QUADTREE WITH QUADRATIC STORAGE

Hyun-Chan Lee
Tony C. Woo

Department of Industrial & Operations Engineering
The University of Michigan
Ann Arbor, Michigan 48109-2117

Technical Report 86-29

March 1987

Quadtree with Quadratic Storage

Hyun-Chan Lee
Tony C. Woo

Department of Industrial and Operations Engineering
The University of Michigan
Ann Arbor, Michigan 48109-2117

March 1987

<Abbreviated Title>

Quadratic Quadtree

<Mailing Address>

Tony C. Woo

Department of Industrial and Operations Engineering
The University of Michigan
Ann Arbor, Michigan 48109-2117

<Key Phrases>

Quadtree, Data Structure, Representation Method, Space Efficiency, Pattern Recognition, Computer Graphics, Algorithm.

Abstract

In addition to being an approximation, existing quadtree methods require a large storage. We develop a new finite branching quadtree method which reduces the storage requirements from exponential to polynomial and guarantees an exact representation of the original object. These are made possible by adopting a new set of termination conditions that ensure finiteness of the quadtree during the subdivision.

This new data structure is analysed theoretically and tested empirically. For space complexity, we analyse its best case, worst case, and average case. Given an n-gon, we show the expected number of nodes in our quadtree is $O(n^2)$. For time complexity, we again analyse the best, worst, and average cases for constructing such a quadtree and find the average to be $O(n^2)$. Finally, random n-gons are generated as test data. We find the regression equation for the number of nodes in a quadratic to be $N = 0.136n^2 + 2.148n + 15.461$. For construction time CT, we find $CT = 1.147N - 18.535$.

List of Symbols

CT : Construction time of a quadtree.

DEN : Double-edge node.

e : An edge.

E[·] : Expected value.

f(·) : Probability distribution function.

G[·] : T[·] - S[·].

H : Height of a quadtree.

HN : Homogeneous node.

K_0, K_1 : Convex sets in 2D, quadrants.

L : Length of a line segment.

MEN : Multi-edge node.

n : Total number of edges in a polygon.

N : Total number of nodes in a quadtree.

NTN : Non-terminal node.

P_0, P_1 : Perimeter of convex set K_0 or K_1.

Pr{·} : Probability.

S[·] : Average space complexity.

SEN : Single-edge node.

T[·] : Time complexity.

U,W : Random variables.

v : A vertex.

$[X_1, X_2)$: X domain of a quadrant ranging from X_1 to X_2.

$[Y_1, Y_2)$: Y domain of a quadrant ranging from Y_1 to Y_2.

1. Introduction

The quadtree method is widely used in image processing and pattern recognition [2,4,5,8,10,11,12,15,18,19,20]. Its popularity is understandable when one considers its elegance. To obtain a quadtree for a 2D object, one starts with a quadrant which contains the entire object. If the quadrant under consideration is completely within or completely outside the object, it is left alone. Otherwise, the quadrant is subdivided into four quadrants each of which is treated similarly. The subdivision continues recursively until all the quadrants are completely within or completely outside the object, or a quadrant of some pre-specified size is reached.

Some terms related to a quadtree are helpful. A node in a quadtree is a quadrant in the Euclidean space E^2. When a quadrant is completely within a given object it is a black node. A quadrant which is completely outside the object is a white node. Because of the homogeneity of their colors within a quadrant, black and white nodes will be referred to as homogeneous nodes. A quadrant which is subdivided into four quadrants is a non-terminal node. A quadrant of the minimal allowed size is a minimum-resolution quadrant. A quadrant which reaches the size of minimum-resolution and is neither a black node nor a white node is a don't-care node. In a quadtree, homogeneous nodes and don't-care nodes are terminal nodes. The number of subdivisions to reach a minimum-resolution quadrant is the height of a quadtree. The condition which prevents a quadrant from further subdivisions is a termination condition for constructing a quadtree. We will call this kind of quadtree construction and representation method the Binary

method.

A major advantage of the Binary method is the simplicity of the Boolean operations on a quadrant: intersection, union, difference, and complement. The intersection operation may be used for collision detection. The union, difference, and complement operations are commonly used in CAD for shape description. All Boolean operations on a quadrant can be performed in constant time. However, a Binary quadtree has a maximum of $(4^{(H+1)}-1)/3$ nodes, where H is the height of the quadtree. Thus, a major disadvantage of the Binary method is the exponential data storage requirement. Because of the sheer volume of data to compute on, even if the operations on a quadrant can be done in constant time, processing on a Binary quadtree-encoded object is not necessarily fast overall.

To reduce the storage requirement, there are two approaches. The first approach is to develop a more storage-efficient data structure while maintaining the same termination conditions. Instead of using a tree structure with homogeneous nodes, an array is used to store black nodes only [3,7,19]. The elements of the array are represented by quaternary integers. This method reduces the storage by as much as the total storage for non-terminal nodes and white nodes. The number of non-terminal nodes in a quadtree can be estimated by (N-1)/4, where N is the total number of nodes in a quadtree in the Binary method. If we assume that the number of black nodes and the number of white nodes are the same on the average, the expected number of black nodes is (3N+1)/8. Therefore, reduction in the number of nodes over the Binary method is about 62%. The second approach is to reduce the storage requirements by

introducing more conditions to terminate the subdivision before a quadrant becomes homogeneous. To do so, edge and gray nodes are used as terminal nodes in addition to black and white nodes. An edge node is a quadrant which includes one edge. A gray node is a minimum-resolution quadrant which may include more than one edge. A reduction of 55%-65% in the number of nodes over that from the Binary method is possible. This is supported by experiments involving a polygon of 20 edges and the height of a quadtree is restricted to eleven [2]. Note that minimum-resolution is invoked as a terminal condition in this approach as well.

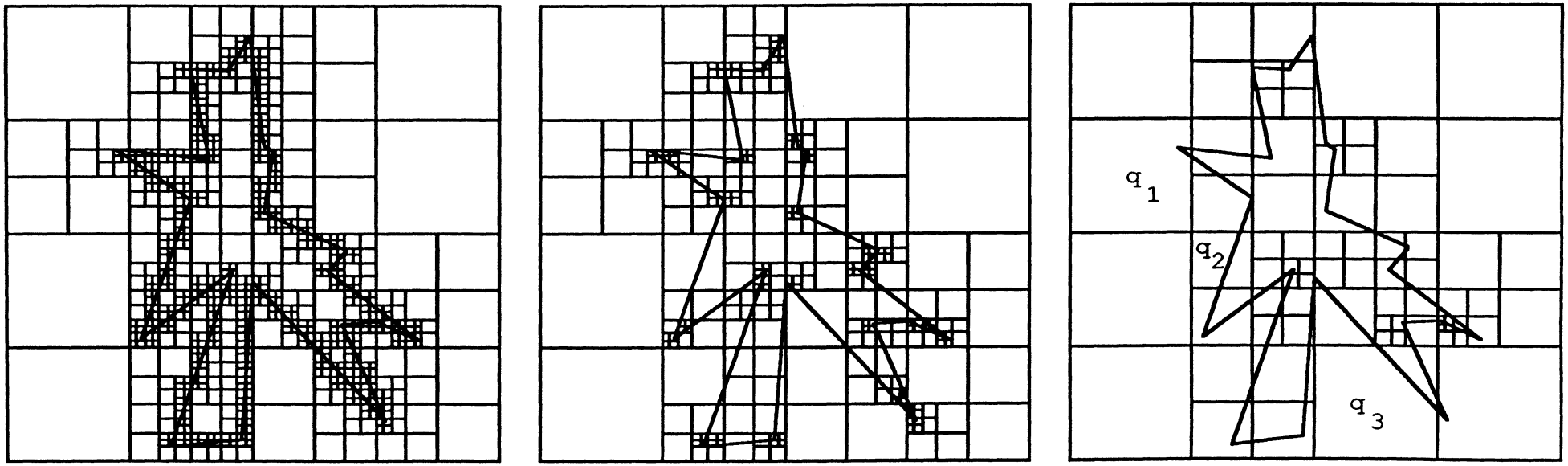
While storage is still a concern, another disadvantage of the existing methods is that a quadtree is inherently an approximation. The inexactness of representation and the large storage requirement of the existing quadtree methods come from the minimum-resolution termination condition (without which, the existing methods will not terminate. Because, on the boundary or at the vertices, the subdivision would continue forever.) To overcome these two disadvantages - storage and approximation, we propose a new finite branching quadtree representation. In our method, we use a new set of termination conditions to ensure finiteness without using the notion of minimum-resolution. As a result, we achieve the goals of compactness in storage and exactness in representation.

To visualize the effectiveness of the new termination conditions, a comparison of methods is illustrated in Figure 1. In the figure, quadtrees generated by the Binary method, by the Ayala, Brunet, Juan, and Navazo (ABJN) method [2], and by the new method are illustrated with a randomly generated polygon of 20 edges. To simulate finite branching

in the first two methods, we set the height to 6 which is the height of the quadtree in Figure 1(c). For the Binary method, the quadrants are "clustered" on the boundary of the polygon. For the ABJN method, they are clustered at the vertices. By comparison, the quadrants by the new method are more "balanced" and "sparse" than these two methods.

<Insert Figure 1>

In Section 2, the new quadtree representation method is explained along with the data structure and the algorithm for constructing the quadtree. In Section 3, the space complexity of a quadtree by the new method is analyzed. In Section 4, the time complexity of the algorithm for constructing such a quadtree is analyzed. In Section 5, experimental results are shown.



(a) Binary method

(b) ABJN method

(c) New method

Figure 1. Comparison of Quadtrees

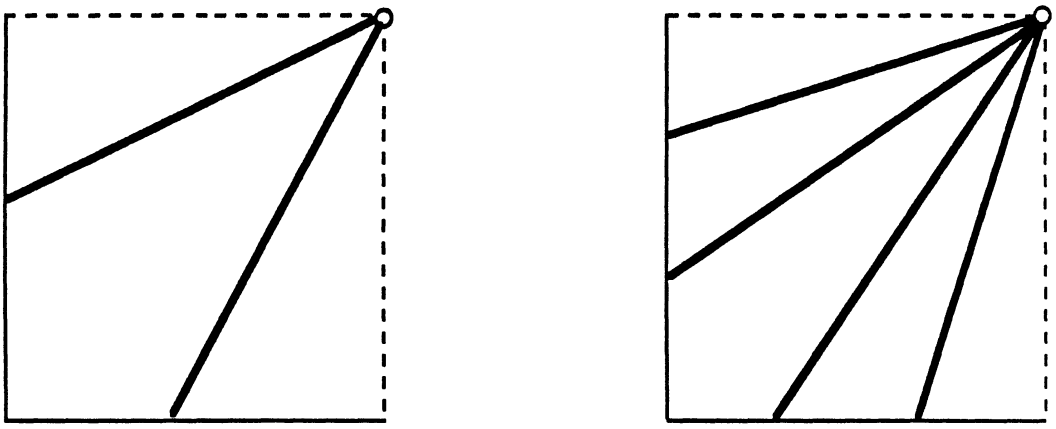
2. Finite Branching Quadtree

A quadrant is a subset of the Euclidean space E^2. The domain of a quadrant is defined as $[X_1,X_2) \times [Y_1,Y_2) \subset E^2$, where X_1 and X_2 are the minimum and maximum X-coordinates of the quadrant, respectively; similarly, for the Y-coordinates. Note that we use an open set to include only one half of the boundary of the quadrant as the domain to avoid overlap.

To terminate the subdivision without minimum-resolution, we provide a condition that ensures finite subdivision on the boundary and at the vertices. We allow a quadrant to contain two edges that intersect at a vertex. (Refer to quadrant q_1 in Figure 1(c).) Two other types of terminal nodes are needed in addition to black and white nodes. One is a single-edge node, which corresponds to a quadrant that contains one edge. (The quadrant q_2 in Figure 1(c) is an example.) The other is a double-edge node, which corresponds to a quadrant that contains two edges. (See q_3 and q_1 in Figure 1(c).) We also allow the special case when a vertex is on the boundary of a quadrant that does not belong to the domain of the quadrant. This is illustrated in Figure 2(a). Our double-edge node ensures the finite termination of subdivision without minimum-resolution. In short, the termination condition for the finite quadtree of a simple polygon is to divide the 2D Euclidean space into quadrants until a quadrant contains a maximum of two edges.

<Insert Figure 2>

The termination conditions for a non-simple polygon are different from those for simple polygons as described in the preceding paragraph.



(a) Double-edge node

(b) Multi-edge node

Figure 2. A Vertex on the Boundary

A polygon is simple if no two non-consecutive edges share a vertex. Otherwise, the polygon is non-simple [14]. The need for representing a non-simple polygon comes from projection and hidden-line removal in computer graphics. If we represent a projected object only by its boundary (a simple polygon) as illustrated in Figure 3, we do not have a representation of the inside. Even in two dimensions, we need to consider quadtrees for simple and for non-simple objects.

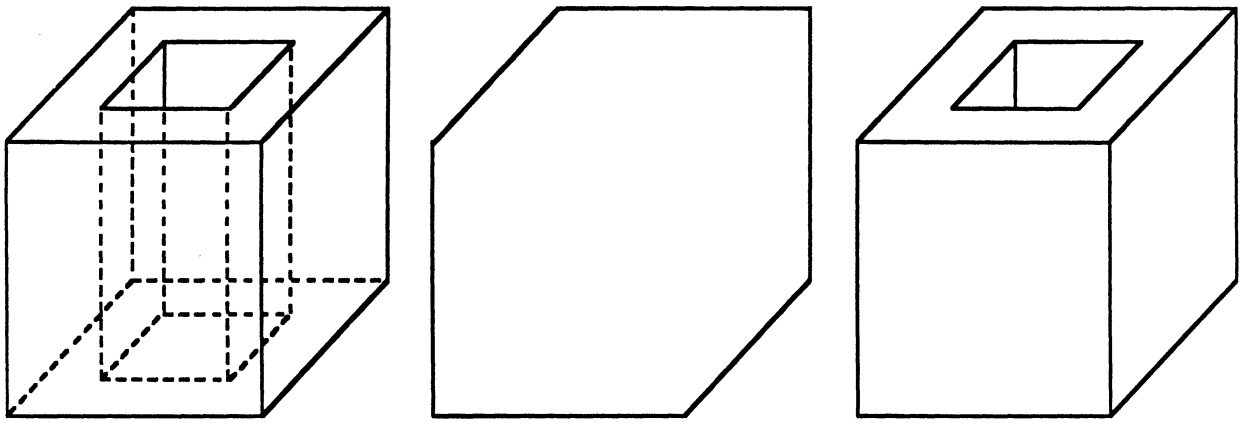
<Insert Figure 3>

For a non-simple polygon, we need to add one more termination condition since there can be any number of edges incident to a vertex. Subdivision is terminated if a quadrant contains a set of edges that are incident to one vertex which is inside or on the boundary of the quadrant. See Figure 2(b). Such a quadrant corresponds to a multi-edge node in a quadtree.

We summarize by comparing the terminal nodes of three methods: the Binary method, the ABJN method, and our new method, in Figure 4.

<Insert Figure 4>

The data structure for a quadtree is given in Table 1, which is described in Pascal. To indicate the inside of a simple polygon, we use the edge equation in a form of $AX + BY + C > 0$ by storing the coefficients A, B, and C. For a non-simple projected object, we record the number of faces of the 3D object to be projected in addition to the edge equations.



(a) 3D object

(b) Projected boundary (a simple polygon)

(c) Projected 2D object (a non-simple polygon)

Figure 3. Projection of a 3D Object onto 2D Screen

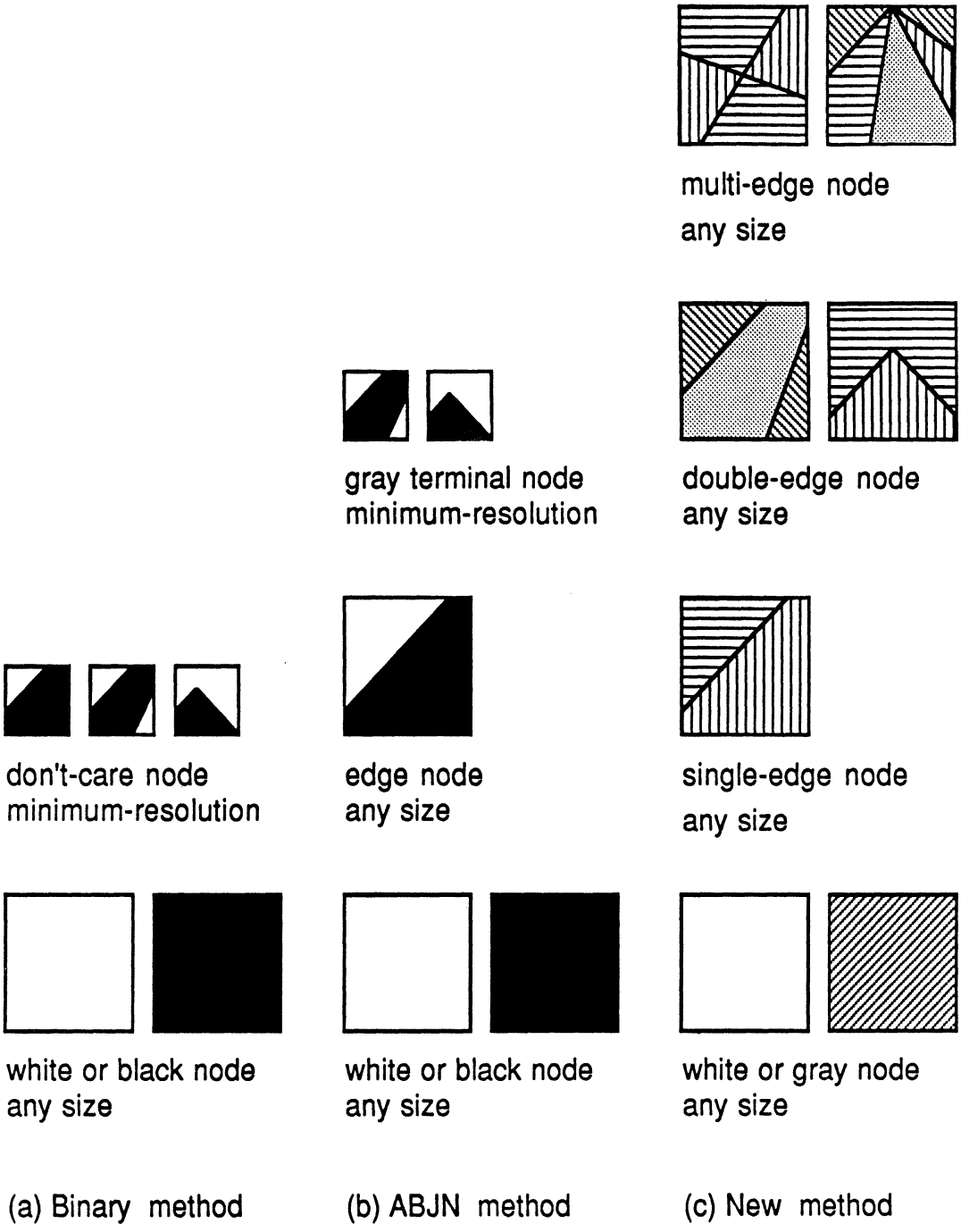


Figure 4. Comparison of Terminal Nodes

<Insert Table 1>

The algorithm for constructing a quadtree using the four types of termination conditions in Figure 4(c) and the data structure in Table 1 is given as Algorithm Quadtree_Construction. In the algorithm, the procedure Check_Type examines Edges, an array containing the number of edges in the parent node of the quadrant currently under consideration, and decides the Type for the current quadrant based on the number of edges, Num_edges, in it. The procedure Subtree(Q) creates four children for the non-terminal node Q. The four children LB, RB, LT, and RT represent the left-bottom, right-bottom, left-top, and right-top quadrants, respectively. In this algorithm we omit the process of identifying whether a homogeneous node HN is a black or white node. (This can be done by traversing the constructed quadtree once. Therefore, it does not effect the time complexity of the construction algorithm.)

<Insert Algorithm Quadtree_Construction>

```
Type Node_Set      = (HN,SEN,DEN,MEN,NTN);

     Vertex_Record = Record
                        X_Co : Real ;
                        Y_Co : Real ;
                     End ;

     Vertex_Array  = Array[1..Tot_Vertices] of Vertex_Record ;

     Edge_Record   = Record
                        First_Vertex  : Integer ;
                        Second_Vertex : Integer ;
                        A,B,C         : Real    ;
                        { coefficients of AX + BY + C > 0 }
                     End ;

     Edge_Array    = Array[1..Tot_Edges] of Edge_Record ;

     Edge_List     = Array[1..Tot_Edges] of Integer     ;

     Node_PTR      = ↑Node_Record ;

     Node_Record   = Record
                        Parent : Node_PTR ;
                        Case Node_Type : Node_Set of
                           HN  : { homogeneous node  }
                                 ( Leaf        : Boolean   );
                           SEN : { single-edge node  }
                                 ( EG          : Integer   );
                           DEN : { double-edge node  }
                                 ( E1,E2       : Integer   );
                           MEN : { multi-edge node   }
                                 ( ESET        : Edge_List );
                           NTN : { non-terminal node }
                                 ( LB,RB,LT,RT : Node_PTR  );
                     End ;
```

Table 1. Data Structure for a Quadtree

```
Algorithm Quadtree_Construction(Q,Edges,Num_Edges,X,Y,Scale)

Begin

   Check_Type(Edges,Num_Edges,X,Y,Scale,Type)

   Case Type of

      HN  : Q↑.Node_Type <-- HN

      SEN : Q↑.Node_Type <-- SEN
            Q↑.EG <-- Edges[1]

      DEN : Q↑.Node_Type <-- DEN
            Q↑.E1 <-- Edges[1]
            Q↑.E2 <-- Edges[2]

      MEN : Q↑.Node_Type <-- MEN
            For i <-- 1 to Num_Edges Do
               Q↑.ESET[i] <-- Edges[i]

      NTN : Q↑.Node_Type <-- NTN
            Scale <-- Scale / 2
            Subtree(Q)
            Quadtree_Construction(Q↑.LB,Edges,Num_Edges,X,Y,Scale)
            X <-- X + Scale
            Quadtree_Construction(Q↑.RB,Edges,Num_Edges,X,Y,Scale)
            X <-- X - Scale
            Y <-- Y + Scale
            Quadtree_Construction(Q↑.LT,Edges,Num_Edges,X,Y,Scale)
            X <-- X + Scale
            Quadtree_Construction(Q↑.RT,Edges,Num_Edges,X,Y,Scale)
   End { Case }

End { Algorithm }
```

3. Analysis of Space Complexity

In this section, based on the new termination conditions we analyze the space complexity. The analysis consists of three cases: the best case, the worst case, and the average case. We use the symbol N for the total number of nodes in a quadtree corresponding to a given polygon with n edges.

The best case, which is shown in Figure 5(a), occurs when every terminal node is a double-edge node. If H is the height of a quadtree, then in the best case H is $\log_4 n$. The total number of nodes N thus is:

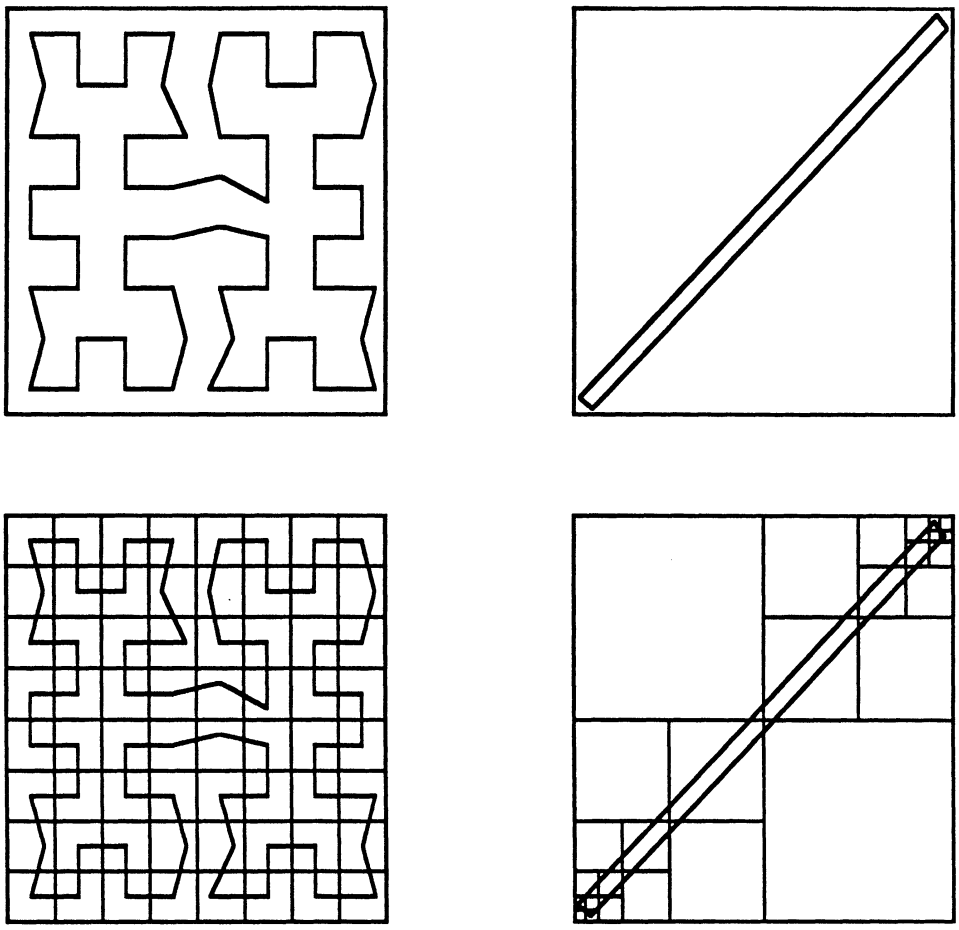
$$N = \sum_{i=0}^{H} 4^i = (4 \cdot 4^H - 1)/3 = 4n/3 - 1/3.$$

Therefore, the best case space complexity of a quadtree is linear in n, the total number of edges of a polygon.

<Insert Figure 5>

The worst case happens when two vertices (or one vertex and a non-incident edge) are very close. This is shown in Figure 5(b). If the edges at the lower-left corner of the figure and those at the upper-right corner become shorter and closer to their respective corners, more and more nodes will be needed. In this worst case, the number of nodes of a quadtree would not depend on the number of edges n. Instead, it depends on the minimum distance between a vertex and an edge to which it is not incident. The following lemma gives us an idea about the maximum possible height of the quadtree.

Lemma 1 The maximum possible height of a quadtree is the smallest



(a) The best case (b) The worst case

Figure 5. The Best and the Worst Case of Space Complexity

integer which is greater than:

$$\log_2(\text{max\_size} / \text{min\_distance}) + 3/2$$

where max_size is the size of the largest quadrant and min_distance is the minimum among the distances from a vertex to non-incident edges.

[Proof] The size s of the smallest quadrant is at most min_distance / $2\sqrt{2}$. When a vertex v and an edge e, which determine min_distance, lie diagonally in a quadrant, the size of the quadrant is min_distance / $\sqrt{2}$. This situation is illustrated in Figure 6(a). The size of the smallest quadrant is one half of the value since we need one more subdivision as shown in Figure 6(b). Therefore, the maximum possible height of a quadtree generated by the new method is the smallest integer which is greater than:

$$\log_2(\text{max\_size} / (\text{min\_distance} / 2\sqrt{2}))$$

$$= \log_2(\text{max\_size} / \text{min\_distance}) + 3/2$$

where max_size is the size of the quadrant containing the entire object as shown in Figure 6(c). ■

<Insert Figure 6>

The reason we say maximum possible height is that even when a vertex is close to an edge or when an edge is short, the maximum height may be less than that stated in Lemma 1. The situation is illustrated in Figure 6(d). The quadtree in Figure 6(c) has maximum possible height of 5. The same polygon translated by a small amount yields a quadtree of height 2, which is less than the maximum possible height.

For the average case analysis of space complexity of a quadtree, we

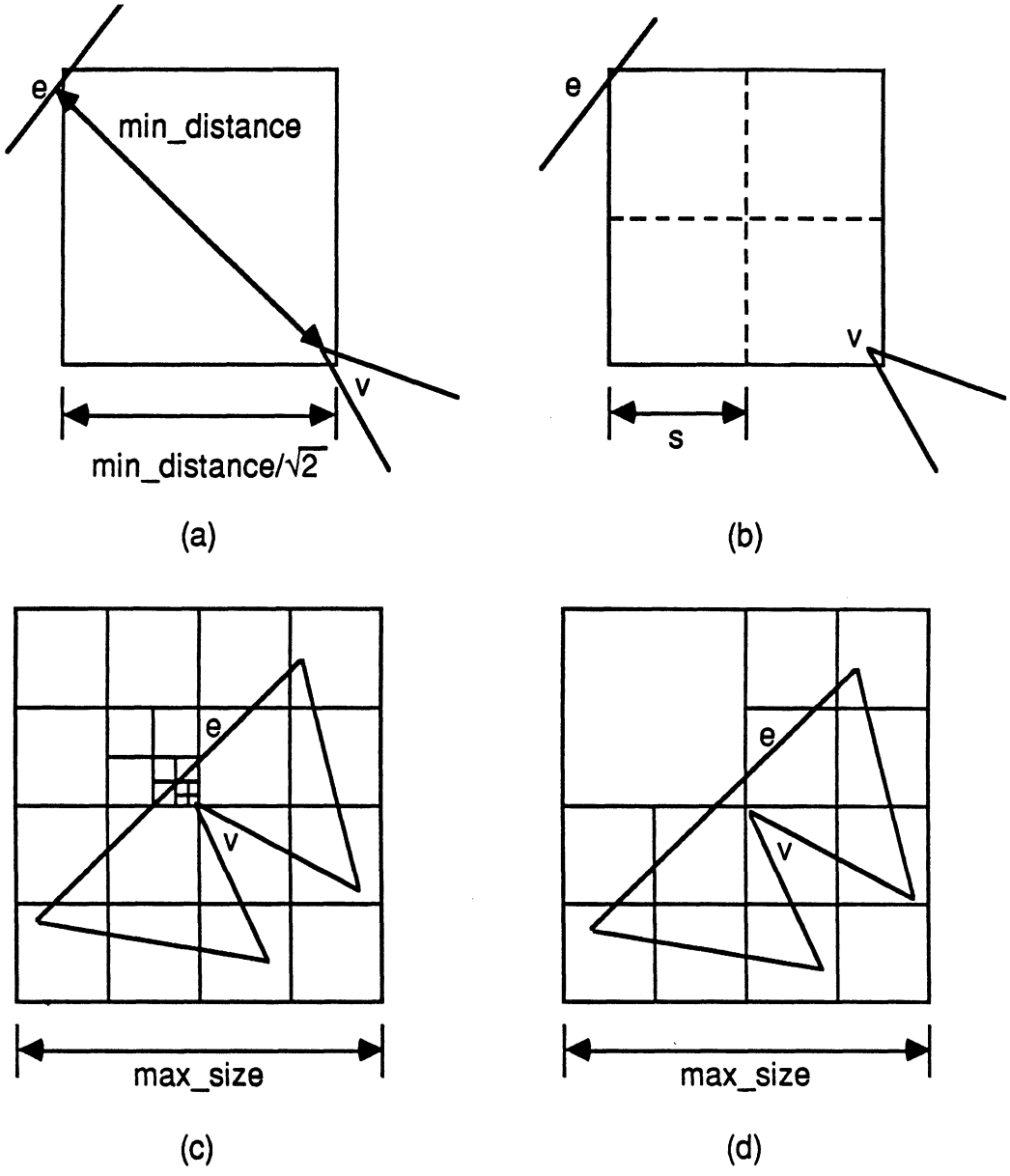


Figure 6. Maximum Possible Height of a Quadtree

employ a known result from geometric probability [21,22] without duplicating its proof.

Lemma 2 Let K_1 be a convex set with perimeter of length P_1 contained in another bounded convex set K_0 with perimeter of length P_0. The probability that a random line intersects K_1, if it is known to intersect K_0, is P_1/P_0.

<Insert Figure 7>

An illustration of Lemma 2 is given in Figure 7. If we draw a random line which intersects the larger quadrant K_0, then the probability that the line also intersects the smaller quadrant K_1 is 1/2. However, an edge of a random polygon is only a subset of a random line since the two endpoints (vertices) of the edge are assumed to be uniformly randomly distributed on the line. Making this observation, we can estimate the average length of an edge.

Lemma 3 The average length of an edge, given the length L of a line segment on which it lies, is L/3.

[Proof] The average length of an edge is:

$$E[|W-U|] = E[W-U|W>U]Pr\{W>U\} + E[U-W|U\geq W]Pr\{U\geq W\}$$

where U and W are random variables representing the two endpoints of an edge lying in the given line segment, and $E[\cdot]$ and $Pr\{\cdot\}$ denote the expectation and probability, respectively. Let $f(\cdot)$ be a probability distribution function. Then the first term on the right-hand side is:

$$E[W-U|W>U]Pr\{W>U\}$$

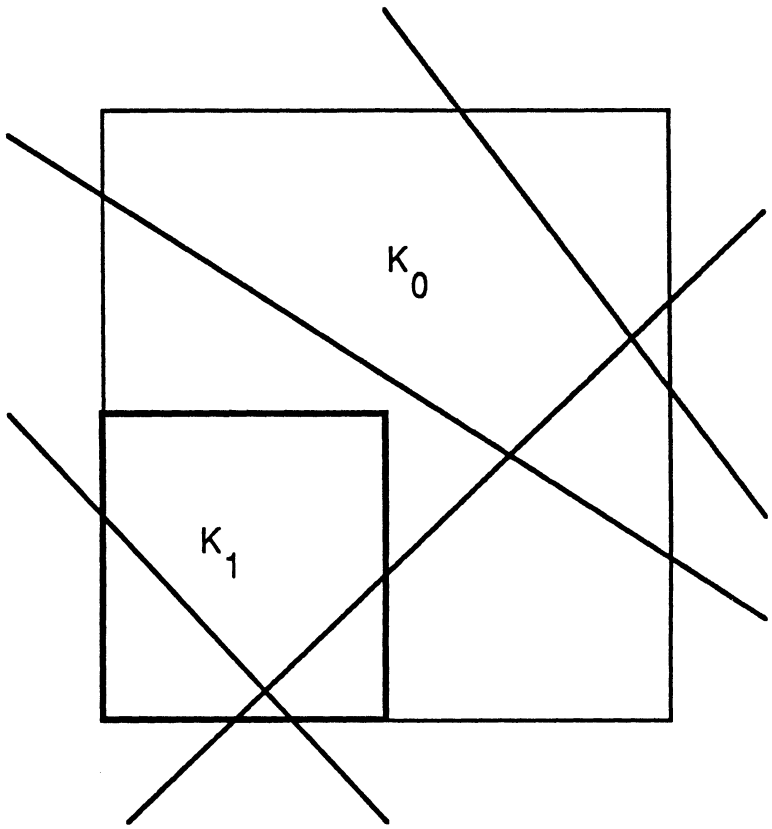


Figure 7. Illustration of Lemma 2

$$= \Pr\{W>U\} \int_{0}^{L}\int_{u}^{L}(w-u)f(u,w)/\Pr\{W>U\}dwdu$$

$$= \int_{0}^{L}\int_{u}^{L}(w-u)f(u)f(w)dwdu$$

$$= (1/L^2) \int_{0}^{L}\int_{u}^{L}(w-u)dwdu$$

$$= (1/L^2)(L^3/6)$$

$$= L/6.$$

With a similar derivation, we get $E[U-W|U\geq W]\Pr\{U\geq W\} = L/6$. Therefore, the average length of an edge is L/3. ■

Using Lemma 2 and Lemma 3, we arrive at the upper bound for the expected number of nodes for a quadtree.

Theorem 1 The expected number of nodes of a quadtree corresponding to a polygon is bounded in space $O(n^2)$.

[Proof] If we extend each edge of a polygon to a line, then the probability that a line intersects a quadrant K_1 of a parent quadrant K_0, given that the line intersects K_0, is 1/2 by Lemma 2. Therefore, the expected number of lines intersecting K_1, when the number of lines intersecting K_0 is n, is n/2. Hence, the average space complexity S(n) of the new method can be calculated by the following recursive relation:

$$S(0) = S(1) = S(2) = 1,$$

$$S(n) = 4S(n/2) + 4.$$

Here, the constant 4 comes from the fact that a quadrant produces 4 sub-quadrants if it is subdivided. The solution of this recursive

relation, i.e. the expected space complexity of our quadtree method, is $O(n^2)$ [1]. Since an edge is shorter than its extended line segment by Lemma 3, the above result is an absolute upper bound on the expected space complexity. ■

4. Time Complexity of the Construction Algorithm

We next analyze the time complexity of constructing a quadtree. The analysis again involves three cases: the best case, the worst case, and the average case.

In the algorithm presented in Section 2, the major element which determines its time complexity is the number of edges examined for each node. By comparing the number of nodes generated and the accumulated number of edges examined, we can get the time complexity T of the algorithm in terms of N, the total number of nodes in a quadtree.

Lemma 4 The best case of the time complexity of the algorithm for constructing a quadtree is O(N).

[Proof] If the average number of edges examined per node is constant, then T(n) is O(N). Such a case does exist, as illustrated in Figure 5(b). Whenever a subdivision is necessary, we examine a constant number of edges, 3 in this example, and produce 4 more nodes. Therefore, the more such subdivisions there are, the closer the average number of edges examined per node is to some constant, 3/4 in this example. Hence, the best case time complexity is O(N). ■

Lemma 5 The worst case of the time complexity of the algorithm for constructing a quadtree is $O(N^2)$.

[Proof] The maximum possible average number of edges examined per node is O(n). If N is O(n), the total number of edges examined is $O(n^2)$. T(n) is therefore $O(N^2)$ in the worst case. Now, we show

the existence of such a case as illustrated in Figure 8. In the figure, whenever there is a subdivision, a vertex is no longer under consideration. In this case, we can show that:

$N = 2n + 3 = O(n)$,

$T(n) = n^2 + 9n - 9 = N^2/4 + 3N - 81/4 = O(N^2)$.

Hence, the worst case time complexity is $O(N^2)$. ■

<Insert Figure 8>

Theorem 2 The average time complexity of the algorithm for constructing a quadtree is $O(N)$.

[Proof] On the average, the accumulated number of edges examined which is equivalent to the time complexity $T(n)$ is:

$T(2) = T(1) = T(0) = 0$,

$T(n) = 4T(n/2) + 4n$.

Here, the constant 4 comes from the fact that a quadrant produces 4 sub-quadrants if it is subdivided. The solution of the above recursive relation is $T(n) = O(n^2)$. To prove that $T(n)$ and $S(n)$ have the same complexity of $O(n^2)$, let $G(n)$ be $T(n) - S(n)$. Then,

$G(n) = 4G(n/2) + 4(n - 1)$.

The solution of this recursive relation is also $O(n^2)$. Having shown that the average $T(n)$ is $O(n^2)$, we recall the definition that there are N nodes in a quadtree. Therefore, the average time complexity of the quadtree construction algorithm is linear in N. ■

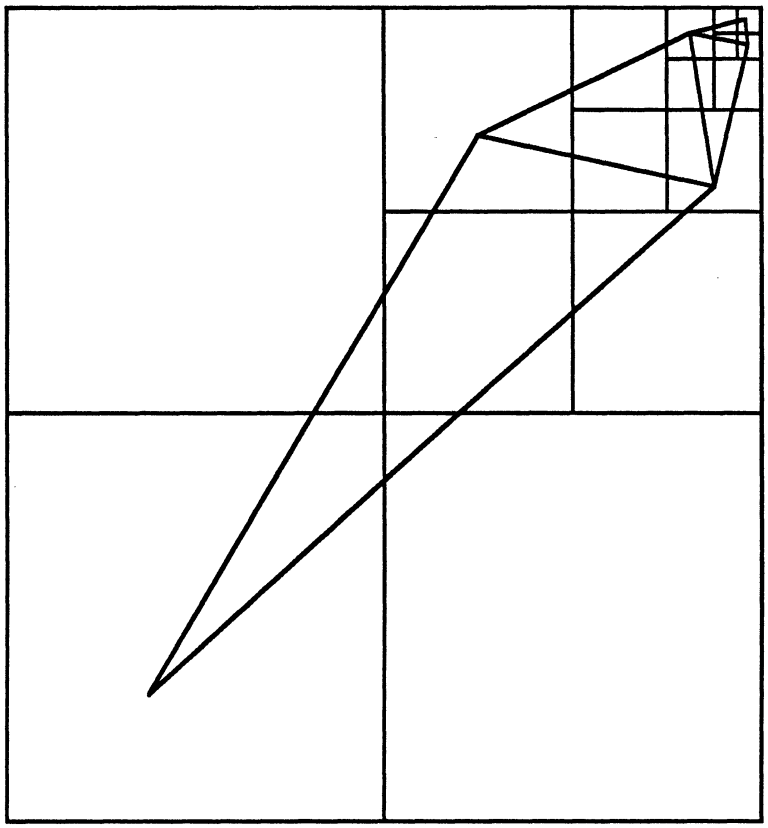


Figure 8. The Worst Case in Time Complexity

5. Experimental Results

We have performed experiments to verify the theoretical storage and time complexities of the construction algorithm. The hardware used for the experiment is Amdahl 470V/8 at the University of Michigan in Ann Arbor, Michigan.

To compare our method to the Binary methods, we use randomly generated polygons. (As there is no provision for handling non-simple polygons by existing methods, we restrict our comparison to using simple polygons.) A method to generate a random simple polygon is as follows. First, we generate uniformly random angles around a point which is located in the center of a quadrant. Then, we generate radii based on the assumption that a radius is distributed uniformly from zero to the one half of the width of the quadrant. By joining the vertices in sorted angular order, we get a random polygon that is simple.

5.1 Space Complexity

A comparison of the number of nodes generated by our method and by the Binary method is illustrated in Figure 9. The result shows, for example, that the number of nodes generated by our method is strictly less than that of the Binary method if the height is greater than 6 and the number of edges is less than 100.

<Insert Figure 9>

A scatter plot of the number of nodes generated against the number of edges of a given polygon is given as Figure 10. In the figure, the numbers indicate the frequency of occurrences and '*' represents an

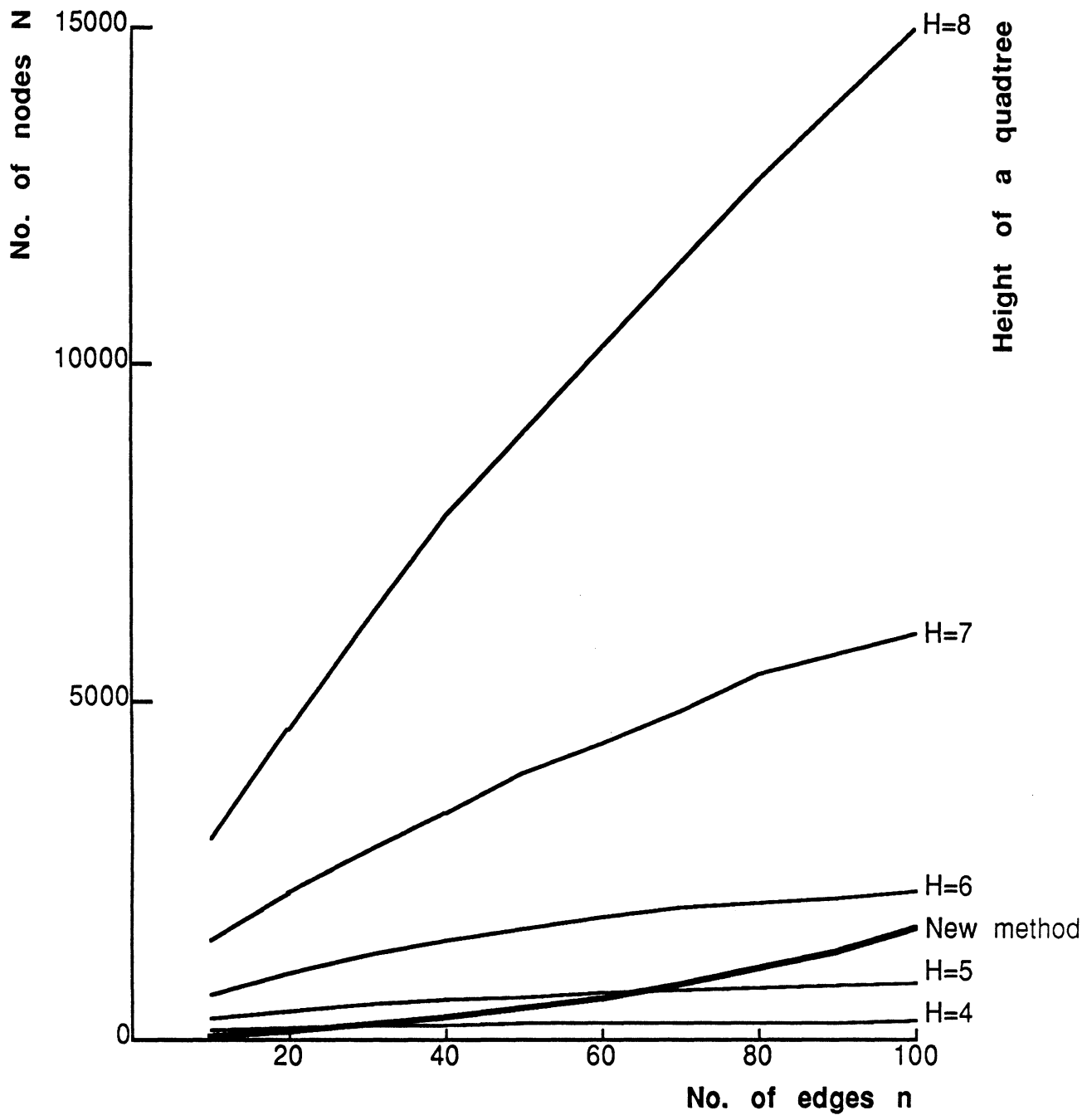


Figure 9. Number of Nodes Generated

occurrence of once. From the data, we find the regression equation to be:

$$N = 0.136\ n^2 + 2.148\ n + 15.461.$$

F_test for the above regression relation shows that we can conclude with a confidence level of 0.99. The coefficient of multiple determination R^2 is 0.989. This result supports Theorem 1 which states that the number of nodes is quadratic in the number of edges.

<Insert Figure 10>

5.2 Time Complexity

The experiment involves a comparison between the number of nodes generated and the construction time. The average number of edges examined per node is also calculated.

The scatter plot of the construction time CT against the number of nodes generated N is given in Figure 11 with the same legend. We find the regression equation to be:

$$CT = 1.147\ N - 18.535.$$

F_test for the above regression relation shows that we can conclude with a confidence level of 0.99. The coefficient of simple determination R^2 is 0.998. This result supports Theorem 2 which asserts the linear relation between time and the number of nodes.

<Insert Figure 11>

The average number of edges examined per node is plotted against the number of edges of a given polygon in Figure 12. The values range from 4.2 to 5.0 and they have a tendency to be strictly less than 5.

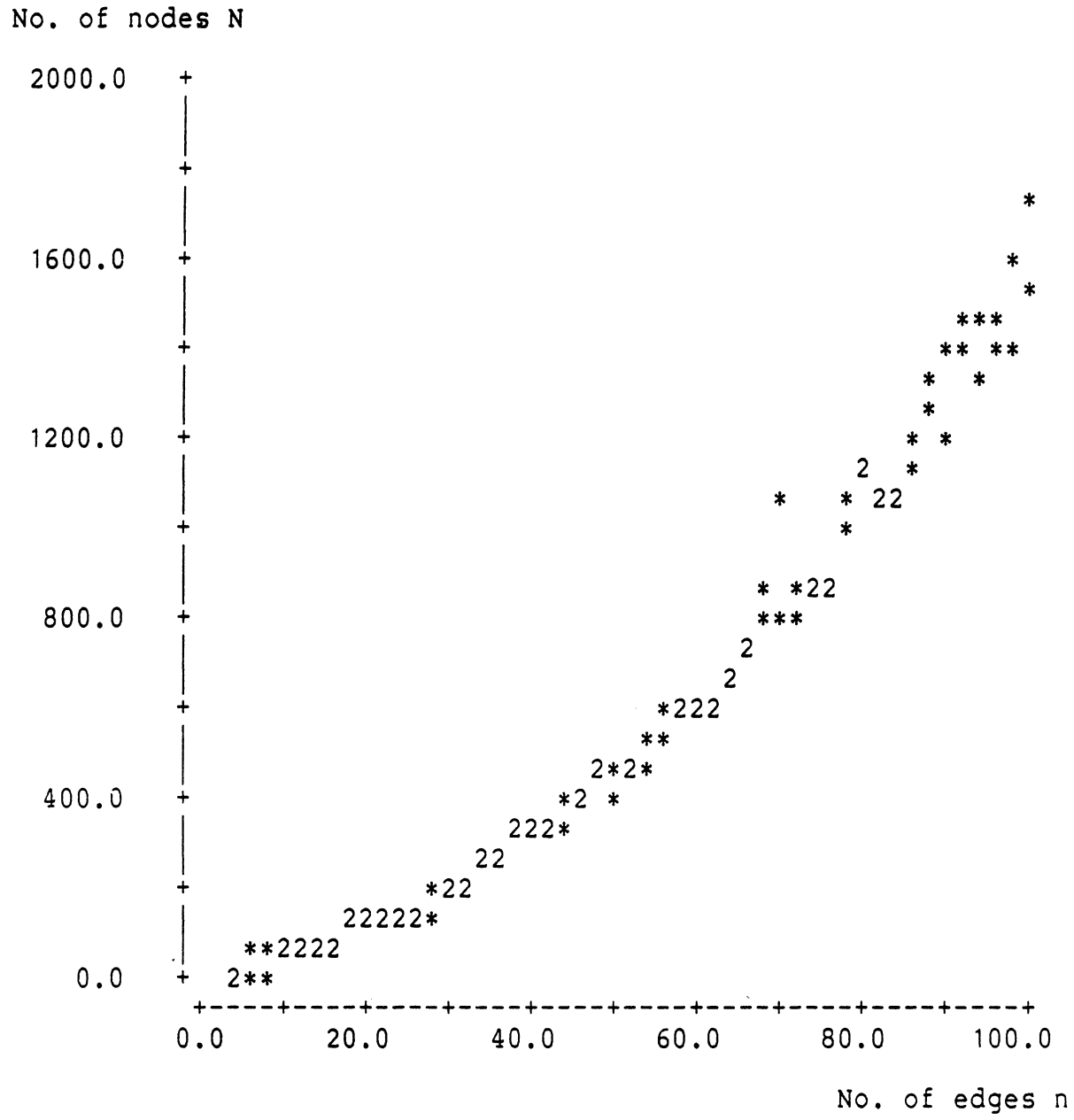


Figure 10. Scatter Plot:

Number of Nodes versus Number of Edges

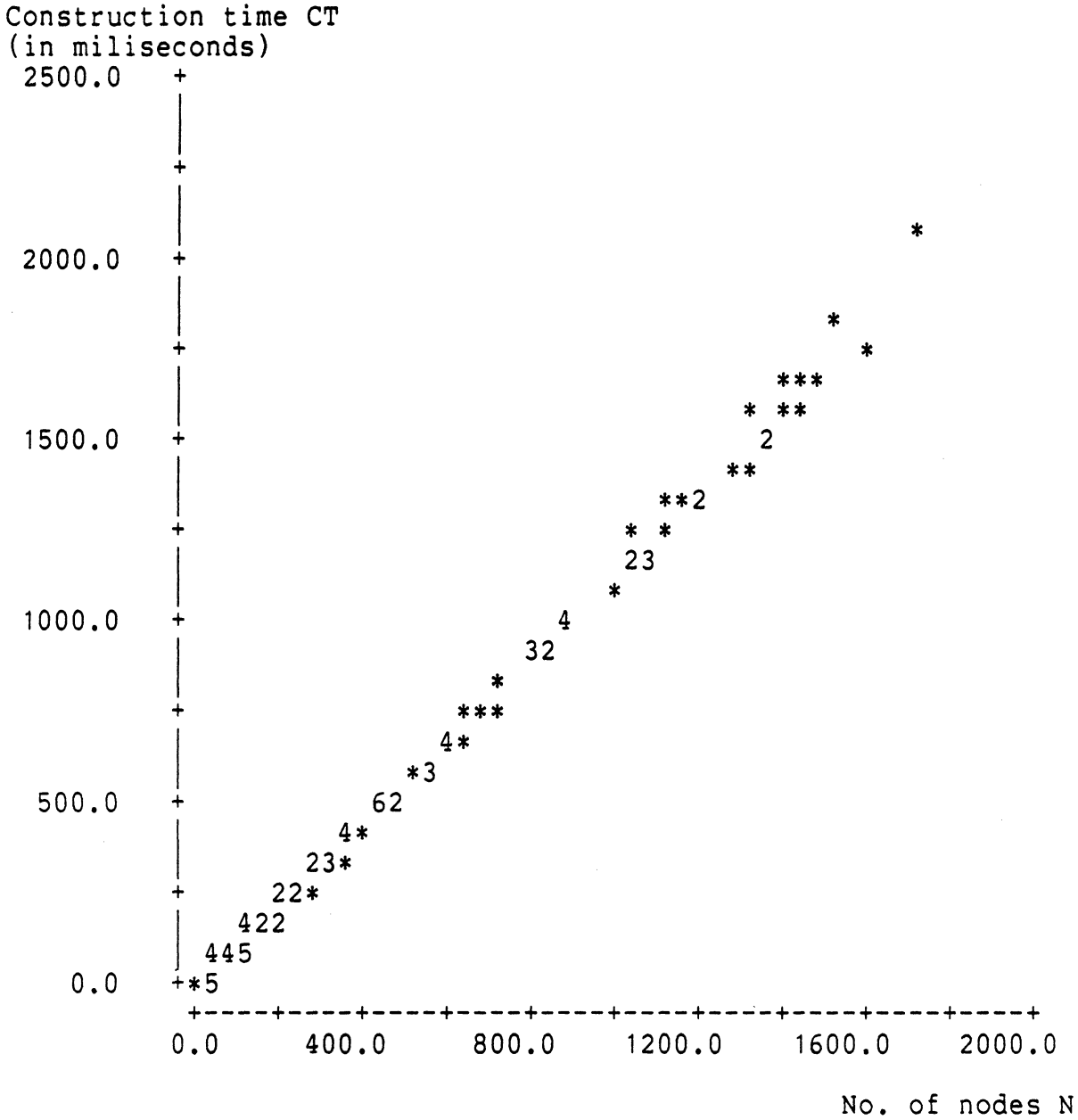


Figure 11. Scatter Plot:

Construction Time versus Number of Nodes

This result also supports Theorem 2.

<Insert Figure 12>

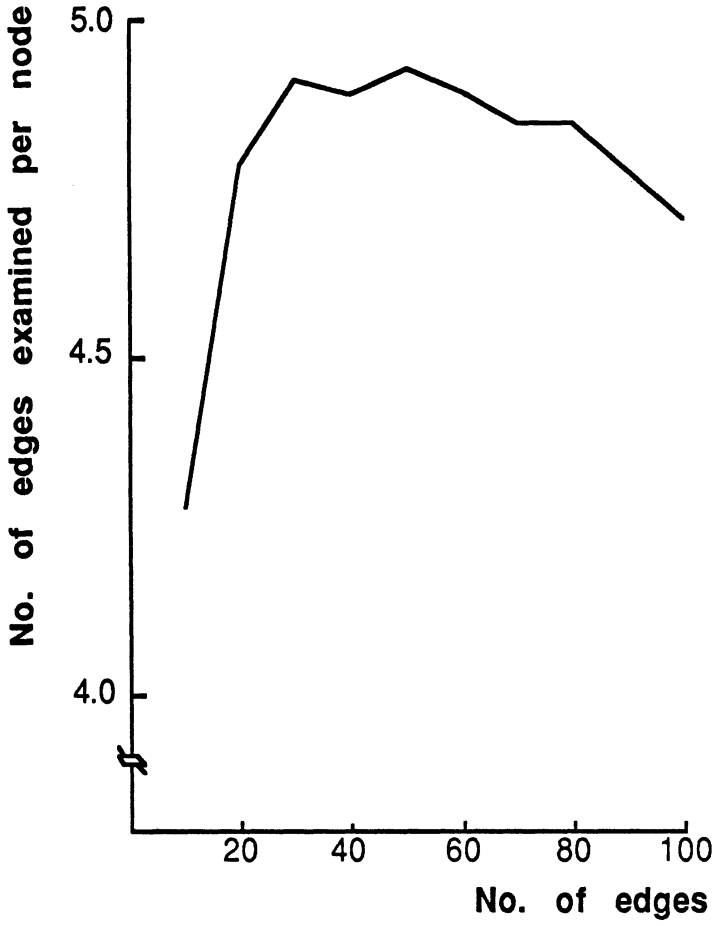


Figure 12. Average Number of Edges Examined per Node against Number of Edges

6. Conclusion

By using the new termination conditions, we are able to reduce the storage for a quadtree from exponential to quadratic in n, on the average, for a given n-gon, yet offer exact representation. The construction time for such a quadtree is quadratic in n as well. While our method retains much of the simple elegance of the existing quadtree methods, our analysis confirmed by experiments offer a new lower bound in time and storage to theoreticians and practitioners alike.

References

[1] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The design and analysis of computer algorithms*, Addison-Wesley, Reading, Mass., 1974.

[2] D. Ayala, P. Brunet, R. Juan, and I. Navazo, Object representation by means of nonminimal division quadtrees and octrees, *ACM Trans. Graphics* **4**, pp. 41-59, 1985.

[3] H. H. Atkinson, I. Gargantini, and T. R. S. Walsh, Filing by quadrants and octants, *Computer Vision, Graphics, and Image Processing* **33**, pp. 138-155, 1986.

[4] B. B. Chaudhuri, Applications of quadtree, octree, and binary tree decomposition techniques to shape analysis and pattern recognition, *IEEE Pattern Analysis and Machine Intelligence* **PAMI-7**, pp. 652-661, 1985.

[5] C. R. Dyer, A. Rosenfeld, and H. Samet, Region representation : boundary codes from quadtrees, *Comm. ACM* **23**, pp 171-179, 1980.

[6] C. E. Eastman, Representation for space planning, *Comm. ACM* **13**, pp. 242-250, 1980.

[7] I. Gargantini, Linear octrees for fast processing of three dimensional objects, *Computer Graphics and Image Processing* **20**, pp. 365-374, 1982.

[8] G. M. Huster and K. Steiglitz, Operations on images using quadtrees, *IEEE Trans. Pattern Analysis and Machine Intelligence* **PAMI-1**, pp. 145-153, 1979.

[9] C. L. Jackins and S. L. Tanimoto, Octrees and their use in representing 3D objects, *Computer Graphics and Image Processing* **14**, pp. 249-270, 1980.

[10] C. L. Jackins and S. L. Tanimoto, Quadtrees, octrees, and k-trees : a generalized approach to recursive decomposition of Euclidean space, *IEEE Trans. Pattern Analysis and Machine Intelligence* **PAMI-5**, pp. 533-539, 1983.

[11] L. P. Jones and S. S. Iyengar, Space and time efficient virtual quadtrees, *IEEE Trans. Pattern Analysis and Machine Intelligence* **PAMI-6**, pp. 244-247, 1984.

[12] A. Klinger and C. D. Dyer, Experiments on picture representation using regular decomposition, *Computer Graphics and Image Processing* **5**, pp.68-105, 1976.

[13] D. Meagher, Geometric modeling using octree encoding, *Computer Graphics and Image Processing* **19**, pp. 129-147, 1982.

[14] F. P. Preparata and M. I. Shamos, *Computational Geometry*, Springer-Verlag, New York, 1985.

[15] C. Puech and H. Yahia, Quadtrees, octrees, and Hyperoctrees : a unified analytical approach to tree data structures used in graphics, geometric modeling and image processing, ACM Proc. Symposium on Computer Graphics, pp. 272-280, 1985.

[16] H. Samet, Region representation : quadtrees from boundary codes, *Comm. ACM* **23**, pp. 163-170, 1980.

[17] H. Samet, Connected component labeling using quadtrees, *J. ACM* **18**, pp. 487-501, 1981.

[18] H. Samet, The quadtree and related hierarchical data structures, *ACM Comput. Survey* **16**, pp. 187-260, 1984.

[19] H. Samet, Data structure for quadtree approximation and compression, *Comm. ACM* **28**, pp. 973-993, 1985.

[20] H. Samet and R. E. Webber, On encoding boundaries with quadtrees,

IEEE Trans. Pattern Analysis and Machine Intelligence **PAMI-6**, pp. 365-369, 1984.

[21] L. A. Santalo, *Integral geometry and geometric probability*, Addison-Wesley, Reading, Mass., 1976.

[22] M. Tamminen, Performance analysis of cell based geometric file organizations, *Computer Vision, Graphics, and Image Processing* **24**, pp. 160-181, 1983.

[23] M. Yau and S. N. Srihari, A hierarchical data structure for multidimensional digital images, *Comm. ACM* **26**, pp. 504-515, 1983.