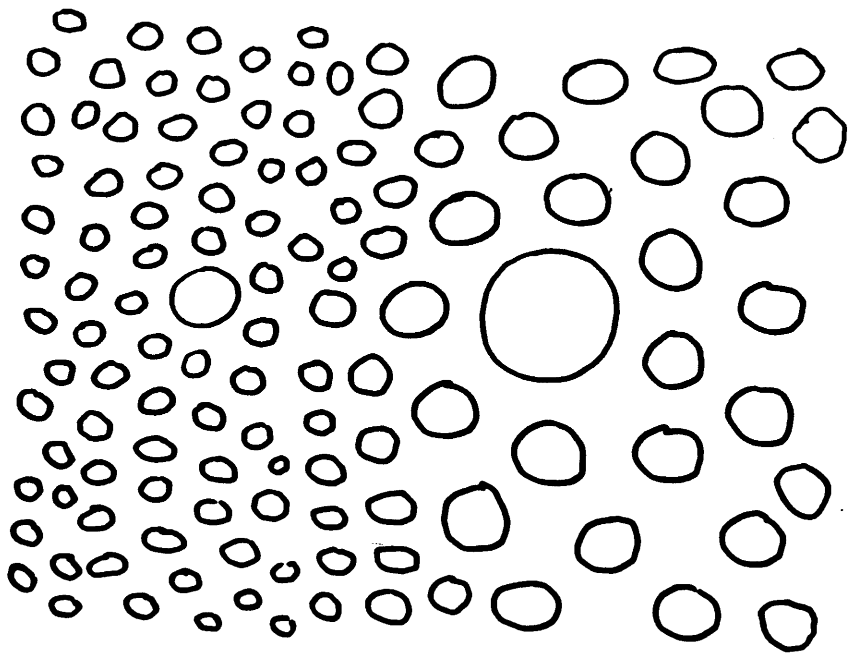


Figure 3.4: Figural prominence is a local phenomenon.

3.2.1 Defining boundary segments.

The computations for making boundary segments explicit are to be applied in a spatially uniform manner all over the input. Because properties like orientation and curvature depend on scale, the segments must be defined at a series of scales. Scale, in this context, is related to arc-length. Prominence computations must be performed independently at each of these scales; the results are then combined. Figure 3.5 illustrates the definition of boundary segments at a location at multiple scales. The boundary segments at a particular location and scale must be made explicit by a local boundary tracing process, because the spatial extent of segments at the larger scales allows for enough variation in segment shape to preclude detection by template matching. Therefore, the largest defined scale is determined by some reasonable limit on the number of tracing steps to be incurred in the course of defining boundary segments.

Boundary segments at the larger scales may span many pixels, so standard pixel-at-a-time methods for boundary tracing may be inadequate for this task. Much better performance

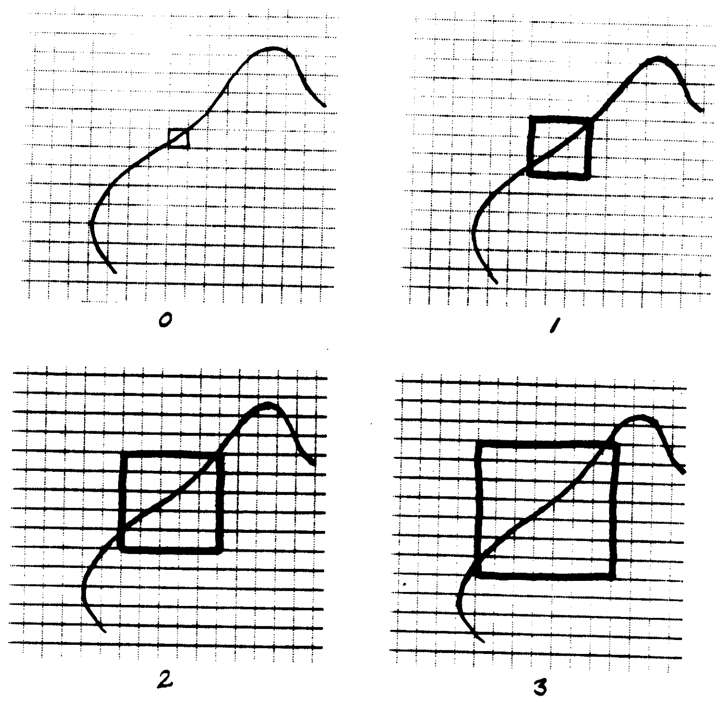


Figure 3.5: Computing boundary segments at multiple scales.

may be achieved by a simple application of concurrency. First, *basic* boundary segments at the smallest possible scale are made explicit by pixel-by-pixel tracing, over *elementary regions* defined by a fixed, *a priori* tesselation of the input. Then, to define segments at any larger scale, these starting segments are sequentially or hierarchically linked together. The issues to be addressed in the detailed design of this kind of simple segment-by-segment tracing process include (i) the choice of the prior tesselation over which the basic segments are defined; (ii) the representation of basic segments, and the details of the pixel-by-pixel tracing process used to define the basic segments; (iii) the mechanisms by which tracing links basic segments into longer ones. Further discussion of these issues is deferred to Chapter 5—the definition of basic segments is a kind of curve chunking process.

3.2.2 Computing boundary segment properties.

The explicit representation of a boundary segment consists of a set of distinguished locations within a region. The useful shape properties of such a set of locations for computing local prominence information include orientation, curvature, and extent, or measures closely related to them. The details of how to implement such measures in simple local hardware will not be explored here. The implementation of local prominence computations demonstrated here makes use of the following measures: (i) orientation of a boundary segment is taken to be the orientation of its axis of least inertia; (ii) a measure related to average curvature of a boundary segment—termed *pseudo-curvature* here—is the difference of the least and greatest moments of inertia divided by the sum of the least and greatest moments (see [Winston and Horn 84] and [Horn 86]); (iii) the extent of a boundary segment is proportional to the diameter of the smallest region including the segment, measured in elementary regions, not pixels.

3.2.3 Computing prominence.

Prominence measures must be computed independently for each defined boundary segment property and for boundary segments at each scale. Then these results must be combined into a single, "best" description of prominence at each location on the boundaries. A reasonable measure of local prominence of a boundary segment with respect to a particular property is the sum of the absolute differences between the value of the property for the given segment and the values for neighboring segments. The particular difference measure used depends on the property in question. The Directional Nearest Neighbor Graph, introduced in Chapter 2, provides an effective neighborhood representation for this application. (A position must somehow be associated with each segment. In my experiments, the center of the square region defining a segment—see Figure 3.5—was used.) The results of the prominence computation at a single scale for examples highlighting prominence in orientation are shown in Figures 3.7 and 4.13 (b). The result of the prominence computation at a single scale for an example highlighting prominence in curvature is shown in Figure 3.9.

One possible way of combining the results for a particular property across all scales is to take the maximum at each location. One possible way of combining the results for various properties is to sum, after scaling values for different properties into the same range. These possibilities require further study.

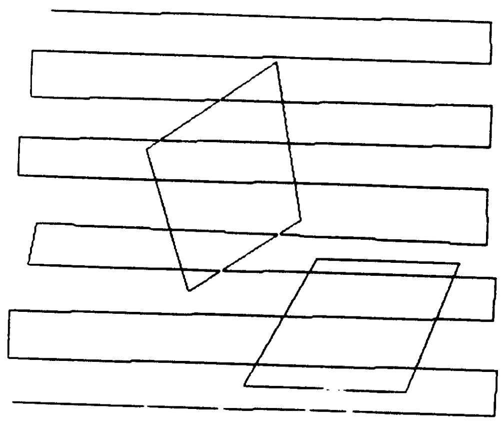


Figure 3.6: A figure with prominent local boundary orientations.

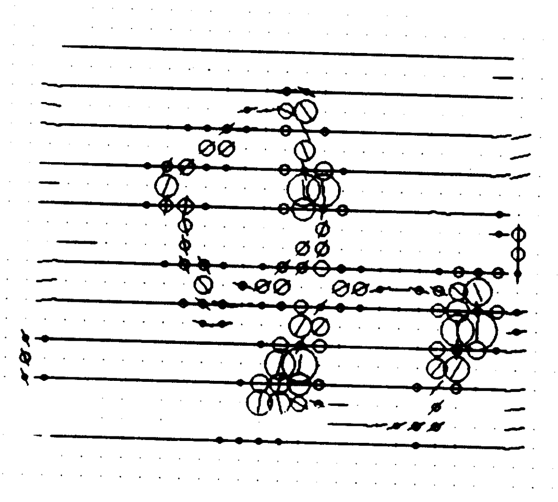


Figure 3.7: Boundary segment prominence in orientation at a single scale.

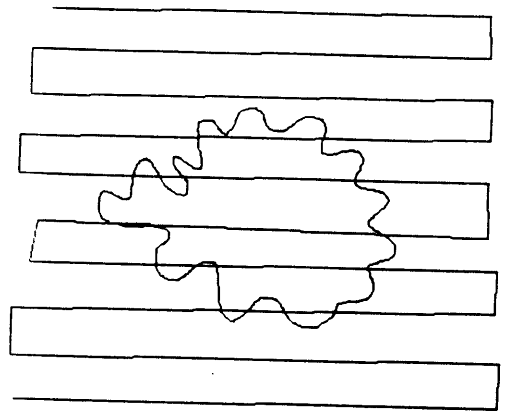


Figure 3.8: A figure with prominent local boundary curvatures.

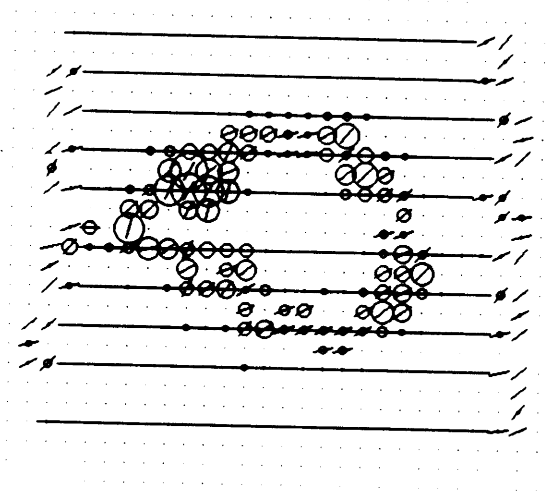


Figure 3.9: Boundary segment prominence in pseudo-curvature at a single scale.

3.2.4 Combining prominence information and boundary information.

The details of how local prominence information should be integrated with boundary information depends on the details of the particular chunking process to which the data is applied. There are two main possibilities. First, the local prominence measures may simply be *added* to corresponding discontinuity measures. Secondly, the prominence measures may be used to *threshold* the discontinuities, thereby extracting salient boundaries.

Chapter 4

Figural chunking

The preceding two chapters partially specified the input to chunking processes, taking a bottom-up approach. This chapter is the first of three devoted to the design of the chunking processes themselves.

The main concern of this chapter is indexing—the effective, time-efficient selection of processing locations in the course of focused vision. I will argue that (i) focused processing requires the capacity to select processing locations on the basis of descriptions of the prominent objects in the field of view, and (ii) *crude* descriptions of these objects or their parts are adequate for effective location selection. An examination of the intermediate vision processes that rely most heavily on processing shifts will provide the main justification for this claim. Furthermore, crude descriptions of scene entities may be useful for initial assignment of object reference frames and initial description of object configurations; both of these applications may support initial access into recognition memory.

I introduce a class of crude object descriptors for indexing termed *figural chunks.* Figural chunks describe relatively isolated configurations of boundary locations. Figural chunking computations are formulated and demonstrated, and the use of these chunks in indexing is discussed.

4.1 The need for low-level descriptors of objects

Spatially focused and goal driven processing brings with it the problems of making intelligent decisions about (i) what location to process next and (ii) what processes to apply there. The decision to process a location is intelligent if the processing done there contributes directly to the agent's current goals. The agent's goals usually pertain to physical objects and their relations, so the low-level vision tokens used to select processing locations should be descriptors of these objects. For example, vision might need to locate and analyze the largest yellow object in a complicated scene, as a step in deciding whether or not a tiger is present. In this example, items of information relevant to deciding to shift processing to a location are the color and relative size of the physical entity present at that location. Only approximate values of object properties are required to identify truly salient entities. The occasional need to locate non-salient objects may preclude the use of crude, rapidly available descriptors—and, hence, indexing—leading to substantially slower time performance.

To support the view that low-level vision should provide descriptors of scene objects, this section explores the visual processes that rely most heavily on processing shifts. The processes examined are tracking, search, and scanning.

Visual tracking of objects and events

The viewer and/or the viewed objects may move around in the environment, resulting in relative motion between them. If the processing focus coincides with an object at one moment, it may not at the next moment. Therefore, if processing is to remain with an object, some computations must be devoted to assuring this. These computations can be based on local or global motion information. Local motion information includes, for example, motion vectors of edge elements. Global motion information requires knowledge of the global spatial properties of the object being tracked, which makes it possible to select a new location based on similarity in global properties to a previous location. When objects move abruptly, object tracking may be more effective based on global information than local information.

Closely related to the tracking of moving objects is the problem of responding to changes (events) associated with objects, including sudden motion of a previously static object, or effectively instantaneous appearance or disappearance of an object due, for example, to rapid occlusion or exposure.

Visual search for objects

Visual search is the problem of shifting the processing focus to the location of a spatial entity whose properties match some arbitrary, top-down specification, where the relevant spatial entity has not been recognized to begin with. Visual search requires an "extract-and-test" procedure; to be efficient, only likely candidates should be extracted. The basis for generating plausible candidates over implausible ones is to compute crude descriptions of spatial entities and their properties ahead of time. For example, in Figure 1.4 rough elongation and size comparisons among the items in the scene might make it possible to locate something that might be a teaspoon before it is actually recognized.

Visual scanning of objects, sets of objects, and areas

Visual scanning is the problem of repeatedly shifting the processing focus to new locations, subject to some fixed, simple condition. The condition controlling the scan involves properties of the spatial entities, measured at each location, or relations between successive locations. The purpose of scanning is generally to integrate information across a set of closely related locations. This fact, and the simplicity of the controlling condition, distinguish scanning from visual search. Examples of visual processes mediated by scanning include counting of similar items (Figure 4.1); tracing or delineation of long-range groupings of similar items (Figure 4.2); and inspection of the constituent locations of extended objects or areas, as in finding simple paths (Figure 4.3).

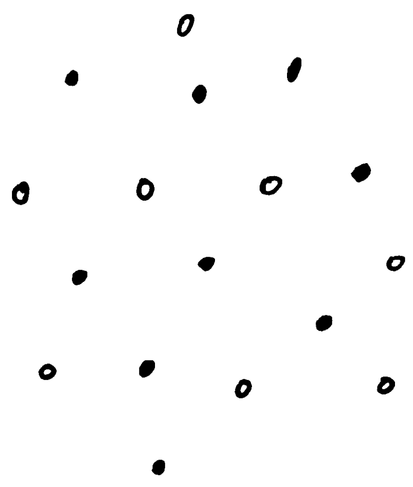


Figure 4.1: A task requiring scanning: count the dots.

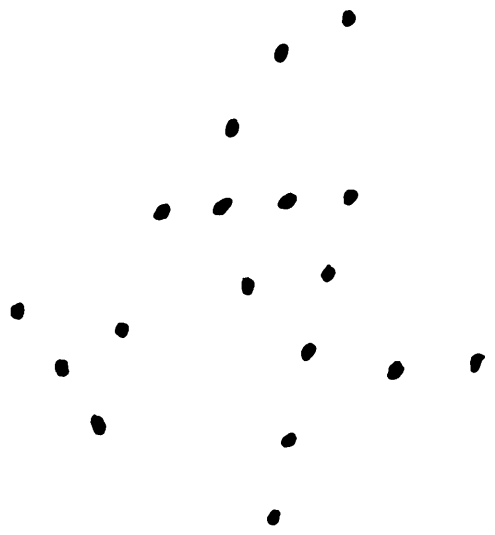


Figure 4.2: Perceiving the "7" shape requires scanning

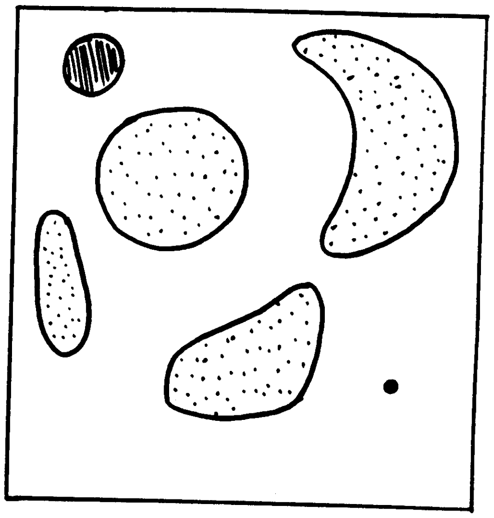


Figure 4.3: A task requiring scanning: is there a path to the dot wide enough for the black disc?

Tracking, search, and scanning compared

Tracking, search, and scanning are closely related, so some comparisons and contrasts between them are in order. Tracking is a strictly data-driven process; only simple properties of objects are relied upon; focus is maintained on a single object over the course of the process. Scanning is potentially a mixed control-structure process; only simple properties of objects are involved; processing shifts among similar items which may consitute the elements of a single object or an abstract group. Search is a primarily top-down, goal-driven process; arbitrary properties of objects may be involved; only one of the objects to which the focus of processing is brought is of ultimate interest.

All three processes are iterative and "shift intensive". Roughly speaking, they involve an iteration of three basic steps: (i) decide upon the next relevant location; (ii) execute a shift to the new location; (iii) perform appropriate computations at this location. For our purposes here, the decision step is the most important—*each of these processes could index relevant locations if descriptors of the scene entities meaningful to them were made explicit at the outset.*

4.2 Indexable visual objects

The preceding section brought out the importance of object properties in the selection of processing locations. The notion of an "object" is quite problematic, however. This section attempts to bring out some of the difficulties and specializes the notion of "object" for the purposes of indexing.

Characters, words, sentences, lines of text, and paragraphs may all be viewed as objects, but the visual processes for extracting them may be quite varied and complex. Which of these entities, if any, could one reasonably expect low-level vision to provide indicators of? In the perception of natural scenes, one intuitive notion of an object is rooted in the phenomenon of physical coherence, often indicated by visible surfaces or occluding boundaries. Visible surfaces or occluding boundaries are not by themselves a reliable basis for defining objects, however. This problem is illustrated by the famous "man on a horse" example. In general, object boundaries—boundaries between image regions corresponding to the surfaces of different physical objects— and internal boundaries—due to changes in local visual properties, between image regions corresponding to the surface of the same physical object—cannot be distinguished from each other solely on the basis of information in the image. [1] As such, the process for distinguishing the man from the horse may be no less complex than that for extracting this sentence.

Descriptions of the surface markings in a scene may be no less useful than descriptions of object silhouettes. For instance a tiger's stripes may constitute visual objects that are as useful for detecting it as its outline. Moreover, there are other possible grounds for establishing physical coherence than visible surfaces or occluding boundaries. For example, schools of fish and flocks of birds display coherence in the form of common motion of their members. Proximity and similarity are well known as grouping criteria for defining visual objects. It is therefore useful to make a distinction between visual objects defined by continuous occluding boundaries, termed *primitive objects* here, and *abstract objects* defined by proximity/similarity, or other grouping of primitive objects.

[1]Disparity information or curve continuity make it possible to distinguish the two types of boundaries only sometimes.

There is no unique, bottom-up object-decomposition of any given input that can serve all visual tasks. Whether or not some configuration in the input should be interpreted as a unit depends on the immediate goals of vision. The criteria used to define objects for recognition (or reading, or manipulation) are certainly not appropriate for the rapid selection of processing locations. For recognition, it is necessary to describe an object's shape in detail. Because scene entities may take on an unbounded variety of shapes and may occur at many locations and orientations, spatially uniform and parallel processes for extracting these entities in detail would be prohibitively costly. The extraction of detailed shape involves relatively slow serial processes, like boundary tracing; the criteria for extracting an object include, for example, connectivity and boundary continuity. For indexing, it is necessary to describe rapidly, but perhaps not accurately, the overall shape properties, such as extent, elongation, and orientation, of some prominent objects; detailed shape is not required at this stage. To provide the required speed, these rough descriptions must be generated by bottom-up, spatially uniform, "hard-wired" processes. The objects in a scene are implicitly described in the input by configurations of boundary locations that may take on a very wide range of shapes. To be feasible, a parallel, spatially uniform process for detecting possible objects must be limited in the range of configurations of boundary locations it can define and evaluate.

A useful, crude criterion for defining objects for the purpose of indexing is *local spatial isolation*, a measure of the distance of a given set of boundary locations from surrounding boundary locations. By the isolation criterion, Figure 4.4 (a) would be described as containing two objects; by the criterion of *connectivity* it contains four.

The *strength* of a boundary location is some measure of discontinuity in a local surface property, some measure of local prominence of a boundary segment, or some combination of the two (see Chapters 2 and 3). An overall difference in strength between a given set of boundary locations and surrounding boundary locations is also a useful criterion for defining objects. For example, in Figure 4.4 (b), the bold circle is no more isolated, strictly speaking, than any other circle in the figure; however, its locations are not near any boundary locations *of similar contrast.* Therefore, it is useful to broaden the notion of "isolation" to include relative strength as well as proximity. This broader notion is termed

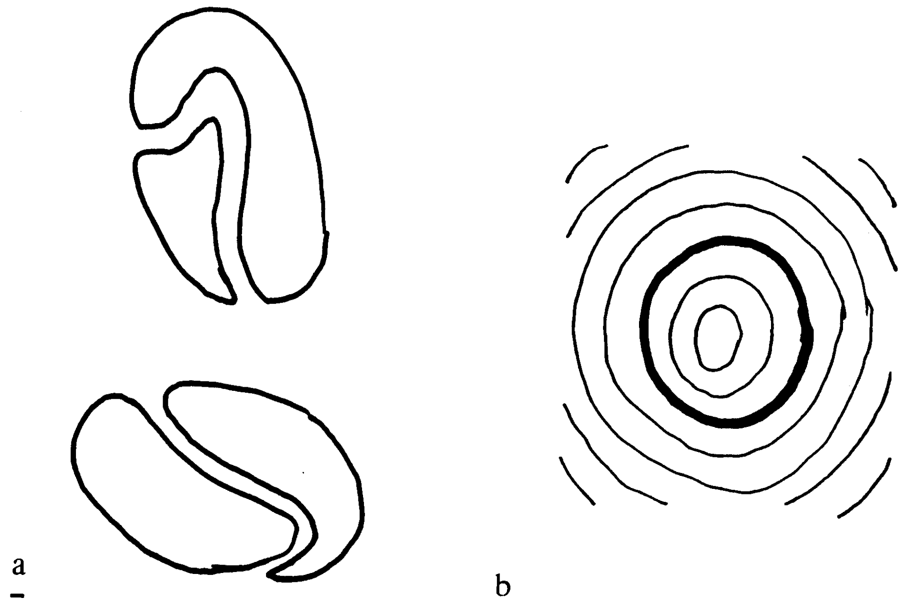


Figure 4.4: (a) A figure containing four objects by the criterion of connectivity, and two by the criterion of isolation. (b) A broadened notion of isloation - the bold curve is isolated with respect to boundary locations of similar contrast.

relative isolation. [2]

4.3 Figural chunks

This section is concerned with the low-level, crude detection of primitive objects and surface markings, which will be referred to in brief as *objects.* (See Figure 4.5. It may be possible to detect certain classes of abstract objects by methods similar to the ones described here, but that problem will not be addressed.) The input assumed is a pointwise boundary representation of the scene—each location in this array describes discontinuity in some local property at a point, or local prominence of a boundary segment.

A parallel, spatially uniform process for detecting objects must accomplish three things: (i) extract configurations of boundary locations; (ii) measure the relative isolation of the configurations; and (iii) associate measures of position, extent, elongation, and orientation—of

[2]One obvious possibility for implementing a measure of relative isolation is to apply a strict isolation measure after thresholding boundary strengths, either locally or globally.

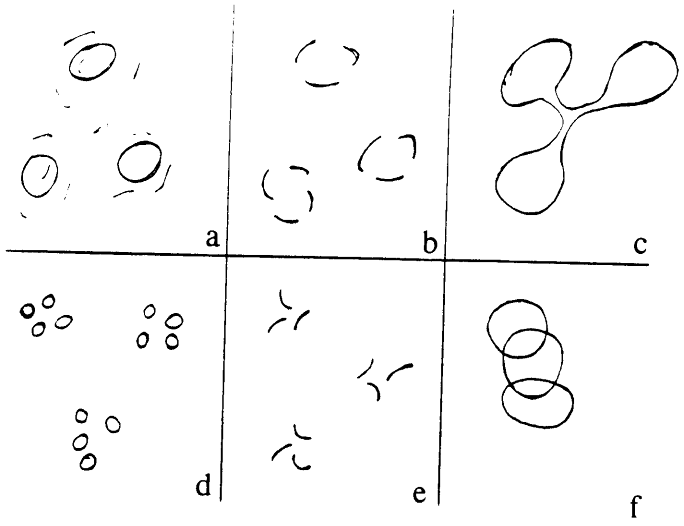


Figure 4.5: Each frame in this figure may be usefully described as containing three objects. The figural chunking scheme introduced in this section does not necessarily apply to cases (d), (e), and (f).

the bounded regions—with the configurations.

A way of extracting a configuration of boundary locations that lends itself to parallel, spatially uniform implementation is to *fit* configurations of predefined shape, size, and orientation to the boundary array. A predefined configuration used for this purpose is termed a *basic configuration.* Suppose that at a given location p, the best-fitting member of the set of basic configurations C is found to be c, according to some measure of fit f. Given f, it is possible to single out the configuration of boundary locations that c is a fit to. Such a configuration is termed a *boundary configuration.* Useful attributes of a basic configuration, such as size, elongation and orientation, can be directly attributed to a boundary configuration it fits.

There are three main problems to be solved. One problem is to define useful basic configurations. The discussion and demonstrations of figural chunking in this report make use of only one basic configuration —the ellipse. The ellipse is a natural choice because it is one of the simplest shapes that has attributes of elongation and orientation. Additional candidates include *bars* and *arcs* (see Figure 4.6). The second problem for figural chunking

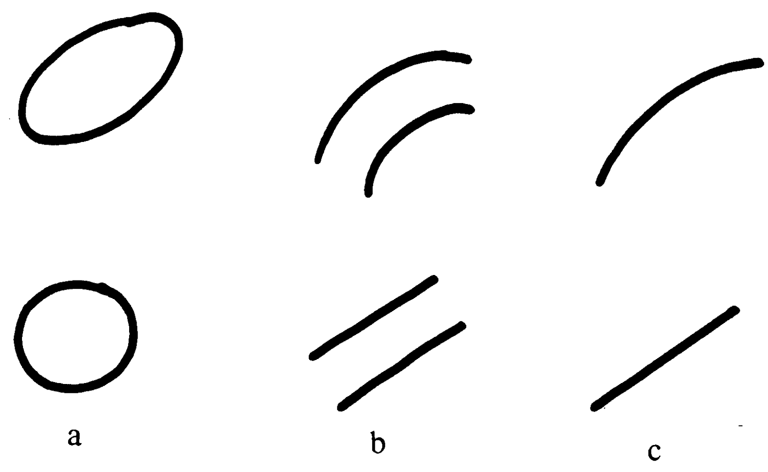


Figure 4.6: Some useful basic configurations. (a) Ellipses. (b) Bars. (c) Arcs.

is to define an effective measure of fit for extracting boundary configurations. An important requirement is that the measure permit considerable deviation in shape between the boundary configuration and the basic configuration, while capturing similarity in overall shape parameters, because object boundaries will seldom closely match the basic configuration. The third problem is to develop a measure of relative isolation.

The alliance of the set of basic configurations, the measure of fit, and the measure of relative isolation support an operational definition of a visual object for the purpose of indexing. A *figural chunk* is defined to be a boundary configuration whose measure of relative isolation is a local maximum. A figural chunk is a *crude* descriptor in two senses. First, it provides only an *indication*, not a guarantee, that the best-fitting boundary configuration at a location may correspond to a meaningful scene entity. Secondly, the shape attributes associated *a priori* with the basic configuration defining the chunk describe the scene entity only approximately.

4.4 Detecting figural chunks

This section develops a technique for detecting figural chunks. The main purpose of the discussion is to demonstrate that figural chunks can be generated by simple, efficient computations. The implementation strategy presented here is not the only possibility. The discussion is centered on one basic configuration—the ellipse. First, a combined measure of fit and relative isolation is presented. Then, an affordable mechanism for spatially uniform, parallel application of this measure is formulated.

4.4.1 Measuring fit and relative isolation

Let B denote the array of local discontinuity/prominence values that is the input to figural chunking. Let E denote an elliptical mask at a particular scale, eccentricity, and orientation. A measure of local fit of E is given by convolving B with a filter f_E that is a smoothed version of E: $\mathcal{F} = B * f_E$, where $f_E = S * E$ and S is some smoothing filter. Intuitively, this measure is high when the locations of the boundary configuration are near those of E.

An indicator of relative isolation is a decrease in $\mathcal{F}(x, y)$ as scale s of the ellipse increases. Hence, a relative isolation measure is given by convolving B with a filter $f'_{E(s)}$ that is the first difference $f_{E(s)} - f_{E(s+1)}$ (see Figure 4.7): $\mathcal{I}_E = B * f'_E$. The filter f'_E is an elliptical center/surround operator that can also be defined by sweeping a one-dimensional first derivative operator around E.

Let $\mathcal{I}$ denote the space generated by computing $\mathcal{I}_E$ for elliptical masks at all descriminable combinations of scale, eccentricity, and orientation. Figural chunks are defined by local maxima in $\mathcal{I}$. Let E_{max} denote the elliptical mask corresponding to a particular local maximum in $\mathcal{I}$ at position x, y, and let $\mathcal{I}(E_{max}(x, y))$ denote the corresponding value of $\mathcal{I}$. A figural chunk is defined to be the set of boundary locations falling inside $E_{max}(x, y)$. $\mathcal{I}(E_{max}(x, y))$ is referred to as the *strength* of the figural chunk.

In the simulation described below, f_E was defined as follows:

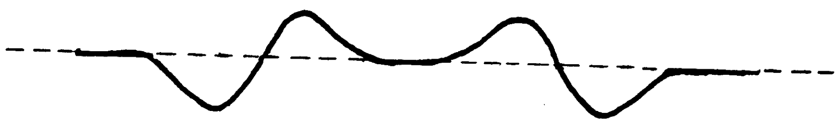


Figure 4.7: Profile of an elliptical operator for measuring relative isolation.

$$f_E(x,y) = \begin{cases} d_E & \text{if } d_E \leq 1 \\ 0 & \text{otherwise,} \end{cases}$$

where $d_E(x,y) = x^2/a^2 + y^2/b^2$. For simplicity of implementation, at most one figural chunk was defined at each location —a figural chunk was defined at a location if $\mathcal{I}(E_{max}(x,y))$ had the largest value in the vicinity of that location.

4.4.2 Spatially uniform, parallel application

Two main practical issues arise in the computation of figural chunks, both pertaining to hardware cost. First, it is necessary to limit the set of discriminable positions, eccentricities, orientations, and scales supported. In the most time-efficient parallel hardware implementation, for each possible combination of descriminable values of the five ellipse parameters (x, y, a, b, θ), a unique processor would be dedicated to computing one value i_E in the convolution $\mathcal{I}_E$. The number of processors required is approximately the product of the number of descriminable values of each of these parameters. Therefore a cost-performance tradeoff must be made.

The second practical issue concerns the manner in which variation in scale is implemented. The processor for applying f'_E at a particular location would take input directly from each location in an elliptical region. In a straightforward implementation of scale variation, both the length and number of the input connections would grow with scale. When the extent of an ellipse approaches the size of the input, the required communication degree (number of input connections) of the associated processor approaches the number of input locations. A more economical way of implementing scale variation is to fit ellipses of limited extent to a set of versions of the boundary array created by sampling at a series of resolutions. This set is termed the *data pyramid.* A simple pyramidal figural chunk detection scheme is formulated in the following paragraphs. The maximum major radius of an ellipse used in this scheme is a. The scheme is used to construct not only a data pyramid but also a *processor pyramid*—a set of processors whose inputs are provided by the data pyramid.

Let C_E denote the number of locations in the filter f'_E—this is the number of input connections to the processor for computing $I_E(x,y)$. An upper bound on C_E is given by a circle of radius $2a$: $C_{max} = 4\pi a^2$.

Let d_e and d_θ denote the number of discriminable ellipse eccentricities and orientations, respectively. At each location of every size-scaled copy of the boundary array, $N_l \approx 2d_e d_\theta$ processors are required for computing $I_E(x,y)$ for all different ellipses. (The factor of 2 roughly accounts for the additional processors required for detecting local maxima in $\mathcal{I}$ at each location.)

Suppose the boundary array input to the figural chunk detection process is a square array of width w_0. An exponential pyramid may be defined using repeated scaling by some factor f, $0 < f < 1$: $w_{l+1} = fw_l$, where w_l denotes the width in pixels of the square array at level l of the pyramid. The scaling factor f is chosen so that scale changes as slowly as possible, such that for all l, $w_l \neq w_{l+1}$. The top level of the pyramid is a square array of width $w_h = f^h w_0 \approx 4a$, where h is the (zero-origin) height of the top level; at this level, an object whose diameter is half that of the input array can be detected. The total number of array locations in the pyramid, denoted by N_P, is given by the following expression:

$$N_P = \sum_{l=0}^{h} (f^l w_0)^2 = w_0^2 \sum_{l=0}^{h} f^{2l}$$

Therefore, approximately $N_l N_P$ processors are required altogether. An upper bound on the total number of *connections* required by these processors is $C_{max} N_l N_P$. In a simulation described in the following subsection, which seemed to provide generous precision in describing the properties of scene entities, the following orders of magnitude were used for the preceding numbers. C_{max} and N_l were both be on the order of 10^2; N_P was on the order of 10^4. The total number of processors required was on the order of 10^6. The total number of connections required was on the order of 10^8.

To construct the data pyramid, an operation is needed for sampling the input array. One simple way of doing this is to assign each location of the array at level l of the data pyramid the maximum of the values in the corresponding *region* of the input.

There is a severe quantization effect at very low sampling resolutions (e.g., see Figure 4.9 (d)). Performance of the measure of fit is significantly improved by the following modification: only the boundary value on each radius of the ellipse for which the product $d_E(x,y)B(x,y)$ is a maximum is allowed to contribute to the sum $I_E(x,y)$.

4.4.3 Performance

This scheme was simulated with the following parameters: $a = 8$ pixels; $d_e = 7$ (i.e., b ranged from 2 to 8 in increments of 1); $d_\theta = 12$ (i.e., θ ranged from $0°$ to $165°$ in increments of $15°$); $w_0 = 128$ pixels; and $f = 0.8$. The height of the pyramid h was 10. The number of convolutions computed was 84. The total number of locations in the pyramid was approximately 45,000. This implies roughly 3.78 million processors in a strictly parallel hardware implementation. It is possible to serialize over the convolutions $\mathcal{I}_E$—if only one processor is provided per location in the pyramid, 45,000 processors are required, and the computation involves 84 iterations.

The simulation was implemented on the Thinking Machines Corporation Connection Machine [Hillis 85], a single instruction stream, multiple data parallel computer. The computation of convolutions was highly serialized, due to the limited number of processors avaliable —one parallel convolution $\mathcal{I}_E$ was computed at a time. Each convolution involved serial scanning over the locations of a stored mask. The Connection Machine run time of the simulation, on half the processors of a 16,384 processor machine, was roughly 4 minutes.

Figure 4.8 shows the ten highest-ranking figural chunks for a hand-drawn binary input. Each figural chunk is represented graphically by (i) a pair of perpendicularly bisecting dashed lines, which represent the major and minor axes of the defining ellipse; (ii) a set of shaded square regions forming a crude, discrete ellipse; and (iii) the set of black input pixels inside the crude ellipse, which constitutes the defined figural chunk. Two numbers are displayed near the center of each ellipse: the lower number is the chunk's strength; the upper number is the chunk's rank—all defined chunks are ranked in terms of strength.

The main considerations in evaluating the performance of the figural chunking scheme are the effects of shape variation, context, and noise. An extensive performance evaluation has not been carried out, but it is possible to make some general observations.

First, the proposed measure of fit copes well with shape variation, as it was designed to do. When the shape of an object deviates a great deal from that of the basic configuration, however, there may be no figural chunk corresponding to the entire boundary (for example, see Figure 4.12). If there is one, it may be ranked lower than other chunks corresponding to *parts* of the boundary. For example, in Figure 4.8, the chunks ranked second and fourth are both subparts of the chunk ranked sixth. A single basic configuration can capture only part of the range of possible shapes occuring in images. For example, Figure 4.12 shows the three strongest figural chunks generated for an input consisting of the intensity boundaries of a connecting-rod—the long, narrow central portion of the connecting-rod was not captured by any ellipse, but probably would have been captured by a bar-shaped basic configuration.

Secondly, context can have a significant effect, in the form of strong figural chunks that do not correspond to any meaningful scene entity. Figure 4.10 shows the four strongest chunks for a cartoon figure. The weakest of the four is a spurious output due to a circular configuration of boundary locations whose relative isolation measure was coincidentally high. The geometric distortion (quantization effect) resulting from the simple sampling scheme used to build the data pyramid certainly worsens the effect of context. (Figure 4.11 shows two extreme and two intermediate levels of the data pyramid.) It would therefore be interesting to investigate other ways of scaling and sampling the input. It seems, however, that spurious responses due to context are unavoidable *in principle*, due to the local nature of the computations being used. At the moment, it is difficult to judge the significance of spurious figural chunks, because the serial scene analysis processes that will make use of figural chunk information have not received much study. It seems reasonable to assume, though, that figural chunking would be very beneficial even with high incidence of "false positives"—it is much better to have indicators of objects that are meaningful only half or even a quarter of the time than to have no object indicators at all.

All of the examples demonstrate that small amounts of noise in the form of (i) omission of some of the locations of an object boundary and/or (ii) addition of noisy segments or internal boundaries near object boundaries do not prevent the detection of an object. Noise does affect what ellipse will be a best fit at a particular location though and, as a result, the properties attributed to the best-fitting boundary configuration.

Figure 4.13 shows an array of local orientation prominence information (see Chapter 3) for a synthetic input similar to Figure 1.1, and the three strongest figural chunks resulting for that input. Two out of three of the strongest chunks coincide with the prominent blob figures—an encouraging but inconclusive result. This result may be explained by the fact that the top and bottom portions of the blob that was not detected run parallel to neighboring curves and are therefore not locally prominent. It may be that additional local measures, e.g., local boundary density, would lead to the detection of this blob. The example does illustrate, nonetheless, that local prominence information can support the parallel detection of figures in context.

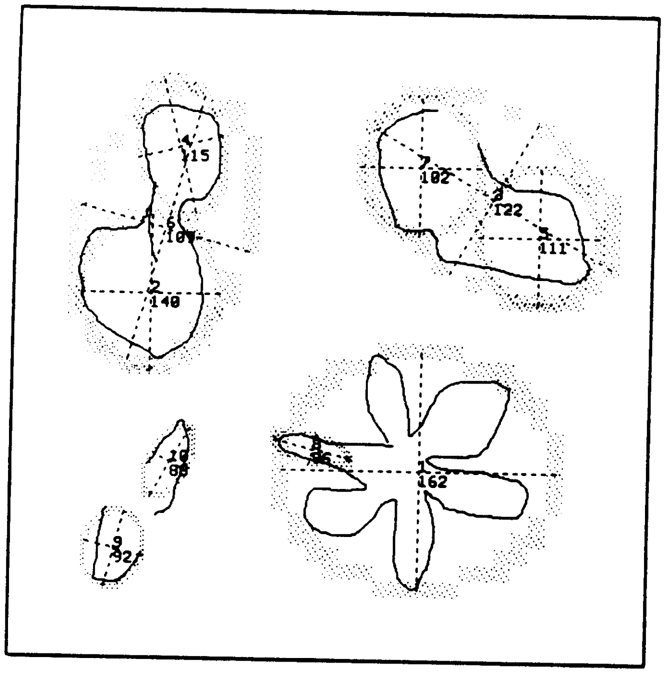


Figure 4.8: An input demonstrating the ability to cope with shape variation.

4.5 Indexing figural chunks

Section 4.2 defined crude descriptors of objects for indexing, and fast, low-level computations for generating them from the boundary representation. This section examines a number of bottom-up and top-down criteria for selecting figural chunks in the course of serial, spatially focused processing.

Bottom-up location selection

The rapid and correct interpretation of a scene must not depend critically on the availability of *a priori* knowledge about it, because scenes can change frequently and dramatically. In the absence of *a priori* information, data driven methods are needed to guide the initial processing [Ullman 84]. Two forms of prominence information provide the basis for bottom-up selection of processing locations. One is the local boundary prominence information introduced in Chapter 3. The other is prominence of the shape attributes associated with figural chunks.

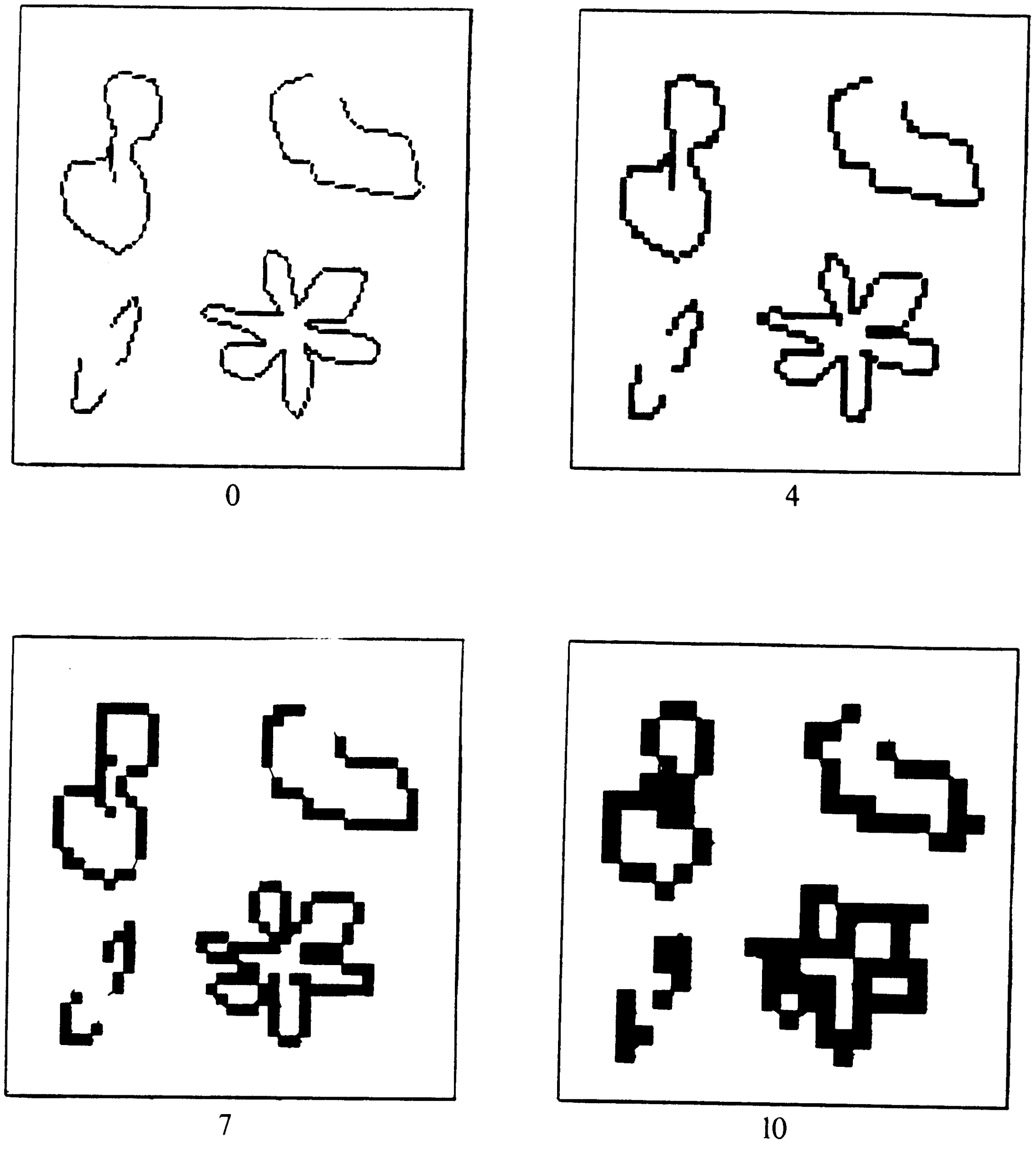


Figure 4.9: Levels 0 (lowest), 4, 7, and 10 (topmost) of the data pyramid for the preceeding example.

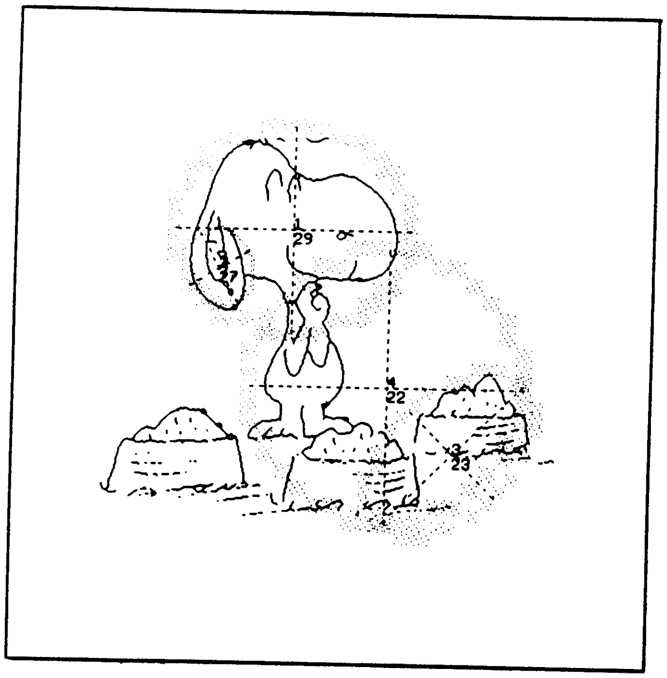


Figure 4.10: An input illustrating the problem of context. The figural chunk ranked fourth is spurious.

Chunk strength/boundary prominence. In all of the preceding examples of figural chunking, chunks were ranked according to the value of $\mathcal{I}(E_{max}(x,y))$. This measure of relative isolation reflects boundary prominence (see Chapter 3). For a set of objects with roughly equal fit to the basic configurations, those with stronger boundaries will be processed first. (In Figure 2.7, one blob has more abrupt changes at its boundary. By the chunk strength ranking criterion, this blob would be indexed first.)

Prominence of figural chunk attributes. The global properties of a scene entity may stand out in comparison to those of neighboring entities (Figures 1.7 and 3.4). In order to make use of this source of figural prominence, a further computation is required whose input is the set of figural chunks, and whose output is a measure, for each figural chunk, of its prominence with respect to its neighbors. Once again, the Directional Nearest Neighbor Graph (see Chapter 3) provides a useful proximity representation for establishing neighbors for this application.

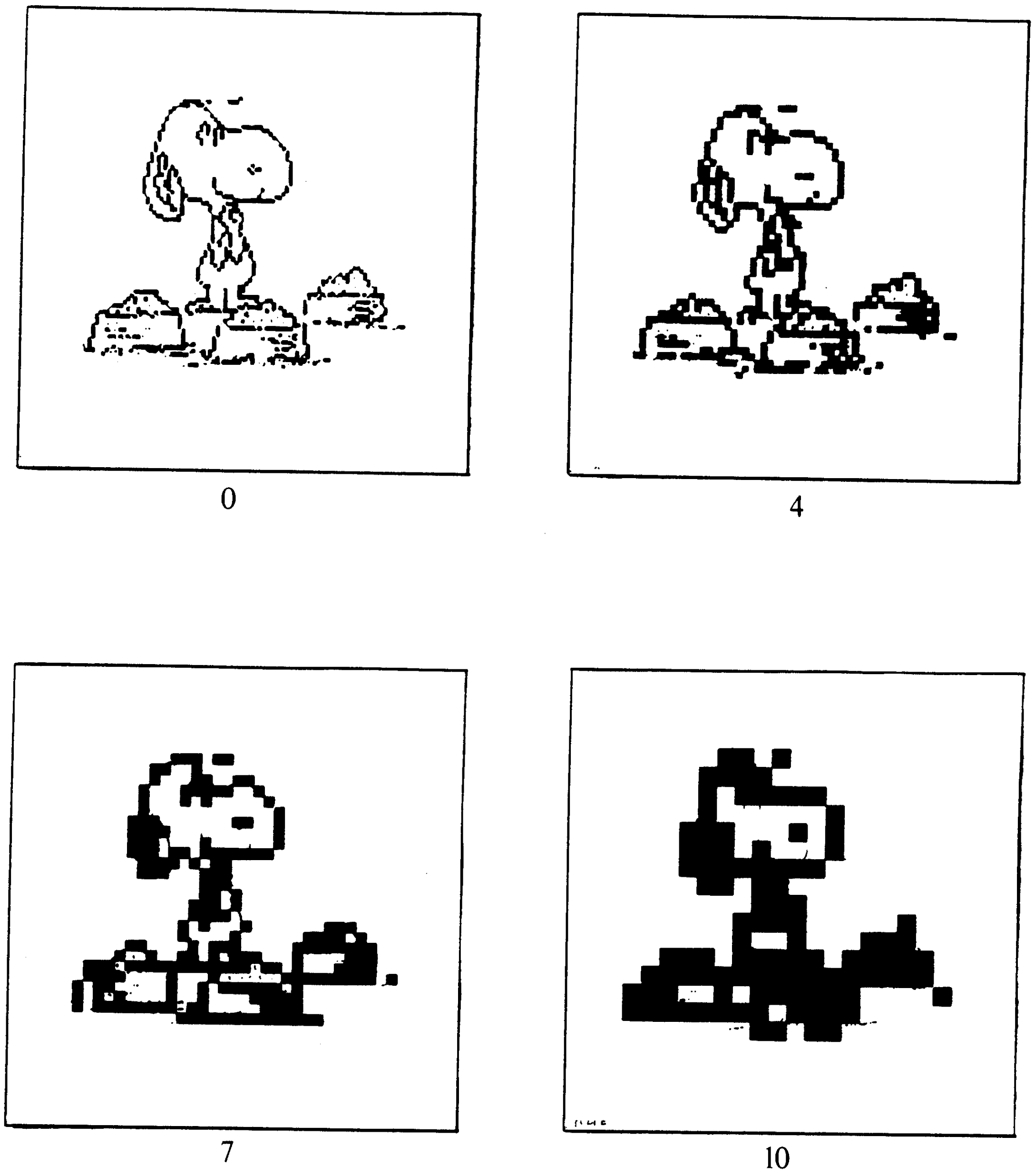


Figure 4.11: Levels 0 (lowest), 4, 7, and 10 (topmost) of the data pyramid for the cartoon input.

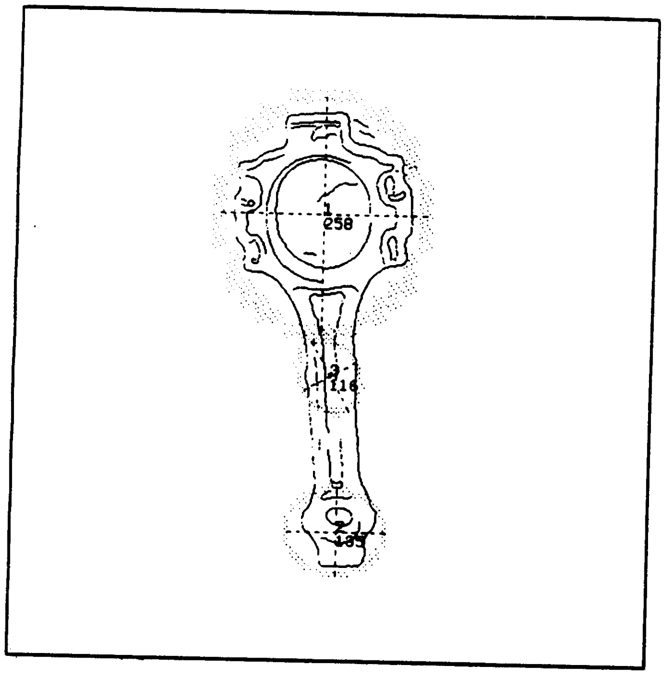


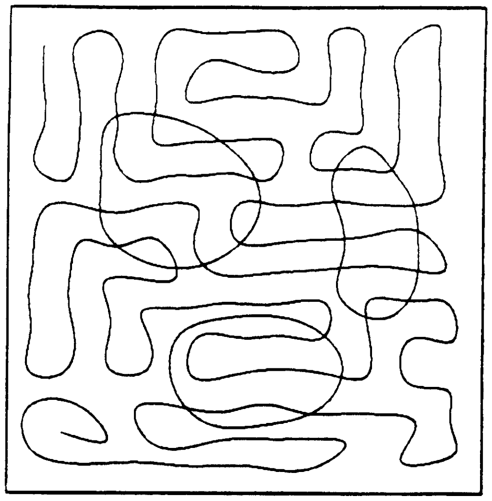
Figure 4.12: Ellipses are not enough—the bar of the connecting-rod might have been captured using a bar-shaped basic configuration.

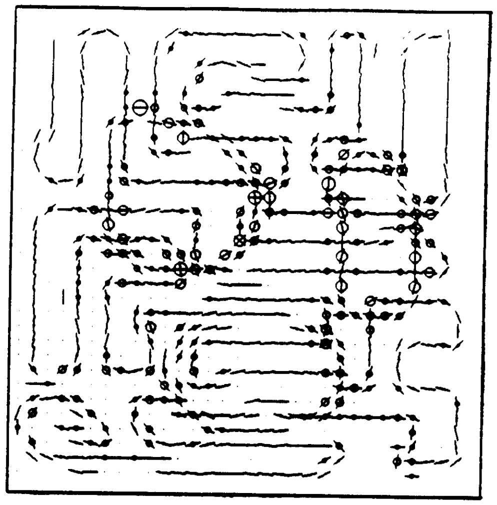
Proximity-similarity facilitation. The processes that distribute similar entities in a scene tend to operate continuously in space. Therefore, proximity and similarity to the most recently processed figural chunk are useful joint criteria for selecting the next figural chunk [Koch and Ullman 84]. This is one possible basis for visual scanning (see Figures 4.1, 4.2, 4.3).

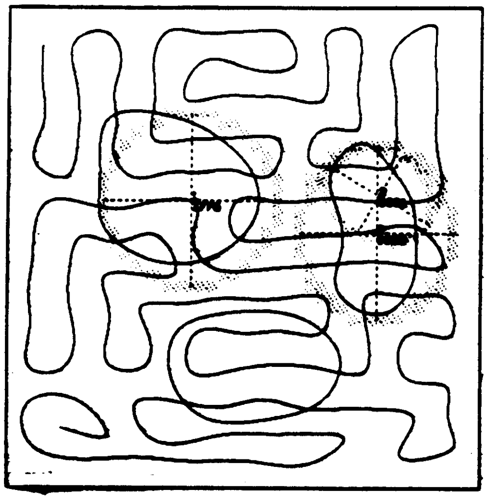
Top-down location selection

When *a priori* information is available about the scene to be processed, this may be used to select figural chunks. Top-down criteria for selecting processing locations include values of chunk attributes, connectivity relations, and position relations.

Facilitation and suppression of labeled locations. Coloring, tracing, and marking operations can be viewed as defining a special "subscene" of the scene—the subscene is the set of locations labeled by the operation. Processing may be restricted to (or excluded from) such


a


b


c

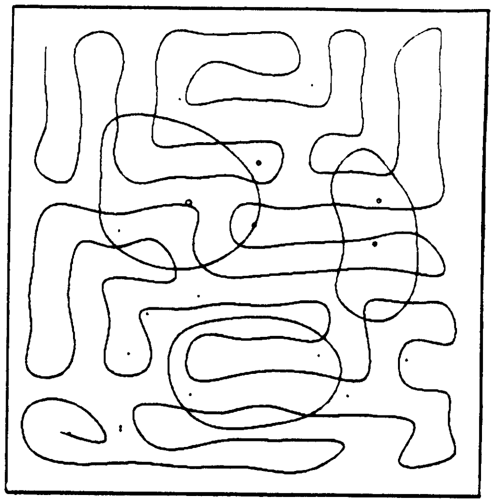

d

Figure 4.13: (a) The input. (b) Orientations and orientation prominence of local boundary segments. (c) The three strongest figural chunks (one of the three prominent blobs was missed). (d) Radii of small circles are proportional to figural chunk strength.

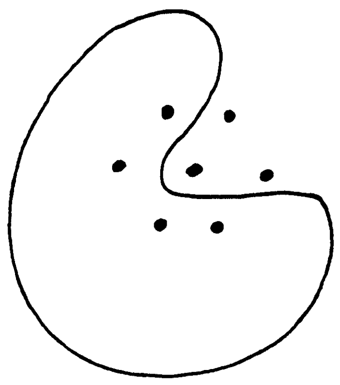


Figure 4.14: Count the dots inside the curve.

a subscene by facilitating (or suppressing) indexing of a figural chunk according to whether its location is specially labeled or not. One way of performing the task of Figure 4.14, for example, it is to first color the interior of the curve and then count, indexing only to dots that are at colored locations.

Figural chunk attributes. The items of information relevant to selecting a location for processing are the properties, such as color, extent, etc., of the scene entity present at that location. This sort of information may mediate selection directly or indirectly. The bottom-up selection strategy discussed above under the heading "Prominence of figural chunk attributes" is an example of indirect application of this information. For instance, if we are asked to describe the largest yellow object in a scene, the location of the relevant object might be quickly selected for processing owing simply to the fact that that object is prominent by virtue of relative size or color. In a direct application of spatial property information, the selection process would select chunks whose associated attributes match those specified in the query, e.g., "yellow" and "large".

4.6 Implications

The use of figural chunks has implications for the computation of spatial properties and relations relevant to both the design of machine vision systems and the study of human vision. Figural chunking raises questions regarding human visual perception at two levels. At the more general level, the hypothesis that figural chunks are made use of may lead to novel interpretations of perceptual phenomena, without regard to the particular proceses by which the figural chunks are generated. Some pertinent observations in this regard are made in this section. At a more detailed level, it may be possible to make hypotheses about how figural chunks are generated in the human visual system, based on psychophysical studies and an understanding of the range of possiblities for computing figural chunks—future work may take up this intriguing problem.

The two-stage spatial analysis framework provides two distinct levels of description of the scene. The figural chunk level is crude and limited, but immediately available and unlimited in range. The other level can be as detailed as necessary but is demand-driven, spatially focused, and potentially time-consuming. The detailed information must be spatially indexed by the crude information. The overall implication of this framework is that the speed and assurance with which a visual task can be accomplished depends on the level of description that it utilizes. Execution of a task will be much faster at the crude level, so the framework provokes a bias toward solutions at that level. Such solutions will apply under more restrictive conditions, though, than their detailed level counterparts. Each of the following paragraphs examines how a particular class of visual tasks can be accomplished using the figural chunk representation.

Figural chunks explicitly describe certain approximate properties of simple objects or object parts. Therefore, some tasks that involve the comparison of these properties could be performed without necessarily extracting the relevant objects in detail. It would be much faster, for example, to determine that two figures have roughly the same size or orientation than that they have matching shapes. Similarly, relative position judgements like "above", "left-of", "inside", etc. can be obtained almost directly from figural chunk information.

For example, given two circular figural chunks A and B, A is inside B if the sum of A's diameter and the distance between the centers of A and B is less than the diameter of B.

Figural chunks make possible the immediate, parallel classification of global configurations of many elements. It is possible to create a summary description in which the distributions of chunk attributes in a global configuration is expressed concisely. Figures 1.2 and 3.4 give rise to an assured but qualitative sense of the overall arrangement of many similar items. This description may be augmented by a serial, detailed local description that moves from one location to another. The subjective impression of immediately available panoramic detail in such scenes may result from the effective integration of the immediate, parallel, global summary description with the incremental, detailed, local descriptions. The detailed description need not be performed exhaustively over the scene—if the summary description indicates that the scene has uniform or slowly varying structure, it is possible to extrapolate the local detailed description.

Another important implication of indexing on the basis of figural chunks is that a figure will be indexable only if it or its simple parts give rise to prominent figural chunks. Consider Figure 3.3 – the long curves stand out. This phenomenon is independent of the number of curves in the input or their lengths. In Figure 4.15, however, a meandering curve that is more than twice as long as any other curve in the figure is effectively hidden. If it is assumed that *connected* and/or *continuous curves* are distinguished and described in low-level vision, say by curve tracing, then the discrepancy between these two figures is hard to account for. Processes amounting to curve tracing should have no more difficulty describing a winding curve than a straight one. In the view I propose, configurations of boundary locations that may be approximately described as *simple arcs* are individually described in early vision, but more complex curves are described only in terms of their simpler components. As such, the long lines in Figure 3.3 give rise to local descriptors that are outstanding in length, relative to the local descriptors generated for the other lines. The longer curve in Figure 4.15, on the other hand, gives rise to no outstanding descriptors. Similar arguments apply to Figure 4.16, in which a curve twice as long as any of the others is hidden. In this case, the early descriptors of the figures express the properties of the small, compact region effectively occupied by each curve, not properties of the curves themselves.

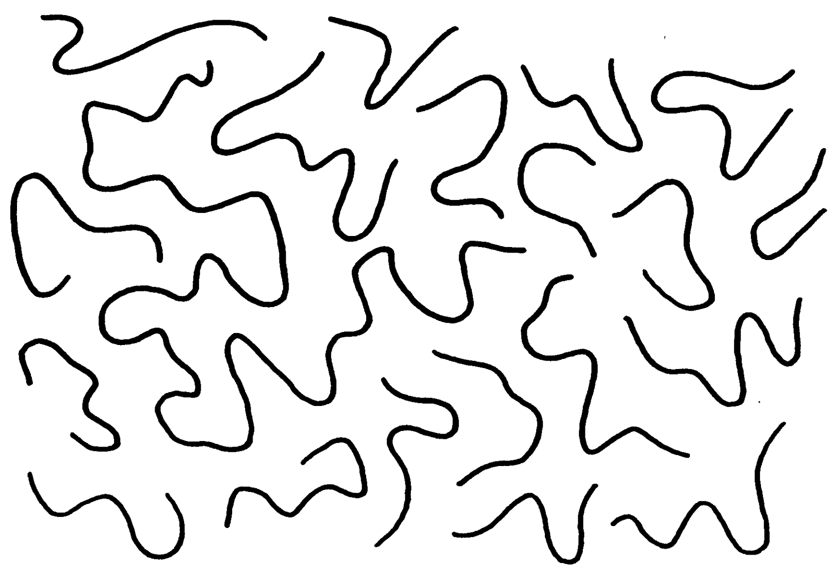


Figure 4.15: Long, meandering streaks are easy to hide.

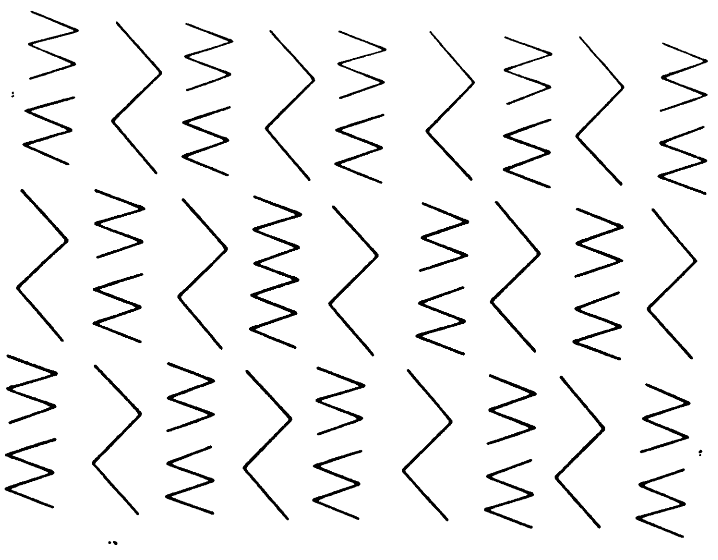


Figure 4.16: One curve in this figure is twice as long as all the others.

A striking phenomenon of human visual perception is the tendency to interpret outline figures spontaneously as *regions* bounded by curves, rather than as curvilinear figures. This happens even when the outlines are incomplete, as in Figure 4.5 (b). This observation seems at odds with an account of object extraction based on segmentation into connected components. An account of object extraction in which region detection processes (like figural chunking using ellipse configurations) are applied to boundary information may provide part of the explanation of this phenomenon.

Chapter 5

Curve chunking

The preceding chapter developed means for efficiently focusing processing on the locations of relevant scene entities, on the basis of crude early descriptors of these entities. As noted in Chapter 1, a fundamental operation for establishing detailed shape properties of an object is that of making its boundary distinct from other boundaries. The main aim of this chapter is to develop time-efficient schemes for curve and boundary activation based on a simple, local model of computation.

The central idea of this chapter is that it is possible to trace a curve segment-at-a-time rather than pixel-at-a-time. A *curve chunk* is defined to be a region containing a single segment of a curve. This definition is region-based owing to the combinatorics of labeling curve segments in parallel. Curve chunking is open to a variety of implementation strategies; four techniques are developed and evaluated here. First, a simple scheme is defined for generating curve chunks whose extent is limited with respect to the size of the input. Then, a direct method of defining curve chunks of unlimited extent, but limited local and total curvature, is presented. Next, a divide-and-conquer method for generating curve chunks of unlimited extent, with no explicit restrictions on curvature, is developed. Finally, the power of the divide-and-conquer scheme is extended by adding a local labeling capability. The behavior of each method is evaluated with regard to its performance benefits to tracing. All of these parallel schemes were simulated and tested on a serial machine. The final section of this chapter discusses the implications of chunk-based tracing for the establishment of shape properties and spatial relations.

5.1 Curve chunks

In this section, curves and curve tracing operations are defined from a computational point of view, and a definition of a curve chunk is derived.

Curves

A curve is a set S of curve elements such that (i) for any two elements p, q in S, there is a path from p to q, and (ii) *either* all elements of S have exactly two neighbors in S *or* $|S|-2$ elements of S have exactly two neighbors in S and two have exactly one. (Assume that $|S| \geq 2$. A path from an element $p = p_0$ to an element $q = p_n$ is a sequence of elements $p_0, p_1, ... p_n$ such that p_i is a neighbor of p_{i-1}, $1 \leq i \leq n$ [Rosenfeld 79].)

The preceding definition characterizes a broad class of curvilinear configurations. It is useful to make a clear distinction between curvilinear geometry in general and the detailed nature of the local curve elements and the neighborhood relation on them. This chapter defines curve chunking and tracing processes that are general in the sense that they apply without regard to how the curve elements are defined.

A *simple curve* is a curve whose elements are boundary pixels in a pointwise representation of the scene's boundaries. An *abstract curve* is a curve whose elements are local configurations of pixels in some pointwise representation of the scene (see Figure 5.1). In both cases, the neighborhood relation between elements involves proximity and perhaps direction. Chunking and tracing processes are demonstrated in this chapter on examples consisting of simple curves—i.e., thin curves in square binary arrays. The extraction of the elements of abstract curves may itself be viewed as an application of chunking, but this problem is beyond the scope of this report.

A general way of representing curves or sets of curves is by using a graph. Nodes of the graph correspond to curve elements. Edges correspond to instances of the neighborhood

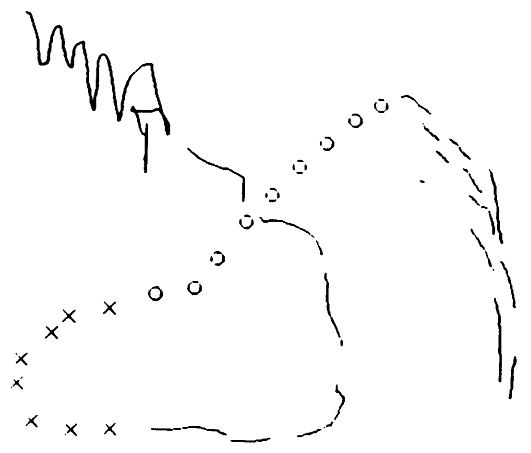


Figure 5.1: An abstract curve.

relation between curve elements.

Curve activation

Curve activation is a labeling operation: a single element of a curve in the input is specified to begin with; the goal is to label every element of the curve containing the given element. Such a curve labeling operation faces two main problems. One problem is to distinguish the set of elements constituting the relevant curve from the elements of irrelevant curves, on the basis of the single curve location that was given. The other problem is to execute the labeling of this set of locations.

Let C be a variable that can be associated with a single curve element. A single element s is given to start, which is initially assigned to C. Each curve element has an associated two-valued label which may take on the values *seen/unseen*, say. The basic form of an element-at-a-time curve labeling process is as follows:

While C has a neighbor n labeled *unseen*:

1. label n *seen*;

2. assign n to C;

3. repeat.

(At the start of such a process, there may be more than one possibility for which neighbor to label next. The above procedure will trace in only one direction. It is useful here to think about the structure of curve tracing processes in the abstract, ignoring this control issue. Note that there is at most a two-fold speedup arising from a direct application of parallelism to this basic tracing process.)

The problem of labeling a curve is inherently serial. To guarantee a correct result in the general case without the use of sequential operations would require prohibitively costly hardware. The stepwise progress along a curve leads to the term "tracing" for curve labeling operations. Evidently, the time-complexity of tracing is a function of the number of elements comprising the curve operated upon.

If curves are represented by a graph, then the operations on this representation useful for tracing are (i) labeling a node, and (ii) indexing a neighbor of a node, i.e., accessing the node that shares a given edge with the given node. For labeling, it is useful to assume that each node of a graph representing curves has an associated two-valued label which may take on the values *seen/unseen.*

Curve location ordering

There is a natural ordering on the elements of a curve, and this ordering can be useful in establishing certain shape properties and spatial relations (for example, see Figure 5.54). The stepwise progress of tracing along a curve can often be used to advantage as a way of establishing ordering. Therefore, the operation of curve tracing is applicable as a solution to two distinct problems, labeling and ordering.

Curve chunking

To reduce the number of iterations incurred by the tracing operation, the tracing process must simultaneously label a chain of curve elements – termed a *curve segment*—at a step, rather than a single element. Any operation for simultaneously labeling a curve segment must be hard-wired. There are so many possible segments, however, that the cost of dedicated labeling hardware for each possibility would be prohibitive. This implies that the parallel labeling machinery must be shared across different curve segments. Perhaps the most straightforward way of achieving this sharing is to use machinery for labeling *an entire region* to label any curve elements falling within the region. A parallel operation for labeling a region may be used to label any curve segment that occupies the region provided nothing else occupies the region. Hence, the need for an essentially parallel labeling operation leads to a definition of a curve chunk as a region containing exactly one curve segment.

More precisely, if the curve elements in a given region are viewed as nodes of a graph, and edges represent the neighborhood relation between curve elements, then *a curve chunk is a region such that (i) the set of curve elements in the region is a chain (a connected graph of maximum degree 2) and (ii) no curve element in the region has more than two neighbors, where neighbors outside the region are counted.* The second condition is introduced because when a curve *junction* coincides with the border of the region, as in Figure 5.2 (b) and (c), it would be incorrect to call the set of curve elements in the region a curve segment.

A curve chunking process must accomplish two things. First, it must construct a decomposition of the input into regions each containing one curve segment. The process of finding such a decomposition may be thought of in terms of two components. One component generates some of the possible tessellations of the input. The other component evaluates each region of a tessellation with regard to whether it meets the curve chunk criterion. (In practice, these aspects of the curve chunking process do not necessarily constitute distinct modules or stages. Nonetheless, the conceptual distinction between them is useful for characterizing curve chunking methods.) The other task of a curve chunking process is to establish, for each adjacent pair of regions in the decomposition, whether there is curve continuity between the two regions. This information is what allows chunkwise tracing to

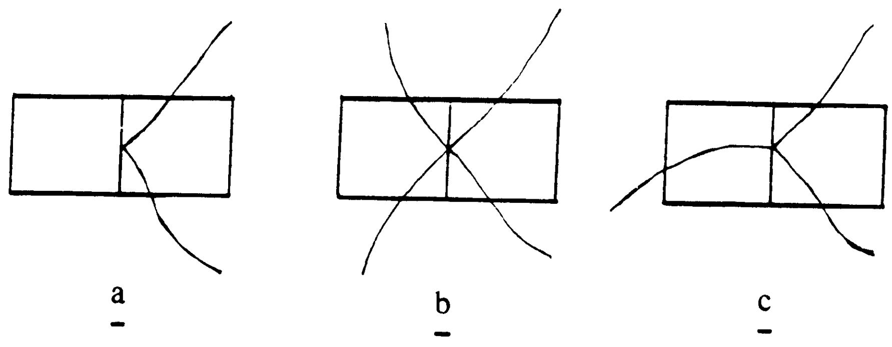


Figure 5.2: A single chain in a region does not necessarily constitute a curve segment, for a junction may coincide with the region border. The right-hand region defines a curve chunk in (a) but not in (b) or (c).

select neighboring chunks to label.

Curve continuity between curve chunks

There is curve continuity between two adjacent regions $r1$ and $r2$ if there exist some curve elements $e1$ in $r1$ and $e2$ in $r2$ such that $e1$ and $e2$ are neighbors. If this condition is met, the common boundary between $r1$ and $r2$ is said to be *crossed.* [1]

Curve chunk representation and chunk-at-a-time tracing

A graph may be used to represent curves at the chunk level in the following way. A node in the graph corresponds to a region in the input. Each node has an associated two-valued label which may take on the values *valid/invalid.* Throughout the discussion of curve chunking, the following two definitions will be useful. A region is said to be ***vacant*** if it contains no curve elements, and ***valid*** if (i) it is vacant or (ii) it contains exactly one curve

[1] It is possible to exclude the second condition from the definition of a curve chunk. This is a less natural definition however, and furthermore it makes it necessary to explicitly check for junctions when establishing curve continuity between chunks. Legal curve continuity between regions must be defined as follows: (i) each curve element $e1$ in $r1$ having a neighbor in $r2$ has exactly one neighbor in $r2$; (ii) each curve element $e2$ in $r2$ having a neighbor in $r1$ has exactly one neighbor in $r1$.

segment (and no elements with more than two neighbors). A curve chunk is a non-vacant valid region. An edge in the graph represents curve continuity between regions.

This representation of curves preserves the basic form of the tracing process. Let C be a variable that can be associated with a single curve chunk. A single valid node s is given to start, which is initially assigned to C.[2] Each curve chunk has an associated two-valued label which may take on the values *seen/unseen*, say. The basic form of a chunk-at-a-time curve labeling process is as follows:

While C has a valid neighbor n labeled *unseen*:

1. label n and all the *curve elements* in the corresponding region *seen* (this is a parallel operation);
2. assign n to C;
3. repeat.

This process stops when an invalid region is encountered. An extension is required to enable the tracing process to label the correct curve segment in an invalid region (i.e., a region containing more than one curve segment). One possibility is to switch to curve-element-at-a-time tracing when an invalid region is encountered, and then switch back to chunk-at-a-time tracing once a curve element is reached which is in a valid region. This "two-level" tracing may be supported in the curve representation by "splicing" each portion of the elementwise graph that corresponds to an invalid region into the chunkwise graph at the appropriate place. This approach involves a substantial slowdown in tracing at invalid regions.

Another possibility for coping with invalid regions is to *skip* them. That is, rather than attempt to *trace* through an invalid region (i.e., label the relevant curve segment in it), the

[2]In this report, I do not discuss the processes that determine the starting chunk s. Generally, some independent process, such as indexing, will shift the processing focus to a location at or near an element of the relevant curve. A mechanism is required for accessing the chunk node s corresponding to the region containing the processing focus.

available information about the local direction of the curve is used to select a neighboring valid region with which to continue tracing. If invalid regions are always small then it is likely that the curve will not change direction in traversing the region. The labeling accomplished by this method may be incomplete—there will be a short unlabeled segment at the site of each invalid region the curve passes through. This method does not necessarily involve a slowdown in tracing at invalid regions, but it is more prone to error and may not always be applicable.

"Two-level tracing" and "skipping" are not incompatible alternatives. The choice between them may be viewed as a tracing time control decision which trades off assurance and speed. In evaluating the various curve chunking schemes, I will distinguish between the time-performance connotations of the chunking method itself and those of the method used to traverse invalid regions.

Parallel region labeling processes

The process for labeling a region may be hierarchical or direct. A direct labeling process requires each node to have a direct connection to every location in the corresponding region of the input (Figure 5.3 (a)). This implies a maximum communication degree of N^2 for a square input array of width N pixels. (Communication degree is defined to be the number of direct communication lines coming into a processor.) A node simultaneously labels every associated input location. The runtime for tracing given a direct labeling operation is equal to the number of iterations incurred by the basic tracing procedure.

A hierarchical region labeling process uses a tree structure to connect a node to corresponding input locations (Figure 5.3 (b)). In the hierarchical case, the label is propagated from a node down through successive intermediate nodes until it reaches all corresponding input locations. Let $l(n1, n2)$ denote the length of the path between the two nodes $n1$, $n2$. If s denotes the start node of tracing, the number of steps it takes for the tracing operation to reach a node n is $l(n, s)$. Let $h(n)$ denote the number of levels of indirection between n and the locations of the region n corresponds to. The time to label the curve elements in

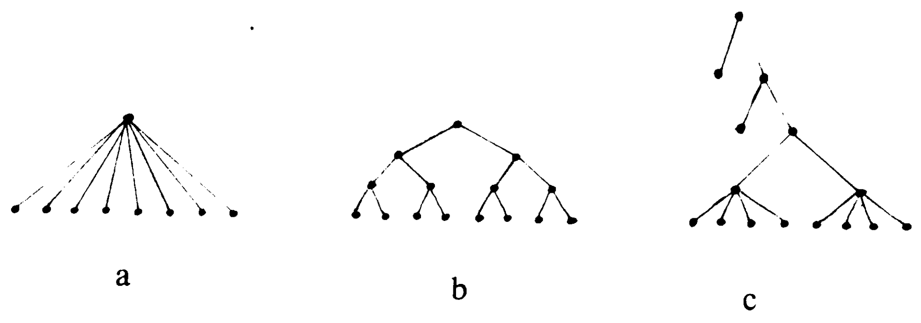


Figure 5.3: Parallel region labeling. (a) Direct labeling. (b) Hierarchical labeling. (c) Hybrid labeling.

the region corresponding to n is $T_{node}(n) = l(n, s) + h(n)$ steps. Let $\mathcal{N}$ denote the set of all nodes reached in tracing a given curve c. The runtime for tracing c, $T_{curve}(c)$ is given by the following expression: $T_{curve}(c) = \max_{n \in \mathcal{N}} T_{node}(n)$.

It is possible to reach a cost/performance compromise between the direct and hierarchical schemes. One simple hybrid region labeling approach is to have nodes corresponding to large regions propagate the label down to nodes representing sufficiently small *elementary regions*, which then label input locations directly (Figure 5.3 (c)). The maximum communication degree is d^2, where d is the diameter of the elementary regions. The runtime for tracing is calculated just as in the pure hierarchical scheme. The three unlimited-extent curve chunking schemes presented in this chapter are evaluated on the basis of this hybrid approach and, for comparison, the ideal, but practically unrealizable, direct approach.

Curve chunking performance considerations

The effectiveness of a decomposition into curve chunks is measured in terms of the number of steps incurred in tracing, which is related to the size of the curve chunks. In general, larger curve chunks imply fewer tracing steps. The number of potential input configurations is very large, as is the number of possible tessellations of the input. Comparatively, the set of tessellations that a chunking mechanism can generate is quite small, because parallel labeling hardware must be committed to each region of each tessellation that can be generated. Therefore, no curve chunking mechanism can generate the optimal set of curve chunks for all inputs. (In fact, it would be a highly unlikely event for a particular mechanism to give the optimal curve chunking of *any* independently given input.) As such, a reasonable performance goal for curve chunking processes is the production of good, not optimal, decompositions over a relatively large class of potential inputs.

The main factor limiting the extent of curve chunks is proximity between curve segments. Any parameter of a configuration of input curves, such as high curvature or curve intersections, that connotes proximity between curve segments influences the sizes of curve chunks into which the input can be decomposed. Position and extent of an isolated input curve may also affect the number of curve chunks, depending on the set of tessellations that a particular chunking process can generate.

5.2 Limited-extent curve chunking

Perhaps the simplest possible scheme for defining curve chunks is one involving (i) the use of the most straightforward techniques, such as conventional pixel-level tracing, for establishing region validity, and (ii) a predetermined tessellation of the input array into regions of limited size. The limitation on the size of the regions is an unavoidable consequence of the use of very simple validity computations. The specific reasons for the size restrictions depend on the particular validity computation used. In addition, given a predetermined tessellation, the larger the regions, the more of them are likely to be invalid.

The techniques of limited-extent curve chunking are an important component of the more powerful, unlimited-extent schemes presented in subsequent sections of this chapter. The divide-and-conquer curve chunking methods begin with a limited-extent decomposition. Direct, unlimited-extent curve chunking is made hardware-efficient by an initial limited-extent decomposition. (Note, also, that the limited-extent approach was assumed, in Chapter 3, to provide the "basic segments" of boundaries used for computing local prominence information.)

Simple, regular tessellations are appropriate for this application because they are easy to implement. The problem of curve chunking does not, however, seem to place any restrictions on the shape of the regions of the tessellation pattern—e.g., rectangular vs. hexagonal. The specific choice may be governed mainly by the available hardware. Also, there is no need for the regions to be *exactly* the same size or shape.

The next two subsections develop a serial method and a parallel method for determining validity of small regions. The serial method works by distinguishing and counting each of the curve segments in the region by pixel-level tracing. The parallel method involves detecting two local features, certain combinations of which indicate the presence of two curve segments. Subsequently, the limited-extent chunking algorithm and an account of its performance are presented.

Figure 5.4 shows an example decomposition by the limited-extent scheme. The scheme was simulated on the Lisp Machine. The input was a square binary array of width 512 pixels.

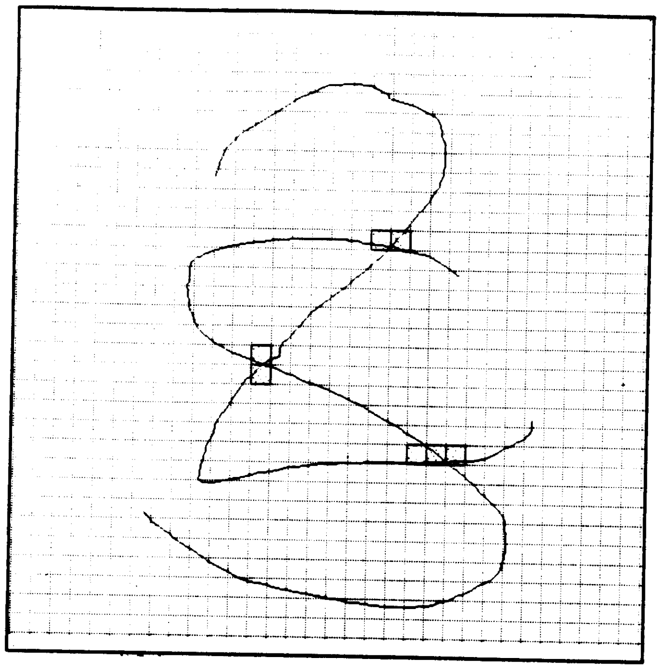


Figure 5.4: A decomposition of an input by limited-extent curve chunking —invalid regions are shaded.

The input curves were entered by hand using a tracking device. The input was decomposed into square regions of width 16 pixels.

5.2.1 Serial, limited-extent validity checking

The curve segments in a region may be counted by tracing within the region at the pixel level to make them distinct. (This imposes a limit on the region size because the time for establishing validity grows roughly with the diameter of the regions.) The tracing process is defined to follow a particular curve segment, starting from a given element, and marking the curve elements until the region is exited or until a previously marked element is reached again. When the current element has two or more unmarked neighbors, the tracing process must apply rules for selecting the new neighbor. If regions are small, then normally the direction of the curve upon entry into the region will be essentially preserved through the region. For example, a curve entering a small square region through the top or bottom edge is likely to traverse it vertically. At the same time, spurious changes in direction may occur at the pixel level. Occasionally, a curve may bend sharply in the region.

A predefined *directional priority scheme (DPS)* for selecting a neighbor can impart a specified overall directionality while allowing for temporary (spurious) or permanent direction changes. A DPS is a list of coordinate offsets. Each pair specifies the positional offsets of an immediately neighboring curve element (pixel) in a particular direction. The tracing process moves at each step from the current element to the neighboring curve element whose coordinate offsets occur earliest in the list. The priority schemes associated with the ***up**, down, left* and *right* directions (and thus the bottom, top, right and left region borders, respectively) are shown in Table 5.1. An algorithm for tracing and counting the curve segments in a small rectangular region (of a rectangular array) is given below. The top, bottom, left, and right *edges* of a region are defined in the obvious way. The count is initially 0. The algorithm makes use of two labels.

For each edge b of the region r, while there is an unmarked curve element e in b having no more than two neighbors, do the following:

1. Shift the starting location for tracing to e.
2. Trace using the DPS associated with b, and a label that is unused so far. (The tracing process terminates not at the region border but one or more pixels beyond it.)
3. Increment the count by 1. Exit if the count exceeds 1.
4. Repeat.

Tracing slightly beyond the region border makes it possible to distinguish the following two situations: (i) the intersection of two curves coincides with a border of the region; (ii) a sharp direction change of a curve coincides with a border of the region. With tracing beyond the border, if a crossing coincides with the border, tracing will proceed a bit through the crossing and stop (and the count of the curve segments in the region will ultimately be 2); if a corner coincides with the border, tracing will simply proceed around the corner. The second condition in the definition of a curve chunk—no elements in the region with more than two neighbors—is thereby implemented.

	1	2	3	4
up	(0 -1)	(1 -1) (-1 -1)	(1 0) (-1 0)	(-1 1) (0 1) (1 1)
down	(0 1)	(1 1) (-1 1)	(1 0) (-1 0)	(-1 -1) (0 -1) (1 -1)
left	(1 0)	(1 -1) (1 1)	(0 -1) (0 1)	(-1 0) (-1 -1) (-1 1)
right	(-1 0)	(-1 -1) (-1 1)	(0 1) (0 -1)	(1 0) (1 -1) (1 1)

Table 5.1: Directional priority schemes for a rectangular tessellation. Order among pairs of coordinate offsets in the same column does not matter. Y-coordinates increase in the down direction.

The runtime of the tracing process depends on the length of the curve segment in the region. Many local tracing processes will operate concurrently, and they will terminate at different times. The runtime of the concurrent tracing process is the maximum of the execution times of all the local processes, i.e., the maximum length of any curve segment in one of the local regions. This value is not known initially, but an upper bound is given by the region size—in a square region of width d locations, the maximum possible curve segment length is about $d^2/2$. This maximum is unlikely, however; it is reasonable to impose a smaller limit proportional to d (e.g., $3d$) on tracing time, and to treat regions in which the local tracing process fails to terminate in that time as *invalid.*

5.2.2 Parallel, limited-extent validity checking

Validity may be established in parallel in terms of two local geometric features—curve terminations in the region and curve exits of the region border. (This imposes a limit on the region size because the cost of distinct hardware units for establishing validity of each region grows with the area of the regions, at least.)

The *border* of a region r is defined to be the set of locations in r that have fewer than eight neighbors in r. An *exit* of a region r is a connected set s of curve elements in the border of a region adjacent to r, such that some curve element e in s has a neighbor in r. A *termination* is a curve element (i) included in the region, and (ii) having no more than one neighboring curve element in the region. The termination count T and the exit count E can be used to establish that a region contains a single curve segment (and no curve

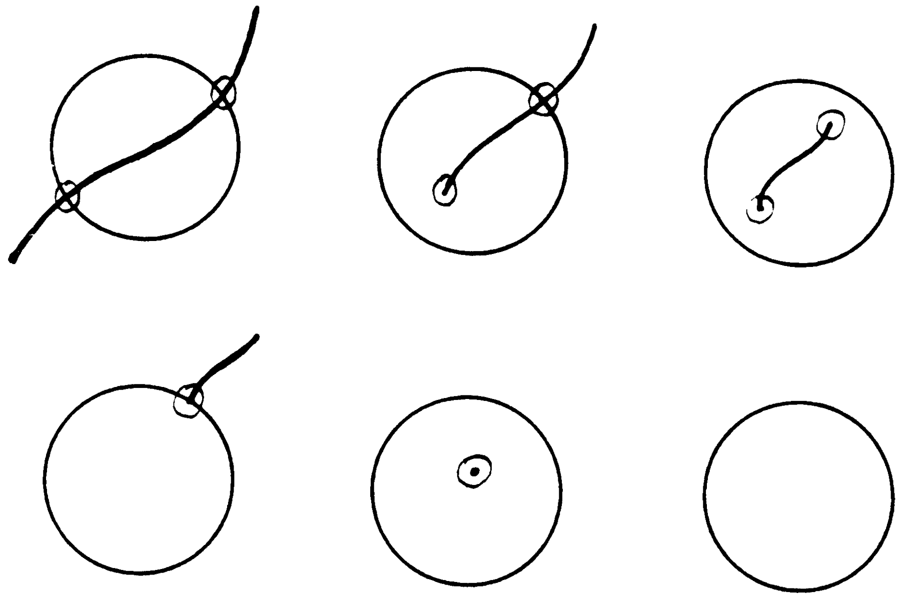


Figure 5.5: Parallel validity check based on termination count and exit count—valid cases.

elements with more than two neighbors), assuming that the region contains no closed curve loops.

A region containing no closed loops is valid if $T + E \leq 2$. (See Figure 5.5.)

Figure 5.6 shows some examples of disallowed combinations. It is necessary to explicitly disallow loops because T and E cannot be used to distinguish between the cases in Figure 5.5 (a) and Figure 5.7, for example. For small regions, this "no loops" assumption is very likely to be correct; when it is violated, tiny loops very near the relevant curve may be incorrectly labeled, but for most applications this is likely to be harmless.

The parallel validity check can be implemented by simple, local processes. Terminations and exits are detectable by simple template matching. The simplest perceptron-like devices may be used to detect the valid combinations of T and E [Minsky and Papert 69]. A general counting mechanism is not required; it is only necessary to implement predicates of the form "There are exactly 2 terminations in the region," etc. The outputs of these predicates can be combined by simple logical operations.

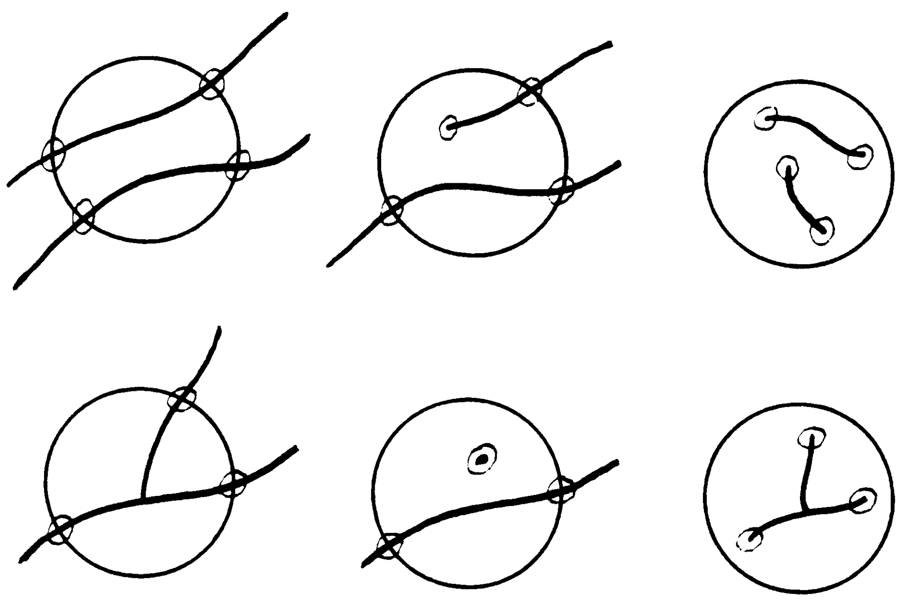


Figure 5.6: Some invalid feature combinations.

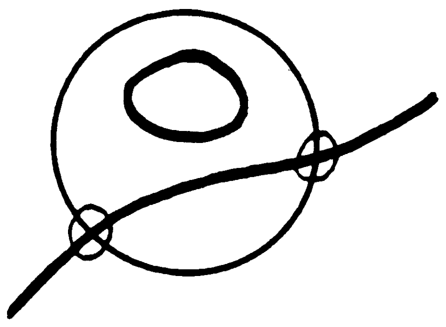


Figure 5.7: An invalid region for which $T + E \leq 2$.

5.2.3 A limited-extent curve chunking algorithm

The output representation of limited-extent curve chunking is a graph. The graph consists of an array of nodes labeled *valid* or *invalid*—one node for each non-vacant region—and a set of edges between nodes indicating legal curve continuity between adjacent regions. The limited-extent chunking algorithm is a single-stage parallel process: the region-validity computation is simultaneously applied to each region and the *valid/invalid* label is assigned the appropriate value. At the same time, the curve continuity computation may be applied simultaneously to all pairs of adjacent regions, and edges are created between corresponding pairs of nodes accordingly. (For each region in the tessellation, three hardware processing mechanisms are required, one for the validity computation, one for the curve continuity computation, and one for the tracing-time region-labeling operation.) The runtime of the algorithm is constant. If the serial validity check is used, kd steps are required, where d is the region width and k is some small integer determined by the maximum expected length of a curve segment in a region. The parallel validity check involves a small constant number of steps that depends on the circuitry used to detect valid combinations of features.

Dynamic choice of region size

The discussion so far has assumed a single predetermined region size. Suppose invalid regions are traversed by tracing temporarily at the pixel level. Tracing time for a given curve is the sum of the number of valid regions traversed and the number of curve elements in invalid regions traversed. The number of invalid regions is a function of the density of input curves in relation to the region size. It is therefore possible to use the data to select a region size that will optimize tracing time—smaller regions imply that curves in the input will traverse *more* regions, but fewer of these may be invalid. One possibility is to use the largest size for which the number of invalid regions is close to the minimum. (The minimum number of invalid regions is obtained for the smallest available region size—for sufficiently small regions, the set of invalid regions coincides with the set of curve crossings.) The motivation for this is that the minimum number of invalid regions is achieved (or almost achieved) as soon as region size becomes small enough that no two extended curves (or parts

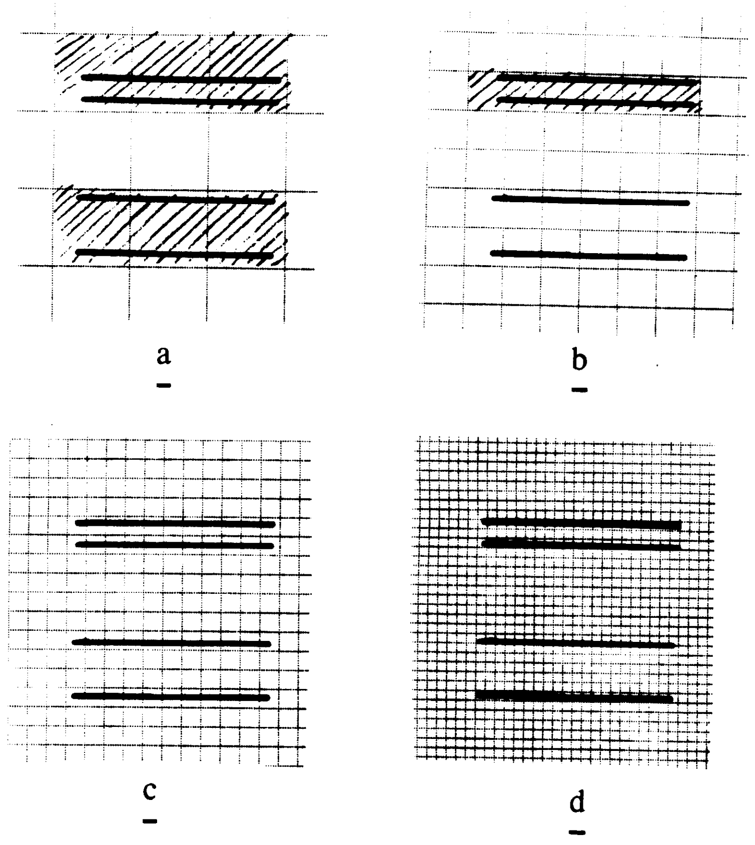


Figure 5.8: Dynamic choice of region size for limited-extent curve chunking. (a) d = 64. (b) d = 32. (c) d = 16. (d) d = 8. Shaded regions are invalid. The chosen size is 16.

of curves) occupy the same set of regions. Figure 5.8 illustrates this idea. Tessellations of an input are shown for a series of region sizes, and invalid regions are shaded. The selected region size is the one corresponding to the last big drop in the number of invalid regions as size is decreased.

5.2.4 Performance

The runtime of tracing using limited-extent curve chunks depends on two factors. The primary factor is the sizes of the curve chunks into which the relevant curve is decomposed (i.e., the valid regions containing segments of the curve). The secondary factor is the overhead involved in traversing invalid regions. In the limited-extent scheme, the chunk size is fixed and uniform over the input, so consideration of the primary factor alone would lead to a simple relationship between curve length and tracing time. The influence of invalid regions cannot be ignored, however, and this makes it difficult to analyze performance in precise terms, for two reasons. First, the number of invalid regions depends on the geometry of the input curves, i.e., incidence of curve crossings and points of curve proximity. A precise performance analysis would require a detailed, statistically justified characterization of the geometry of some universe of inputs. This would be difficult to provide even for a specific domain of application; it is beyond the scope of this thesis to attempt to provide a general characterization of input geometry, if there is such a thing. (This comment also applies to the rest of the curve chunking schemes developed in this chapter—for them, the geometry of the input curves is the primary performance factor.)

The second difficulty is that there are at least two possible strategies for traversing invalid regions (see Section 5.1), each with different time-performance, and the choice between them may depend on the circumstances—different strategies might be used even in the course of tracing a single curve. In the following analysis I present the extremes of performance that these alternative strategies lead to.

Let the region diameter used in the limited-extent scheme be d pixels. The best case arises when no region traversed by the relevant curve is invalid—i.e., it never traverses the same region more than once, and never traverses a region that another curve traverses. In this case, limited-extent curve chunking provides a roughly d-fold speedup over pixel-at-a-time tracing. This relationship is approximate, for one thing, because tracing time is insensitive to possible variations in the length in pixels of a curve segment in a region. Also, there is a small position dependence analogous to quantization effect, as illustrated in Figure 5.9. This effect depends on the shape of the regions in the tessellation; a rectangular tessellation

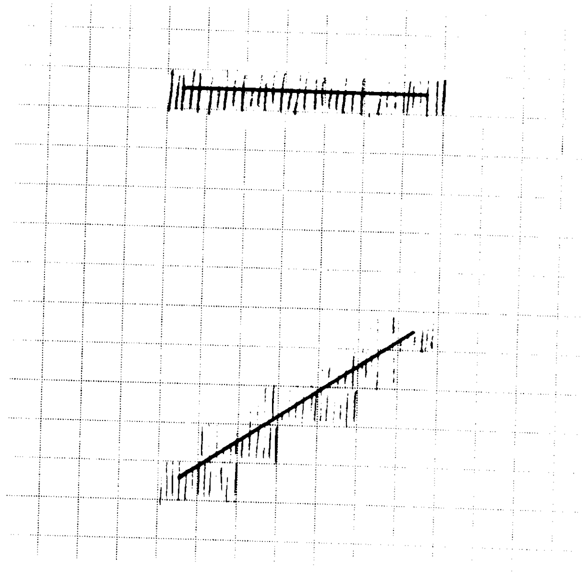


Figure 5.9: Position dependence in limited-extent curve chunking.

is in this respect inferior to a hexagonal one.

The worst case arises when every region traversed by the relevant curve is invalid—in this case no valid curve chunks are available at all, and the entire curve must be traced at the pixel level. Figure 5.10 shows two situations which give rise to the worst case: (i) two parallel curves are separated by a distance of less than d; (ii) a curve is repeatedly crossed at intervals along its length of less than d.

If v and i denote the number of valid and invalid regions, respectively, traversed by a curve c, then the time to trace c is given by $T \approx v + di$.[3] For typical inputs, i is normally small compared to v, for any given curve, so tracing time is still much improved over pixel-level tracing. For example, the "2" in Figure 5.4 is 817 pixels long. 65 of the regions it traverses are valid and 7 are invalid. 81 of its pixels are in invalid regions. Therefore, the time to trace this curve is 72 steps at best (i.e., invalid regions are skipped, using local curve direction information) and $65 + 81 = 146$ steps at worst (invalid regions traversed by pixel-at-a-time

[3]If tracing proceeds concurrently in two directions along the curve, tracing time can vary by a factor of up to 2, depending upon where tracing begins.

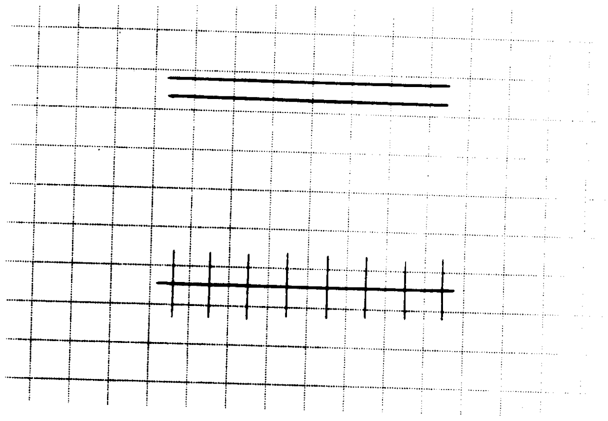


Figure 5.10: Examples of the worst case for limited-extent curve chunking— no curve chunks are defined for any of the horizontal curves.

tracing).

It is useful to note the effect of region size here. Given 4-pixel-wide regions (Figure 5.11), the "2" traverses 256 valid regions and 6 invalid ones. 16 of its pixels are in invalid regions. In this case, the time to trace this curve is 262 steps at best and $262 + 16 = 278$ steps at worst.

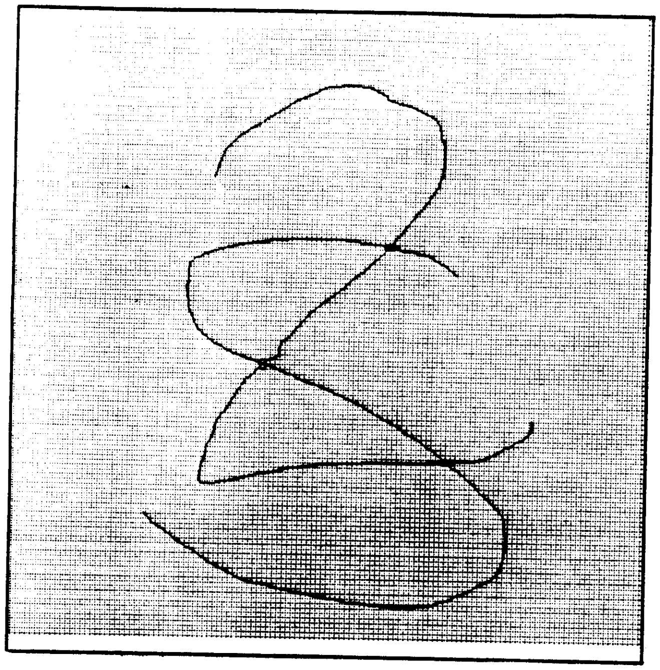


Figure 5.11: A decomposition of an input into 4-pixel-wide elementary regions—invalid regions are outlined by solid lines.

5.3 Direct, unlimited-extent curve chunking

This section and the following two develop schemes that generate curve chunks with no *a priori* limit on extent. The capacity for defining curve chunks of unlimited extent is gained by introducing somewhat more elaborate computations than the simple ones that were used to establish region validity in the case of limited-extent curve chunking. Those simple methods do not scale up directly to regions of arbitrary size. Pixel-at-a-time tracing becomes too slow—that is, after all, why curve chunking is being explored. The "no loops" assumption made by the parallel validity computation becomes unrealistic.

The methods for establishing validity of large regions may be classified according to whether or not they involve a series of intermediate computations. The method introduced in this section does not, and it is therefore referred to as a *direct* method. The methods of the next two sections are indirect—they are based on the technique of divide-and-conquer.

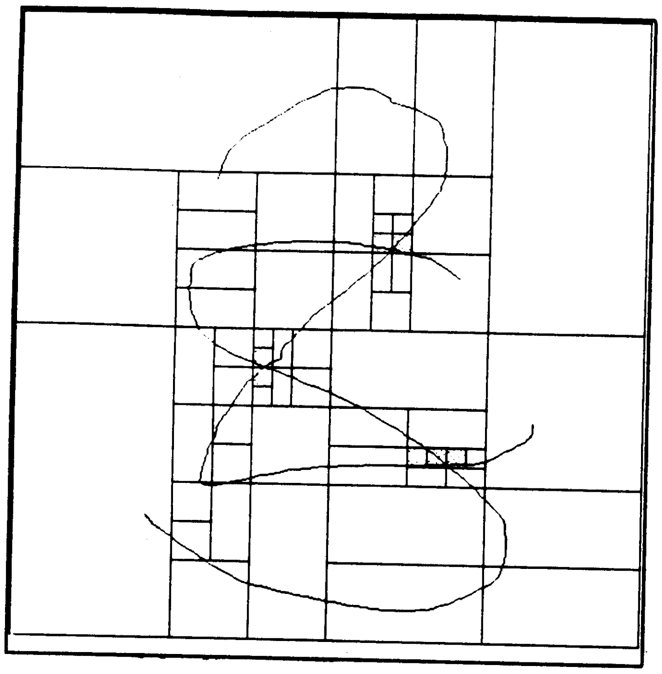


Figure 5.12: Decomposition of an input by direct, unlimited-extent curve chunking.

The direct, parallel, unlimited-extent validity check developed in this section is based on a simple extension of the parallel validity check developed for the limited-extent scheme. Restrictions on local curvatures and the range of local curve orientations are introduced to exclude regions containing closed loops.

There are three main considerations in the design of a direct, unlimited-extent curve chunking scheme: (i) the computations for ascertaining region validity; (ii) the size, shape, and arrangement of the regions; and (iii) the algorithm of the chunking process.

Figure 5.12 shows an example decomposition by the direct, unlimited-extent scheme, for an input that was previously used to demonstrate limited-extent curve chunking.

5.3.1 A parallel, unlimited-extent validity check

This section extends the feature-based parallel validity check introduced in Section 5.2.2 to regions of arbitrary size. That check may be applied to regions of any size, but the problem with it is that closed loops in the region are "invisible" to it, because they do not give

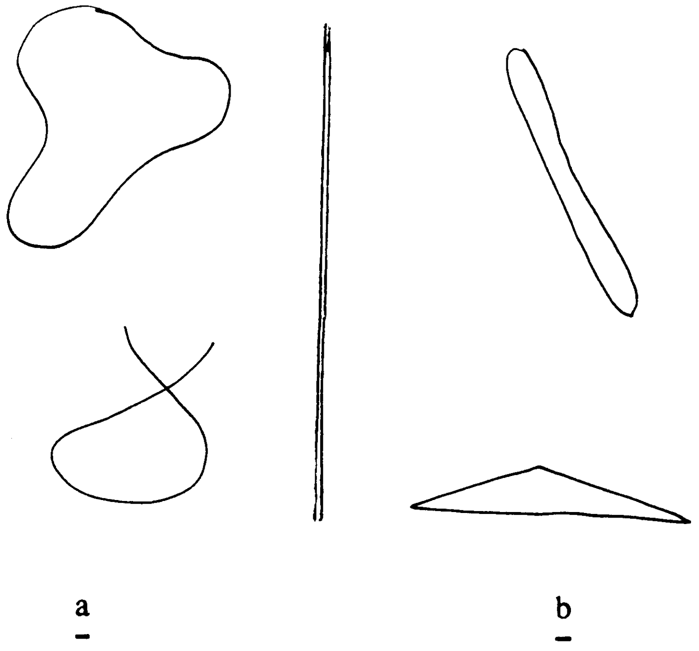


Figure 5.13: (a) Closed loops with no points of very high or discontinuous curvature have local orientations spanning the range 0 to 180. (b) Closed loops with curvature discontinuities need not span the entire possible range of orientations.

rise to terminations or exits. Because it is not safe to assume that a large region contains no closed loops, it is necessary to introduce additional computations that will explicitly determine whether or not the region contains a closed loop. The computations introduced are required to be direct.[4]

The possible presence of a closed loop in a region can be established in a very simple, direct, parallel way, based on the set of local curvatures and orientations in the region. Suppose local curve orientation is measured between 0° and 180°. If a loop has sufficiently low curvature at every location, then the set of local orientations must span the entire possible range (Figure 5.13 (a)). (Note that the presence of orientations across the entire possible range implies only the *possibility* of a closed loop.) A loop need not span the entire range, however, if it includes points of very high or discontinuous curvature (Figure 5.13 (b)).

[4][Edelman 85] used an *indirect* method—divide-and-conquer—to exclude closed loops in a related application. See Chapter 7.

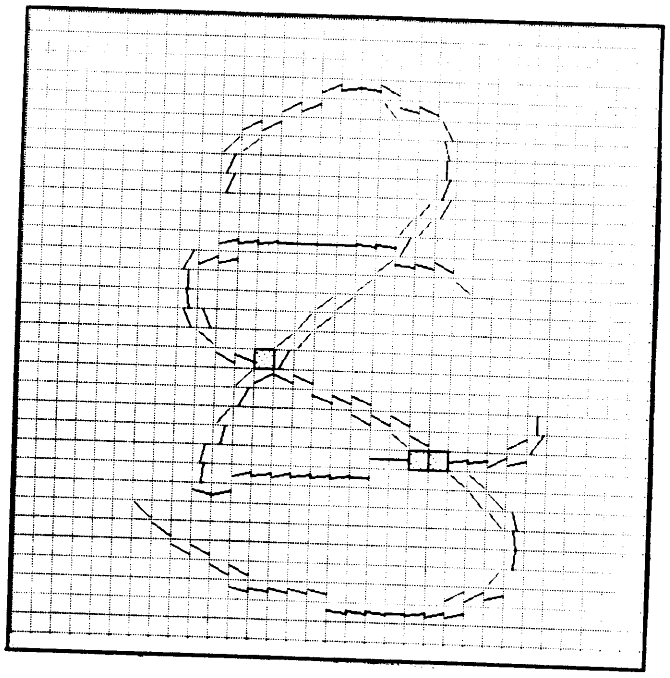


Figure 5.14: Local curve orientations defined by the curve segments in 16-pixel-wide elementary regions. Regions in which an orientation could not be reliably defined are shaded.

Computing local curve orientation and curvature

There are a variety of possibilities for computing local orientation and curvature, but the particulars of how these measures are computed are not crucial to this discussion of direct, unlimited-extent curve chunking. Perhaps the most problematic issue that arises in defining local curve orientations and curvatures is the choice of the appropriate scale at which to compute these measures. This issue is not addressed here. Many edge/line detectors can provide edge orientation information. In the simulation, local curve segments were defined by dividing the input into small regions, and the orientation of the axis of least inertia of the set of pixels in each region was used as the local curve orientation measure (see [Horn 86], [Winston and Horn 84]). A region diameter of 16 pixels was chosen, somewhat arbitrarily. Figure 5.14 shows the local curve orientations computed for an example input. The local curve segments that gave rise to these orientations are those in Figure 5.4, because limited-extent curve chunking was applied to the same example with the same region size.

The validity check

The *orientation difference* $d(\theta_1, \theta_2)$, $0 \le \theta_1, \theta_2 < 180$, is defined as follows:

$$d(\theta_1, \theta_2) = \min(\theta_h - \theta_l, (\theta_l + 180) - \theta_h),$$

where $\theta_h = \max(\theta_1, \theta_2)$ and $\theta_l = \min(\theta_1, \theta_2)$.

Let the set of local curve orientations defined in a given region R be denoted by $\Theta(R)$. The *orientation spread* $S(\Theta(R))$ is defined by the following expression:

$$S(\Theta(R)) = \max_{\theta_1, \theta_2 \in \Theta(R)} d(\theta_1, \theta_2)$$

In principle, for any region R containing a closed loop $S(\Theta(R)) \ge 90°$. In practice, because of error in the computation of local curve orientations, it is necessary to limit the orientation spread further. Recall the definitions of T, the termination count, and E, the exit count, introduced in Section 5.2 for the limited-extent parallel validity check.

A region R is valid if (i) it includes no points of very high or discontinuous curvature; (ii) $S(\Theta(R)) < (90 - \epsilon)°$; and (iii) $T + E \le 2$.

Orientation spread and curvature thresholds

The value of ϵ used in the simulation was 25°; this value was found empirically to be safe in the context of the orientation computations that were used.

There are a variety of possible curvature measures, and the curvature threshold depends on the particular measure used. The choice of the curvature threshold has not been explored. (The curvature restriction was not implemented; the simulation is not guaranteed to give correct results for inputs including points of high or discontinuous curvature.)

(It is worth pointing out here that the orientation spread and curvature restrictions were not used to exclude loops in the limited-extent case because they are not suitable for very small regions. Within a small region, local curve orientation may only be defined at the pixel-level, that is, by the directions between neighboring pixels. Pixel neighborhood provides a very small total range of eight local orientations. The spurious local variation in curve direction at the pixel level is large in comparison to this total range. Therefore no restrictions on orientation spread and curvature can both exclude loops and permit curve segments in a small region.)

Reducing combinatorics

A truly direct implementation of this scheme would have maximum communication degree of N^2, for a square input array of width N pixels, where N is on the order of 10^2, say. By adding a single level of indirection to the computation for regions above a certain size, its hardware cost can be brought into a reasonable range. By dividing the input into small *elementary regions* of diameter d pixels, and computing validity of larger regions on the basis of these, the maximum communication degree can be reduced to $O((N/d)^2)$. The $O(N^2)$ communication requirement comes from the computation of T. Adding indirection to the computation of T is straightforward: the termination count of a region R is the sum of the termination counts for the elementary regions R includes. T is computed for elementary regions directly.

It is slightly more involved to add indirection to the exit count. The extension is not strictly necessary—the communication requirement for computing E is only $O(N)$—but it is worth sketching here. The *elementary edges* of a region R are edges of R's elementary regions that are included in R's edges. The exit count of a region R is the number of exits that occur at its elementary edges.

5.3.2 Regions

Given a validity check for large regions, it becomes necessary to decide upon the range of sizes, shapes, and positions of the regions over which it will be applied. Intuitively, tracing time can be optimized by detecting curve chunks at the maximum possible number of sizes, shapes, and positions, subject to a restriction on total cost. The direct curve chunking scheme could be applied in a manner very similar to the one Shafrir [Shafrir 85] developed for fast region coloring (see Chapter 7). He used a direct, parallel computation to detect empty regions at many positions and sizes, subject to cost restrictions. At each coloring step, all uncolored regions overlapping a previously colored region may be colored.

In this report, the direct curve chunking approach is demonstrated in the context of an economical, pyramid scheme for decomposing the input into subregions known as the binary image tree. This is done so that it can be compared to the upcoming divide-and-conquer methods, which use the same scheme. The detailed description of the pyramid is given in Section 5.4.1.

5.3.3 A direct, unlimited-extent curve chunking algorithm

The output representation is a graph. The graph consists of an array of nodes labeled *valid* or *invalid*—one node for each non-vacant region—and a set of edges between nodes indicating legal curve continuity between adjacent regions. The unlimited-extent curve chunking algorithm is a single-stage parallel process: the region-validity computation is applied simultaneously to each region and the *valid/invalid* label is set appropriately. At the same time, the curve continuity check is applied to pairs of adjacent regions and links are appropriately established between nodes representing these regions. The runtime of the algorithm is a small constant that depends on the circuitry used to measure orientation spread and detect valid combinations of features.

5.3.4 Performance

The runtime of tracing using the unlimited-extent curve chunks provided by the direct scheme depends on the sizes of the curve chunks into which the relevant curve is decomposed. Chunk size, in turn, depends on two factors. The primary factor is the geometry of the input curves – total curvature, local curvature, and incidence of curve crossings and points of curve proximity. (Taken by itself, curve length does *not* affect chunk size, so tracing time is scale independent.) The influence of curve geometry makes a precise performance analysis difficult, for the reasons previously stated in the performance discussion for limited-extent chunking. The secondary factor is the scheme by which the regions to be checked for validity are defined (e.g., binary image tree, quadtree, overlapping or non-overlapping regions, etc.). Each such scheme has different performance ramifications, and there is a wide, essentially unexplored range of possibilities. For that reason, the following paragraphs attempt to characterize the *general* aspects of the behavior of direct, unlimited-extent curve chunking, i.e., the aspects that do not depend on the particular scheme used to define regions.

The best case occurs when the relevant curve (i) does not cross itself or any other curve; (ii) includes no points of very high or discontinuous curvature; (iii) does not have local orientations spanning the range S_{loop}. If a region R including the entire curve is checked for validity, the result is a single curve chunk. "Tracing" this curve involves one parallel region-labeling operation. In the hybrid model of region labeling (Section 5.1), this requires about $\log D/d$ steps, where D and d are the diameters of R and the elementary regions of the hybrid labeling process, respectively. (In the direct model of region labeling, only a single step is required.)

The worst case arises when of all regions checked, none larger than a pixel and traversed by the relevant curve prove valid. In this case, direct, unlimited-extent chunking provides no improvement over pixel-level tracing. The examples of worst case inputs presented above in the performance discussion for limited-extent curve chunking also apply to unlimited-extent curve chunking.

Using the binary image tree decomposition, *for typical inputs the number of steps required for tracing a curve is normally on the order of 10.* (This does not depend much on the size of the image.) Figure 5.15 shows the result of tracing one of the curves in the example of Figure 5.12. In the left frame, the small circle indicates the starting point of tracing, and shaded regions are curve chunks that were labeled. The right frame shows the curve extracted by this labeling process. The figure caption gives performance metrics. T_H and T_D denote the number of steps required to trace the portion of the curve falling in shaded regions, counted using the hybrid and direct region-labeling models, respectively.[5] P denotes the number of pixels in the traced curve. I denotes the number of pixels of the traced curve in invalid regions.

In the hybrid labeling scheme, the time to trace a curve is T_H steps at best (i.e., invalid regions are skipped, using local curve direction information) and $T_H + I$ steps at worst (invalid regions are traversed by pixel-at-a-time tracing). I, the additional time for tracing the curve pixel-by-pixel through invalid regions, depends on the width of the smallest regions checked for validity. As shown in Section 5.4.4, when the region width goes from 16 to 4 pixels, I drops from 81 to 16 for the "2" in this example, and T_H increases by about 10. Therefore, for 16-pixel-wide elementary regions, tracing the "2" involves 114 steps; for 4-pixel-wide elementary regions, tracing the "2" involves about 60 steps.[6]

As pointed out above, the binary image tree decomposition is not the most effective possibility for defining regions for direct, unlimited-extent curve chunking—equal or better performance can be achieved by any scheme that defines regions at an equivalent or larger range of positions and sizes.

Curve chunk size is affected by the limit on orientation spread S_{loop}. Figure 5.16 shows the decompositions of a region containing a circle for several values of S_{loop}.

[5]Where $\mathcal{N}$ is the set of nodes reached in tracing, s is the starting node, and $l(n, s)$ is the path length between n and s, $T_D = \max_{n \in \mathcal{N}} l(n, s)$. Suppose a hybrid labeling scheme is used in which square elementary regions of width d pixels can be directly labeled, but larger regions must be labeled indirectly. Let $D(n)$ denote the maximum of the horizontal and vertical dimensions, in pixels, of the region corresponding to a node n. The number of steps it takes to label a region defined by the binary image tree is given by $h(n) = 2 \log D(n)/d$, and $T_H = \max_{n \in \mathcal{N}} l(n, s) + h(n)$.

[6]The direct validity check can be applied to regions of any size—even down to a single pixel. By making the minimum region a pixel, the added term I is eliminated. T_H also increases, however, so tracing time would not be much different from that for a minimum region width of 4 pixels.

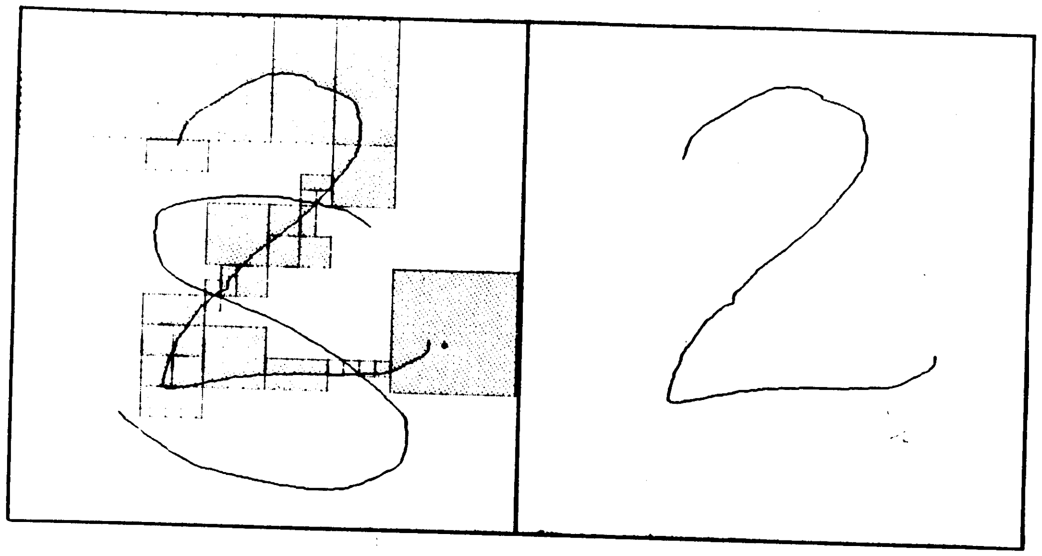


Figure 5.15: Tracing the "2" by unlimited-extent curve chunks given by the direct scheme: $P = 817$; $T_H = 33$; $T_D = 28$; $I = 81$.

5.4 Divide-and-conquer, unlimited-extent curve chunking

The strategy of *divide-and-conquer* provides an effective alternative way of establishing validity of large regions without explicit restrictions on local curvature or orientation spread. In the divide-and-conquer approach, large valid regions are assembled by combination of smaller regions that have previously been shown to be valid, subject to a simple rule. The main components of a divide-and-conquer curve chunking scheme are (i) an initial limited-extent decomposition into curve chunks, providing the basis for the divide-and-conquer process; (ii) a pyramid representation; (iii) rules for combining valid regions into valid regions; (iv) an algorithm for building up the decomposition by applying these rules.

Figure 5.17 shows an example divide-and-conquer decomposition for a now-familiar input.

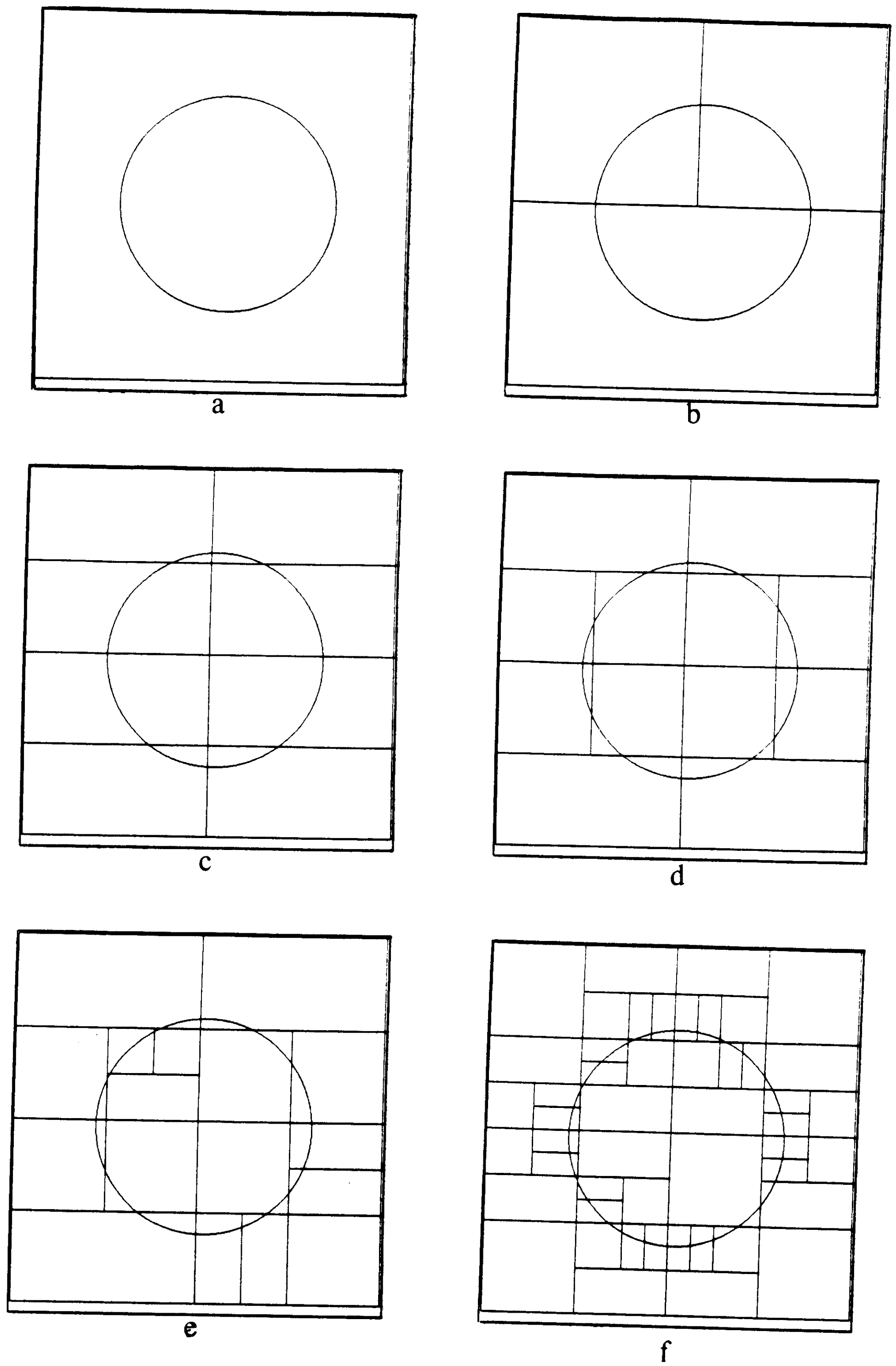


Figure 5.16: Decompositions of a region containing a circle for several values of S_{loop}: (a) 90°; (b) 89°; (c) 60°; (d) 45°; (e) 30°; (f) 15°.

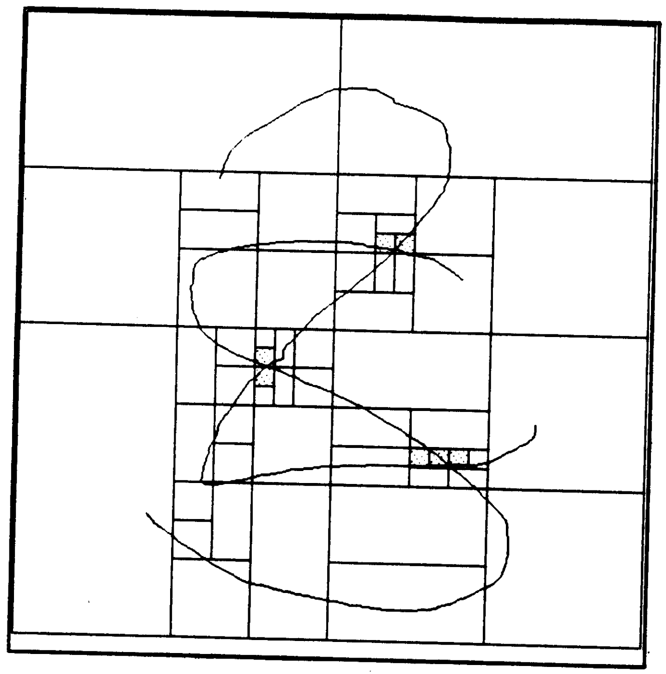


Figure 5.17: Decomposition of an input by divide-and-conquer curve chunking.

5.4.1 The pyramid representation

A pyramid representation is used to construct the tessellation. Each node in the pyramid represents a region of the image. Associated with each node is a label which can take on the values *valid/invalid*. The size of the region represented by a node is determined by the node's level in the pyramid. The position of the region represented by a node is determined by the node's position in the array of nodes that are at the same level. The array of nodes at a particular level represents a regular tessellation of the input at some scale. A node is linked to a number of child nodes in the level immediately below it. The children of a node in the pyramid represent the subregions of the region that the parent node represents. The base-level nodes represent the regions - termed *elementary regions*—in the initial decomposition. The pyramid as a whole can represent a large number of *irregular* tessellations of the input. (A subset of the pyramid's nodes constitutes a description of a particular tessellation if the union of the regions these nodes represent is the entire input array.)

Some of the more natural possibilities for defining the subregions of a region in a rectangular array include the following (Figure 5.18): (i) two subregions per region and one common boundary (binary image tree); (ii) four subregions per region and four common boundaries (quadtree); (iii) nine subregions per region and twelve common boundaries (9-tree). (See [Pavlidis 82] for discussions of the binary image tree and the quadtree.) These schemes may be applied with or without overlap between the subregions.

The two-subregion-per-region scheme is adopted here because of two main advantages that it offers. First, it leads to the simplest computations for merging the subregions—curvilinearity relations need be checked at only one boundary, which means a savings in hardware in parallel implementation, a savings in time in serial implementation. Perhaps more importantly, a failure to merge in a two-subregion scheme is less costly than it is in any of the others—it leads to one additional chunk, not three or eight. The two-subregion scheme does have two disadvantages, however. First, regions of two different shapes are required (squares and rectangles), which makes design and implementation somewhat more involved than it would be for a quadtree. Also, more merging steps are required in the assembly of regions of a given size than in any of the other schemes.

Most of the demonstrations of divide-and-conquer curve chunking presented here are the output of an implementation based on non-overlaping regions. In practice, the use of overlaping regions requires somewhat more hardware, but it may be much easier to construct because it does not require precise alignment of region boundaries. Figure 5.19 illustrates a pyramid built on the two-subregion decomposition without overlap.

A binary image tree representing a square array of width N pixels has $h = 2\log N$ levels, 2^h nodes, and 2^h links between nodes. In order to represent curve continuity, additional direct links are required, to connect nodes in the pyramid that correspond to adjacent regions. (Associated with each of these links is a two-valued label, say *continuity/no-continuity.*) The rectangular region a node n at level l of the tree represents is adjacent on each side to up to 2^l other regions represented by nodes in the tree, so n may require up to 2^{l+2} continuity links. This requirement leads to an $O(N)$ upper bound on maximum communication degree (MCD); specifically, the upper bound is $2^{(h-1)+2} = 2N$ ($h - 1$ is

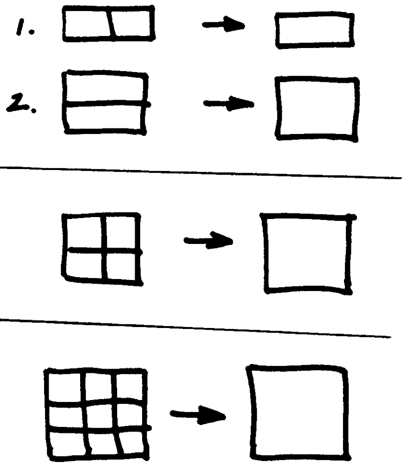


Figure 5.18: Some possible ways of defining the subregions of a region in a rectangular array (without overlap).

used instead of h because the region represented by the top-level node is the entire image and does not require continuity links). Thus for $N = 512$, the MCD is at most 1024—a costly but not unrealistic communication requirement.

The communication requirement can be reduced considerably, as follows. The binary image tree need not represent the image all the way down to the pixel level. If fringe nodes of the tree represent square *elementary image regions* of width d pixels, then the MCD is bounded by $2N/d$. For example, for $N = 512$ and $d = 16$, the MCD is at most 64, and for $d = 4$ it is at most 256. The use of elementary regions provides a reduction in hardware cost but involves a certain time performance sacrifice, as the performance results presented below show. Therefore, a cost-performance compromise must be made.

5.4.2 The rule for merging regions

The union of two adjacent valid regions is valid if (i) either region is vacant; or (ii) the common boundary is crossed (see section 5.2). (See Figure 5.20.)

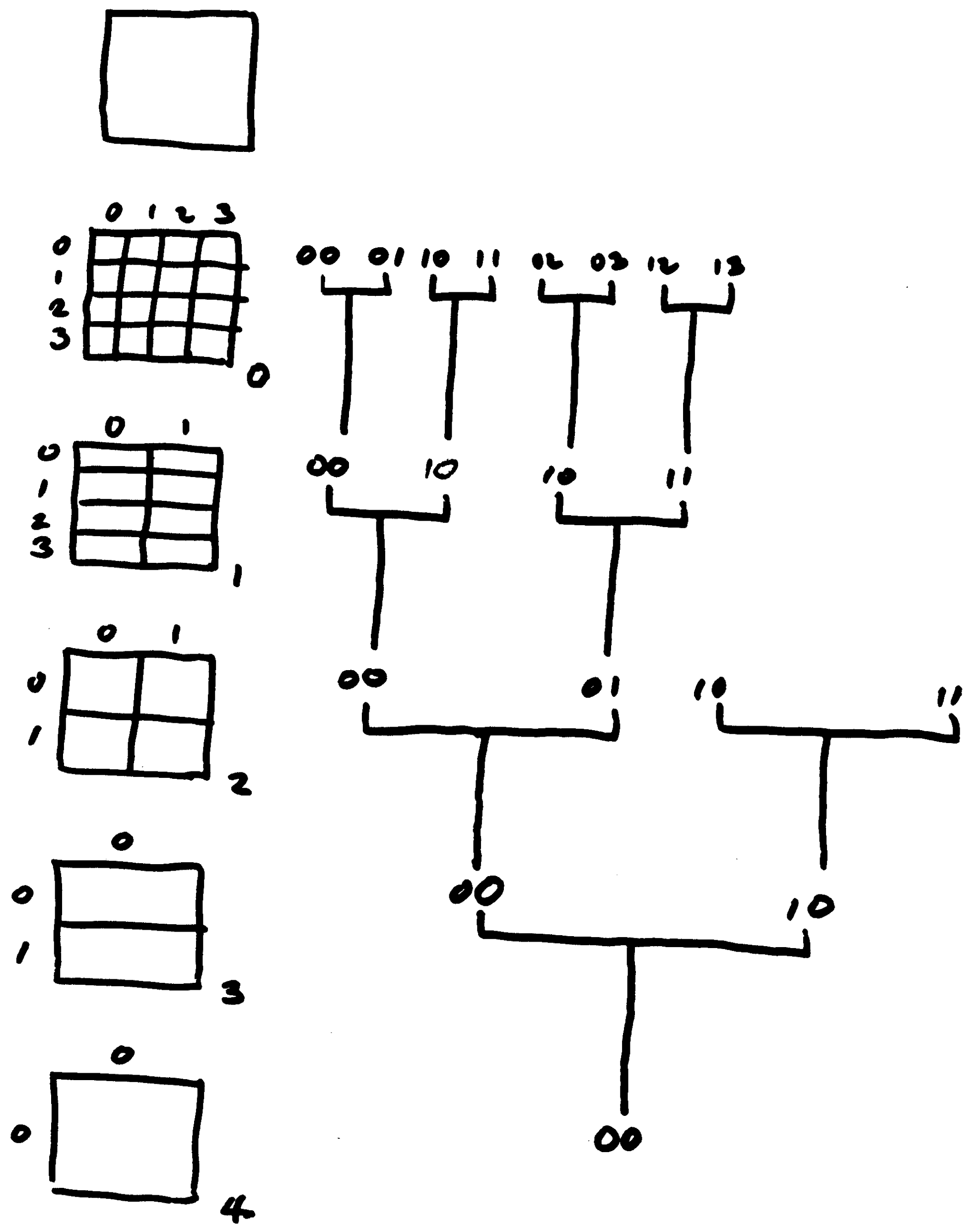


Figure 5.19: A binary image tree, starting from a four-by-four array of elementary regions.

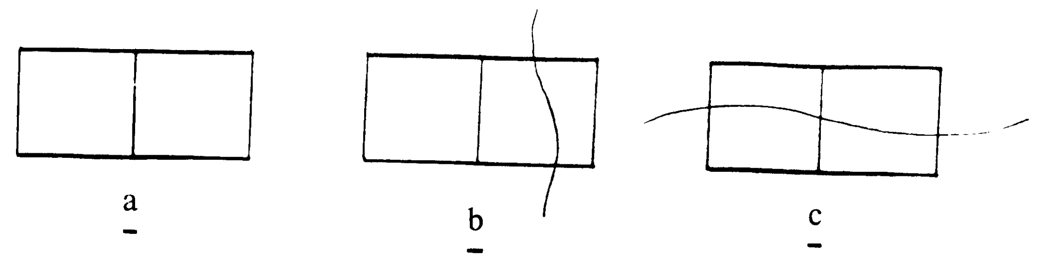


Figure 5.20: Illustration of the rule for merging adjacent regions: (a) vacant regions are valid; (b) a vacant region and a valid, non-vacant region form a valid region; (c) two valid, non-vacant regions form a valid region if their common boundary is crossed.

Proof:

Condition (i): Obvious.

Condition (ii): Let $R1$, $R2$ denote two adjacent valid regions. A curve segment may be viewed as a chain, i.e., a connected graph of maximum degree 2, whose nodes are curve elements and whose edges are instances of the neighborhood relation between curve elements. By definition, that a region is valid implies that the set of curve elements in the region either is the empty set or constitutes a chain, and no element in the region has more than two neighbors. That the boundary between $R1$ and $R2$ is crossed implies that both regions contain one or more curve elements, and that for some curve elements $e1$ in $R1$ and $e2$ in $R2$, $e1$ and $e2$ are neighbors. Therefore, the union of the sets of curve elements in $R1$ and $R2$ constitutes a single chain, so the union of the two regions is valid.

(The extension of the preceding rule to overlapping regions is straightforward. When two regions overlap, each one includes part of the boundary of the other. Condition (ii) is replaced by a requirement that each section of "overlapped" boundary must be crossed.)

5.4.3 A divide-and-conquer curve chunking algorithm

This section presents an algorithm for divide-and-conquer curve chunking. The output representation of divide-and-conquer chunking is a graph. The graph consists of an array of nodes labeled *valid* or *invalid*—one node for each maximal valid region defined by the chunking process—and a set of edges between nodes indicating curve continuity between adjacent regions. The algorithm has three main components: (i) elementary regions are checked for validity by a direct method; (ii) non-elementary regions are checked for validity by divide-and-conquer; (iii) maximal valid regions are identified; (iv) nodes representing adjacent regions are linked if there is curve continuity between them.

Checking elementary regions for validity

The computations described in Section 5.2 may be used to establish validity of the elementary regions. The validity labels of corresponding nodes at the base of the pyramid are set appropriately.

Building valid regions

Suppose the height of the pyramid is h, where the lowest level—the representation of the initial decomposition—has height 0. The variable l indicates the pyramid level currently being operated upon; its initial value is 1. The *valid/invalid* label of each node in the pyramid is initialized to *invalid.* The algorithm for building up valid regions is as follows:

Repeat for l from 1 to h:

1. (In parallel) for each node r at level l both of whose children are valid, apply the merging rule computations to its subregions. Modify the *valid/invalid* label as appropriate.

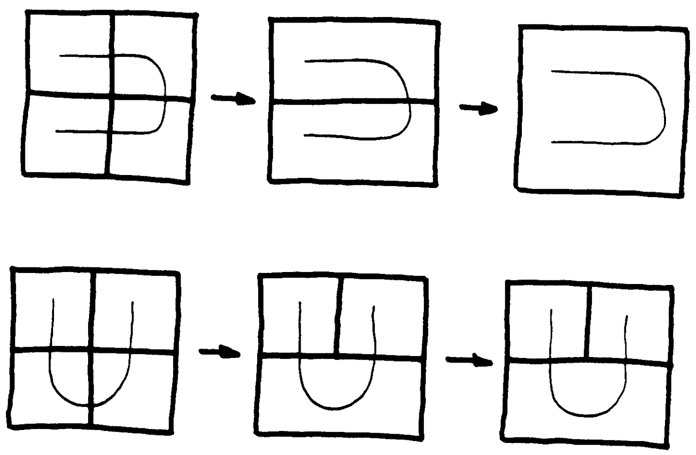


Figure 5.21: Input demonstrating the asymmetric behavior of the binary image tree.

Applied in the context of the binary image tree, the behavior of the above algorithm is asymmetric. In Figure 5.21 (a) it gives rise to one curve chunk but in Figure 5.21 (b) it gives rise to two; one figure is simply a 90° rotation of the other. To correct this asymmetry, a *symmetric binary image tree* is used instead (Figure 5.22). This necessitates two different merging operations. (The new image tree is described for the non-overlaping case only; the required extension for the case of overlaping regions is very similar.) Each square region may be defined either by a pair of horizontal regions or by a pair of vertical ones. Also, each square region may define half of a horizontal rectangular region or half of a vertical one. As before, at odd levels of the pyramid, squares are merged into rectangular regions; at even levels, rectangles are merged into squares. Now twice as many nodes are required at odd levels of the pyramid, half of them representing vertical rectangles, the other half horizontal ones. Each rectangle has a pair of square children; validity is established for every rectangle. At even levels, the number of nodes required remains unchanged; such a node now has two pairs of children, one vertical and one horizontal, and therefore two opportunities to be labeled valid. The merging rule computations are applied independently to the horizontal pair and to the vertical pair of subregions; the common parent node is labeled *valid* if the merging rules were satisfied by *either* pair of subregions.

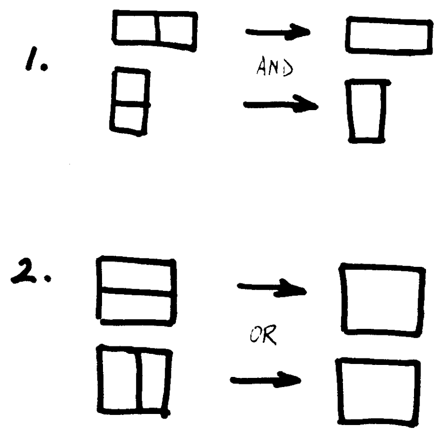


Figure 5.22: Defining the *symmetric* binary image tree.

Identifying maximal valid regions

Assume that each node in the pyramid may take one of two states, active and inactive, and all nodes are initially active. The algorithm for identifying maximal valid regions (MVR's) does so by suppressing non-maximal valid regions. Nodes remaining active after this process are the maximal valid regions.

Repeat for l from h down to 1:

1. (In parallel) for each active valid node r at level l, deactivate every descendant node of r. (A descendant node of r is any node than can be reached by following parent-child links downward through the pyramid.)

Establishing curve continuity between regions

The curve continuity process is not dependent on the preceding two processes. For any given pair of regions, the curve continuity check involves determining whether their common boundary is crossed. The check may be applied in parallel for every adjacent pair of regions represented in the pyramid.

Examples illustrating the divide-and-conquer process

Figures 5.23 and 5.25 illustrate the progress of this algorithm for two small, similar examples. The base of the pyramid is at the top of the diagram – pyramid level increases downward. Dashed and solid lines in the arrays at the left indicate the decomposition so far. A string of the form $l - ji$ will be used to refer to the region in the array at level l whose y-coordinate is j and whose x-coordinate is i. In the first example, an attempt to merge regions 0-10 and 0-11 fails because 0-11 is invalid. In the second example, an attempt to merge regions 0-10 and 0-11 fails because the common border of these two regions is not crossed. Figures 5.24 and 5.26 show the output.

If the serial check is used to establish validity of elementary regions, the runtime of the entire divide-and-conquer process is about $2h + kd$, where d is the elementary region diameter and k is some small constant. If the parallel validity check is used for elementary regions, the runtime is $2h + k$, for some small k.

5.4.4 Performance

The runtime of tracing using the unlimited-extent curve chunks provided by the divide-and-conquer scheme depends on the sizes of the curve chunks into which the relevant curve is decomposed. Chunk size, in turn, depends on two factors. The primary factor is the geometry of the input curves—total curvature, local curvature, and incidence of curve crossings and points of curve proximity. (Taken by itself, curve length does *not* affect chunk size, so tracing time is scale independent.) The influence of curve geometry makes a precise performance analysis difficult, for the reasons previously stated in the performance discussion for limited-extent chunking. The secondary factor is the scheme by which the regions to be checked for validity are defined. Each such scheme has different performance ramifications, and there is a wide, essentially unexplored range of possibilities. For that reason, the following paragraphs attempt to characterize the *general* aspects of the behavior of divide-and-conquer curve chunking, i.e., the aspects that do not depend on the particular scheme used to define regions.

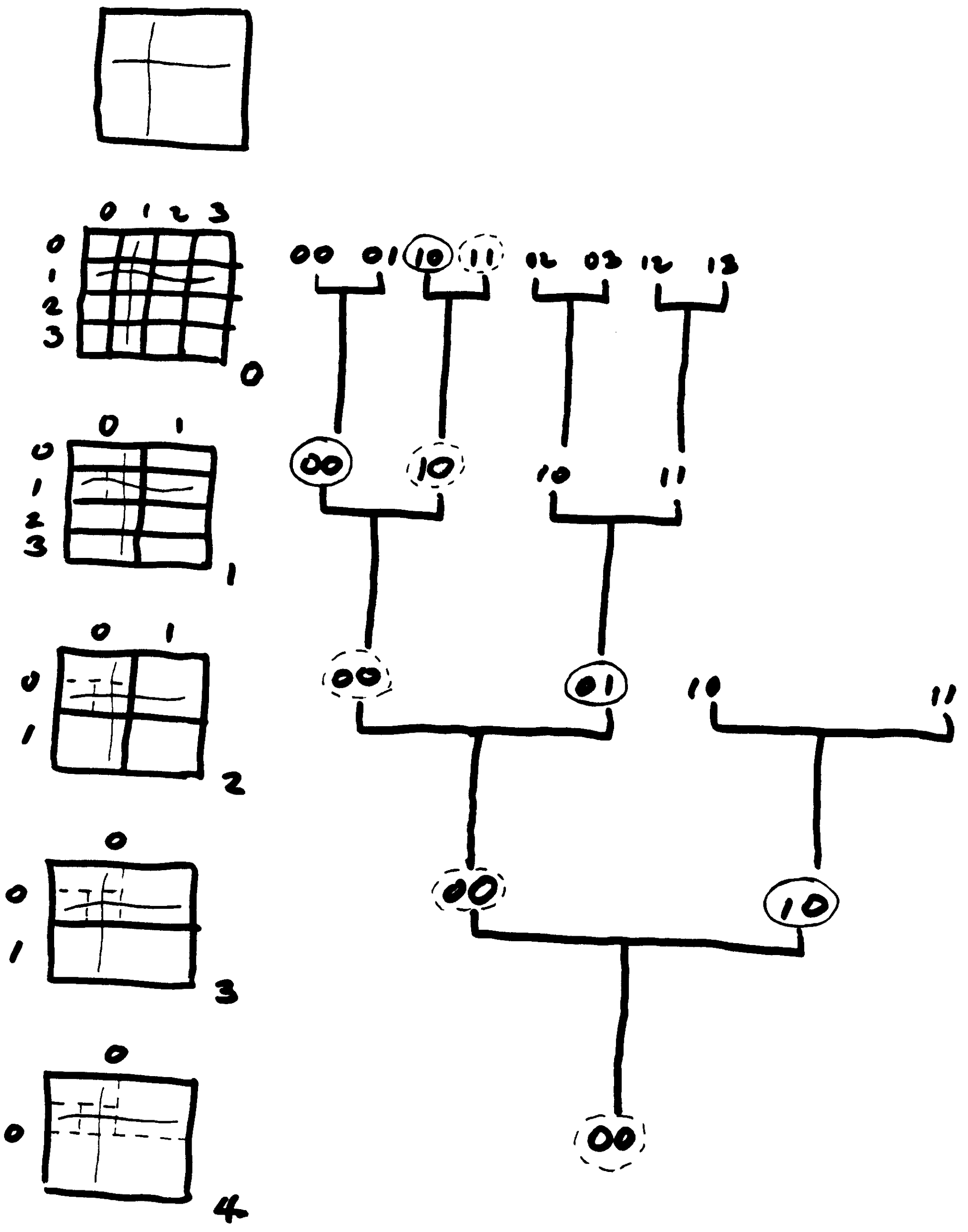


Figure 5.23: Divide-and-conquer curve chunking, Example 1. A solid circle surrounding the coordinate of a node in the tree indicates a maximal valid region. A dotted circle indicates an invalid region.

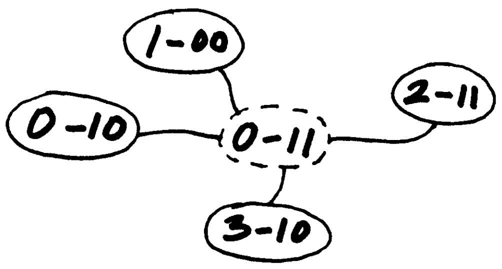


Figure 5.24: The output for Example 1. A solid circle surrounding the coordinate of a node in the graph indicates a maximal valid region. A dotted circle indicates an invalid region.

The best case occurs when (i) no elementary region traversed by the relevant curve is invalid—i.e., it never traverses the same elementary region more than once, and never traverses an elementary region that another curve traverses; and (ii) for some region R that is checked for validity, R includes the relevant curve and nothing else. "Tracing" involves one parallel region-labeling operation applied to R. In the hybrid model of region labeling (Section 5.1), this requires $\log D/d$ steps, where D and d are the diameters of R and the elementary regions of the hybrid labeling process, respectively. [7] (In the direct model of region labeling, only a single step is involved.)

The worst case arises when every elementary region traversed by the relevant curve is invalid—in that case no valid curve chunks are available at all, and the entire curve must be traced at the pixel level. In this case, divide-and-conquer curve chunking provides no performance improvement. The examples of worst case inputs presented above in the performance discussion for limited-extent curve chunking apply here as well.

Using the binary image tree decomposition, *for typical inputs the number of steps required*

[7]It is reasonable to define the elementary regions of the hybrid labeling process to be the elementary regions of the divide-and-conquer process.

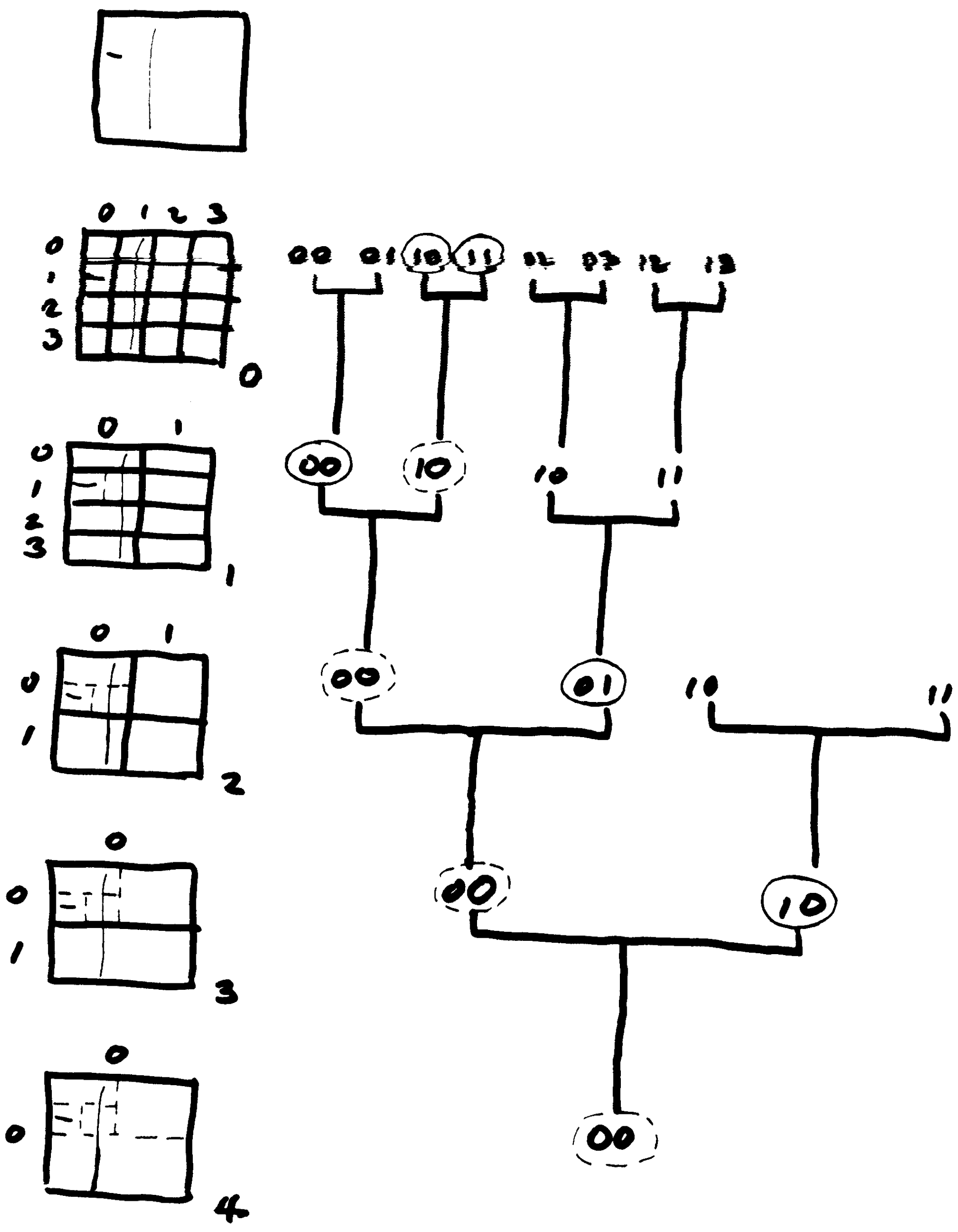


Figure 5.25: Divide-and-conquer curve chunking, Example 2. A solid circle surrounding the coordinate of a node in the tree indicates a maximal valid region. A dotted circle indicates an invalid region.

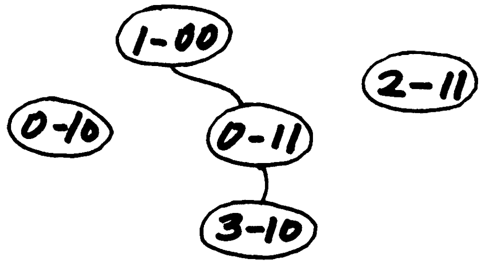


Figure 5.26: The output for Example 2. A solid circle surrounding the coordinate of a node in the tree indicates a maximal valid region. A dotted circle indicates an invalid region.

for tracing a curve is normally on the order of 10. (This does not depend much on the size of the image.) Figure 5.27 shows the result of tracing one of the curves in the example of Figure 5.17. In the left frame, the small circle indicates the starting point of tracing, and shaded regions are curve chunks that were labeled. The right frame shows the curve extracted by this labeling process. The figure caption gives performance metrics (these numbers were introduced in Section 5.3.4.) In the hybrid labeling scheme, the time to trace a curve is T_H steps at best (i.e., invalid regions are skipped, using local curve direction information) and $T_H + I$ steps at worst (invalid regions are traversed by pixel-at-a-time tracing). I, the additional time for tracing the curve pixel-by-pixel through invalid regions, depends on the width of the smallest regions checked for validity. As shown in Figure 5.28, when the region width goes from 16 to 4 pixels, I drops from 81 to 16 for the "2" in this example, and T_H increases by about 10.[8] Therefore, for 16-pixel-wide elementary regions, tracing the "2" involves 113 steps; for 4-pixel-wide elementary regions, tracing the "2" involves 61 steps.

[8]For this example, the direct and divide-and-conquer schemes have very similar behavior, so these comments apply equally to the former.

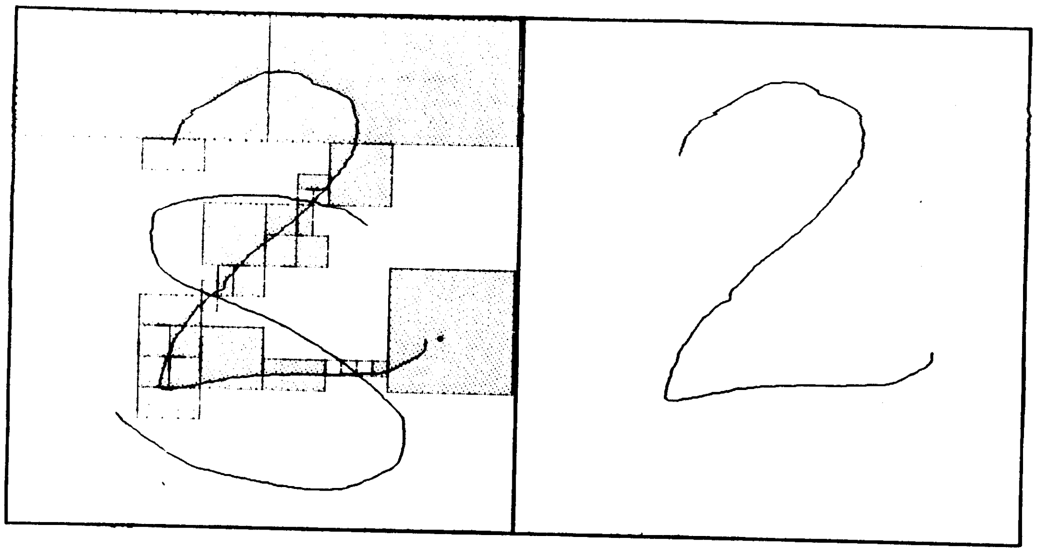


Figure 5.27: Tracing a curve by unlimited-extent curve chunks given by the divide-and-conquer scheme: $P = 817$; $T_H = 32$; $T_D = 27$; $I = 81$.

The divide-and-conquer scheme can be applied to regions of any size – even down to a single pixel. By making the minimum region a pixel, the added term I is eliminated. T_H also increases, however, so tracing time would not be much different from that for a minimum region width of 4 pixels. As pointed out in Section 5.4.1, maximum communication degree is inversely proportional to elementary region width, so a width of 4 pixels is preferable to a width of 1.

The worst case described above is unlikely. A more likely "bad case" for this technique is a situation in which most elementary regions traversed by the relevant curve are valid, but for which most attempts to *merge* regions fail. In this case, divide-and-conquer curve chunking provides little or no speed-up over limited-extent curve chunking. For example, Figure 5.29 shows an example that is best case for limited extent curve chunking and "bad case" for divide-and-conquer (Figure 5.30).

Using the binary image tree decomposition, divide-and-conquer curve chunking is somewhat

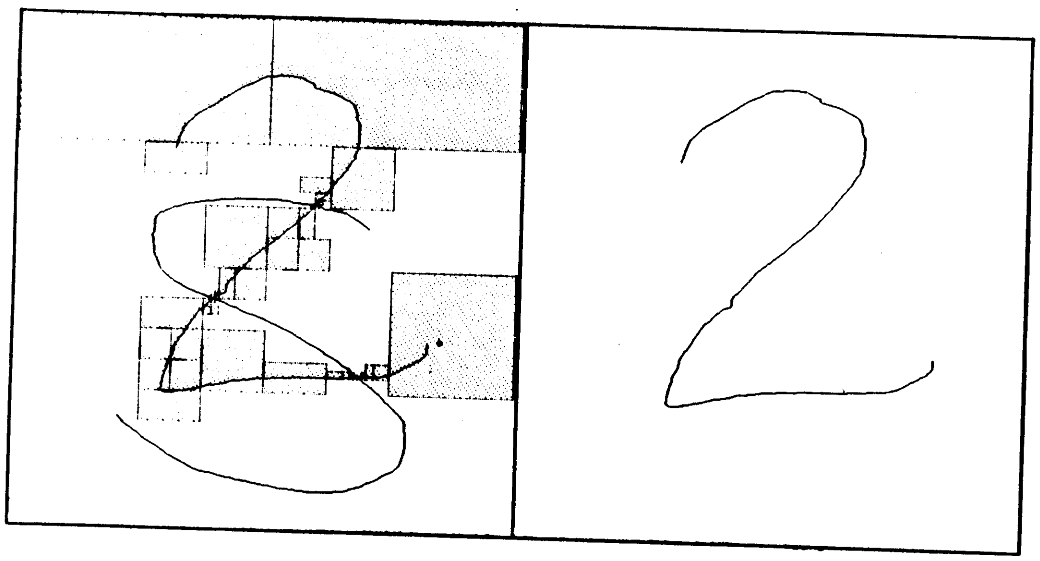


Figure 5.28: Divide-and-conquer curve chunking: effect of elementary region size: $P = 817$; $T_H = 45$; $T_D = 36$; $I = 16$.

position dependent. This is not revealed by a test configuration of a single extended straight line. Test configurations revealing position dependence are shown in Figures 5.31 and 5.32. It is possible to correct for this position dependence by somehow normalizing the input for position, or by applying the divide-and-conquer process independently at a number of different positional offsets and accepting the result of the application that gave rise to the fewest curve chunks. For all but the simplest inputs, however, the effect of position is in fact quite small, and the additional cost of these corrective measures may not be justified. Figure 5.33 shows the decompositions of an input resulting for a number of different (manually chosen) positional offsets. The difference in the number of curve chunks in the best and worst cases is small in comparison to the total number of chunks defined for the curve.

Figure 5.34 illustrates the effect of curvature on curve chunk size. Figures 5.35 and 5.37 show the curve chunks computed from real image intensity boundaries, and the results of tracing the object boundaries. In both cases, a single invocation of the tracing operation

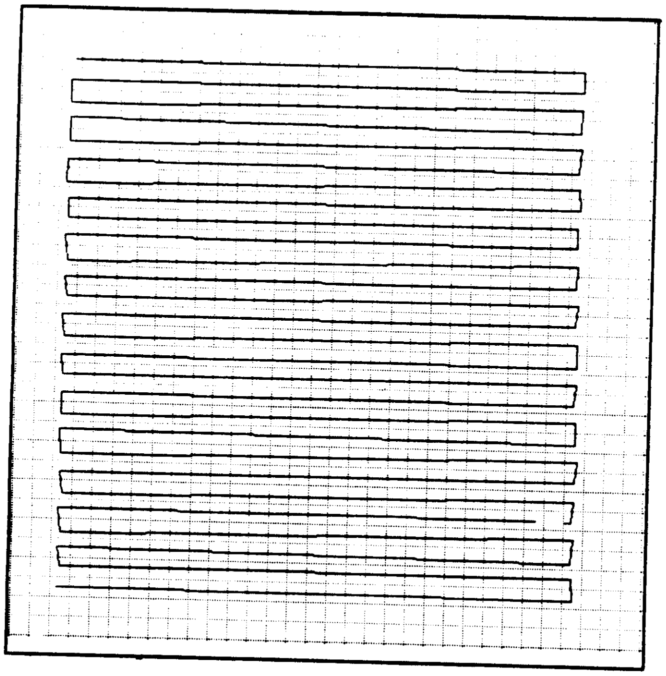


Figure 5.29: A best-case input for limited-extent chunking that is "bad case" for divide-and-conquer.

did not extract the entire object boundary—it was stopped by small gaps. Also in both cases, curve chunk size was limited by the incidence of internal boundaries of the objects. Both of these phenomena indicate that the time-performance of tracing could benefit if, prior to curve chunking, the following processes were applied to the boundary array: (i) a "clean-up" process of gap filling and the suppression of noisy or weak boundaries; (ii) a process for distinguishing occlusion boundaries (or some other type of "relevant" boundary) from irrelevant boundaries.

The divide-and-conquer scheme seems somewhat more powerful than the direct scheme of the preceding section, because it does not involve explicit restrictions on local curvature or orientation spread. For example, Figure 5.39 shows a number of inputs for which the divide-and-conquer scheme gives rise to a single chunk, but for which the direct scheme gives several smaller subregions. On the other hand, though, the direct scheme is less position dependent than the divide-and-conquer method—for example, it results in a single curve chunk for both inputs in Figure 5.31. A detailed comparison of the performance of

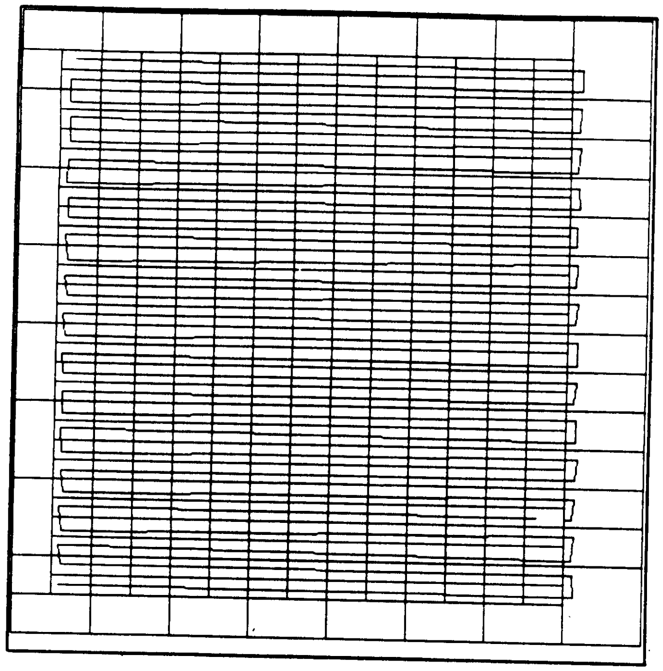


Figure 5.30: Divide-and-conquer provides only a factor of 2 improvement over limited-extent chunking for this "bad-case" example.

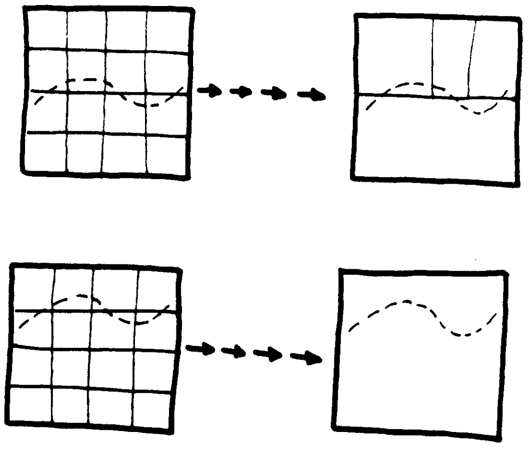


Figure 5.31: Position dependence of divide-and-conquer curve chunking.

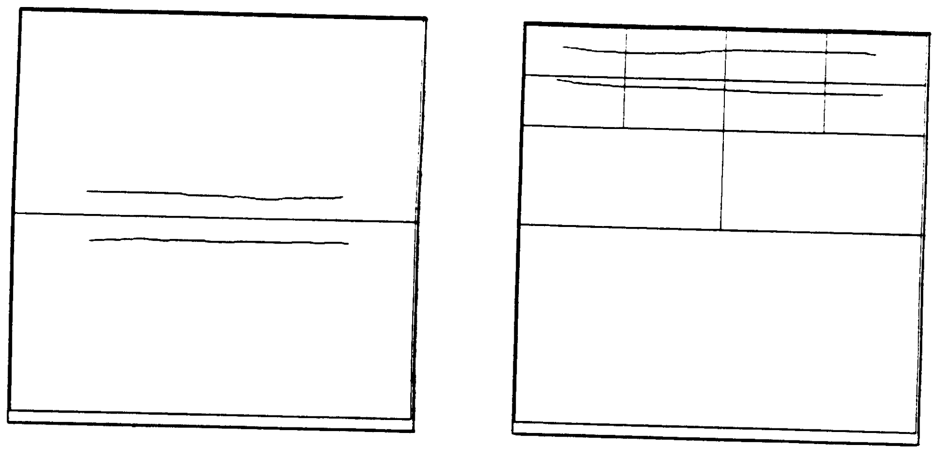


Figure 5.32: Position dependence of divide-and-conquer curve chunking.

the two methods is an interesting problem for further work.

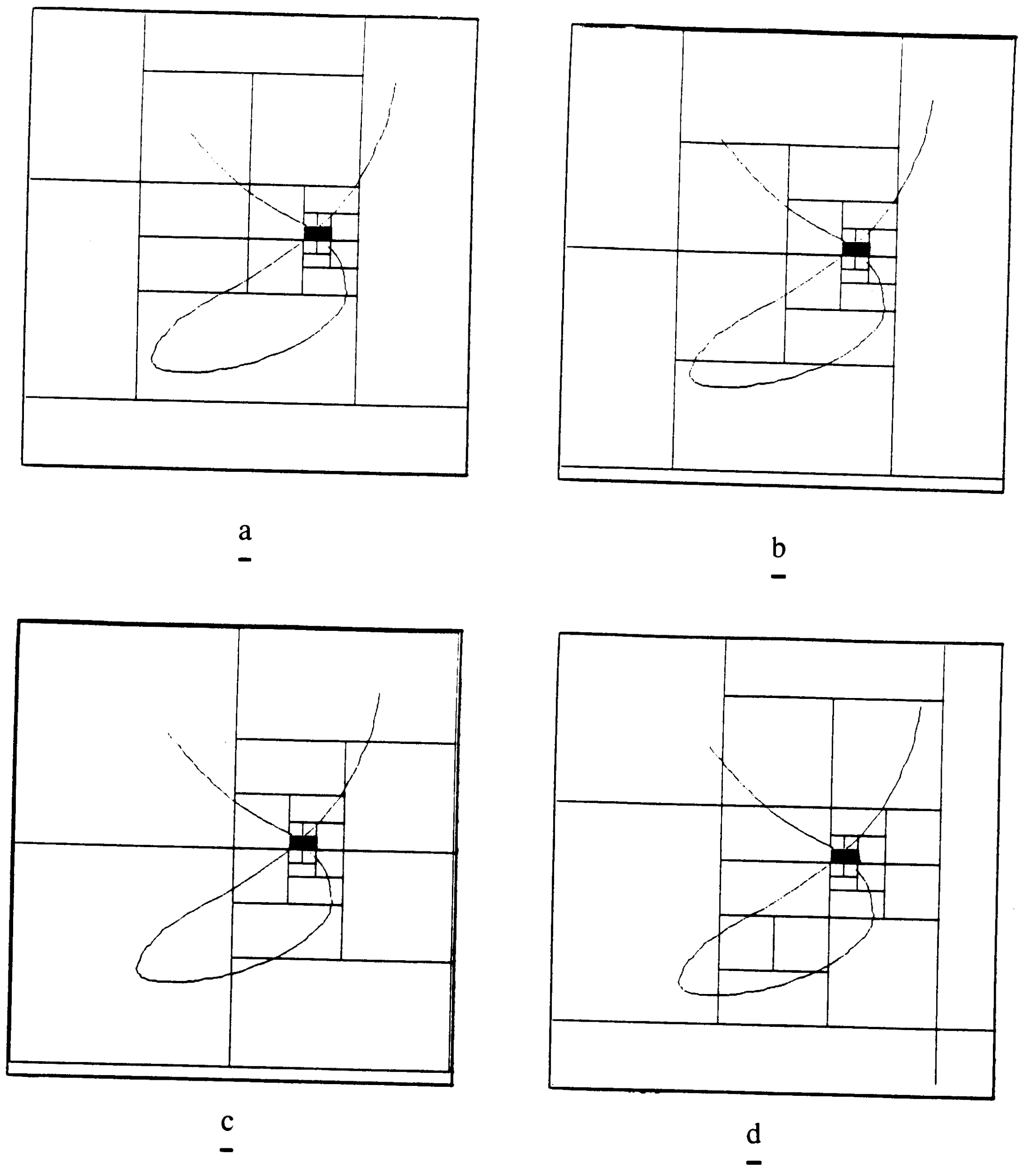


Figure 5.33: Position dependence is insignificant for all but the simplest inputs: (a) 19 chunks (best case); (b) 20 chunks; (c) 21 chunks; (d) 26 chunks (worst case).

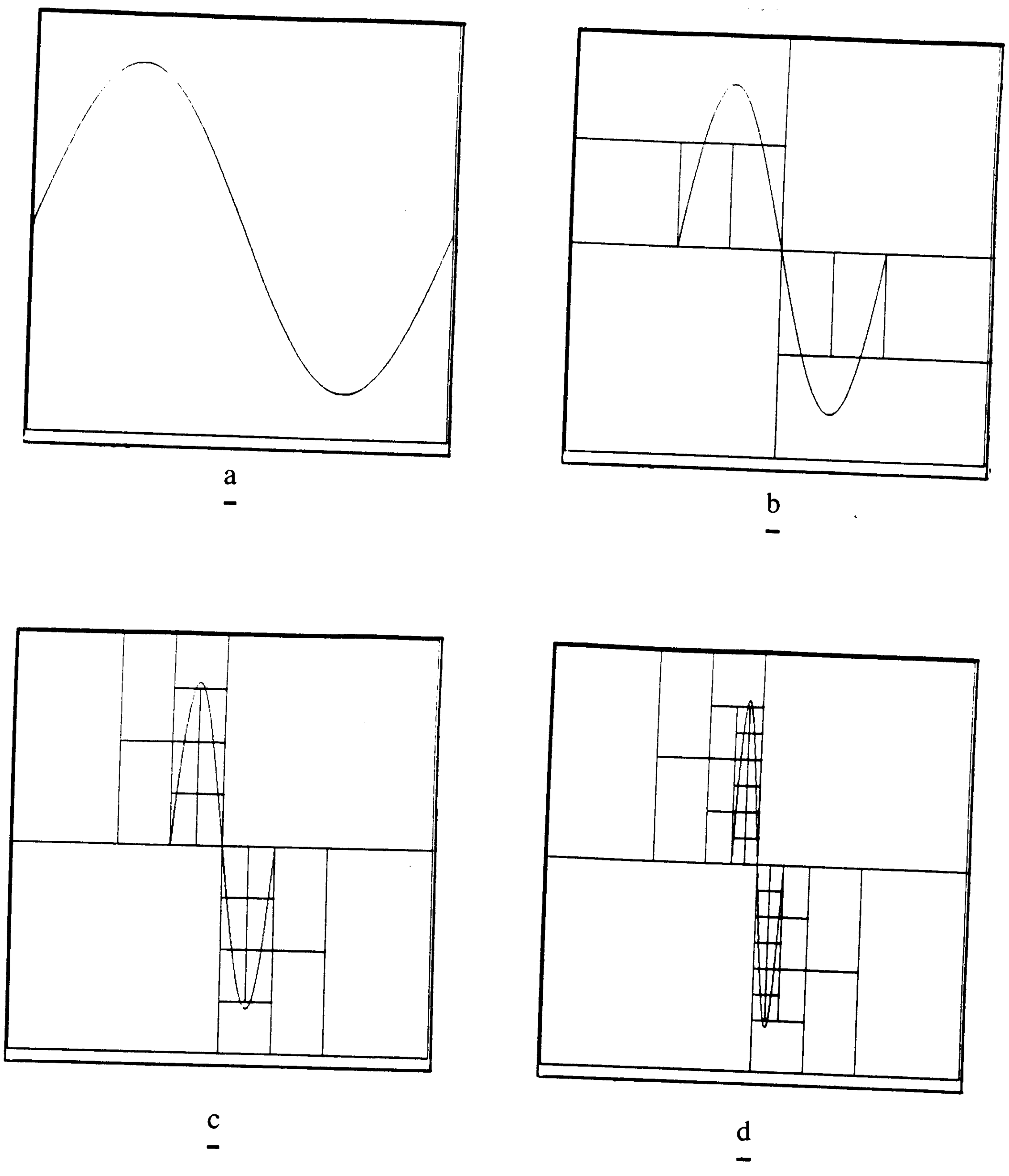


Figure 5.34: Divide-and-conquer curve chunking: effect of curvature.

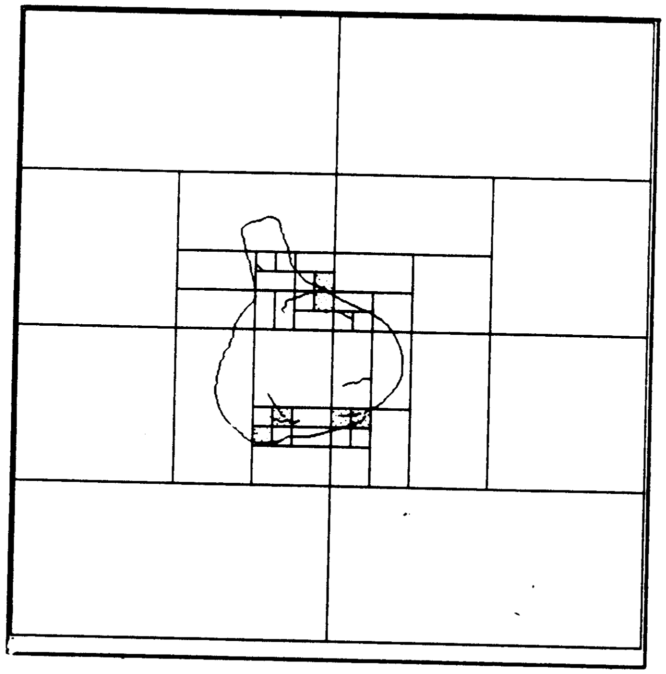


Figure 5.35: Divide-and-conquer curve chunking applied to intensity boundaries from an image of a garlic.

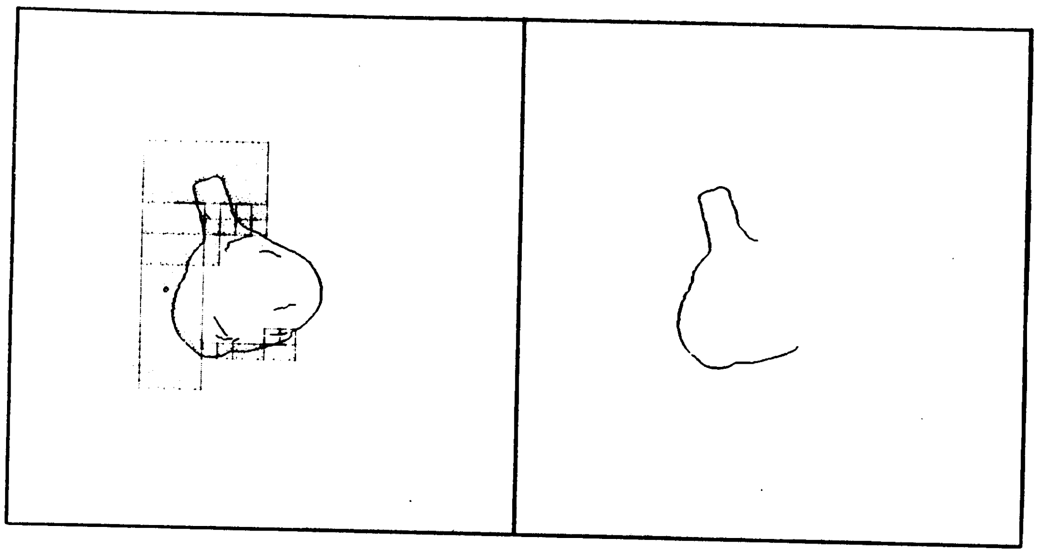


Figure 5.36: Tracing (incomplete due to small gaps): $P = 361$; $T_H = 10$; $I = 36$.

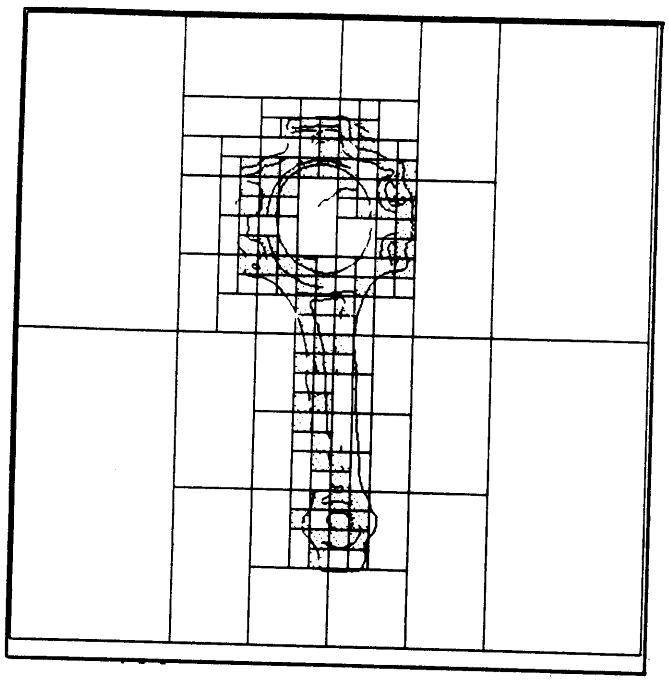


Figure 5.37: Divide-and-conquer curve chunking applied to intensity boundaries from an image of a connecting-rod.

5.5 Two-label, divide-and-conquer, unlimited-extent curve chunking

This section develops an extension to the preceding divide-and-conquer scheme that encounters less fragmentation at points of intersection or close proximity between curves. Merging failures are avoided by distinctly labeling curve segments that are in close proximity and then applying the merging rules on a per-label basis. The extended scheme simultaneously generates (i) a tessellation of the input and (ii) a two-valued distinguishing labeling within each region of the tessellation (Figure 5.40). Given this representation, a tracing process may issue a label to the elements of *one of* the curve segments in a region by specifying the relevant label to a *selective* region labeling operation. The region labeling operation modifies the *seen/unseen* label associated with a curve element only if another label associated with the element, call it the *0/1* label, has a specified value (Figure 5.40).

Two-label, divide-and-conquer curve chunking involves three simple extensions to the "one-

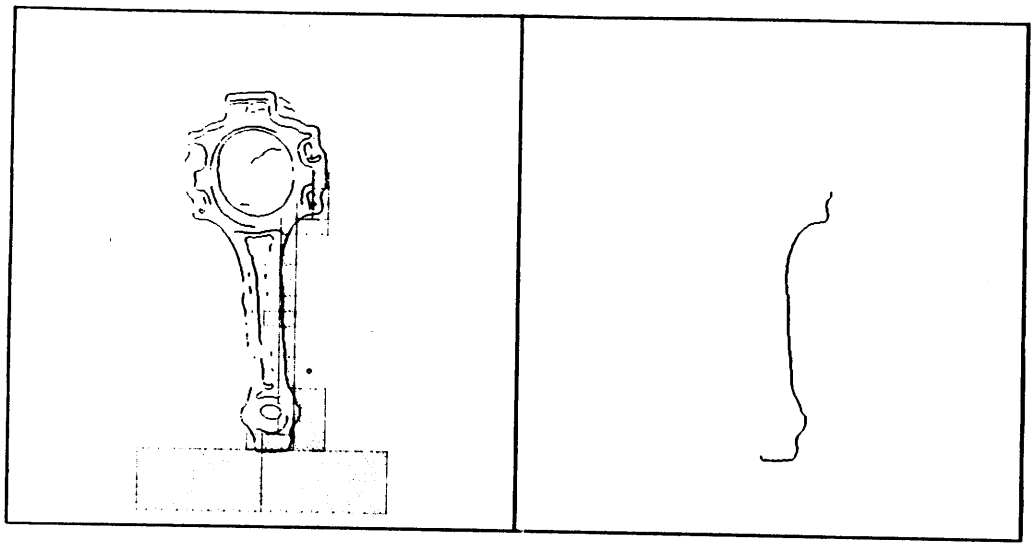


Figure 5.38: Tracing of connecting-rod boundary (incomplete due to small gaps): $P = 327$; $T_H = 12$; $I = 50$.

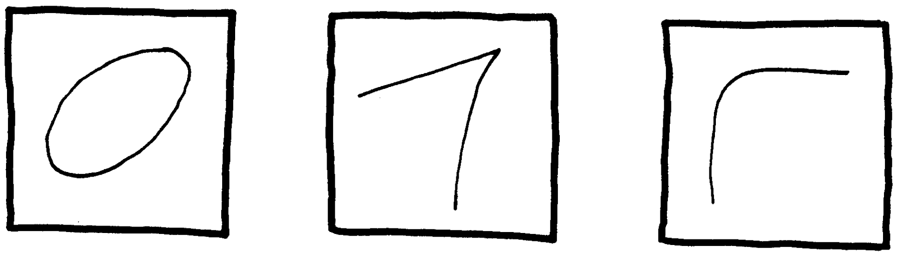


Figure 5.39: For each of these simple configurations, the divide-and-conquer scheme gives rise to a single chunk, but the direct scheme gives several smaller chunks, because its local curvature and orientation spread restrictions are violated.

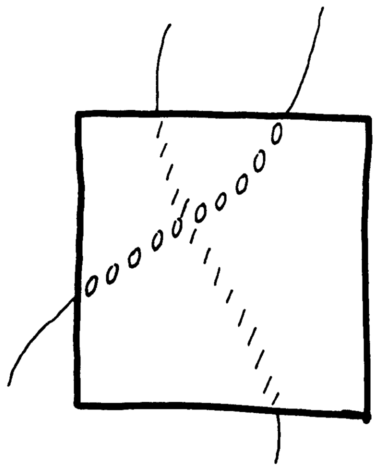


Figure 5.40: Two-valued distinguishing labeling of the curve segments in a region permits selective labeling at tracing time.

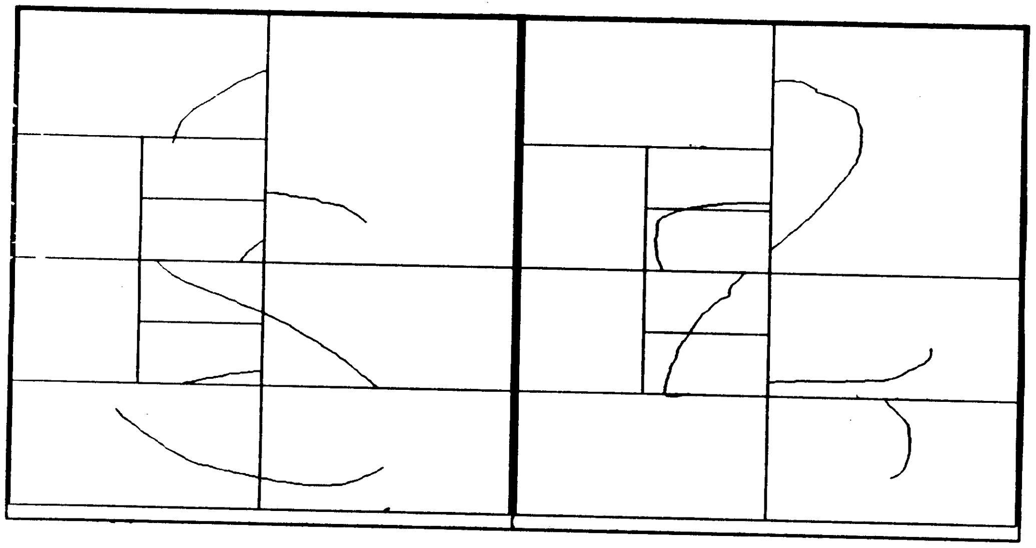


Figure 5.41: The result of two-label divide-and-conquer curve chunking. Each array represents one label. The regions of the tessellation in both arrays are in one-to-one correspondence.

label" divide-and-conquer method: (i) an initial *two-label, limited-extent decomposition* and (ii) an algorithm for building the two-label tessellation; and (iii) a modification to the curve continuity links in the pyramid. The rules for merging regions remain unchanged. The required extensions are developed in the following sections.

5.5.1 The two-label divide-and-conquer curve chunking algorithm

This section presents an algorithm for two-label, divide-and-conquer curve chunking. The choices for defining the subregions of a region and the rules for combining regions apply without modification. The divide-and-conquer algorithm involves the same four components: check elementary regions for validity; build valid regions; identify maximal valid regions; establish curve continuity between regions.

The initial two-label, limited-extent decomposition

A two-label, limited-extent validity computation is required to provide the starting point for the two-label divide-and-conquer process. It generates two effects as output: (i) each region in a fixed, predetermined tessellation is marked *valid* if it contains no more than *two* curves, and (ii) the curve segments in each valid region are distinctly labeled. For the purpose of this labeling, each curve element is associated with a two-valued label. It does not matter which segment gets which value of this label.

The curve segments in a given region may be distinctly labeled by pixel-at-a-time tracing; in the process of doing so, these curves may also be counted, thereby establishing region validity. To implement this, it is only necessary to modify the exit condition of the pixel tracing process described in Section 5.2.3: the procedure is exited when the count exceeds 2, not 1. The region-validity/curve labeling computation is simultaneously applied to each elementary region—the *valid/invalid* label is assigned appropriately, and each curve segment in a valid elementary region ends up labeled uniquely.

Building valid regions

As before, suppose the height of the pyramid is h, where the lowest level—the representation of the set of elementary regions—is 0. The variable l indicates the level of the pyramid currently being operated upon. The *valid/invalid* label of each node in the pyramid is initialized to *invalid*. The algorithm for building up valid regions is as follows:

Repeat for l from 1 to h:

1. (In parallel) for each node r at level l both of whose children are valid do the following:
   - (a) Apply the merging rule computations to its subregions independently per label.
   - (b) If the result of the merging rule check is positive *for both labels*, set the *valid/invalid* label to *valid*.
   - (c) Otherwise, do the following:
     - i. Interchange the labels of the curve elements in one subregion. (That is, if the labels are 0 and 1, then curve elements with label 0 are assigned label 1 and vice versa. This operation is referred to as *flipping* labels.)
     - ii. Apply the merging rule computations independently per label once again.
     - iii. Set the *valid/invalid* label according to this new result.
2. Increment l by 1.

There are two aspects of the performance of this basic algorithm that can be moderately improved. Firstly, according to the merging rules a vacant region and a non-vacant valid region may be merged, so there are three configurations in which the segments of a curve in the two subregions may be left with different labels by this algorithm upon a merge. I refer to this as "split-labeling" of a curve segment. Split-labeling leads to additional steps during tracing (the tracing operation must apply the region labeling operation to a split-labeled region twice, once for each label). It cannot be *avoided*, because labels are assigned independently within different subregions. Figure 5.42 shows three configurations for which the preceding algorithm can result in split-labeling. Split-labeling can be *corrected* by relabeling in cases (a) and (b), but not in (c). Correction requires the explicit detection of configurations (a) and (b). The required modification to the algorithm for accomplishing this correction is to replace Step 1b by the following:

If the result of the merging rule check is positive *for both labels*, then

1. Set the *valid/invalid* label to *valid*, and
2. if the common border of the two subregions is crossed but the elements involved are labeled differently *then*
   (a) flip labels in one subregion;
   (b) apply the merging rule computations to the two subregions independently per label;
   (c) if the result of the merging rule check is negative for either label, flip labels in one subregion again (i.e., we had something like the configuration in Figure 5.42 (c)—put things back the way they were).

The algorithm can give asymmetric results when applied in the context of the binary image tree. This is the second aspect of its performance can be improved, by extending it to apply in the context of the symmetric binary image tree. Recall that at odd levels of the symmetric binary image tree, horizontal and vertical rectangles are formed from square subregions; at even levels, each square is formed by the succesful merge of either a pair of horizontal rectangles or a pair of vertical rectangles. The required extension to the two-label algorithm is quite elaborate, and the added complexity does not seem to be justified by the improvement in performance.[9] The complicating factor is that each square region is a subregion of both a vertical rectangular region and a horizontal one—the flipping operation modifies the curve element labels, so merging cannot be performed independently for a horizontal rectangle and a vertical rectangle that have a square subregion in common.

[9]This extension was implemented and tested—it did not give a substantial performance improvement.

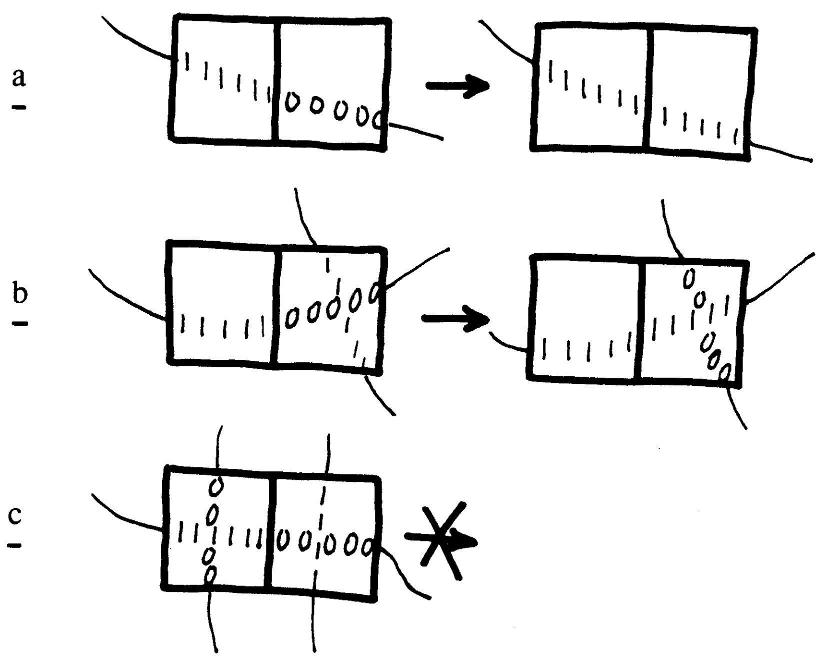


Figure 5.42: (a), (b), and (c) are valid configurations, but segments of the same curve in different subregions have been labeled differently. This is correctible in cases (a) and (b), but not in (c).

Identifying maximal valid regions

The process described for the one-label divide-and-conquer scheme applies in the two-label case without modification.

Establishing curve continuity

The process for establishing curve continuity between adjacent maximal valid regions in the one-label divide-and-conquer scheme applies in the two-label case with a minor modification. It is necessary to associate an additional bit of information with each edge between adjacent MVR's; this bit indicates whether the label associated with the curve elements is preserved or reversed across the region boundary. The curve tracing operation uses this bit to determine, for each node reached, the correct argument to the per-label region labeling operation.

Example illustrating the two-label divide-and-conquer process

Figure 5.43 illustrates the progress of this algorithm on a small example. The base of the pyramid is at the top of the diagram—pyramid level increases downward. The left-hand pair of columns of arrays corresponds to the first attempt to merge. The right-hand pair of columns of arrays corresponds to the second attempt to merge (following a flip), if any. The left-hand column of arrays in each pair shows the contents of regions per label 0. The right-hand column of arrays in each pair shows the contents of regions per label 1. Shading indicates a pair of regions for which an attempt to merge per the corresponding label fails. A string of the form $l - ji$ will be used to refer to the region in the array at level l whose y-coordinate is j and whose x-coordinate is i. In the example, an attempt to merge regions 0-10 and 0-11 per label 0 fails because both regions are non-vacant but their common border is not crossed per that label. When labels are flipped in 0-11, (i) the merge succeeds per label 0 because both regions are still non-vacant but their common border is crossed; (ii) the merge succeeds per label 1, because 0-10 is vacant.

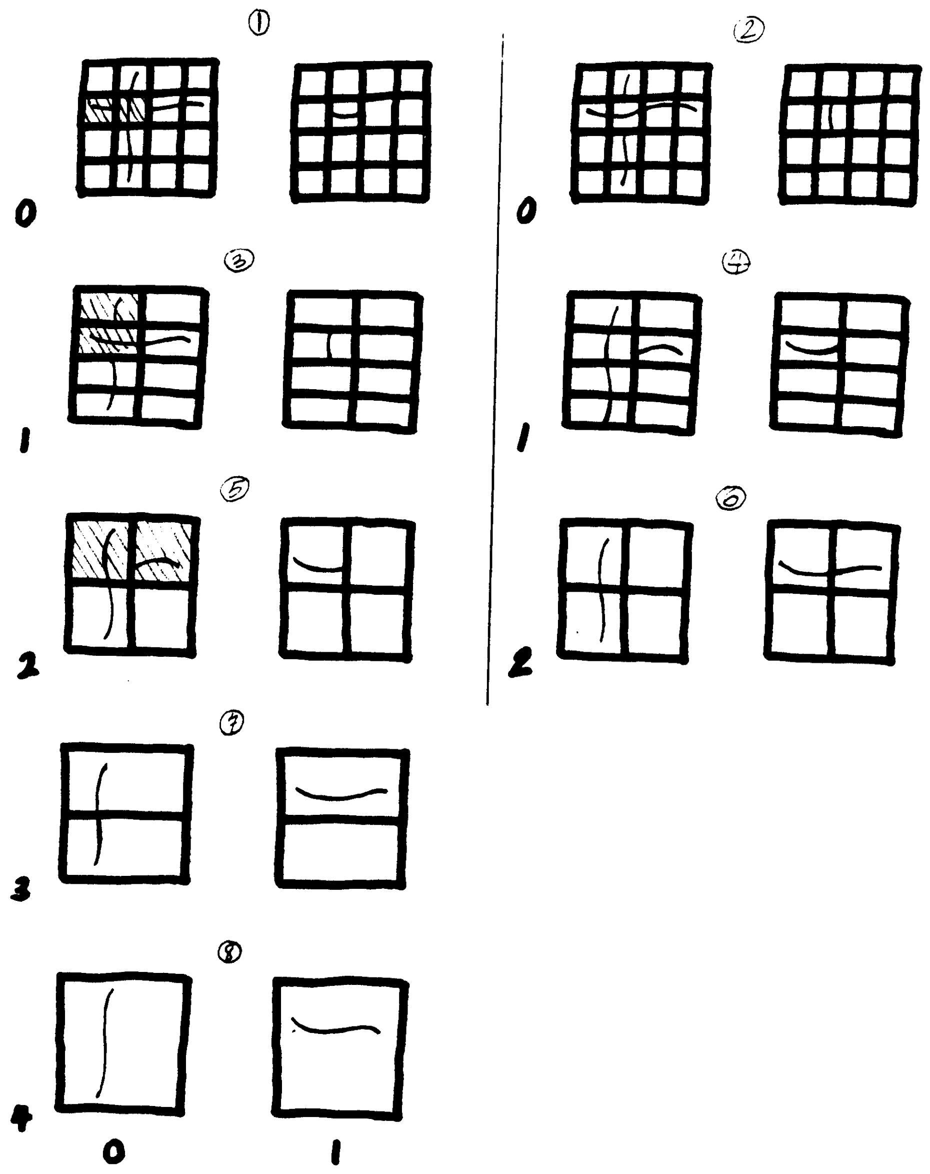


Figure 5.43: The two-label divide-and-conquer curve chunking process.

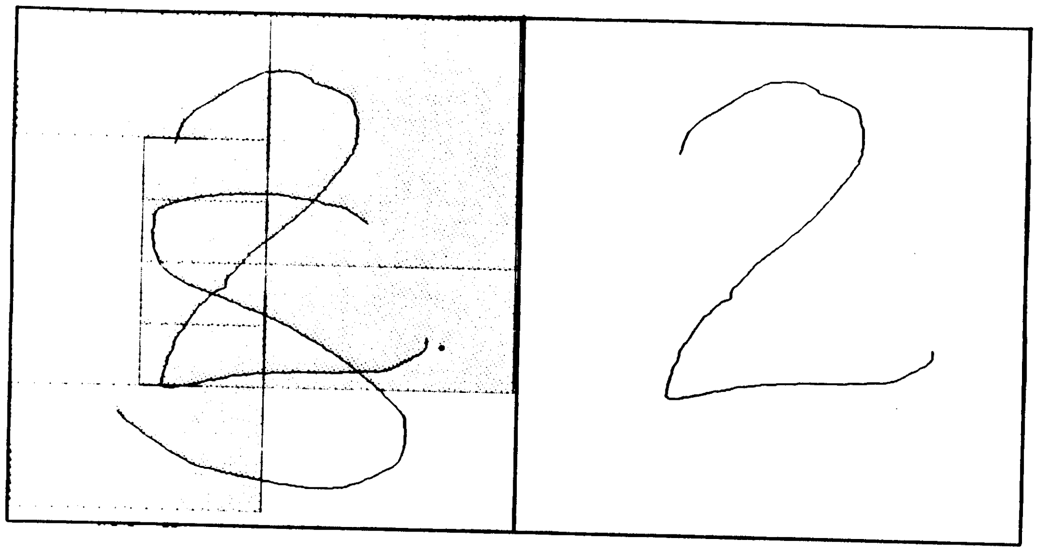


Figure 5.44: Tracing a curve by unlimited-extent curve chunks given by the two-label divide-and-conquer scheme: $P = 817$; $T_H = 14$; $T_D = 9$; $I = 0$.

5.5.2 Performance

Figure 5.44 shows the result of tracing one of the curves in the example of Figure 5.41. In the left frame, the small circle indicates the starting point of tracing, and shaded regions are curve chunks that were labeled. The right frame shows the curve extracted by this labeling process. The figure caption gives performance metrics (these numbers were introduced in Section 5.3.4.)

On the whole, the behavior of the two-label scheme closely parallels that of the one-label scheme, but its performance for any given input is substantially better. Notice that in the example of Figure 5.44, not only is T_H much smaller, but I has vanished—crossings (or near proximity) of just two curves do not give rise to invalid regions, so two-label tracing is not slowed down at these locations, even with the use of relatively large elementary regions. (Therefore, the use of two labels provides low maximum communication degree without a sacrifice in time performance.)

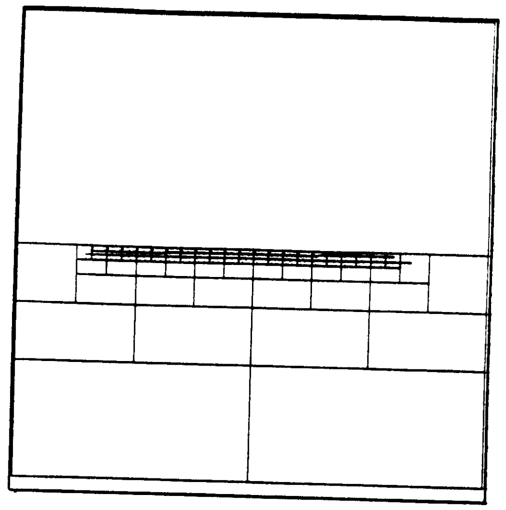


Figure 5.45: Simple worst-case for all previous methods - nearby parallel lines.

The best-case performance of the one-label scheme is achieved by the two-label scheme for somewhat more complex inputs. For example, Figures 5.46, 5.47, and 5.48 show that for simple configurations for which the one-label scheme is affected by proximity, curvature, and position, respectively, the two-label scheme is unaffected. Similarly, worst-case inputs for the one-label method are not quite worst-case for the two-label method—two labels provide a speed-up over pixel-tracing for complex inputs for which one-label does not. Figure 5.49 shows the result of two-label curve chunking on the input that was introduced in Section 5.4.4 as a "bad-case" for the one-label method (see Figure 5.30).

The performance advantage of the two-label scheme over the one-label scheme is large for simple inputs and decreases as the input becomes more complex. For example, an intersection of three lines leads to a degree of fragmentation similar to that resulting from the one-label divide-and-conquer scheme in the case of two intersecting lines (Figures 5.50 and 5.51). Figures 5.52 and 5.53 show that chunking and tracing performance are only moderately improved over the one-label scheme for the noisy intensity boundaries of the connecting-rod image.

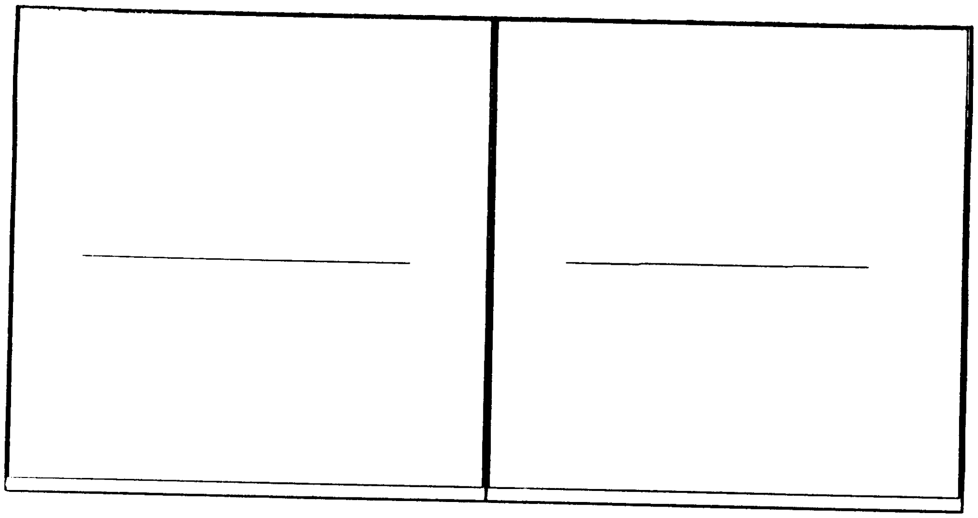


Figure 5.46: Two-labels lead to less proximity dependence.

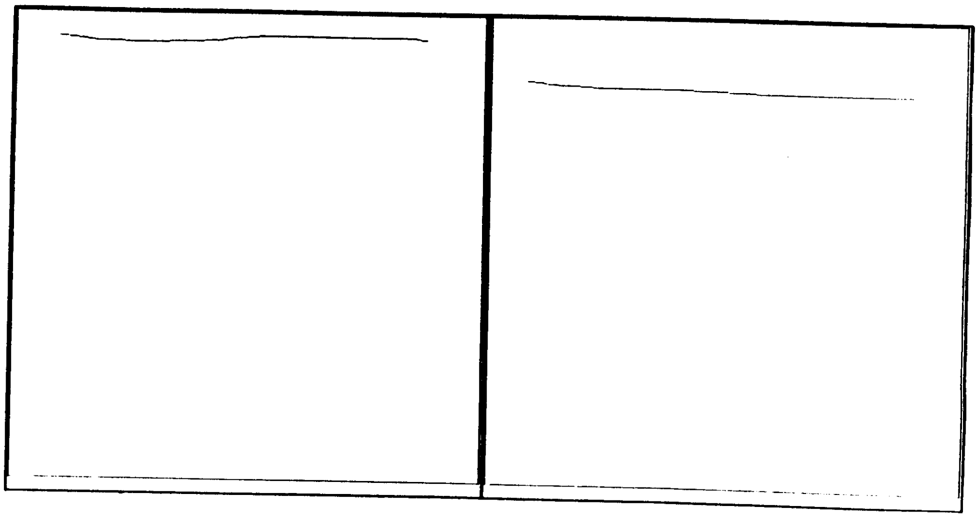


Figure 5.47: Two-labels lead to less position dependence.

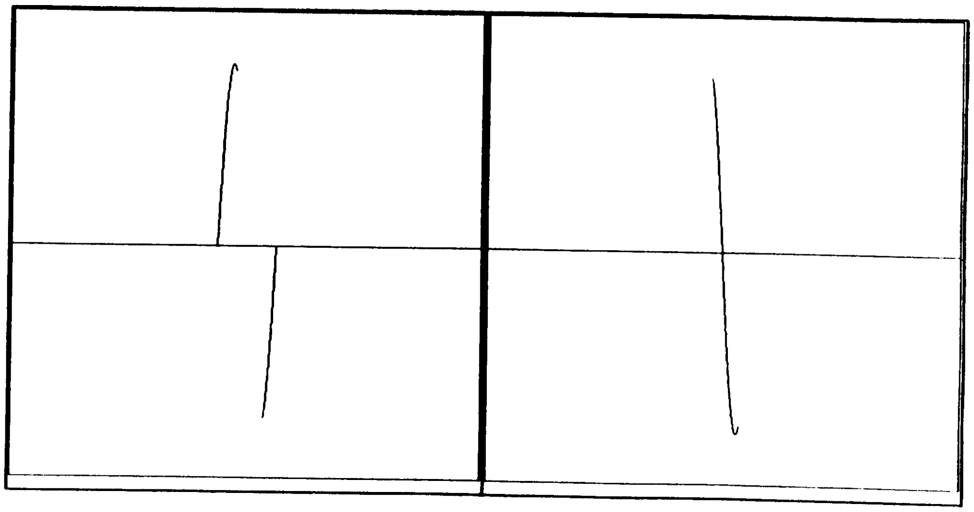


Figure 5.48: Two-labels lead to less curvature dependence.

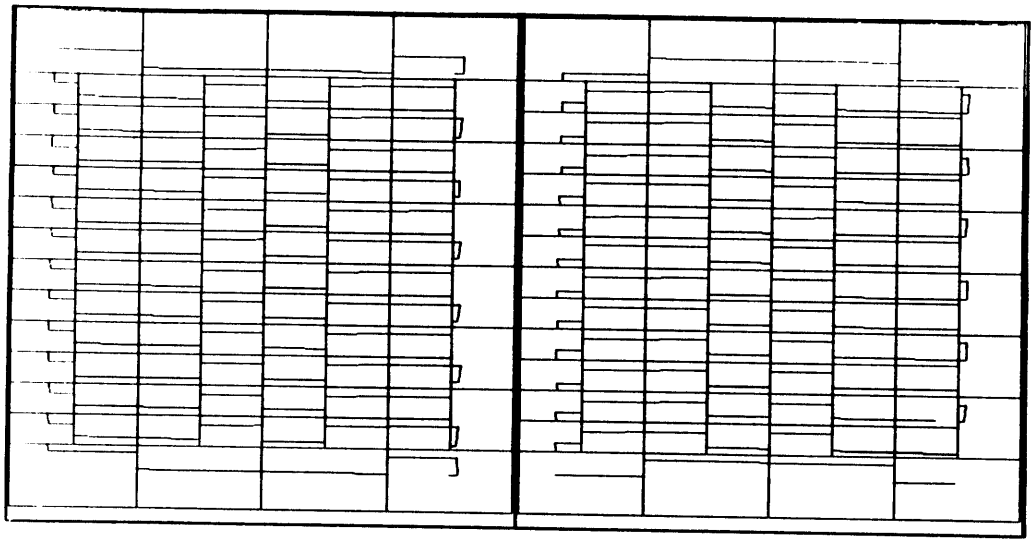


Figure 5.49: Two-labels lead to larger chunks even for “bad-case” inputs.

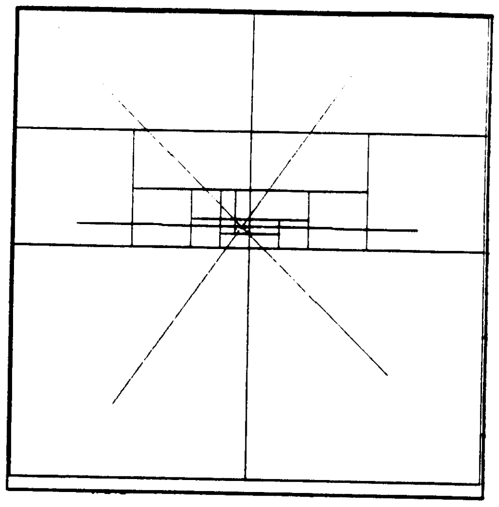


Figure 5.50: Fragmentation at a crossing of three intersecting curves: one-label divide-and-conquer.

Extension to more labels. The performance of the two-label scheme is much improved over the "one-label" scheme. How much is to be gained by adding more labels? n-label schemes, for $n > 2$ are unlikely to provide benefits worth the additional implementation cost, for three reasons. First, the number of independent merging checks to be done grows as $n!$. Secondly, much of the benefit of labeling arises at the locations of crossings, and crossings of three or more curves are uncommon in comparison to crossings of two curves. Finally, it is difficult for a pixel tracing process in the initial validity checking step to correctly distinguish curve segments at a crossing of three or more curves.

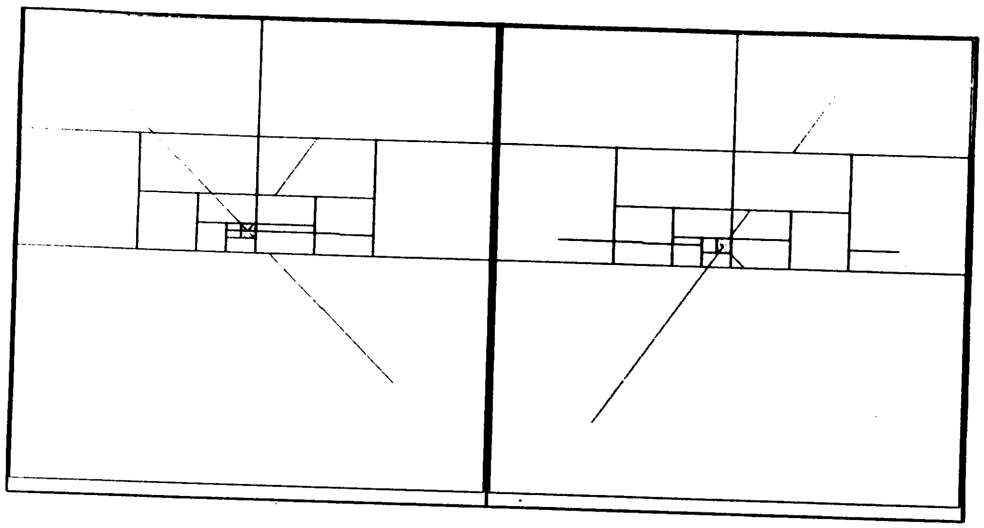


Figure 5.51: Fragmentation at a crossing of two intersecting curves: two-label divide-and-conquer.

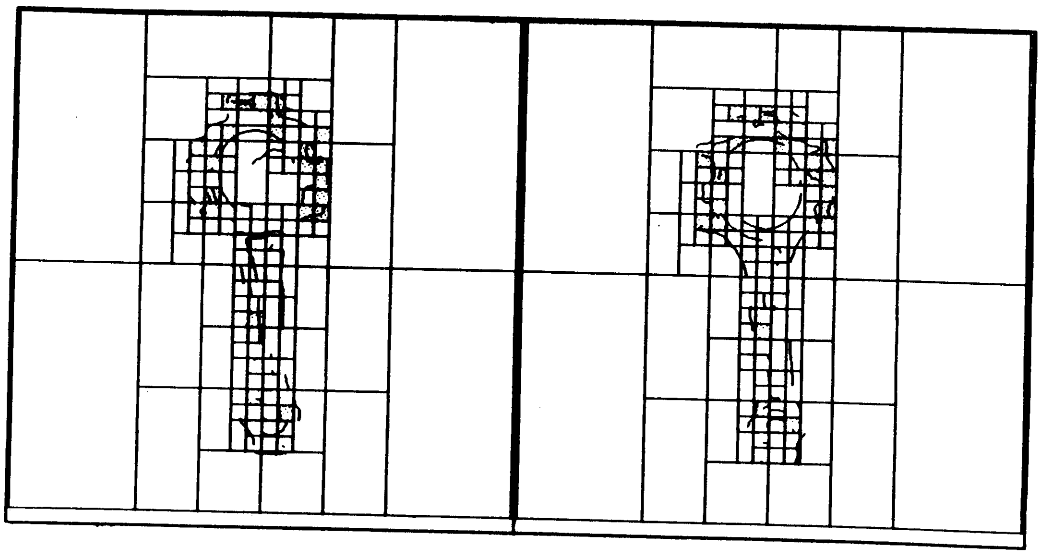


Figure 5.52: Two-label curve chunking applied to intensity boundaries from an image of a connecting-rod.

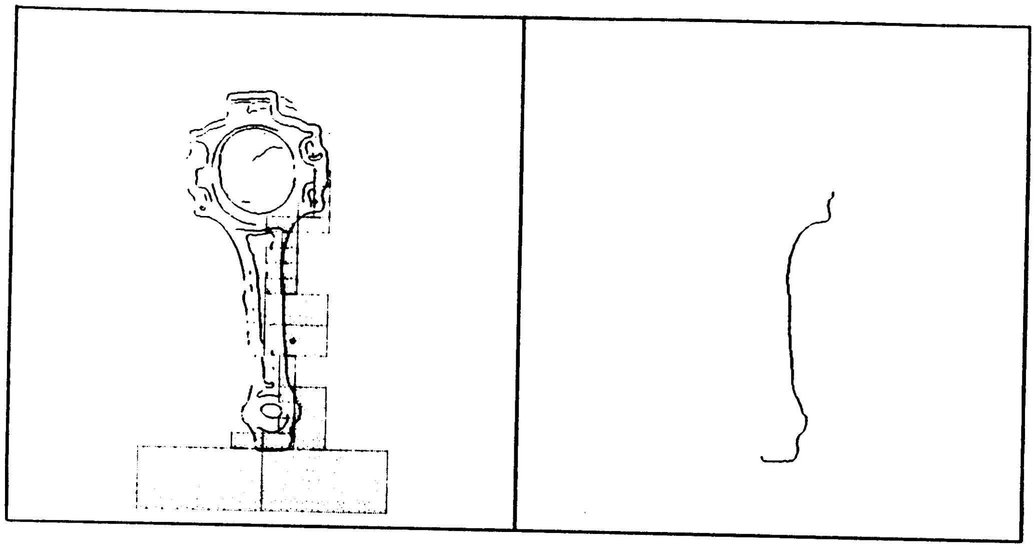


Figure 5.53: Tracing of connecting-rod boundary (incomplete due to small gaps): $P = 327$; $T_H = 15$; $I = 50$.

5.6 Summary of the four curve chunking techniques

Each of the four preceding sections introduced a technique for defining image chunks for curve tracing. These methods were introduced in increasing order of potential performance benefits. The limited-extent curve chunking scheme, the weakest method, provides a d-fold speed-up over pixel-level tracing, at best, where d is the region diameter. The three unlimited-extent curve chunking schemes lead to tracing times that depend not on the length of the input curves, but on their geometry – for all of these methods, the number of tracing steps is typically on the order of 10.

The direct, unlimited-extent curve chunking scheme is somewhat less powerful than the one-label divide-and-conquer scheme, because of its explicit restrictions on local curvature and orientation spread (the range of local orientations defined on a given curve.) On the other hand, the direct method is considerably faster and simpler to implement than the divide-and-conquer method. The difference in runtime between the two methods may not be very important, because tracing time may dominate chunking time. In order to make a proper assessment about this, more study in the application of these processes may be required. It does seem safe to say, though, that the difference in ease and cost of implementation may justify the slightly poorer performance of the direct method. Perhaps the most important difference between the two methods is that the divide-and-conquer approach can be extended in power through the addition of a local labeling capability, whereas the power of the direct technique cannot be extended in any simple way.

The two-label divide-and-conquer curve chunking scheme is considerably more powerful than the one-label divide-and-conquer scheme, because of its local curve segment labeling capability. The two-label method is also somewhat more complicated to implement, however; again, more study in the application of these processes would be needed to justify the additional cost. The ability of the two-label scheme to make pairs of distinct curve segments explicit in local regions at a series of scales is useful, however, quite apart from the consequent performance advantage to tracing—it could, for example provide the boundary segments that were required for the computation of local prominence information in Chapter 3. (Due to fragmentation at crossings, the one-label curve chunking scheme is very much less useful for this application.)

For very simple inputs, the performance advantage of the three unlimited-extent curve chunking schemes over the limited-extent scheme is dramatic. For very complex (but plausible) inputs, the performance advantage of the unlimited-extent schemes is rather small. This suggests that the quest for ever-more-powerful curve chunking techniques is an enterprise with rapidly diminishing returns—when the input curves have complex geometry, the task of "untangling" them is one to which local, parallel computations are inherently not suited.

In the worst case, none of the four curve chunking methods provides any improvement over pixel-level tracing, but the worst case configurations are extremely unlikely to occur in the course of visual spatial analysis.

5.7 Implications

The use of curve chunks has implications for the computation of spatial properties and relations relevant to both the design of machine vision systems and the study of human vision. Curve chunking raises questions regarding human visual perception at two levels. At the more general level, the hypothesis that curve chunks are made use of may lead to novel interpretations of perceptual phenomena, without regard to the particular processes by which the curve chunks are generated. Some pertinent observations in this regard are made in this section. At a more detailed level, it may be possible to make hypotheses about how curve chunks are generated in the human visual system, based on psychophysical studies and an understanding of the range of possibilities for computing curve chunks. Future work in curve chunking may take up this intriguing problem.

Many spatial judgements about curves involve the ordering of locations on the curve. The tasks of Figure 5.54 are examples. An important implication of the use of curve chunks for tracing is that the goal of extracting a curve as rapidly as possible may conflict with the goal of making judgements like these—the set of curve chunks representing a curve will only partially preserve ordering information. In the most extreme case, a curve may be represented by a single chunk, in which case no ordering information will be available at all! Therefore, it may sometimes take less time to establish curve connectivity (e.g., "same-curve") relations than relations involving curve ordering—the establishment of ordering relations may require a deliberately serial retracing of the curve, using chunks of artificially restricted extent. Thus, there is a possible distinction between the process by which a curvilinear entity is extracted and the process by which its properties and relations are computed. As such, in Figure 5.54 (a) it would take less time to decide that the two dots are on the same curve than that the curve in question is a right-handed spiral. Similarly, in Figure 5.54 (b) it would take less time to establish that the dot and the circle are on the same curve than to decide which is closer to an endpoint of the curve.

It is worth noting in this context that some properties and relations pertaining to curves may in certain circumstances be established without the use of curve tracing. The problem of evaluating tracing processes in the human visual system is complicated by this fact. For

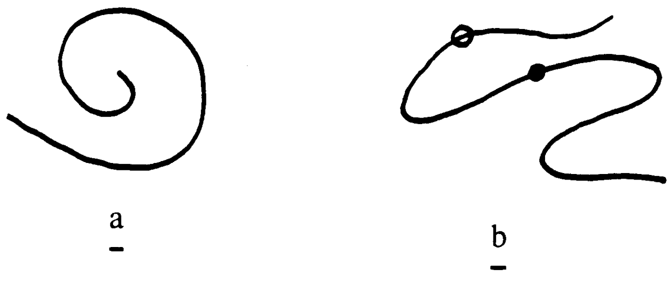


Figure 5.54: Judgements involving the ordering of locations on a curve. (a) Is the spiral curve right-handed or left-handed? (b) Which is closer to a termination of the curve, the small circle or the dot.

example, in Figure 5.55, it is possible to establish that the two dots lie on the same curve by counting curves and checking for breaks. Also, it may be possible to recognize certain curvilinear properties and configurations strictly in parallel. Perhaps the most striking demonstration of this is Fraser's illusion, Figure 5.56, in which nested concentric sets of disconnected spiral segments give rise to the irresistible impression of continuous nested spirals. This percept suggests that in this case local curve information is integrated in parallel in the human visual system.

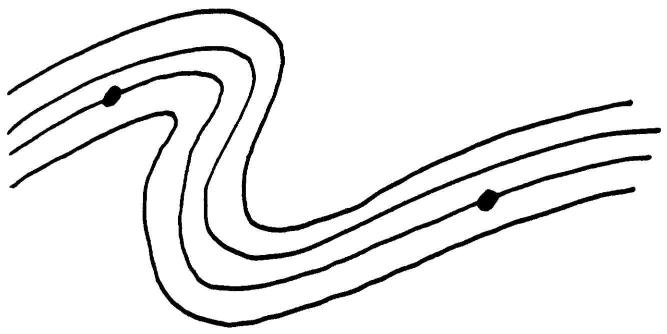


Figure 5.55: Establishing that the dots lie on the same curve in this example may not require curve tracing.

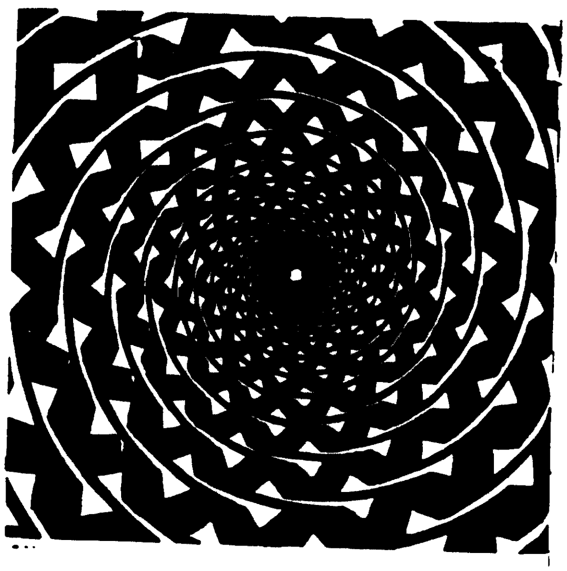


Figure 5.56: The Fraser illusion [Fraser 08]: perhaps the reason spirals are perceived where there are none is that descriptions of curves are being generated without the use of tracing.

Chapter 6

Region chunking

The preceding chapter explored one fundamental operation for establishing detailed shape properties of an object: that of making its boundary distinct from other boundaries. An equally important operation is that of making distinct the region in the image corresponding to an object's surface. This chapter explores time-efficient schemes for region activation, based on a simple, local model of computation.

Shafrir's [Shafrir 85] recent work has established that high-performance region coloring may be achieved by describing the input in terms of extended *empty* regions. This is the notion of a *region chunk* adopted in this report; it is open to quite a variety of implementation strategies, a number of which Shafrir explored in depth. The purpose of this chapter is to put representational support for high-performance region activation in the broader perspective of image chunking. Two main observations are made. The first is that it is possible for region and curve chunking processes to share their underlying machinery; parsimonious extensions to the curve chunking processes of the preceding chapter lead to effective region chunking processes at almost no additional cost. The second observation is that region chunking is most effective when applied to the results of curve activation operations, rather than to the input boundary representation itself.

6.1 Region chunks

In this section the computational nature of regions and region coloring operations is investigated and a definition of a region chunk is established.

Regions

A region is any connected set S of region elements, i.e., for any two elements p, q in S, there is a path from p to q.[1] A representation of the scene's boundaries is a two dimensional array some of whose elements are distinguished as curve elements; a region element is simply an element of this array that is not a curve element. (For example, if the boundary representation is a binary array in which locations having the value 1 constitute curve elements, then locations with value 0 constitute region elements.) The curve elements induce a division of all the region elements into a set of *maximal regions*; for any two maximal regions $R_i, R_j (i \neq j), \forall p \in R_i$ and $\forall q \in R_j, p$ and q are not neighbors.

Region activation

Region activation (or coloring) is a labeling operation: a single region element is given to begin with, and the goal is to label uniquely all elements of the maximal region containing the given element. As in curve activation, there are two problems: (i) establishing the set of locations constituting the relevant region on the basis of the single given location; and (ii) labeling this set of locations.

A parallel process for region coloring generally takes the following form: (i) label all unlabeled elements that neighbor labeled elements; (ii) if there are any unlabeled elements that neighbor labeled elements, repeat. The time-complexity of coloring is a function of the *diameter* of the maximal region to be colored, where the diameter of a region is defined to be the length of the longest path connecting two elements of the region.

[1]A definition of a path was given in Section 5.1.

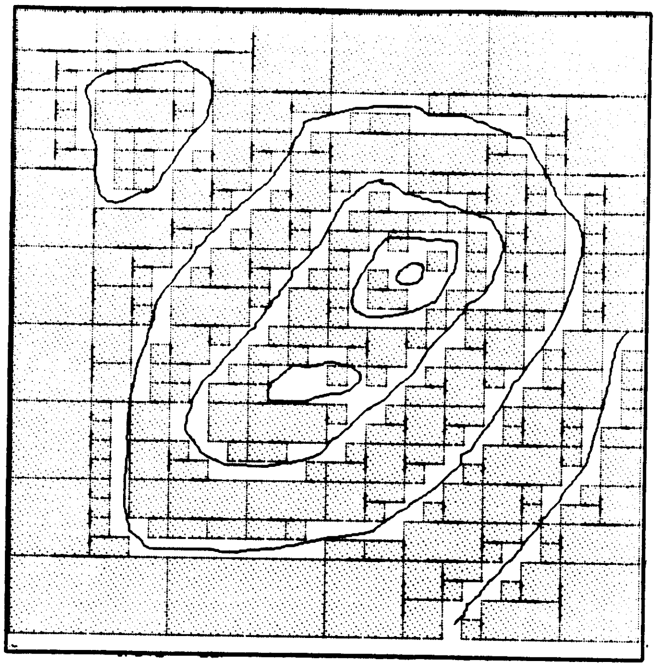


Figure 6.1: Input divided into non-overlaping region chunks.

Region chunking

The motivation for region chunking is to reduce the number of iterations incurred by the coloring operation. To do this, the coloring process must simultaneously label a subregion of a region at a step, rather than a single region element. A *region chunk* is defined to be a region containing no curve element (Figure 6.1). In the terminology introduced in Chapter 5, a region chunk is a *vacant* region. A "region chunking" is a tessellation of the input combined with an assertion, for each region, about vacancy. Region chunking defines a set of vacant subregions of each maximal region. The union of the set of subregions is a subset of the maximal region (some of the elements of a maximal region may be included in the non-vacant regions that contain the curve elements bounding the region). The effectiveness of a decomposition into region chunks is measured in terms of the number of steps incurred in coloring, which is related to the size of the region chunks—in general, the larger the region chunks, the fewer coloring steps are required.

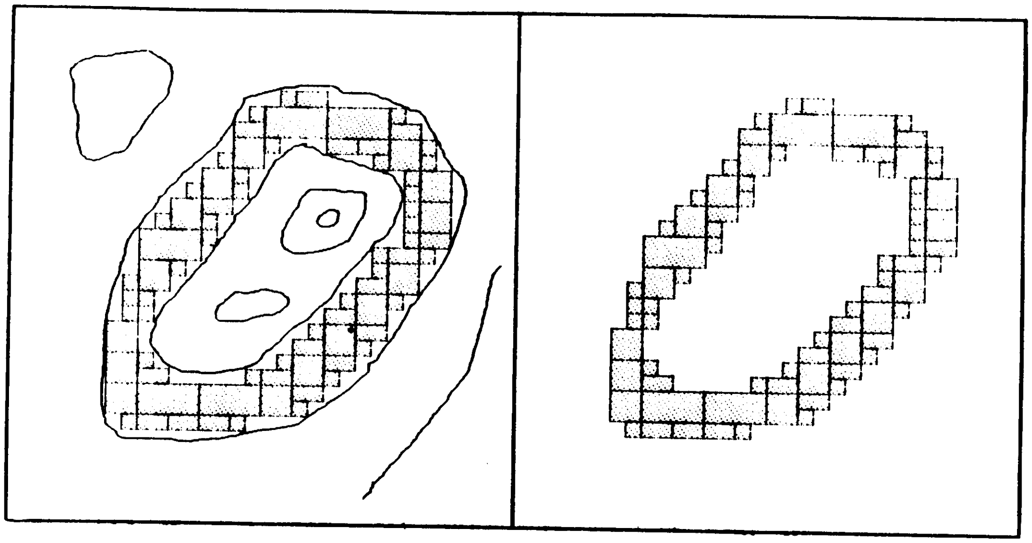


Figure 6.2: A region colored in chunks.

Region chunk representation and chunk-at-a-time coloring

A graph may be used to represent regions at the chunk level in the following way. A node in the graph corresponds to a region in the input. Each node has an associated two-valued label which may take on the values *vacant/non-vacant*. An edge in the graph corresponds to an instance of the connectivity (adjacency) relation between regions.

This representation of regions preserves the basic form of the coloring process, which is just a connected component labeling operation. Let C be a variable that can be associated with a set of region chunks. A single valid node s is given to start, which is initially assigned to C.[2] Each region chunk has an associated two-valued label which may take on the values *seen/unseen*, say. The basic form of a chunk-at-a-time region labeling process is as follows:

While any node in C has a vacant neighbor n labeled *unseen*:

[2]In this report, I do not discuss the processes that determine the starting chunk s. Generally, some independent process, such as indexing, will shift the processing focus to a location coinciding with or near an element of the relevant region. A mechanism is required for accessing the chunk node s corresponding to the region containing the processing focus.

1. for all vacant neighbors n of nodes in C:

    (a) label n and all the *region elements* in the corresponding region *seen* (this is a parallel operation);

    (b) add n to C;

2. repeat.

At the end of this process, a connected set of vacant nodes bounded by non-vacant nodes will have been labeled. If the region chunking process is capable of defining region chunks at a range of sizes that goes down to one pixel, the labeled set will correspond to a maximal region; otherwise, the labeled set may correspond to a ***subset*** of a maximal region. In the latter case, elements of the relevant maximal region that are in non-vacant regions will not have been colored by the end of the coloring process. An extension is required if the coloring process is to label the region elements of a maximal region that are in a non-vacant region.

One possibility for coping with non-vacant regions is to switch to region-element-at-a-time coloring when an invalid region is encountered (and switch back to chunk-at-a-time coloring if a region element is reached which is in a vacant region). This "two-level" coloring may be supported in the region representation by "splicing" each portion of an element-wise region representation that corresponds to an invalid region into the chunk-wise graph at the appropriate place.

The input to region chunking

When the goal is to color the region corresponding to an object's surface, only the physical (e.g., occluding) boundaries of the object should stop the coloring spread. Internal boundaries of the object of interest are irrelevant. Boundaries of other scene entities are irrelevant when the goal is to color the background of an object (the complement of the area in the image corresponding to the object's surface.) Irrelevant boundaries make the coloring process slower and more difficult. The process is slower because the relevant region

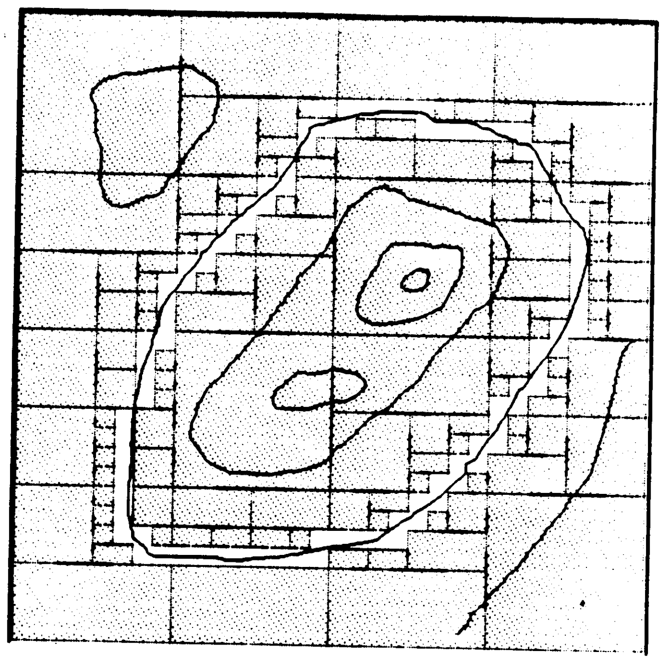


Figure 6.3: In this example, one boundary was first singled out for region chunking. In Figure 6.1, region chunking was applied to all boundaries.

is described by smaller, and therefore more, region chunks. The process is more difficult because a relevant region may not be colorable in a single invocation of the coloring process from a single starting location in the relevant region—irrelevant boundaries may divide the region of interest into two or more regions each of which must be separately colored (as in Figure 6.1). Therefore, it is desirable to first single out relevant boundaries and then apply region chunking. Figure 6.3 shows the result of region chunking of an input for which a relevant boundary was first singled out. Both the breakup of the relevant region into disconnected subregions and the reduced size of its region chunks are illustrated by comparing this example to Figure 6.1.

The machinery of region chunking

Like curve chunking, the process of finding a decomposition into region chunks may be thought of in terms of a component for generating a tessellation and a component for

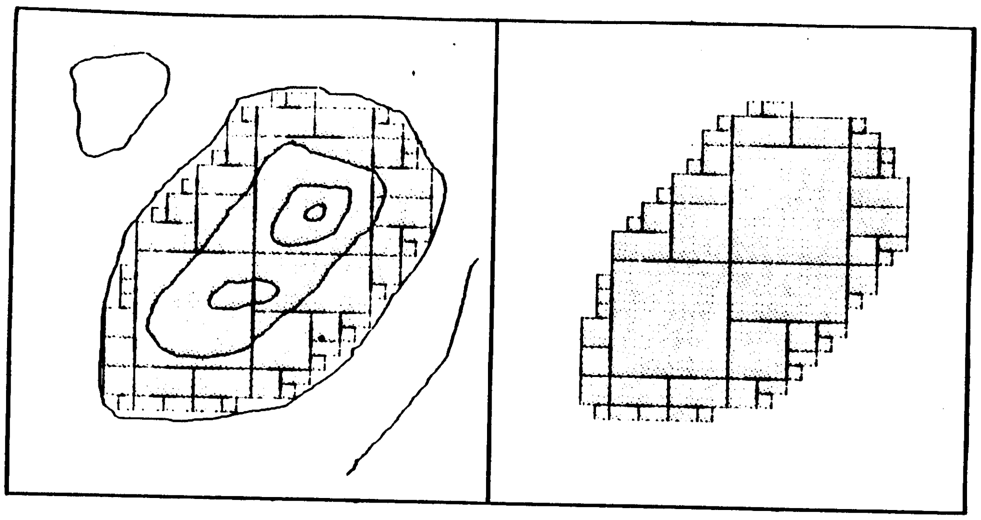


Figure 6.4: Singling out the relevant boundary before region chunking made coloring of its interior easier and faster.

checking region validity. It is possible for region and curve chunking processes to use common underlying machinery. The first point in support of this claim is that the mechanism for generating a tessellation obviously does not depend on the application which the regions in the tessellation are to serve. Secondly, the validity check for curve chunking subsumes the one for region chunking—recall that a valid region was defined to be a vacant region or a region containing exactly one curve segment.

This is not to say that curve and region chunking *must* use common machinery. Shared machinery connotes a savings in hardware cost, but it may also imply a compromise in the time performance of coloring, tracing, or both. Shafrir [Shafrir 85] showed that coloring speed is a function of the shape of the tessellation regions, and that, in the average case, coloring is swiftest using very elongated regions. The effect of region shape on tracing time has not been investigated, but suppose, for the sake of argument, that optimal average-case tracing required non-elongated regions.[3] Then, to support both optimal tracing and

[3]I conjecture that tracing performance is not optimized by the elongated region tessellation that optimizes

optimal coloring, two specialized tessellation mechanisms would be required.

The remainder of this chapter examines how the close relationship between curve and region chunking may be exploited to minimize hardware cost. Two main tessellation schemes were introduced in Chapter 5. One involved regions of fixed, limited extent. The other involved a binary image tree representation for defining regions at a series of sizes up to the size of the input. Associated with the limited-extent tessellation were two direct validity computations. Associated with the binary image tree tessellation were a direct validity computation and a divide-and-conquer validity computation. The following sections explain how each of these validity computations can support region chunking and assess the performance of such a region chunking method.

6.2 Limited-extent region chunking

The serial validity check for limited-extent curve chunking explicitly counts the curve segments in a region. A count of zero indicates a vacant region. The parallel validity check detects valid configurations on the basis of the measures T (termination count), and E (exit count). A region is vacant if $T = 0$ and $E = 0$.

The single-stage, parallel process which performs limited extent curve chunk validity checking can simultaneously generate region chunks—the only additional operation is to appropriately set the *vacant/non-vacant* label of a corresponding node in the region chunk output representation. Edges representing region adjacency in this representation are independent of the data—unlike curve continuity edges, they do not require state.

Let d denote the diameter of the regions defined by the tessellation. For inputs whose convex kernel is somewhat larger than d, the number of coloring steps resulting from the limited extent region chunking scheme is roughly linear in the diameter of the region in pixels. There is a roughly d^2 speedup over pixel-at-a-time parallel coloring.

coloring. For coloring, most regions may be decomposed into a relatively small number of straight elongated regions at just two orientations. It seems that elongated regions at a *series* of orientations and curvatures would be required to decompose an arbitrary curve into a correspondingly small number of chunks.

6.3 Direct, unlimited-extent region chunking

The validity check for direct, unlimited-extent curve chunking detects valid regions on the basis of the measures $S(\Theta(R))$ (orientation spread), T (termination count), and E (exit count). A region is vacant if $S(\Theta(R)) = 0$, $T = 0$, and $E = 0$.

The one-step, parallel process which performs direct, unlimited-extent curve chunking can simultaneously generate region chunks—the only additional operation is to appropriately set the *vacant/non-vacant* label of a corresponding node in the region chunk output representation. Edges representing region adjacency in this representation are independent of the data.

The number of coloring steps resulting from the unlimited extent region chunking scheme, using the binary image tree tessellation, is better than linear in the diameter of the region in pixels.

6.4 Divide-and-conquer, unlimited-extent region chunking

The validity check for divide-and-conquer, unlimited-extent curve chunking builds up maximal valid regions on the basis of some simple merging rules. Maximal vacant regions may be identified by an independent and even simpler process using the same pyramid. A node may be labeled vacant if every one of its descendant nodes is also vacant. A vacant node represents a *maximal* vacant region if its parent is labeled not-vacant.

Suppose the height of the pyramid is h, where the lowest level—the output representation of the initial chunking—has height 0. The variable l indicates the level of the pyramid currently being operated upon; its initial value is 1. A two-valued label, which may take on the values *vacant/not-vacant* is associated with each node of the pyramid. For all nodes at levels greater than 0 this label is initialized to *not-vacant.* The limited-extent region chunking process provides the initialization for level 0. The following process correctly sets the *vacant/not-vacant* label of every node in the pyramid.

Repeat for l from 1 to h:

1. (In parallel) for each node r at level l both of whose children are vacant, set the *vacant/non-vacant* label to it vacant.

Assume that each node in the pyramid may take one of two states, active and inactive, and all nodes are initially active. The algorithm for identifying maximal vacant regions does so by suppressing non-maximal vacant regions, as follows:

Repeat for l from h down to 1:

1. (In parallel) for each active vacant node r at level l, deactivate every descendant node of r.

Nodes remaining active after this process are the maximal vacant regions. They constitute the nodes of the output graph of region chunking. Edges between nodes, representing region adjacency, are independent of the data.

The number of coloring steps resulting from this unlimited extent region chunking scheme, using the binary image tree tessellation, is better than linear in the diameter of the region in pixels.

Chapter 7

Comparisons with other work

For the purpose of contrasting this work with related efforts, this study of image chunking may be appreciated on two different levels. At the more general level, this research is motivated by the need to make scene entities visually distinct rapidly, so that their spatial properties and relations may be established. At a more detailed level, this research is the study of effective representation for three basic spatial operations—indexing, tracing, and coloring—with time-performance being a primary consideration.

A great variety of image analysis techniques that have been proposed may be seen as, in large part, low level methods for *articulating* a pointwise description, i.e., for creating a new description whose constituent elements are more meaningful—that is, more directly useful to the goals of visual processing—than those in the initial description. By "low level," I mean methods that are in their essentials data driven, spatially uniform, and parallel. These schemes may be compared to image chunking at the more general level; they are reviewed in sections 7.4 and 7.5. Sections 7.1, 7.2, and 7.3 set the stage for this review. Section 7.1 sets out the criteria which will be the basis for evaluating previous image articulation methods. Section 7.2 examines the range of possibilities regarding input and output representation of an image articulation scheme—this discussion makes it possible classify and assess the various proposals. Section 7.3 recapitulates and assesses the image articulation framework which image chunking supports.

There has not been a great deal of work devoted to the problem of representing the image in support of very fast formulations of indexing, tracing, and coloring. Most of the work that has been done is in fact part of the same research program that mine is a part of—the study of visual routines. This closely related work, pertaining to line and region coloring, is reviewed in Section 7.6.

7.1 Grounds for comparison with other image articulation schemes

An image articulation scheme must meet two general requirements. First, the method must generate descriptions of scene entities that are effective for the tasks to be performed. Second, it must do so rapidly enough for the tasks to be accomplished succesfully. Two image articulation techniques can be directly compared only if they are designed to support similar tasks. The chunking methods I have proposed are not themselves directly comparable to most existing image articulation methods. Each chunking process performs a limited sort of image articulation designed to support a particular basic spatial operation. Most previous image articulation proposals are instead targeted directly toward ultimate goals of vision, like recognition or navigation, and they are usually intended to generate complete, detailed descriptions of scene components meaningful for these applications.

The thesis I have put forward in this report is that complete, detailed descriptions of meaningful scene components must be the product of a two-stage process, of which image chunking constitutes the first stage. The second stage assembles detailed descriptions from image chunks in a goal-driven, spatially focused manner. This two-stage process as a whole, but not image chunking by itself, may be usefully compared to most previous image articulation proposals.

The examples shown in Figure 1.1 and Figure 1.2 were designed to demonstrate that the best existing image articulation techniques do not provide adequate time-performance for tasks such as recognition, manipulation, and navigation in a general setting. These examples suggested two general requirements for rapid, effective extraction of scene components. The first requirement is that useful analysis of a scene should not rely on an initial, detailed, exhaustive extraction of the scene's components —there should be some data-driven basis for selectively applying detailed extraction processes at the likely locations of relevant entities. The second requirement is that detailed extraction should not take place on a pixel-at-a-time basis. It is with particular reference to these two requirements that previous image articulation methods will be evaluated.

7.2 The input and output of image articulation

Two distinctions regarding the nature of the output representation are useful in characterizing image articulation methods. The first distinction concerns the means by which the *segregation* of the set of all pointwise image elements into subsets constituting meaningful entities is achieved in the output representation. The pointwise information may be segregated in a local or global sense. In the local case, called *linking*, the output representation of an entity consists of its pointwise elements, each explicitly linked to its immediate neighbors. Elements of two different entities are (necessarily) distinguished from one another only by virtue of the fact that there is no path of links connecting them. In the global case, called *labeling*, each element of an entity is associated with an identifier that is unique to that entity. A labeled representation is more powerful than a linked one, in that spatial operations may be applied on a per label basis, and it is possible to operate directly on the set of elements constituting a relevant scene entity by specifying the associated label. To operate on a linked set of elements, the set must first be extracted from the array, which is equivalent to labeling it uniquely. (This is exactly what activation operations—e.g., tracing and coloring—accomplish.) Naturally, it is more costly in time, hardware, or both, to generate labeled representations than linked ones.

Secondly, the process of *segregating* the input into meaningful components may be distinguished from the process of *describing* the properties of these components. Description involves computing useful attributes of a *set* of pointwise elements and explicitly associating these attributes with the set. To implement this association, a *token*—some sort of declarative structure having a slot for each attribute—must be spatially identified with the set. Labeling is a prerequisite for description. Description is an optional adjunct to labeling. Image articulation schemes may therefore be classified as *descriptive* or *non-descriptive.*

(Note that labeling and description do not necessarily constitute distinct processing stages. For example, descriptive, labeled image articulation can be accomplished directly by template matching. A template match is descriptive if certain properties of the figure are explicitly associated *a priori* with the template. The matching template may serve as a label for (i.e., be uniquely identified with) the set of input locations that contributed to the match.)

The preceding distinctions lead to three useful classifications of articulated scene representations. These classifications are *non-descriptive, linked*; *non-descriptive, labeled*; and *descriptive, labeled.* Since there is no such thing as a "descriptive, linked" representation, the preceding classifications may be referred to in brief as linked, labeled, and descriptive, respectively.

Another distinction useful in characterizing image articulation methods regards the nature of the input representation to the articulation process. An image articulation scheme is said to be *region-based* if the input describes pointwise surface properties of the scene; it is *boundary-based* if the input represents discontinuities in these surface properties instead.

7.3 Two-stage image articulation

This section reviews and consolidates the two-stage image articulation framework, through the use of a complete example.

The first-stage representation of a scene entity consists of three components: (i) a set of figural chunks; (ii) a set of curve chunks; (iii) a set of region chunks. Of these, only the first component, the figural chunks, comes close to being a description directly applicable to recognition, manipulation, etc. The figural chunk representation is primarily a boundary-based, descriptive, labeled articulation of the scene into likely objects or object parts. The second-stage incrementally generates a descriptive, labeled articulation of the input into objects, by tracing boundaries, coloring regions, etc. This incremental process is guided by the figural chunk representation and by the visual task to be performed. A more detailed, general account of the two-stage articulation process is beyond the scope of this thesis. It is possible, however, to illustrate the general flavor of the two-stage articulation framework, by means of examples like Figure 1.1 and the task of counting the large blobs. In contrast to traditional image articulation methods, which would have to extract every figure in detail in order to perform this task, the two-stage scheme can detect and count the blobs in quite a small number of steps, in the course of which most irrelevant figures are never processed. The processing involved is as follows (see Figure 4.13):

1. Boundary detection.
2. Local prominence computations.
3. Chunking (figural and curve chunking done concurrently)
4. Extraction and counting of prominent blobs—three repetitions of the following sequence:
   (a) Shift to the most prominent, unprocessed, large figural chunk.
   (b) Trace the boundary underlying the figural chunk at this location to verify that we are indeed counting a large blob.
   (c) Mark the current location (so that it may be avoided at Step 4 (a).)
   (d) Increment the count.

7.4 Region-based image articulation schemes

This section presents a brief general review of region-based image articulation schemes. Region-based schemes subscribe to the uniformity assumption—the assumption that scene entities give rise to homogeneous image regions—so the arguments presented in Section 2.1 pose a serious general objection to these methods. Looking beyond this, there are a number of interesting points of comparison.

7.4.1 Non-descriptive schemes: region segmentation

The methods refered to under the heading of "region segmentation" are generally non-descriptive. The basic methods of region segmentation include local linking, split-and-merge, and global, histogram-based clustering [Ballard and Brown 82] [Horn 86]. (Many of the specific proposals do not fall strictly into one of these basic categories, but combine some aspects of each.)

In local linking, a local region property, e.g., average intensity, is measured over small pixel neighborhoods. Neighboring locations with sufficiently similar values of the measure are linked together. The result of such a process provides no support for selecting locations of relevant scene entities. Moreover, extracting an entity must be done on a pixel-at-a-time basis.

In split-and-merge schemes, the image is repeatedly subdivided into regions—say using a quadtree data structure—until every region gives a sufficiently high measure of homogeneity in some region property. (One of the simplest homogeneity measures used is the difference between the lowest and highest intensity values in the region.) After this spliting process, adjacent regions are merged (linked together) if the measure of the relevant region property is sufficiently similar over each of them. Split-and-merge may be viewed as a region-based analog of region chunking. The spliting phase is analogous to the process of establishing maximal vacant regions. The merging phase is analogous to the process of establishing connectivity links between adjacent maximal vacant regions. There are, however, two important advantages to the boundary-based region chunking proposal made in this report, beyond the fact that the uniformity assumption inherent in region-based split-and-merge schemes is generally incorrect. First, the extraction of regions based on region chunks can be more efficient because region chunking can be selectively applied to relevant scene boundaries, whereas region-based split-and-merge schemes apply directly to the image. Secondly, because it is boundary based, region chunking can completely share machinery with curve chunking processes—it does not seem that region-based split-and-merge schemes can share machinery as effectively with processes for extracting boundaries.

In global clustering methods, local region property measures taken at every image location are histogrammed, clusters in the histogram are used to divide the range of values of the measure into intervals, and each image location is labeled according to the interval of the measure to which its value belongs. A simple clustering technique is to establish interval boundaries at values of the measure coinciding with valleys in the histogram. Beyond the fact that the uniformity assumption inherent in these schemes is generally incorrect, an undesirable effect of global clustering methods is that they may label disconnected, physically unrelated image regions with the same value. Since it therefore cannot be generally assumed that locations having the same label belong to the same object, the labeling produced by global clustering cannot be used to directly extract meaningful scene components—a coloring process is still required. In other words, the output of global clustering is not a labeled representation in the sense defined above—instead, the labels merely provide a form of local linking.[1]

7.4.2 Descriptive schemes: blob detection

The region-based methods often referred to under the heading of "blob-detection" are descriptive, at least to some degree. These methods are generally pyramid-based. This section reviews the methods of [Pizer et al 85], [Dawson and Treese 85], and [Rosenfeld 86].

[Pizer et al 85] and [Dawson and Treese 85] had the explicit goal of providing indicators of possible objects *prior to* detailed identification. These workers adopted a two-stage approach to image articulation too, but the role of the first stage in their schemes is confined to the problem of object localization. Both proposals involve the use of copies of the intensity image at a series of resolutions (created by recursive Gaussian filtering and resampling).

In the method of [Dawson and Treese 85], a blob is defined to be a region in one of these images that is significantly lighter or darker than its surrounding area—that is, a local intensity extremum. Such regions are considered to correspond to objects. Spatial correspondence is established between such regions at adjacent resolutions; nesting relationships

[1]Only in artificially restricted situations, in which each image region is guaranteed to have a unique value of some local region property—e.g., uniquely colored items on a conveyor-belt —does the output of global clustering constitute a true labeled representation

are established between regions at the same resolution. A region constituting a blob at one resolution is defined using a local threshold—the mean intensity computed over a corresponding region at the immediately lower resolution.

In the method of [Pizer et al 85], blobs—termed "extremal regions" – are delineated in the original image by tracking intensity extrema from fine to coarse resolutions. The path of such an extremum moves continuously and eventually the extremum vanishes as resolution decreases. A blob is defined in the original image by the set of isointensity contours that (i) have the same intensity as a point on the extremal path and (ii) surround the extremal path's finest scale starting point.

In both of these methods, blobs can be *detected* by a local computation which notes the disappearance of an intensity extremum in scale space. The resolution at which this happens provides an indication of the spatial extent of the blob in the image. This blob detection technique does not directly provide other attributes of the blob such as elongation or orientation—to obtain these, spatial computations must be applied to the constituent locations of the blob at some substantially finer resolution than the one at which it vanishes.

Blob detection schemes of this sort can in principle support rapid detailed extraction of the region corresponding to a blob. [Pizer et al 85] describes a parallel process for generating a tree of links from blob-pixels in the original image to the extremum in the blurred image in which that blob vanishes. Presumably, to extract that blob in detail, it would only be necessary to propagate a label down through this tree from its root, and the time to do this is logarithmic in the diameter of the image. Such an account of the detailed extraction of scene components faces two main difficulties. The first is that it relies on a particularly restrictive form of the uniformity assumption —that objects are defined by *intensity* contrast with their surroundings. Secondly, the tree structure requires a large number of connections: scale must vary very gradually to provide continuous extremal paths in scale space, so many scale space images are required; image locations at adjacent resolutions must be individually linked.

[Rosenfeld 86] describes a pyramid-based divide-and-conquer scheme for constructing large, compact, homogenous regions. Measures of the mean and variance of some region property are used to determine whether two subregions may be merged. He proposes a modification of this basic idea to allow smoothly varying regions to be constructed —the new criterion for merging regions is that the *Fisher distance* between them should be small. In Chapter 2, the use of such distribution measures for defining regions was called into question on the grounds that it is possible in human vision, and should be possible in general, to perceive distinct regions even when sparsity and/or the rate of spatial variation of the relevant local property preclude a reliable estimate of any useful distribution measure.

7.5 Boundary-based image articulation schemes

This section reviews boundary-based image articulation schemes. Many boundary-based schemes subscribe to an assumption about the structure of images which I refer to as the *Gestalt assumption.* This is the assumption that meaningful scene components can be extracted in detail by *bottom-up processes which establish local relations* such as proximity, similarity, and colinearity. Systematic local proximity, similarity, or colinearity does typically arise from meaningful structure in the visual world, but it often is not enough to allow this structure to be recovered—there are many possible ambiguities. There are two options for dealing with these ambiguities. One is to generate all the scene components for which an ambiguity leaves open the possibility—this is impractical in general because there may be many such possibilities. The other option is to generate only the parts of scene components that can be generated unambiguously, and later assemble these parts into complete scene components using top-down information. This alternative is workable, and is frequently adopted, but it is inherently serial and slow. This is because the parts of scene components that can be extracted may take on a wide range of shapes or configurations.

7.5.1 Non-descriptive schemes: boundary segmentation

Spatially uniform schemes for linking local edge fragments together have been proposed, and these are pertinent to the problem of providing the input to chunking processes. [Zucker 80] and [Zucker 82], for example, developed cooperative processes for disambiguating local orientation operator responses in the context of line detection and dot grouping, respectively.

Cooperative computations provide a way of enabling more global information to bear on the descriptions of local spatial elements. These methods have the advantage, over other techniques that have been applied to similar goals in computer vision [Ballard and Brown 82], of being parallel and spatially uniform. These non-descriptive linking methods, however, do not by themselves constitute a significant step toward the goals of rapidly selecting processing locations or rapidly extracting scene components at these locations.

Marr's [Marr 76] proposal for extracting forms from the raw primal sketch (a set of pointwise assertions, called *place tokens*, about intensity changes in the image) constitutes a descriptive, boundary-based method. In his method, recursive grouping processes, applied in a spatially uniform manner, *replace* certain aggregates of place tokens by new place tokens. The grouping processes are irreversible, and so must be conservative. They repeatedly modify the same representation. Marr's account of the grouping processes suggests that the place token representing an aggregate would explicitly encode certain spatial properties of the aggregate; this information is made use of at higher grouping stages. For example, the grouping processes applied local colinearity, local connectivity, and closure constraints in order to extract complete object boundaries.

[Hong et al 82] and [Rosenfeld 86] describe a divide-and-conquer scheme whose aims are similar: the extraction of curvilinear forms on the basis of "good continuation." In this pyramid-based scheme for extracting smooth curves, each node in the pyramid stores a *summary description* of each curve segment in the corresponding region of the image. (There is a small fixed limit on the number of curves for which this may be done.) A node computes the description of the curve segment passing through its corresponding image region by combining the descriptions computed by its children for their subregions. Nodes at the base of the pyramid compute the description directly from the input. The description consists of the segment's overall orientation, endpoints, and local orientations at its endpoints. In a single pass from the base of the pyramid to the top, descriptions of the input curve segments at all scales are computed. This scheme is reminiscent of the divide-and-conquer curve chunking schemes presented in this report, but the resemblance is superficial. The goal of the method of Hong, et al, is to generate *descriptors* of *scene curves*—as individual components. The summary descriptions of curve segments and curves generated are *approximate* and *incomplete.* Detailed arithmetic computations are involved in generating these descriptors. The storage requirements per pyramid node appear to be quite extensive, if the method is to cope with complex inputs. The technique is quite involved in comparison to curve chunking, which has a goal that is more limited and therefore more appropriate to spatially uniform, parallel computation. ([Hong and Schneir 82] describe a closely related scheme for extracting compact objects as individual components from pointwise grey-scale and boundary data by explicitly computing *surroundedness* at each pyramid node. In comparing this scheme to region chunking, similar comments apply.)

[Samet and Weber 82] describe an *exact* quadtree representation for boundaries called a *line quadtree.* This representation is isomorphic to the conventional region quadtree. The tree's terminal nodes, however, store information about which *sides* of the corresponding region in the image *correspond to* a boundary (rather than whether the region includes a boundary location.) The elements of this representation conform to the definition given in Chapter 5 of curve chunks: each active node corresponds to a region containing a single curve segment; it simply happens that the curve segment in a region always runs along the region's border. The fact that image boundaries are normally quite irregular, however, limits the effectiveness of the line quadtree as a representation for fast curve tracing—most curve chunks it defines for a typical image are quite small, in fact of minimal size. Samet included storage for border information in the non-terminal nodes to make it possible to trace boundaries without individually examining all the terminal nodes of the tree. This modification, however, leads to undesireably large storage requirements. The divide-and-conquer curve chunking schemes I proposed support rapid curve tracing with minimal storage requirements.

7.5.2 Descriptive schemes: spine transforms

The methods discussed in this section are related to figural chunking in that they are low-level techniques for detecting and describing shapes in the input.

Crowley's [Crowley 84] proposal for describing shapes in an intensity image involves detecting and linking local maxima in band-pass (difference of low-pass) filtered versions of the image at a series of resolutions. Crowley's method may be thought of as boundary based because bandpass filtering provides edge enhancement—it is less sensitive to slow-variation in intensity than the blob-detection schemes that make use of recursive low-pass filtering. A peak or ridge point in this scale space indicates a local best fit of the circularly symmetric center lobe of the band-pass filter to the grey-level shape. This scheme is somewhat similar to the figural chunking scheme of Chapter 4 – the circular center lobe of the filter corresponds to a basic configuration model. There are, however, three significant differences. First, figural chunking may be applied to the result of *any* boundary computations, whereas Crowley's scheme is limited to intensity images. Secondly, Crowley's scheme is designed to provide exact descriptions of shapes for recognition, whereas the main goal of figural chunking is to provide crude descriptions of shapes for indexing. Finally, Crowley's scheme describes shapes, in effect, in terms of a single circular basic configuration model. For most shapes, the description consists of many local fits to this model, no one of which necessarily captures the overall shape or any of its prominent parts particularly well. Figural chunks are crude but self-contained descriptors of an overall shape or its simple parts. Individual figural chunks can therefore provide more guidance in the selection of processing locations than an individual match (a peak or ridge point) in Crowley's scheme. A *linked set* of matches in Crowley's representation cannot directly support indexing. A set of figural chunks provides a more concise (though perhaps less precise) description of an object or configuration than a linked set of local matches.

Crowley's shape representation is in essence an instance of a class of *spine representations* of shape. This class also includes Blum's *symmetric axis transform (SAT)* [Blum and Nagel 78] and Brady's *smoothed local symmetry representation (SLS)* [Brady and Asada 84]. Apart from the fact that these representations, like figural chunks, are computed from a boundary representation, similar comments apply to them as do to Crowley's. Associated with the smoothed local symmetry representation is Fleck's [Fleck 85] *local rotational symmetry (LRS)* representation. The LRS representation may be viewed as a descriptive representation restricted to the class of "roughly round" shapes. Apart from its restricted scope, the LRS technique has the disadvantage in comparison to figural chunking that its performance depends on the accuracy of the local boundary orientation information, and it involves detailed arithmetic computations using this information. The scheme is therefore inherently more sensitive to noise and apparently less suited to implementation in a simple local model.

7.6 Sibling studies of fast coloring

Two recent studies approached the problem of extracting scene components from a standpoint very similar to the one taken in the research reported here. One of these was a study of fast coloring of connected curvilinear components [Edelman 85]; the other was a study of fast region coloring [Shafrir 85]. Both workers achieved their goals by dividing the problem into two stages—a preprocessing stage defined appropriate building blocks for a subsequent coloring stage. They did not emphasize the notion of chunking in their discussion of the preprocessing. In my review of their work I have done so, to make the parallels clear.

7.6.1 Line coloring

Edelman's [Edelman 85] work on fast boundary coloring is related to my study of curve chunking for fast tracing. *Boundary coloring* is the problem of activating a connected curvilinear component.[2] The local relation used to establish the relevant set of curve elements in boundary coloring is connectivity, not curvilinearity. Edelman describes two closely related schemes for boundary coloring, both of which involve two phases. In the first phase, large regions of the image that contain a single connected curvilinear figure—called "OK-regions"—are detected by a quadtree-based, divide-and-conquer process. The second phase activates a connected curvilinear component starting from a given location on the relevant component by coloring one maximal OK-region at a step. Edelman's two methods differ in the computation used to detect OK-regions. In one algorithm, called MAC-1, a region is deemed *OK* if (i) all its subregions are *OK*; (ii) the Euler number computed over the region is 1; and (iii) the number of intersections of boundaries in the region with a

[2]Edelman uses the terms "boundary activation" and "boundary coloring" interchangeably. I reserve the term activation to refer to the general problem of distinctly labeling a set of locations—boundary tracing and boundary coloring are two different boundary activation operations.

vertical and a horizontal line each passing through the center of the region is 0 or 1 (this number is called the *I-count*). The purpose of the third requirement is to exclude the case in which the region contains a closed loop. In the other algorithm, called MAC-2, the Euler number measured over the region is required to be 1 (except in the case of a vacant region in which case it may be 0), and the I-count check is omitted. MAC-1 is provably correct—it is guaranteed never to mark a region containing more than one connected component *OK*. MAC-2 leads to substantially faster coloring than MAC-1, because it is less conservative and generates larger OK-regions, but it may sometimes wrongly label a region containing more than one connected component *OK*.

Edelman's proposal for computing the Euler number involves unrestricted counting of curve junctions and terminations in large regions, and the results must be passed between pyramid nodes. This is undesireable in the context of a simple, local computing model.[3]

It should be noted that curve chunking cannot be implemented by any straightforward modification of the preprocessing phase of the MAC algorithms. In those algorithms, the OK-check involves applying a partial connectivity test directly to the region as a whole; divide-and-conquer simply augments this computation by *excluding* certain illegal cases which the partial connectivity test wrongly accepts (i.e., loops in the region). In view of this secondary role of divide-and-conquer, the MAC algorithms are actually more similar to the direct, unlimited-extent curve chunking technique of Chapter 5 than they are to the divide-and-conquer schemes.

7.6.2 Region coloring

Shafrir [Shafrir 85] reports a thorough and very revealing study of the problem of fast region coloring. He investigated three high-performance schemes all of which involve (i) a preprocessing stage that detects regions of the input containing no boundary elements, and (ii) a coloring process whose basic operation is the simultaneous activation of all the pixels

[3]Incidentally, Edelman's implementation made use of pixel coordinates in the detection of junctions and terminations. The passing of absolute addresses between nodes is also undesirable in the context of a simple local computing model. It seems, though, that Edelman's basic scheme is implementable without address passing.

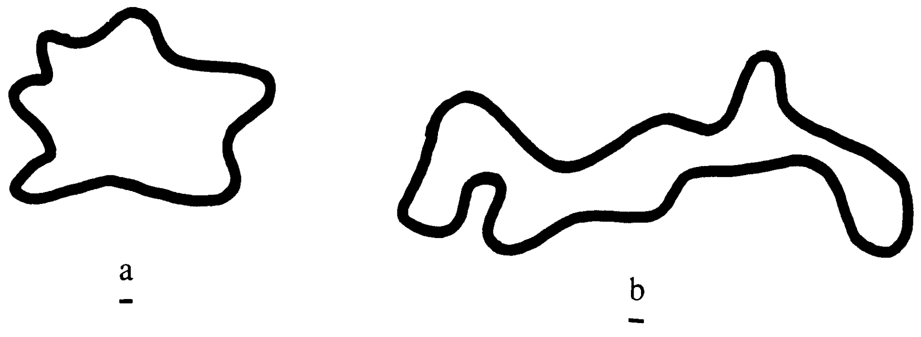


Figure 7.1: Some of Shafrir's test shapes: (a) stars (b) snakes.

in such a region. The schemes differ according to the shapes of the region chunks that the preprocessing phase can define, their degree of overlap, and the maximum communication degree of the hardware mechanism used to detect these regions. (Communication degree is defined to be the number of direct communication lines coming into a processor.) The average case performance of the different coloring methods was established by simulation. Mean coloring times were recorded for five families of visually important input shapes. These shapes included circles, horizontal squares, horizontal rectangles, "stars" (randomly generated shapes whose convex kernels were large in comparison to their total area), and "snakes" (randomly generated shapes whose convex kernels were small in comparison to their total area). Examples of stars and snakes are shown in Figure 7.1.

One of the schemes Shafrir investigated—the *quadtree model* – makes use of a quadtree-like input decomposition and processor network. The network's maximum communication degree is small (exactly 9). There is no overlap between regions. Region-vacancy is established by a hierarchical process. The basic computing step in this model is the operation of communicating a value between nodes that share a communication link. For compact shapes (circles, squares, stars) coloring time has only a small dependence on area of the shape. The position of the shape in the input and the position of the starting location within the shape did not significantly influence coloring time. For elongated shapes (rectangles and snakes), however, coloring time depended significantly on area.

Taking human performance as a guide, Shafrir found it unsatisfactory that the quadtree model's performance is significantly worse for simple elongated shapes (rectangles) than it is for compact ones. He developed another scheme—called the *rectangular model*—that is more satisfactory in that it colors simple elongated regions just as rapidly as compact ones. This scheme, illustrated in Figure 7.2, *directly* detects one-pixel-wide vertical and horizontal region chunks at a series of lengths and positions governed by a scaling parameter S and an overlap parameter O. For S small enough and O large enough, all possible region chunks are detectable, but this requires $O(n^3)$ processors, where n is the diameter of the square input array in pixels. Shafrir tested the performance of the the rectangular model for virtually the full range of settings of S and O subject to an $O(n^2)$ limit on the number of processors and an $O(n)$ limit on communication degree. He found that the particular settings of S and O did not significantly affect performance—all choices with similar "density" (number of chunk detectors and communication degree) gave similar behavior. The rectangular model is insensitive to the area of the input in the case of circles, squares, rectangles, and stars. Coloring requires fewer than 10 steps for these simple inputs. For snakes, there appears to be a small dependence on the area of the input figure. Naturally, performance improves even further with higher density.

The third high-performance coloring scheme Shafrir investigated—the *square model*—was built along the same lines as the rectangular model, but makes use of square regions instead. Squares are defined at a series of diameters and positions governed by S and O. The square model proved more sensitive to area of the input figure, and substantially slower overall, than the rectangular model.

The region chunking proposal of Chapter 6 is very similar to Shafrir's quadtree model. The discussion of region chunking in this report goes beyond Shafrir's study only in that (i) it explores the possibility of sharing machinery between curve and region chunking processes, and (ii) it integrates region chunking into a general framework. It is interesting to note that the representational requirements of the best known curve and region chunking schemes (the

```
         ***********
       ***222222222***
     ***2222222222222***
    **32222222222222233**
   **2212222222222222222**
  **222122222222222222222**
  *2222122222222222222222*
 **22221222222222222222222**
 *2222212222222222222222222*
**2222212222222222222222222**
*111111011111111111111111112*
*2222221222222222222222222223*
*2222221222222222222222222223*
*2222221222222222222222222223*
*2222221222222222222222222223*
*2222221222222222222222222223*
*2222221222222222222222222223*
*2222221222222222222222222223*
*2222221222222222222222222223*
**2222212222222222222222222**
 *2222212222222222222222222*
 **22221222222222222222222**
  *2222122222222222222222*
  **222122222222222222233**
   **2212222222222222223**
    **21222222222222222**
     ***2222222222222***
       ***333333333***
         ***********
```

Figure 7.2: Shafrir's rectangular region coloring model, applied to a circular input. Each numeral indicates the step of the coloring process at which a location was colored.

two-label divide-and-conquer scheme and the rectangular model respectively) appear at the moment to be quite incompatible.

Chapter 8

Conclusion

On the whole, low-level vision has been the study of extracting images from images. This thesis demonstrates that low-level processes can and must go much further than this—spatially uniform, parallel, bottom-up processes can build representations which make the inherently serial operations of scene analysis very time-efficient, even under the conservative restrictions of simplicity and locality. Image chunking is the "missing link" between traditional "very low-level" vision – the computation of pointwise properties—and the generation of detailed descriptions of complex spatial entities, properties, and relations.

Image chunking may be distinguished from the traditional notions of segmentation and grouping. Image chunks function much more as building blocks and "beacons" than as descriptors. Furthermore, there is a methodological distinction to be made, having to do with representation design—a more specific formulation of the application of a representation makes it possible to design a more effective representation for that application. The chunk representations are specifically designed to support particular basic operations of intermediate vision, whereas many proposals for "perceptual organization" have been designed to support *general goals* of vision, like recognition or reasoning, without detailed formulations of the processes that are to use the representations to achieve these goals.

Image chunking is not merely the exploitation of parallelism—it is the application of a crucial parallel-then-serial problem decomposition. This thesis is as much concerned with what low-level processes *cannot* do effectively, as with what they can. The goals of image chunking are deliberately limited compared to those of many other scene analysis schemes that exploit parallelism, because parallelism is exploited most effectively if its limits are not exceeded.

In this thesis I have emphasized the role of image chunks in providing time-efficiency in spatial analysis, but their importance goes beyond this—they also provide *expressive power.* Image chunks allow spatial analysis processes—visual routines—to be expressed in terms of entities that are more meaningful to the immediate goals of vision than are individual pixels. For example, the indexing operation —shifting of the processing focus to the location of a likely object —cannot be expressed on the basis of pixel-level representations of the scene, because the objects of interest in vision are typically extended. Similarly, it is more natural to express a "find-space" operation in terms of region chunks than pixels. Ultimately, I believe this notion of expressive power will prove crucial to the effective development and modification of visual routines, both in man and machine.

Future work

There are many ways in which the ideas explored in this report may be further developed. It is only possible to hint at some of them here.

Input to image chunking processes. Chapters 2 and 3 presented partial specifications of the input to chunking processes. The general problems of detecting boundaries and computing local prominence of boundary segments must be addressed more thoroughly. Boundary detection may be somewhat outside the scope of image chunking, but the study of image chunking will provide specific requirements that have not been available before.

Real inputs. Most of the demonstrations of image chunking in this report were on artificial inputs, so as to highlight the capabilities of the methods. Image chunking can provide substantial performance benefits even for very noisy, cluttered inputs, but the benefits are truly dramatic when the input is relatively noise-free. There are indications that it is possible to suppress a great deal of noise and irrelevant information in images at a very low level. The potential performance benefits to serial spatial analysis provide a very strong motivation for investigating this essentially unexplored and very intriguing problem. Some of the possibilities for generating high-quality boundary input to chunking are the following:

- more robust boundary detection, perhaps through the integration of multiple sources of information (ie. intensity, color, texture, motion, disparity) [Poggio 86];
- selection of boundaries according to their physical causes, especially occlusion [Voorhees 87];
- gap filling (for example, see [Ullman 76a]);
- suppression of short noisy boundary segments (one available technique is the detection of edges at low spatial resolutions, but it may be useful to explicitly suppress noise at high resolutions, in order to preserve detail).

Performance analyses of chunking processes. In this report I have given mainly qualitative accounts of the performance of chunk-based indexing, tracing, and coloring. More detailed quantitative performance analyses, both analytic and empirical, would be useful.

Refinement of the chunking processes. In this initial study of image chunking, it made sense to focus on simple, general formulations of the basic operations. There are, however, many specific situations arising in the analysis of images that may pose problems for the general formulations. These cases must be dealt with on an individual basis, and this may lead to more specific formulations of basic operations and the chunking processes supporting them. For example, the general formulation of curve tracing defines a curve to be a connected sequence of local elements. The basic tracing operation in this formulation would stop at every gap. In the processing of visual images, however, tracing across short gaps might be so frequently useful that it would be appropriate to implement this behavior in the basic operation itself, rather than at the level of the control in a tracing routine. This might be accomplished by somehow allowing for small gaps in the generation of curve chunks.

Visual routines. Image chunking is meant to support the efficient operation of visual routines—therefore, the ultimate justification for image chunking processes is in the benefits that they provide for the formulation and execution of visual routines for demanding visual tasks. The development of visual routines for a wide variety of tasks may lead to more detailed requirements on the chunking processes proposed here, and perhaps to the invention of new kinds of chunks.

Image chunks for other applications. Indexing, tracing, and coloring are not the only possible applications of image chunks. For one thing, it may be useful to detect chunks representing local shape in detail for recognition. It may also be useful to detect commonly occuring orientation patterns—distributions of local orientations often attributed to growth or propagation—such as parallelism, concentricity, radial convergence, spiral convergence, dendritic and other kinds of branching, etc. Global symmetry is another possible candidate for low-level detection.

Dedicated image-chunking hardware. Ultimately, it will be worthwhile to experiment with the construction of special-purpose hardware for image chunking. At the moment, technologies for constructing simple, local computing machinery have not been much explored. Therefore, for practical applications such as industrial vision systems, a likely implementation direction for the immediate future is the formulation of image chunking strategies specifically matched to the state of the art in hardware architectures. Perhaps research into the engineering of simple local machinery will eventually be motivated by the need for optimal visual time-performance in mobile robots and other compact, low-cost vision systems.

Psychological investigations. A number of implications of image chunking relevant to the study of human visual perception were discussed in Chapters 4 and 5. Psychophysical and neurophysiological studies may be devised to investigate whether image chunks support the analysis of spatial information and, if so, what the detailed nature of the chunking processes at work are.

Bibliography

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

[Barrow and Tenenbaum 78] Barrow, H. G., and Tenenbaum, J. M., 1978. "Recovering Intrinsic Scene Characteristics From Images," in *Computer Vision Systems,* A. R. Hanson and E. M. Riseman (eds.) New York: Academic Press.

[Blum and Nagel 78] Blum, H., and Nagel, R. N., 1978. "Shape Description Using Weighted Symmetric Axis Features," *Pattern Recognition, 10*: 167–180.

[BBN 85] Bolt Beranek and Newman Inc., 1985. "Development of a Butterfly Multiprocessor Test Bed," Rep. 5872, Quarterly Technical Report No. 1.

[Brady and Asada 84] Brady, M., and Asada, H., 1984, "Smoothed Local Symmetries and their Implementation." *Int. J. Robotics Research,* 3 (3).

[Canny 83] Canny, J. F., 1983. "Finding Edges and Lines in Images." Cambridge, MA: M.I.T. Artificial Intelligence Laboratory, AI-TR-720.

[Cook 83] Cook, S., 1983. "The Classification of Problems which have Fast Parallel Algorithms," Proc. FCT-83, Springer-Verlag Lecture Notes in Computer Science, 1983.

[Cornsweet 70] Cornsweet, T. N., 1970. *Visual Perception.* New York: Academic Press.

[Crowley 84] Crowley, J, L., 1984. "A Representation for Shape Based on Peaks and Ridges in the Difference of Low-Pass Transform," *IEEE Transactions on Pattern Analysis and Machine Intelligence, PAMI-6, (2)*: 156–170.

[Dawson and Treese 85] Dawson, B. and Treese, G., 1985. "Locating Objects in a Complex Image," SPIE Vol. 534, Architectures and Algorithms for Digital Image Processing II: 185–192.

[Duda 76] Duda, R. O., and Hart, P. E., 1973. *Pattern Recognition and Scene Analysis.* New York: Wiley.

[Edelman 85] Edelman, E., 1985. "Fast Distributed Boundary Activation," M.Sc. Thesis, Department of Applied Mathematics, Feinberg Graduate School, Weizmann Institute of Science, Rehevot, Israel.

[Fleck 85] Fleck, M., 1985. "Local Rotational Symmetries." Cambridge, MA: M.I.T. Artificial Intelligence Laboratory, AI-TR-852.

[Fraser 08] Fraser, J., 1908. "A New Visual Illusion of Direction," *British Journal of Psychology 2*: 307–320.

[Gottlieb et al 83] Gottlieb, A., Grishman, R., Kruskal, C., McAuliffe, K., Rudolph, L., and Snir, M., 1983. "The NYU Ultracomputer – Designing an MIMD Shared Memory Parallel Computer," *IEEE Transactions on Computers* C-32 (2): 175–189.

[Hillis 85] Hillis, W. D., 1985. *The Connection Machine.* Cambridge, MA and London: The MIT Press.

[Hirschberg 76] Hirschberg, D. S., 1976. "Parallel Algorithms for the Transitive Closure and Connected Component Problems." *ACM 8th Symposium on Theory of Computation:* 55–57.

[Hong et al 82] Hong, T. H., Schneier, M., Hartley, R., Rosenfeld, A., 1982. "Using Pyramids to Detect Good Continuation," Computer Science TR-1185, University of Maryland.

[Hong and Schneir 82] Hong, T. H., Schneier, M., 1982. "Extracting Compact Objects Using Linked Pyramids," in Proc. Image Understanding Workshop, Stanford, CA, 1982: 58–71.

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

[Jastrow 00] Jastrow, J., 1900. *Fact and Fable in Psychology.* Boston: Houghton Mifflin.

[Jolicoeur et al 86] Jolicoeur, P., Ullman, S., and Mackay, M., 1985. "Boundary Tracing: An Elementary Visual Process," *Memory and Cogniition 4*: 219–227.

[Julesz 81]	Julesz, B., 1981. "Textons, the Elements of Texture Perception, and their Interactions." *Nature*, 290: 91–97.
[Koch and Ullman 84]	Koch, C., and Ullman, S., 1984. "Selecting One Among the Many: A Simple Network Implementing Shifts in Selective Visual Attention." A. I. Memo 770. Cambridge, MA: MIT Artificial Intelligence Laboratory.
[Lim 86]	Lim, W., 1986. "Fast Algorithms for Labeling Connected Components in 2-D Arrays." Report No. NA86-1, Thinking Machines Corporation. Cambridge, MA.
[Lowe 83]	Lowe, D. G., 1984. "Perceptual Organization and Visual Recognition." Computer Science Department Report. Stanford, CA: Stanford University.
[Marr 76]	Marr, D., 1976. "Early Processing of Visual Information." *Phil Trans. Roy. Soc. B, 275*: 483-524.
[Marr 82]	Marr, D., 1982. *Vision.* San Francisco: W. H. Freeman.
[Marroquin 76]	Marroquin, J., 1976 "Human Visual Perception of Structure." S.M. Thesis. Cambridge, MA: Dept. of Electrical Engineering and Computer Science, MIT.
[Minsky and Papert 69]	Minsky, M. and Papert, S., 1969. *Perceptrons.* Cambridge, MA and London: The MIT Press.
[Muller 86]	Muller, M. J., 1986. "Texture Boundaries: Important Cues for Human Texture Discrimination." *IEEE Proceedings of Conference on Computer Vision and Pattern Recognition, 1986:* 464–468.
[Pavlidis 82]	Pavlidis, T., 1982. *Algorithms for Graphics and Image Processing.* Rockville, MD: Computer Science Press.
[Pizer et al 85]	Pizer, S. M., Koenderink, J. J., Lifshits, L. M., Helmink, L., Kaasjager, A. D., 1985. "An Image Description for Object Definition, Based on Extremal Regions in the Stack," in Int. Proc. in Med. Imaging, 9th IPMI Conf., Washington D.C., 1985.
[Poggio 86]	Poggio, T., 1985. "Integrating Vision Modules with Coupled MRFs." (Unpublished.) Cambridge, MA: M.I.T. Artificial Intelligence Laboratory.
[Riley 81]	Riley, M. D., 1981. "The Representation of Image Texture." S.M. Thesis. Cambridge, MA: Dept. of Electrical Engineering and Computer Science, MIT.

[Rock 84]	Rock, I., 1984. *The Logic of Perception.* Cambridge, MA and London: The MIT Press.
[Rosenfeld 79]	Rosenfeld, A. 1979. *Picture Languages.* New York: Academic Press.
[Rosenfeld and Kak 76]	Rosenfeld, A. and Kak, A. C., 1976. *Digital Picture Processing.* New York: Academic Press.
[Rosenfeld 86]	Rosenfeld, A., 1986. "Pyramid Algorithms for Perceptual Organization," in preparation.
[Samet and Weber 82]	Samet, H., and Weber, R. E., 1982. "On Encoding Boundaries with Quadtrees," Computer Science TR-1162, University of Maryland.
[Sedgewick 83]	Sedgewick, R., 1983. *Algorithms.* Reading, MA and London: Addison-Wesley.
[Shafrir 85]	Shafrir, A., 1985. "Fast region coloring and the computation of inside/outside relations," M.Sc. Thesis, Department of Applied Mathematics, Feinberg Graduate School, Weizmann Institute of Science, Rehevot, Israel.
[Shiloach and Vishkin 82]	Shiloach, Y., and Vishkin, U., 1982. "An O(log n) parallel connectivity algorithm," Journal of Algorithms 3 (1), March, 1982.
[Stevens 78]	Stevens, K. A., 1978. "Computation of Locally Parallel Structure," *Biological Cybernetics 29*: 19–28.
[Treisman and Gelade 80]	Treisman, A. and Gelade, G., 1980. "A feature integration theory of attention." *Cognitive Psychology*, 12: 97–136.
[Ullman 76a]	Ullman, S., 1976. "Filling in the Gaps: The Shape of Subjective Contours and a Model for their Generation," *Biological Cybernetics 25:* 1–6.
[Ullman 76b]	Ullman, S., 1976. "Relaxation and Constrained Optimization by Local Processes." *Computer Graphics and Image Processing 10:* 115–125
[Ullman 84]	Ullman, S., 1984. "Visual Routines." *Cognition*, 18 (1984): 97–159.
[Voorhees 87]	Voorhees, H. L., 1987. "Finding Texture Boundaries in Images," S.M. Thesis (forthcoming). Cambridge, MA: Dept. of Electrical Engineering and Computer Science, MIT.

[Winston and Horn 84] Winston, P. H., and Horn, B. K. P., 1984. *Lisp*. Reading, MA and London: Addison-Wesley.

[Yuille and Poggio 83] Yuille, A. L., and Poggio, T. A., 1983. "Fingerprints Theorems for Zero-crossings." Cambridge, MA: M.I.T. Artificial Intelligence Laboratory, AI-Memo 730.

[Zucker 80] Zucker, S. W., 1980. "Labeling Lines and Links: An Experiment in Cooperative Computation." Montreal, Quebec: McGill University Computer Vision and Graphics Laboratory, Technical Report 80-5.

[Zucker 82] Zucker, S. W., 1982. "Early Orientation Selection and Grouping: Type I and Type II Processes," Montreal, Quebec: McGill University Computer Vision and Graphics Laboratory, Technical Report 82-6.