Constant Time Neighbor Finding in Quadtrees:

An Experimental Result

Kunio Aizawa, Koyo Motomura, Shintaro Kimura, Ryosuke Kadowaki, and Jia Fan
Department of Mathematics and Computer Science
Interdisciplinary Faculty of Science and Engineering
Shimane University, Matsue
Shimane 690-8502 Japan
aizawa@cis.shimane-u.ac.jp

Abstract—**Neighbor finding is an important and a basic part of image processing in quadtrees. A constant time algorithm is proposed for neighbor finding in quadtrees in [1]. In this paper, empirical tests are given for the constant time algorithm in comparison with usual neighbor finding algorithm using quadtrees [2] and another constant time algorithm using linear quadtree [3]. Experiments using synthetic images simulating worst case situations show that the proposed algorithm is in constant time complexity while others are not. Even for experiments using natural images, the proposed algorithm is more than twice as fast as algorithm using quadtrees and is slightly as fast as algorithm using linear quadtrees.**

Keywords-component; image processing, quadtrees, linear quadtrees, neighbor finding

I. INTRODUCTION

Quadtrees were originally proposed in [4]. An image is stored in a tree such that each node has four sons, each of which represents a quadrant (NE, NW, SE, and SW) of a given square at corresponding level (Fig. 1(a) and 1(b)). To facilitate better understanding of our proposal, the tree structures are reviewed briefly.

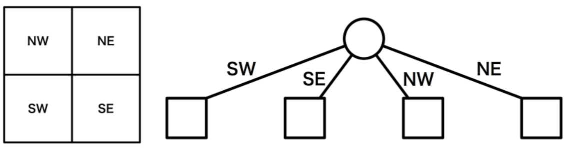

Figure 1. An example of the quadrants of an image and the corresponding quadtree.

Each quadrant is checked whether the image fills it completely, partially, or not at all. If a quadrant is filled completely, the corresponding node of a quadtree is assumed "BLACK". If a quadrant is filled partially, the corresponding node is assumed "GRAY". BLACK and WHITE nodes become *leaf nodes*, while GRAY nodes are subdivided into four subquadrants. This process continues until all leaf nodes are labeled BLACK or WHITE, or a given level is reached, called the *resolution r* (or *height*) of the quadtree (see Fig. 2 for $r = 3$). There are at most r subdivision levels. For in-depth expositions, see [5] and [6].

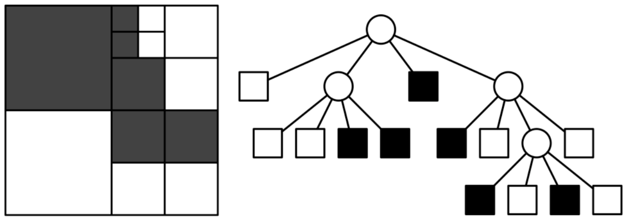

Figure 2. A black and white image and its quadtree representation.

A variant, linear quadtree was proposed in [7]. Each leaf node of a linear quadtree is represented by an ordered pair, (n, l), where n is its spatial address called the *location code* and l is its level. The level of the root node is 0, that of its four sons is 1, etc. A *linear quadtree* is a list of the code/level pairs of all BLACK nodes.

Finding the neighbors of a specific leaf node is a fundamental operation for many algorithms which manipulate quadtree data structures. In quadtrees, finding neighbors takes $O(r)$ computational time for the worst case (see, e.g., [2]). Schrack [3] proposed a constant time algorithm for finding equal-sized neighbors in linear quadtrees. His algorithm calculates the location codes of equal size neighbors only, without determining their existence. To find the location codes of different-sized neighbors requires computational time proportional to the level difference of these neighbors (i.e., at most $O(r)$), necessary for searching the list of location codes of the given linear quadtree in general. In [1], a new algorithm to find the neighbors of a given leaf node in a quadtree is proposed, which requires only $O(1)$ (i.e., constant) computational time for the worst case. Moreover, the algorithm does not claim consideration of the existence or non-existence of neighbors. Therefore, no additional checking is needed.

In this paper, empirical tests are given for our constant time algorithm in comparison with Samet's neighbor finding algorithm using quadtrees and Schrack's constant time algorithm using linear quadtree. Experiments using synthetic images simulating worst case situations show that the proposed algorithm is in constant time complexity while others are not. Even for experiments using natural images, the proposed algorithm is more than twice as fast as algorithm using quadtrees and is slightly as fast as another constant time algorithm using linear quadtrees.

The rest of the paper is organized as follows. In Section II, basic definitions and properties of linear quadtrees are reviewed, as well as Schrack's algorithm. In Section III, constant time algorithm for finding neighbors in quadtree with location codes and level differences (QTLCLD) is reviewed briefly. Experiments results are given in Section IV. Finally a brief conclusion is given in Section V.

II. LINEAR QUADTREES

A location code is denoted as a quaternary integer. Quadrants are labeled according to a labeling scheme, where the SW, SE, NW, NE quadrants are labeled 0, 1, 2, 3, respectively. The most significant quaternary digit of a location code represents the quadrant of level 1, the following digit is the quadrant of level 2, and so on. Therefore, a location code has always exactly r digits and the given image is represented by $2^r \times 2^r$ pixels. A node at level $l < r$ has a location code the last $r - l$ digits of which are all 0s. Although, a *linear quadtree* usually is a list of location code/level pairs of its BLACK nodes only, in this paper, we will include all WHITE nodes as well. The linear quadtree of the image in Fig. 3 is then becomes:

linear quadtree = {(000, 1), (100, 2), (110, 2), (120, 2), (130, 2), (200, 1), (300, 2), (310, 2), (320, 3), (321, 3), (322, 3), (323, 3), (330, 2)}.

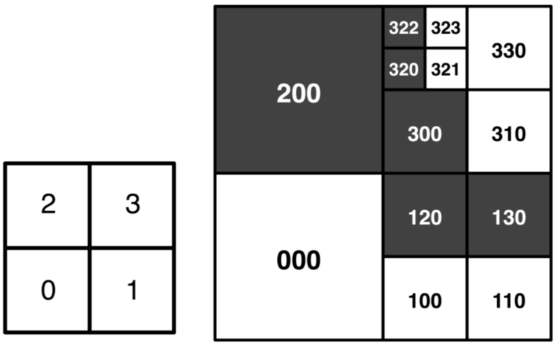

Figure 3. Location codes for an image of resolution $r = 3$.

Gargantini [7] has shown that the binary representation of the location code n of a pixel is an *interleaved coordinate*, that is to say it has the structure

$$n = y_{r-1}x_{r-1}...y_1x_1y_0x_0,\ x_i, y_i \in \{0,1\} \qquad (1)$$

where $x = x_{r-1}...x_1x_0$, $y = y_{r-1}...y_1y_0$ are the binary representations of the coordinates of the quadrant with location code n. For example, the binary representation of the quaternary location code "320" is "111000." Thus the binary integer "100" is the x-coordinate and "110" is the y-coordinate of the quadrant "320."

For Schrack's algorithm [3], the following operators are assumed:

$+$	normal addition of two binary integers,
$\vert$	bitwise OR,
$\wedge$	bitwise AND,
$<< n$	"shift left" n times,
$>> n$	"shift right" n times.

In addition, two constants (in binary representation) are required:

$t_x = 01...0101$	"01" repeated r times,
$t_y = 10...1010$	"10" repeated r times.

The quad location addition operator $\oplus_q$ becomes

$$\begin{aligned} m_q &= n_q \oplus_q \Delta n_i \\ &= (((n_q \mid t_y) + (\Delta n_i \wedge t_x)) \wedge t_x) \mid (((n_q \mid t_x) + (\Delta n_i \wedge t_y)) \wedge t_y) \end{aligned} \qquad (2)$$

where n_q is the binary representation of a given location code, Δn_i is one of the *basic direction increments* defined below, and m_q is the location code (in binary) of the neighbor of the quadrant n_q. For $r = 3$, the eight basic direction increments are defined by

$\Delta n_0 = 000001$	East neighbor,
$\Delta n_1 = 000011$	North-East neighbor,
$\Delta n_2 = 000010$	North neighbor,
$\Delta n_3 = 010111$	North-West neighbor,
$\Delta n_4 = 010101$	West neighbor,
$\Delta n_5 = 111111$	South-West neighbor,
$\Delta n_6 = 101010$	South neighbor,
$\Delta n_7 = 101011$	South-East neighbor.

To obtain the equal-sized neighbors of any level is summarized by the following theorem.

Theorem 1 (Calculation of neighbors of equal size) [3]**:** Given a location code n_q and its level l, the eight neighbors of equal size are given by

$$m_q = n_q \oplus_q (\Delta n_i << (2(r-l))),\ i = 0, 1, ..., 7, \qquad (3)$$

where $\oplus_q$ is the quad location addition operator, Δn_i are the eight basic direction increments, and r is the (fixed) resolution. This calculation is of constant time-complexity.

Note that the $2(r - l)$ times "shift-left" operation can be replaced by the single multiplication by $2^{2(r-l)}$.

III. CONSTANT TIME ALGORITHM FOR FINDING NEIGHBORS

In [1], a new data structure for quadtrees is proposed, which holds the location codes as linear quadtrees and also holds the differences of levels between adjacent quadrants. For example, the data structure takes the form represented in Fig. 4 for the image of Fig. 2. Length of the location code varies in proportion to the resolution of image, i.e., if r =3 then the code has three digits. The Fig. 4 is in the case of r =3. The meanings of numbers for each quadrant are in Fig. 5.

To describe a quadrant with level differences, we will use the following notation:

(location code, level, color, Δeast, Δnorth, Δwest, Δsouth), where Δeast, Δnorth, Δwest, and Δsouth represent the level differences between east, north, west, and south neighbors, respectively.

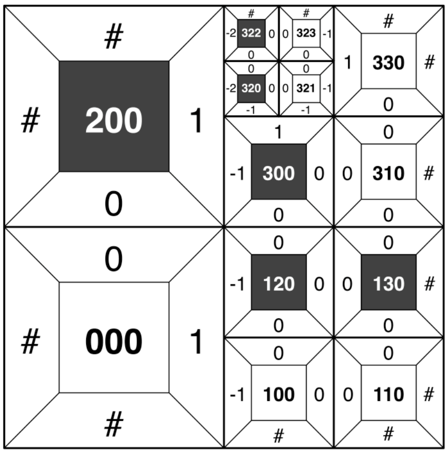

Figure 4. An example of level differences between adjacent quadrants.

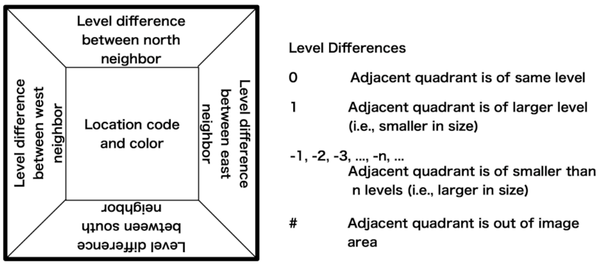

Figure 5. Legend of numbers for a quadrant of quadtree.

So, the complete quadtree with location codes and level differences (*QTLCLD*) for the image of Fig. 2(a) is as follows:

QTLCLD = {(000, 1, WHITE, 1, 0, #, #), (100, 2, WHITE, 0, 0, –1, #), (110, 2, WHITE, #, 0, 0, #), (120, 2, BLACK, 0, 0, –1, 0), (130, 2, BLACK, #, 0, 0, 0), (200, 1, BLACK, 1, #, #, 0), (300, 2, BLACK, 0, 1, –1, 0), (310, 2, WHITE, #, 0, 0, 0), (320, 3, BLACK, 0, 0, –2, –1), (321, 3, WHITE, –1, 0, 0, –1), (322, 3, BLACK, 0, #, –2, 0), (323, 3, WHITE, –1, #, 0, 0), (330, 2, WHITE, #, #, 1, 0)}

Algorithm to construct the quadtree with location codes and level differences for a given image is in Fig. 6. The algorithm is based on breadth-first expansion of a quadtree. In the middle of this expansion, Schrack's algorithm is used to find equal size neighbors in constant time. In these neighbors, the level differences are recalculated according to the following method. The recalculations are done in four cases:

Case 1: If the level difference of a parent is #, then
 the corresponding difference of its children is also #,
Case 2: if the level difference of a parent is 0, then
 the corresponding difference of its children is -1,
Case 3: if the level difference of a parent is 1, then
 the corresponding difference of its children is 0,
Case 4: if the level difference of a parent is -*i*, then
 the corresponding difference of its children is –(*i*+1).

The recalculations are represented in Fig. 7. As stated before, value "1" means only the fact that "they are smaller than me." But in the level differences adjusting algorithm in Fig. 6, quadtree expansion proceeds in "breadth-first" style. So whenever an quadrant is intended to divide, therefore the smaller neighbors must be smaller than at most ONE level.

algorithm QuadtreeWithLCodeLevelDifferences (image *bh*, quadtree *qtlcld*)
 qtlcld:={(000, 0, GREY, #, #, #, #)};
 while *qtlcld* includes GREY areas **do**
  for each equal size neighbor of first GREY area **do**
   Corresponding level differences of neighbors are added by 1;
  end for;
  First GREY quadrant is replaced by its four children;
  Newly introduced four children are added to the head of queue *qtlcld*;
  for each equal size neighbor of each child (other than their brothers) **do**
   Corresponding level differences of neighbors are added by 1;
  end for;
 end while;
end QuadtreeWithLCodeLevelDifferences;

Figure 6. Algorithm to construct *QTLCLD*.

It is easy to see that the algorithm in Fig. 6 is just a breadth-first expansion of a quadtree using Schrack's constant time algorithm to find equal size neighbors in two **for**-loops. The number of equal size neighbors in the first **for**-loop is at most four and in the second **for**-loop is at most eight. Then, obviously, the following theorem holds.

Theorem 2 (Construction of *QTLCLD*) [1]**:** For an image of resolution r having n quadrants, a linear quadtree with level differences can be constructed within $O((r + 1)n)$. Its time complexity is the same as that of usual quadtree construction algorithm.

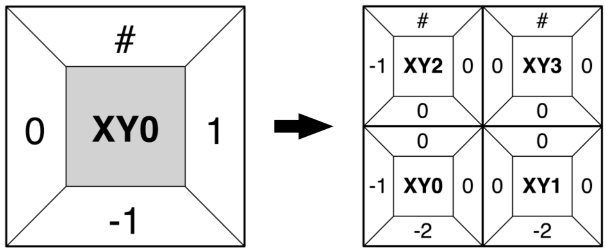

Figure 7. Recalculation of level differences when a quadrant is divided into four children.

A constant time algorithm for neighbors finding in quadtrees is presented below by making use of the data structure defined in the previous section. At first, neighbors in given direction of a given quadrant must be defined. Unfortunately, there are often more than one neighbors in given direction. So, we follow Samet's definition in [6].

Definition 1: The neighbor Q in given direction of given quadrant P is the smallest quadrant (it may be GRAY) that is adjacent to P in given direction and is of size greater than or equal to the quadrant P.

The algorithm to calculate location code/level in given direction for given location code/level pair is in Fig. 8, where r is the resolution of given image and Δn_i's are equal to that of Schrack's algorithm. In our case, Δn_i is defined only for $i = 0$, 2, 4, 6. The algorithm is based on Schrack's *dilated integer addition* $\oplus_q$ but by making use of data structure introduced in

Fig. 5, it can calculate location code/level for neighbor in given direction.

```
algorithm NeighborFinding (quadtree qtlcld, code/level quadrant,
                           direction d, code/level neighbor)
  dd:=qtlcld(quadrant).d;
    /* level difference with given quadrant and its neighbor in direction d */
    /* such value derived from qtlcld using location code of quadrant       */
  if dd≠"#" then
    n_q:=area.locationCode;
    l:=area.level;
    if dd<0 then
      m_q:=((n_q>>2(r-l-dd))<<2(r-l-dd))⊕_q(Δn_d<<(2(r-l-dd)));
    else
      m_q:=n_q⊕_q(Δn_d<<(2(r-l)));
    end if;
    neighbor.code:=m_q;
    neighbor.level:=qtlcld(m_q).level;
  else
    /* if dd="#" then here is no neighbor since it is a border       */
    /* appropriate processes are done here for such case             */
  end if
end NeighborFinding;
```

Figure 8. A constant time algorithm for finding neighbors in quadtrees.

The algorithm is based on the following formulae:

$$\begin{cases} m_q = ((n_q >> 2(r-l-dd)) << 2(r-l-dd)) \\ \qquad \oplus_q (\Delta n_d << (2(r-l-dd))) & dd < 0 \\ m_q = n_q \oplus_q (\Delta n_d << (2(r-l))) & dd \geq 0 \end{cases} \quad (4)$$

where n_q is the location code (in binary) of given quadrant, m_q is the location code (in binary) of the neighbor in the direction d (d = 0, 2, 4, 6), dd is the level difference for the neighbor in the direction d, r is the resolution of quadtree, l is the level of given quadrant, $\Delta n_0 = 000001, \Delta n_2 = 000010, \Delta n_4 = 010101, \Delta n_6 = 101010$, for r = 3.

For the cases of $dd \geq 0$, it is equal to Schracks' algorithm because in such cases the neighbors are in the same size by Definition 1. For the cases of $dd < 0$, the location code of given quadrant is shifted right $2(r\text{-}l\text{-}dd)$ digits then left $2(r\text{-}l\text{-}dd)$ digits to set the size of given quadrant equal to that of the neighbor. Basic direction increments are also shifted left $2(r\text{-}l\text{-}dd)$ digits to set the proper size.

This algorithm consists of several substitution statements and two calculation statements that includes dilated integer addition, shift-right, and shift-left. It has no iterative process. It is obvious that the algorithm is done in $O(1)$ (i.e., constant time). So the following theorem holds.

Theorem 3 (Neighbor finding in *QTLCLD*) [1]**:** Algorithm "NeighborFinding" has constant-time complexity.

IV. Experimental Results

We implement all three algorithm, i.e., our algorithm, Samet's algorithm, and Schrack's algorithm. These implementations are done on the following environment:

- CPU: Intel Core 2 Duo 4300 1.80GHz
- RAM: 1GB
- OS: Microsoft Windows XP Professional x64 Edition Version2003
- Language: Microsoft Visual C++ 2005

A. Experiments on Synthetic Images

We prepare two sets of images synthetic and natural. Synthetic images simulate worst-case situations. Roughly speaking, all neighbor finding operations taking place in these images are of searching for the largest quadrant from the smallest one. An example of these images is represented in Fig. 9 (for the case *level* = 3).

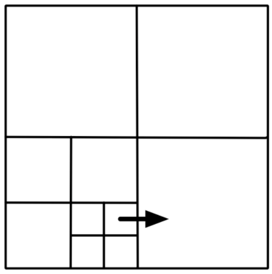

Figure 9. Neighbor finding in a synthetic image (*level* = 3).

We made eight synthetic images of *level* 3 (8 x 8) to *level* 10 (1024 x 1024). We repeated each algorithm for 1,000,000 times for each image and took average for execution time. The results are presented in Table I.

TABLE I. NEIGHBOR FINDING EXECUTION TIME IN SYNTHETIC IMAGES

Levels	No. of pixels	Quadtree (Secs)	Linear quadtree (Secs)	QTLCLD (Secs)
3	64	6.2130E-08	2.9533E-08	2.6359E-08
4	256	7.3245E-08	3.2344E-08	2.6605E-08
5	1024	8.3680E-08	3.5365E-08	2.7067E-08
6	4096	9.4344E-08	3.8490E-08	2.7067E-08
7	16384	1.0451E-07	4.2212E-08	2.6437E-08
8	65536	1.1682E-07	4.5219E-08	2.6691E-08
9	262144	1.2574E-07	4.9570E-08	2.6701E-08
10	1048576	1.4163E-07	5.2839E-08	2.6513E-08

The results in Table 1 show that our algorithm (*QTLCLD*) is two to five times as fast as Samet's algorithm (quadtree). More importantly, it is the only constant time algorithm. Due to size difference between two quadtrants, Schrack's algorithm (linear quadtree) needs execution time, which is proportion to image levels. Fig. 10 shows these situations clearly.

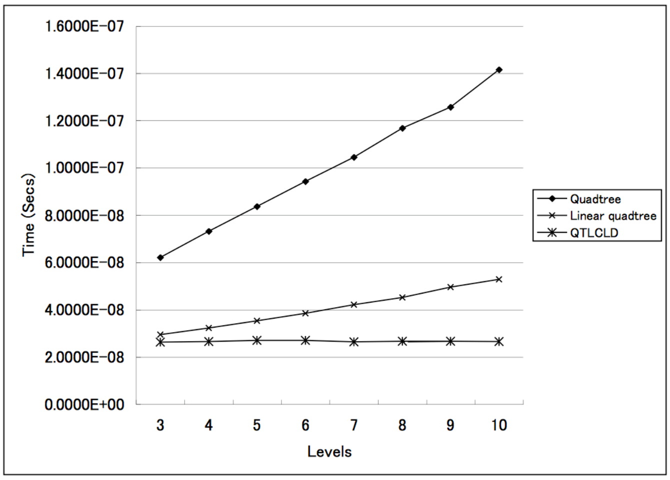

Figure 10. Neighbor finding in synthetic images.

B. *Experiments on Natural Images*

Fifty natural images from the image database of the USC (University of Southern California) which are 1024 x 1024 pixels in size (level 10), were used for evaluation (see, Fig.11). The images were transformed into binary images by making use of the method in [8]. From these images we produced smaller images from 32 x 32 (level 5) to 512 x 512 pixels (level 9).

Figure 11. An example of images from the image database of the USC.

In these images, there are some possibilities that neighbor finding from larger quadrant to smaller quadrants happens. So, we extended each algorithm to handle such situations. Again, we repeated each algorithm for randomly chosen 1,000,000 points for each image and took average for execution time. The results are presented in Table II.

The results in Table II show that our algorithm is still more than twice as fast as algorithm using quadtrees. Algorithms using *QTLCLDs* and linear quadtrees have constant time complexities generously. Their execution times are increased less than 0.2×10^{-8} while the sizes of images are increased 2^{10} times. However the difference between linear quadtrees and *QTLCLDs* is a slight. Fig. 12 and Fig. 13 show these situations clearly.

TABLE II. NEIGHBOR FINDING EXECUTION TIME IN NATURAL IMAGES

Levels	No. of pixels	Quadtree (Secs)	Linear quadtree (Secs)	QTLCLD (Secs)
5	1024	6.4111E-06	2.6962E-06	2.6070E-06
6	4096	6.7178E-06	2.7115E-06	2.6289E-06
7	16384	6.8755E-06	2.7235E-06	2.6498E-06
8	65536	7.0260E-06	2.7354E-06	2.6604E-06
9	262144	7.3569E-06	2.7534E-06	2.6795E-06
10	1048576	7.5942E-06	2.7643E-06	2.6895E-06

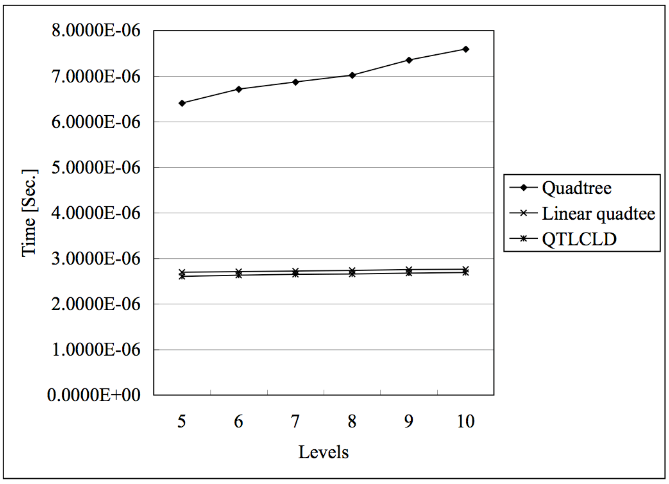

Figure 12. Neighbor finding in natural images.

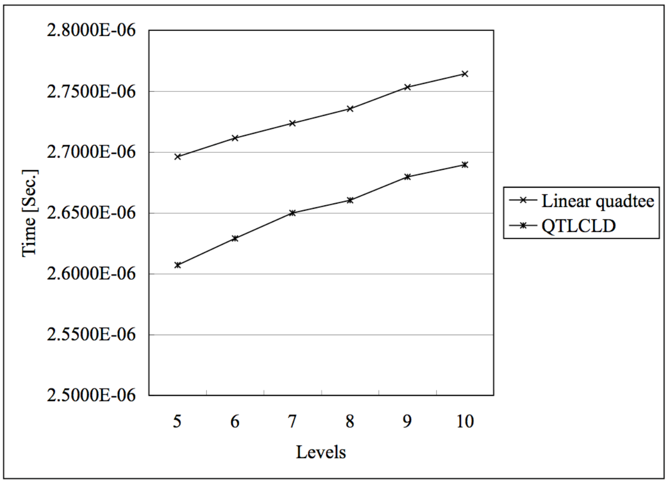

Figure 13. Difference between *QTLCLD* and linear quadtree.

V. CONCLUSION

This paper has given empirical tests for three neighbor finding algorithm in quadtrees. It demonstrated the algorithm based on *QTLCLD* takes less execution time. Even in the worst-case situations, it takes constant execution time, while other two algorithms need execution time proportional to image level.

ACKNOWLEDGMENT

Kunio Aizawa thanks Prof. Shojiro Tanaka of Shimane University and Prof. Akira Nakamura for his valuable suggestions.

REFERENCES

[1] Aizawa, K. and S. Tanaka, A constant time algorithm for finding neighbors in quadtrees, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2007 (Submitted).

[2] Samet, H., Neighbor finding techniques for images represented by quadtrees, *Computer Graphics and Image Processing*, 18, pp. 35-57 1982.

[3] Schrack, G., Finding neighbors of equal size in linear quadtrees and octrees in constant time, *CVGIP: Image Understanding*, 55, pp. 221-230 1992.

[4] Finkel, R. A. and J. L. Bentley, Quad trees: a data structure for retrieval on composite keys, *Acta Informatica*, 4, pp. 1-9, 1974.

[5] Samet, H., *The design and analysis of spatial data structures*, Addison-Wesley, 1990.

[6] Samet, H., *Applications of spacial data structures: computer graphics, image processing, and GIS*, Addison-Wesley, 1990.

[7] Gargantini, I., An effective way to represent quadtrees, *Communications of the ACM*, 25, pp. 905-10, 1982.

[8] Otsu, N., A threshold selection method from gray-level histgrams, *IEEE Transaction on Systems Man Cybernetcs*, 9, pp. 62-66, 1979.