Then we obtain

$$\epsilon_{IJ} = \left| \frac{1}{H^2} \sum_{ij,kl} \iint_{\square_{ij,kl} \cap \Omega_{IJ}} \Phi\big(x(\xi,\eta), y(\xi,\eta)\big) J(\xi,\eta)\, d\xi\, d\eta - \left(\frac{h_N}{H}\right)^2 \sum_{\text{Cases I\&II}} (\Phi J)\big(\xi(\overline{G}), \eta(\overline{G})\big) \right|$$

$$\le \left| \frac{1}{H^2} \sum_{\text{Case I}} \left[\iint_{\square_{ij,kl} \cap \Omega_{IJ}} \Phi J\, d\xi\, d\eta - h_N^2 (\Phi J)\big(\xi(\overline{G}), \eta(\overline{G})\big) \right] \right| + \left| \frac{1}{H^2} \sum_{\text{Case II}} \left[\iint_{\square_{ij,kl} \cap \Omega_{IJ}} \Phi J\, d\xi\, d\eta - h_N^2 (\Phi J)\big(\xi(\overline{G}), \eta(\overline{G})\big) \right] \right| + \left| \frac{1}{H^2} \sum_{\text{Case III}} \iint_{\square_{ij,kl} \cap \Omega_{IJ}} \Phi J\, d\xi\, d\eta \right|. \quad (6.15)$$

Consequently, the desired result (6.13) is obtained from Lemmas 6.1, 6.2, and

$$\left| \iint_{\square_{ij,kl} \cap \Omega_{IJ}} \Phi J\, d\xi\, d\eta \right| \le h_N^2 \left| \Phi J \right|_{0,\infty,\square_{ij,kl}}. \quad \blacksquare \quad (6.16)$$

Below let us prove Theorem 4.1 by using the above lemmas. We have from (4.5)

$$\text{Area}(S_{ij,kl}) = \iint_{\square_{ij,kl}} J(\xi,\eta)\, d\xi\, d\eta \ge \epsilon_0 h_N^2. \quad (6.17)$$

It then follows

$$h_N^2 \le \frac{1}{\epsilon_0} \text{Area}(S_{ij,kl}). \quad (6.18)$$

We obtain

$$h_N^2 \sum_{\text{Cases I\&II}} \left| \Phi J \right|_{2,\infty,\square_{ij,kl}} \le \frac{1}{\epsilon_0} \left| \Phi J \right|_{2,M,\tilde{\Omega}_{IJ}} \text{Area}(\tilde{\square}_{IJ}), \quad (6.19)$$

$$h_N^2 \sum_{\text{Cases II\&III}} \left| \Phi J \right|_{0,\infty,\square_{ij,kl}} \le \frac{1}{\epsilon_0} \left| \Phi J \right|_{0,\infty,\tilde{\Omega}_{IJ}} \text{Area}(\partial\tilde{\square}_{IJ}). \quad (6.20)$$

Moreover, we can see

$$\text{Area}(\partial\tilde{\square}_{IJ}) \le 4 d_{\max} H,\ \text{Area}(\tilde{\square}_{IJ}) = O(H^2), \quad (6.21)$$

where $d_{\max}$ is the maximal width of $\partial\tilde{\square}_{IJ}$. Since $T \in C^1$, we have

$$\text{Area}(S_{ij,kl}) = \iint_{\square_{ij,kl}} J(\xi,\eta)\, d\xi\, d\eta \le h_N^2 \left| J \right|_{0,\infty,\square_{ij,kl}}. \quad (6.22)$$

By noting the fact that the boundary layer $\partial\tilde{\Omega}_{ij}$ is made up of two or three layers of $S_{ij,kl}$, there exists a constant C independent of N, Φ, and T such that

$$d_{\max} \le C\big[\text{Area}(S_{ij,kl})\big]^{1/2} \le Ch_N \big(\left| J \right|_{0,\infty,\square_{ij,kl}}\big)^{1/2}. \quad (6.23)$$

Combining (6.19)-(6.23), we obtain

$$h_N^2 \sum_{\text{Cases I\&II}} \left| \Phi J \right|_{2,\infty,\square_{ij,kl}} \le \frac{CH^2}{\epsilon_0} \left| \Phi J \right|_{2,M,\tilde{\Omega}_{IJ}}, \quad (6.24)$$

and

$$h_N^2 \sum_{\text{Cases II\&III}} \left| \Phi J \right|_{0,\infty,\square_{ij,kl}} \le \frac{Ch_N}{\epsilon_0} H \left| \Phi J \right|_{0,\infty,\partial\tilde{\Omega}_{ij}} \big(\left| J \right|_{0,\infty,\partial\tilde{\Omega}_{ij}}\big)^{1/2} \le \frac{Ch_N H}{\epsilon_0} \left| \Phi \right|_{0,\infty,\partial\tilde{\Omega}_{IJ}} \big(\left| J \right|_{0,\infty,\partial\tilde{\Omega}_{IJ}}\big)^{3/2}. \quad (6.25)$$

The desired result (4.6) is obtained from Lemma 6.3 and the assumption $h_N = H/N$. This completes the proof of Theorem 4.1. ■

Second, we provide the proof of Corollary 4.2 in Section IV.

Proof: From (4.9) the total amount of calculation of the splitting-shooting method is:

$$O\left(\sum_{k=1}^{p} N_k^2 M\right) = O\left(\frac{1}{1-\beta_2^2} N_p^2 M\right) = O(N_p^2 M). \quad \blacksquare$$

Last, let us consider the transformation T^* (5.7). Denoting the corresponding errors ϵ_{ij}^* for T^* when using the splitting-shooting method in Section II, we then have the following.

Theorem 6.1: Let (5.7) and all conditions in Theorem 4.1 hold. Then there exist bounds

$$\epsilon_{IJ}^* \le \frac{C}{\epsilon_0} \left\{ \frac{H^2}{N^2} \left| \Phi J \right|_{2,M,\tilde{\Omega}_{IJ}^*} + \frac{\alpha}{N} \left| \Phi \right|_{0,\infty,\partial\tilde{\Omega}_{IJ}^*} \big(\left| J \right|_{0,\infty,\partial\tilde{\Omega}_{IJ}^*}\big)^{3/2} \right\}, \quad (6.26)$$

where $\tilde{\Omega}_{IJ}^* \xleftarrow{(T^*)^{-1}} \tilde{\square}_{IJ}$, $\partial\tilde{\Omega}_{IJ}^* \xleftarrow{(T^*)^{-1}} \partial\tilde{\square}_{IJ}$.

Proof: By means of (5.7), we have

$$\left| J^* \right| = \alpha^2 \left| J \right|,\ \epsilon_0^* = \alpha^2 \epsilon_0. \quad (6.27)$$

The desired result (6.26) is obtained by using Theorem 4.1 for T^*. ■

Corollary 6.1: Let all conditions in Theorem 6.1 hold. Then when N is large, there exist

$$\left| \epsilon_{IJ}^* \right| = O(\alpha/N). \quad (6.28)$$

ACKNOWLEDGMENT

We express our gratitude to the referees for their valuable suggestions.

REFERENCES

[1] R. L. Burden, J. D. Faires, and A. C. Reynolds, *Numerical Analysis*, 2nd ed. Prindle, Weber, & Schmidt, 1981.
[2] S. Lang, *Calculus of Several Variables*, 3rd ed. New York: Springer-Verlag, 1987.
[3] S. Y. Lee, S. Yalamanchili, and J. K. Aggarwal, "Parallel image normalization on a mesh connected array processor," *Pattern Recogn.*, vol. 20, pp. 115-124, 1987.
[4] Z. C. Li, T. D. Bui, Y. Y. Tang, and C. Y. Suen, *Computer Transformation of Digital Images and Patterns*. Singapore: World Scientific, 1989.
[5] G. Strang and G. J. Fix, *An Analysis of the Finite Element Method*. Englewood Cliffs, NJ: Prentice-Hall, 1973.
[6] O. C. Zienkiewicz, *The Finite Element Method*, 3rd ed. New York: McGraw-Hall, 1977.

A New Region Expansion for Quadtrees

CHUAN-HENG ANG, HANAN SAMET, AND CLIFFORD A. SHAFFER

***Abstract*—A one-pass algorithm is presented to perform region expansion in images that are represented by quadtrees. The algorithm**

Manuscript received June 27, 1988; revised January 30, 1990. Recommended for acceptance by C. R. Dyer. This work was supported by the National Science Foundation under Grant IRI-88-02457.

C.-H. Ang is with the Department of Computer Science, National University of Singapore, Singapore 0511.

H. Samet is with the Department of Computer Science, and the Institute of Advanced Computer Studies, and the Center for Automation Research, University of Maryland, College Park, MD 20742.

C. A. Shaffer is with the Department of Computer Science, Virginia Polytechnic Institute and State University, Blacksburg, VA 24061.

IEEE Log Number 9034830.

changes to BLACK those WHITE pixels within a specified distance of any BLACK node in the image. The algorithm yields a significant improvement over previous approaches by 1) reducing the number of BLACK nodes that must be considered for expansion, and 2) reducing the number of nodes that must be inserted as a result of the expansion. This is achieved by the introduction of the concepts of a merging cluster and a vertex set. Empirical tests show that the execution time of this algorithm generally decreases as the radius of expansion increases, whereas for previous approaches the execution time generally increases with the radius of expansion.

***Index Terms*—Cartography, computer-aided design, computer graphics, geographic information systems, hierarchical data structures, image dilation, polygon expansion, quadtrees, region expansion.**

I. Introduction

This correspondence addresses the efficient computation of the region expansion operation (also known as image dilation). It is useful in computer-aided design (CAD) when we want to find all objects near a cursor or near a particular set of objects. It can also be used to simplify the process of planning a collision-free path for a robot in a two-dimensional environment consisting of obstacles. In this case, a finite-width robot is treated as a point and the obstacles are expanded by an amount of equal to the size of the robot. It is also very useful in computer cartography applications where it is desirable to provide graphic answers to queries such as "find all wheatfields within five miles of the floodplain." Such an answer is computed by expanding the floodplain region in the image and intersecting the result with another image that represents the wheatfields.

In this correspondence we refer to the region expansion task as the *WITHIN* function. Given a binary image I, WITHIN generates a new image which is BLACK at all pixels within a specified radius of expansion R of the BLACK regions of I. The distance is computed by using the *chessboard distance metric* which defines the distance between points (x_1, y_1) and (x_2, y_2) as $\text{MAX}(|x_1 - x_2|, |y_1 - y_2|)$.

In order to simplify the presentation, we assume that the image is binary and is represented by a region quadtree [5], [8], [9]. The region quadtree decomposes an image into homogeneous blocks. If the image is all one color, it is represented by a single block. If not, then the image is decomposed into quadrants, subquadrants, $\cdots$, until each block is homogeneous.

When a quadtree is constructed with pointers, it is referred to as a *pointer-based quadtree*. Often the volume of data is so high that it is preferable to store the quadtree in disk files. In such a case, a pointer-based representation may require many disk pages to be accessed. Thus alternative representations such as the linear quadtree [3], [1] are used. The linear quadtree represents an image as a collection of the leaf nodes that comprise it. Each leaf node is represented by its locational code which corresponds to a sequence of directional codes that locate the leaf along a path from the root of the tree. Assuming that the origin of the coordinate system of the image is located at its upper left corner, then the locational code of a node is the same as the result of interleaving the bits that comprise the x, y coordinates of the upper left corner of the node. In addition, the depth of the node relative to the root must also be recorded. The collection of nodes making up the linear quadtree is usually stored as a list sorted in increasing order of the locational codes. Such an ordering is useful because it is the order in which the leaf nodes of the quadtree are visited by a depth-first traversal of the quadtree. This representation is employed in the QUILT system [12] which was used to test the algorithms described in this correspondence.

Section II briefly describes three alternative prior implementations of the WITHIN function, and outlines their strengths and weaknesses. Section III presents a new algorithm that overcomes the shortcomings of the methods described in Section II. Section IV analyzes the execution time of the new algorithm. Section V contains some concluding remarks.

II. Three Polygon Expansion Algorithms

The simplest region expansion algorithm, termed *WITHIN1* [10], visits each node of the quadtree. Each BLACK node, say B, is expanded by R units and the new square (of width $\text{WIDTH}(B) + 2 \cdot R$) is decomposed into quadtree blocks and inserted into the output quadtree. The execution time of WITHIN1 increases directly with R since the number of blocks in a quadtree is proportional to the total perimeter of the regions that comprise it [4]. Many of the regions expanded from the black nodes overlap each other. As a result, WITHIN1 requires many duplicate insertions and subsequent mergings of BLACK nodes. This algorithm is the quadtree analog of the traditional method used in most image processing applications [7].

A second algorithm, termed *WITHIN2* [11], works on the WHITE nodes. The BLACK nodes are simply copied into the output tree. WHITE nodes of width less than or equal to $(R + 1)/2$, as well as their brothers, are inserted into the output quadtree as BLACK nodes. For WHITE nodes of width greater than $(R + 1)/2$, WITHIN2 has to search through all the neighboring nodes to determine which portions of these large WHITE nodes that lie within radius R of a nearby BLACK node can be output as BLACK nodes. The problem with this approach is that many nodes of the input quadtree will be visited more than once in search for the nearby BLACK nodes. In addition, there are many redundant node insertions.

A third algorithm, termed *WITHIN3* [6], makes two passes over the nodes in the linear quadtree's node list to avoid making redundant insertions. The first pass processes the node list in increasing order of locational codes. For each WHITE node, say W, it converts to BLACK all portions of W that are within a distance R of a BLACK node that has been encountered at a prior position in the list. The second pass is analogous except that the list is processed in reverse order. A substantial amount of computation is required to determine and maintain the appropriate information about previously encountered BLACK nodes, and to split the WHITE nodes. Its execution time increases directly with the radius of expansion since the number of previously encountered BLACK nodes that must be examined increases.

III. The New Region Expansion Algorithm

Our new algorithm, termed WITHIN4, traverses the input quadtree in preorder and writes the result of the expansion to an output quadtree. It takes no action for large WHITE nodes. All large BLACK nodes are expanded using WITHIN1. The algorithm treats clusters of small nodes as one unit termed a merging cluster (explained below). For each merging cluster, the algorithm computes the vertex set (explained below) and performs a node expansion only when enough information has been collected.

The keys to our new algorithm are the concepts of a *merging cluster* and a *vertex set*. These concepts allow us to consider a collection of BLACK nodes for expansion instead of expanding each BLACK node individually. As a result, WITHIN4 is able to reduce the number of input BLACK nodes that must be considered for expansion, and to reduce the number of BLACK nodes to be output. These reductions yield an algorithm whose execution time is asymptotically unaffected by the magnitude of the radius of expansion (see [2]).

A. Merging Cluster

Given an input quadtree, and a radius of expansion R let us examine a subtree rooted at an internal node M which represents the largest block whose width is less than or equal to $R + 1$. Each WHITE leaf node, say W, in this subtree is within R pixels of a BLACK leaf node in this subtree and therefore W will be changed to BLACK after expansion. Of course, M becomes a BLACK node as a result of the expansion.

Define $w(R)$ to be the largest integer that is a power of 2 and is less than or equal to $R + 1$, i.e., $w(R) = 2^r \leq R + 1 < 2^{r+1}$ for $r \geq 0$. If M is a nonleaf node whose corresponding block is of width $w(R)$, then M is a *merging cluster* of width $w(R)$. Merging

cluster M consists of the set of leaf nodes in the subtree rooted at M. In the rest of this correspondence, whenever we speak of the width of a node, we mean the width of the node's corresponding block.

Fig. 1 is an example of a merging cluster with 13 leaf nodes when $R = 3$. In this case, $w(R)$ is 4. In Fig. 1, A, B, C, and D are the BLACK nodes in the merging cluster. The corners of their blocks, termed *vertices*, are designated by using the corresponding lower case letter with a subscript that designates the vertex. For example, a_{SW} is the SW vertex of node A.

B. Computing the Vertex Set

Consider the expansion of the merging cluster in Fig. 1 by 3 pixels. The result is given in Fig. 2 from which we make the following observations:

1) The node corresponding to the root of the merging cluster is now a BLACK node.

2) The area expanded beyond the boundary of the merging cluster in the vertical and horizontal directions always forms a rectangle. The size of each rectangle is determined by the distance between the boundary and the nearest BLACK node within the merging cluster.

3) The area of expansion of the merging cluster in a diagonal direction always forms a staircase-like region. The extreme points of the staircase are those points which can be obtained by the translation of the vertices of some of the BLACK nodes in the merging cluster.

From this example, we discover that the expansion from the merging cluster is totally determined by $VS = \{a_{NW}, b_{NW}, c_{NW}, c_{SW}, d_{SE}, d_{NE}, a_{NE}\}$. The set VS is termed the *vertex set* of the merging cluster. The purpose of using the vertex set is to minimize the number of BLACK elements of the merging cluster requiring expansion.

Let d be in {NW, NE, SW, SE}. Let OPQUAD(d) denote the vertex direction opposite to d (e.g., OPQUAD(NW) = SE). The *vertex set* (VS) of a merging cluster M is defined to be the union of four vertex subsets VS_d. Given BLACK node P in M, vertex v of P is in VS_d if v is the d vertex of P and v is not in the closed OPQUAD(d) quadrant of any vertex of another BLACK node in M.

From the definition, it is clear that the four vertex subsets are disjoint. This means that they can be constructed independently or even in parallel. Table I shows the changes to each vertex subset when the BLACK nodes of the merging cluster in Fig. 1 are processed in the order A, B, C, D.

It is easy to see that no two vertices in VS_{d_1} and VS_{d_2} can have the same x coordinate value where d_1 and d_2 are NW and NE, or SW and SE. As a result, we have:

Corollary: The size of the union of any two horizontally (or vertically) adjacent vertex subsets VS_{d_1} and VS_{d_2} is less than or equal to $w(R) + 1$, and this bound is attainable. □

Within a merging cluster, if there is only one black node P, then P contributes four vertices to the vertex set. Otherwise, at most two black nodes can contribute three vertices apiece to the vertex set. All other black nodes cannot contribute more than two vertices to the vertex set. Therefore, the number of the vertices in a vertex set is bounded by 2 plus twice the number of BLACK nodes in M. Thus we have proved the following theorem.

Theorem: The size of the vertex set of meging cluster M is bounded by the minimum of $2 \cdot w(R) + 2$ and 2 + twice the number of BLACK nodes in M. The bound is attainable. □

C. Merging Cluster Expansion

The expansion of a merging cluster M in the eight directions can be decomposed into two groups, namely those that deal with directions {NW, NE, SW, SE} and those that deal with directions {N, W, S, E}. We shall describe one expansion from each group.

To expand in the NW direction from a merging cluster, only the elements of VS_{NW} need to be considered. The result of the expansion is a staircase-shaped region formed in the NW direction with the steps of the staircase marked by the vertices in VS_{NW} which have been translated by $(-R, -R)$; i.e., they are obtained by subtracting R from the coordinates of each vertex in VS_{NW}. To insert all the nodes that are components of the staircase, we find the smallest quadtree block that covers the staircase and begin the regular decomposition. Blocks which are completely within (or outside) the staircae are inserted as BLACK (or WHITE). Blocks which partially overlap the staircase are decomposed into four equal-sized blocks which are processed recursively.

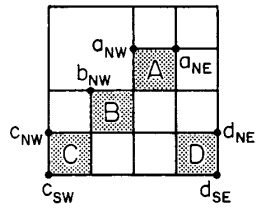


Fig. 1. Example of a merging cluster.

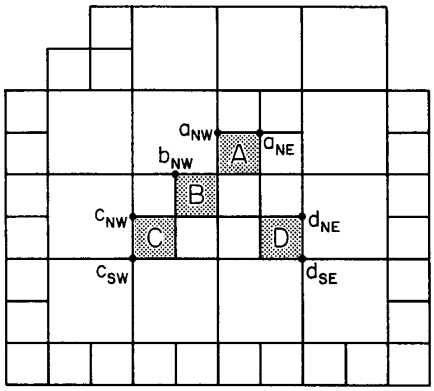


Fig. 2. Result of expanding the merging cluster in Fig. 1 by 3 pixels.

TABLE I
BUILDING VERTEX SUBSETS FOR FIG. 1

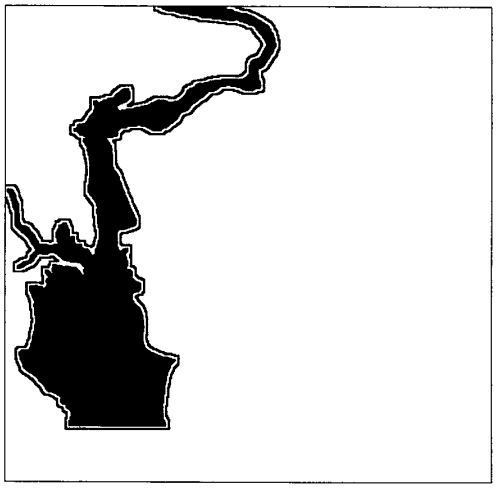
node	VS_{NW}	VS_{NE}	VS_{SW}	VS_{SE}
A	$\{a_{NW}\}$	$\{a_{NE}\}$	$\{a_{SW}\}$	$\{a_{SE}\}$
B	$\{a_{NW}, b_{NW}\}$	$\{a_{NE}\}$	$\{b_{SW}\}$	$\{a_{SE}, b_{SE}\}$
C	$\{a_{NW}, b_{NW}, c_{NW}\}$	$\{a_{NE}\}$	$\{c_{SW}\}$	$\{a_{SE}, b_{SE}, c_{SE}\}$
D	$\{a_{NW}, b_{NW}, c_{NW}\}$	$\{a_{NE}, d_{NE}\}$	$\{c_{SW}\}$	$\{d_{SE}\}$

To expand in the W direction, we use the vertex v in VS_{NW} (or VS_{SW}) which is closest to the western boundary of the merging cluster. The result of the expansion is a rectangle T with height $w(R)$ and width $R - d_W$ where d_W where d_W is the distance from v to the W edge of M.

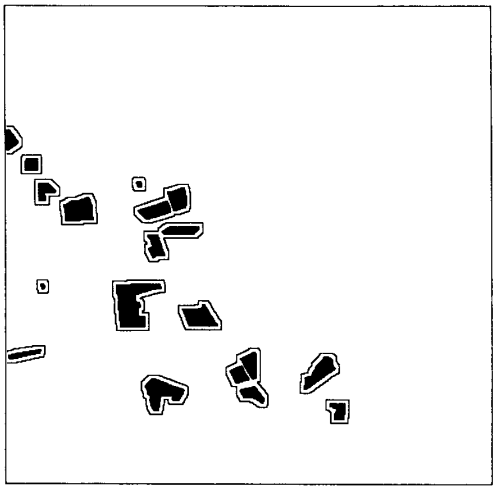
A further reduction in processing time can be achieved by taking advantage of the interactions between the blocks of neighboring merging clusters. If the d neighbor of the merging cluster is a big black node or a merging cluster where d is one of the eight directions, then the expansion towards the d direction need not be performed.

IV. ANALYSIS

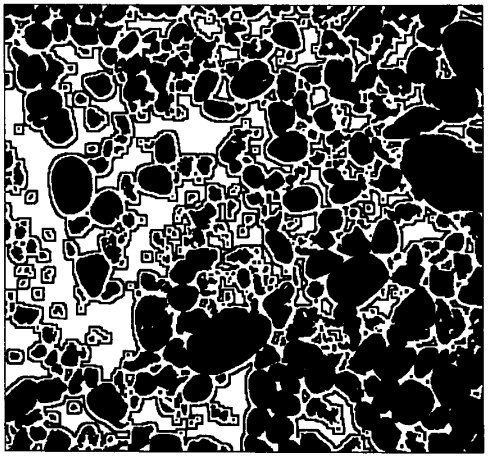
The four WITHIN algorithms were tested on three 512 × 512 images named Center, Acc, and Pebble, shown in Fig. 3(a), (b), and (c), respectively. These figures also show the result of expanding them by 4 pixels. The natural logarithm of their execution times is plotted in Figs. 4(a), (b), and (c) as a function of the radius of expansion (i.e., R). Notice that the execution times for even values of R are generally smaller than those for values of $R - 1$



(a)

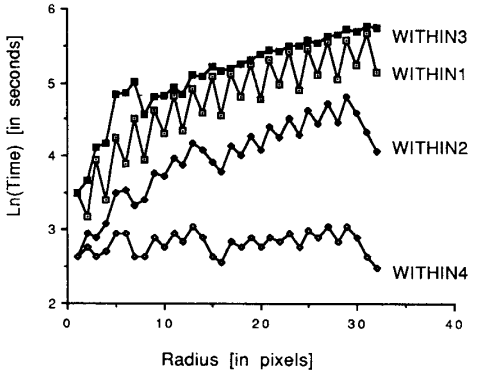


(b)

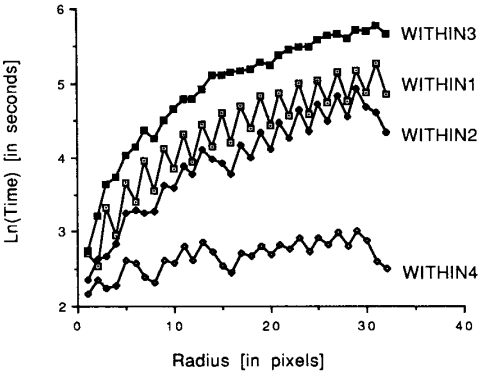


(c)

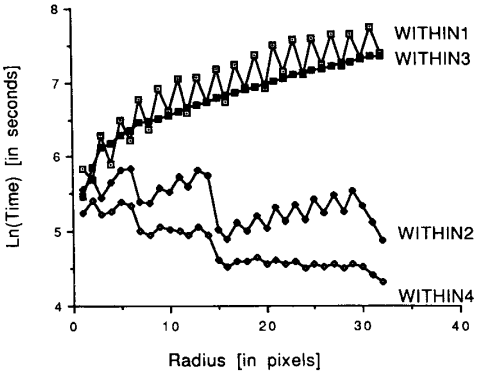
Fig. 3. Three images: (a) center, (b) acc, (c) pebble. All regions are expanded by 4 pixels.



(a)



(b)



(c)

Fig. 4. Execution times for expanding the three images: (a) center, (b) acc, (c) pebble.

and $R + 1$ (which are odd) due to the effects of node aggregation, which means that fewer blocks need to be inserted into the output quadtree.

As Fig. 4 demonstrates, the execution time of WITHIN4 generally increases at a much slower rate than the other algorithms. In fact, it decreases as R gets sufficiently large. It can be shown that the execution time of WITHIN4 is asymptotically independent of R [2]. This can be seen by observing that as R increases the merging clusters get bigger and there are fewer of them. On the other hand, the complexity of the vertex set usually increases as the size of the merging cluster increases. These two effects tend to cancel each other out; hence accounting for the relative indpendence of R.

Nevertheless, the radius of expansion does have an effect on the execution time. As R increases from $2^r - 2$ to $2^r - 1$, $w(R)$ is doubled, thereby reducing the number of merging clusters. In particular, this means that at most four merging clusters of size 2^{r-1} are merged into one merging cluster of size 2^r, or equivalently, four node insertions are being replaced by one. Considering the four merging clusters together, at most eight expansions in the directions of a corner can be avoided depending on whether the surrounding nodes are WHITE or not. All of these savings lead to a reduction in the execution time.

When R is between $2^r - 1$ and $2^{r+1} - 2$, $w(R)$ is constant (i.e., 2^r) and the execution time increases slowly as R increases since more nodes will be inserted. The execution times for odd values of R show a cyclical behavior which is characterized by a slow increase as R increases from 2^{r-1} to $2^{r+1} - 2$, followed by a drop back to nearly the lowest execution time for $2^{r+1} - 1$. The behavior for even values of R follows the same pattern.

V. Concluding Remarks

In this correspondence we have described and compared four region expansion algorithms (i.e., WITHIN1, WITHIN2, WITHIN3, and WITHIN4). The execution times of WITHIN1 and WITHIN3 increase as R increases and hence they are less attractive. WITHIN4 is more efficient than WITHIN2 for the following reasons:

1) Every BLACK node in a merging cluster is individually inserted by WITHIN2, whereas only one insertion is needed for WITHIN4.

2) A node can be inserted by WITHIN4 at most eight times since there are at most eight neighboring merging clusters, whereas for WITHIN2, the number of repeated insertions of a node increases as R increases.

Although WITHIN4 outperforms the other three WITHIN algorithms, it still makes many redundant insertions. By using a more efficient algorithm to compute the vertex sets, as well as to expand from them, the performance of WITHIN4 can be further improved.

At this point, we must still ask if all this effort is really worth the trouble. A good yardstick for measuring the performance of our algorithm is to compare it with an approach that would convert the quadtree to an array, perform the array WITHIN algorithm, and then rebuild the quadtree. Using the QUILT system building the quadtrees for the center and pebble images took 16 and 110 seconds, respectively [11]. Of course, we must still perform the region expansion operation. For each of these images, using WITHIN4 was faster whereas WITHIN1, WITHIN2, and WITHIN3 were all considerably slower, Thus we see that WITHIN4 is indeed worth the trouble.

References

[1] D. J. Abel and J. L. Smith "A data structure and algorithm based on a linear key for a rectangle retrieval problem," *Comput. Vision, Graphics, Image Processing*, vol. 24, no. 1, pp. 1-13, Oct. 1983.

[2] C. H. Ang, "Analysis and applications of hierarchical data structures," Ph.D. dissertation, Dep. Comput. Sci., Univ. Maryland, College Park, Tech. Rep. TR-2255, June 1989.

[3] I. Gargantini, "An effective way to represent quadtrees," *Commun. ACM*, vol. 25, no. 12, pp. 905-910, Dec. 1982.

[4] G. M. Hunter, "Efficient computation and data structures for graphics," Ph.D. dissertation, Dep. Elec. Eng. Comput. Sci., Princeton Univ., Princeton, NJ, 1978.

[5] A. Klinger, "Patterns and search statistics," in *Optimizing Methods in Statistics*, J. S. Rustagi, Ed. New York: Academic, 1971, pp. 303-337.

[6] D. C. Mason, "Dilation algorithm for a linear quadtree," *Image Vision Comput.*, vol. 5, no. 1, pp. 11-20, Feb. 1987.

[7] A. Rosenfeld and A. C. Kak, *Digital Picture Processing*, 2nd ed. New York: 1982.

[8] H. Samet, *The Design and Analysis of Spatial Data Structures*. Reading, MA: Addison-Wesley, 1990.

[9] ——, *Applications of Spatial Data Structures: Computer Graphics, Image Processing, and GIS*. Reading, MA: Addison-Wesley, 1990.

[10] H. Samet, A. Rosenfeld, C. A. Shaffer, R. C. Nelson, and Y. G. Huang, "Application of hierarchical data structures to geographical information systems, Phase III," Dep. Comput. Sci., Univ. Maryland, College Park, Tech. Rep. TR-1457, Nov. 1984.

[11] C. A. Shaffer and H. Samet, "An algorithm to expand regions represented by linear quadtrees," *Image Vision Comput.*, vol. 6, no. 3, pp. 162-168, Aug. 1988.

[12] C. A. Shaffer, H. Samet, and R. C. Nelson, "QUILT: A geographic information system based on quadtrees," Dep. Comput. Sci., Univ. Maryland, College Park, Tech. Rep. TR-1885, July 1987.

A Spatial Sampling Criterion for Sonar Obstacle Detection

ROMAN KUC

***Abstract*—This correspondence describes a spatial sampling criterion for sonar systems that allows all obstacles within a given radius from the sensor to be detected. The environment considered is a two-dimensional floor plan that is extended into the third dimension, in which the scanning is performed in the horizontal plane. In this environment, edge-like reflectors, such as edges of doors or doorways, and oblique surfaces are the most difficult to detect. By considering the physics of sound propagation, we determine the sonar scanning density required to detect these objects. An experimental verification is included. The limitations of detecting objects with sonar in a more general environment are discussed. These results can used to determine the necessary spacing in a transducer ring array and the maximum step size that a mobile robot can translate without danger of collision.**

***Index Terms*—Acoustics, intelligent sensors, map building, obstacle avoidance, robot navigation, sensors, signal processing, sonar, time-of-flight ranging.**

I. Introduction

Acoustic sensors provide an inexpensive means for determining the proximity of objects and have shown utility for implementing sonar systems for robot navigation [1]-[3]. One of the most popular is the *time-of-flight* (TOF) system implemented by Polaroid [4]. However, problems arise in the straightforward, but naive, interpretation of TOF readings: objects that are present are not always detected and range readings produced by the TOF system do not always correspond to objects at that range [5]-[7]. Because of these problems, many researchers abandon sonar-only navigation systems and include additional sensing systems, such as collision detectors [8], [9] and vision systems [9]-[11]. For some applications, we feel that adequate understanding of sonar echo production will allow obstacle avoidance schemes to be accomplished with sonar only.

This correspondence describes a spatial sampling criterion that indicates the proper procedure to interrogate the environment to detect any obstacles within the vicinity of the sensor. The environment to be considered is a general two-dimensional floor plan that

Manuscript received July 22, 1988; revised August 24, 1989. Recommended for acceptance by R. De Mori. This work was supported by the Yale Science and Engineering Association and by the National Science Foundation under Grant ECS-8802627.

The author is with the Intelligent Sensors Laboratory, Department of Electrical Engineering, Yale University, New Haven, CT 06520.

IEEE Log Number 8933760.