Improved quadtree image segmentation approach to region information

Z. F. Muhsin[1], A. Rehman*[2], A. Altameem[3], T. Saba[4] and M. Uddin[5]

Images are full of information and most often, little information is desired for subsequent processing. Hence, region of interest has key importance in image processing. Quadtree image segmentation has been widely used in many image processing applications to locate the region of interest for further processing. There are also variable block-size image coding techniques to effectively reduce the number of transmitted parts. This paper presents quadtree partition technique as a pre-processing step in image processing to determine what part should be more heterogeneous than the others. It also introduces an idea to solve the problem of squared images. Finally, proposed approach is implemented and analysed. The simulation of the Matlab code of the quadtree is represented by all algorithms and the figures. Thus, achieved results are promising in the state of the art.

Keywords: Quadtree, Image segmentation, Image compressing, Information acquisition, Region of interest (ROI), Image visualisation

Introduction

The quadtree (QT) decomposition or partitioning is widely used in the digital image processing applications. It has also been found that manipulations and calculations operations done upon an image by QT prove to be efficient. The QT segments the image into homogenous and heterogeneous parts to allocate fewer bits to homogenous regions and more bits to those regions that contain additional detail and sharper features, which means that specific information embedded in the images, is of high worth than the rest.[1,2] Hence, this information is required to extract its background. The QT method scans the bitmap area by area to identify matching pixels (area has homogenous features).[3,4]

The QT achieves a structure with a variable block-size decomposition of an image, where the movement to a smaller block size represents fine details more accurately and it can be carried out according to the outcome of any homogeneity test criteria. The weakness of the fixed size partitioning lies in the fact that there are regions which are difficult to cover well. There may be homogenous image regions, which could be sufficiently represented by using large blocks, while in the high contrast regions, smaller blocks might be fruitful to achieve desired image quality.[5,6,13]

In the QT, a parent node denotes image and its four quadrants are symbolised by four child nodes. The QT comprises of a number of internal nodes and a number of leaf nodes. Leaf nodes correspond to the homogenous/uniform quadrants (one where all the pixels have the same colour), whereas internal nodes correspond to heterogeneous/no uniform quadrants (do not have the same colour in all their pixels; it is an interior node). The size of the QT is directly proportional to the complexity of the images: the more complex the image, the greater the QT size and *vice versa*.[7,14]

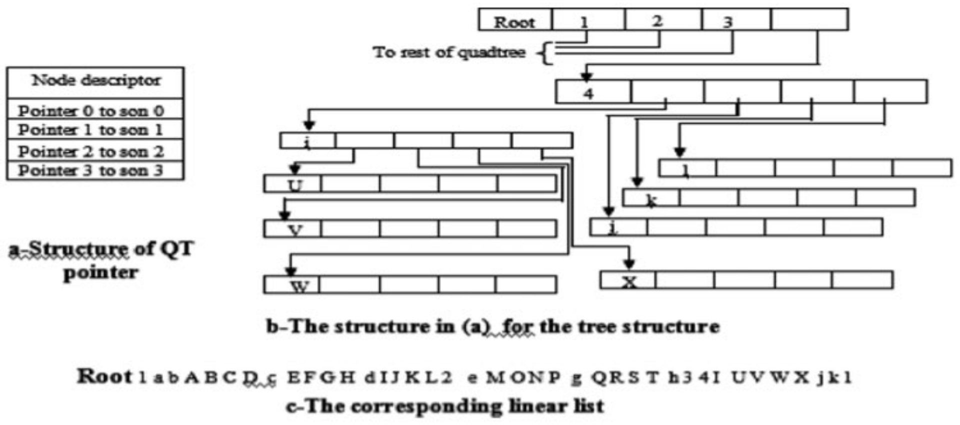
During segmentation, the necessary parameters are computed for an image block; the parameters values are compared with the thresholds, the resulting homogenous/heterogeneous criterion is determined usually by checking the distribution of certain threshold and the procedure is repeated on the next block. The apparent way of approaching the image representation problem via a QT is to know the background regions where the image detail varies slowly and large areas could be coded 'at a stroke'. Thus, some kind of activity test needs to be applied to successive regions in the QT hierarchy to indicate the need for further sub-division.[8]

The further paper is organized as follows. The section on 'Top-down and bottom-up partitioning strategy' shows the proposed strategy of the QT. The section on 'Types of QT data structure' explains the type of the QT. The section on 'Implementation of the QT' presents the proposed implementation of the QT techniques by the flowchart and the structure, and also explains how to solve the problem of non-square images. The conclusion of this paper is drawn in the section on 'Conclusion'.

[1]Department of Computer Science, Isra University, Amman, Jordon
[2]MIS Department, College of Business Administration, Salman bin Abdul Aziz University, Al Kharj-Riyadh, Saudi Arabia
[3]College of Applied Studies and Community Services, King Saud University, Riyadh, Saudi Arabia
[4]College of Engineering and Computer Sciences, Salman bin Abdul Aziz University, Al Kharj-Riyadh, Saudi Arabia
[5]Kulliah of Information and Communication Technology, International Islamic University, Malaysia, Kuala Lumpur, Malaysia; Faculty of Computing, Asia Pacific University, Kuala Lumpur, Malaysia

*Corresponding author: Amjad Rehman, MIS Department, College of Business Administration, Salman bin Abdul Aziz University, Al Kharj-Riyadh, Saudi Arabia; email: ar.khan@sau.edu.sa

© 2014 RPS
Received 26 March 2013; accepted 26 September 2013
DOI 10.1179/1743131X13Y.0000000063


Node descriptor
Pointer 0 to son 0
Pointer 1 to son 1
Pointer 2 to son 2
Pointer 3 to son 3
a-Structure of QT pointer
Root 1 2 3
To rest of quadtree
4
i
j
k
l
U
V
W
X
b-The structure in (a) for the tree structure
Root 1 a b A B C D c E F G H d I J K L 2 e M O N P g Q R S T h 3 4 I U V W X j k l
c-The corresponding linear list

1 Regular quadtree[10] and two types of quadtree representation, pointer-based representation or regular QT, and pointer-less representation or linear QT: (a) structure of QT pointer; (b) the structure in (a) for the tree structure; (c) the corresponding linear list

Top-down and bottom-up partitioning strategy

QT is typically represented by top-down or bottom-up methods.[8,9,15] Top-down construction algorithm is first made to look whether a single leaf can represent the entire block or whether it must be divided into four sub-blocks. The same decision is made on the entire divided block that must be applied to its four sub-blocks as well, Each time, a block is checked to decide whether it needs further division or not, and so on. Bottom-up construction algorithms consist of decisions to merge. Accordingly, similar features blocks are merged to build a larger one; this process of block merging continues to build a larger block of same features. This approach builds a complete quadrant tree by assuming that all quadrants are non-uniform. Each time, a quadrant is evaluated for uniform features and four nodes corresponding to its four quarters are deleted from the QT. This strategy proceeds from bottom to root node.[10]

Types of QT data structure

Shukla *et al.*[10] employed two types for QT representation detailed as below.

Pointer-based QT representation (regular QT)

The pointer-based QT representation or regular QT is implemented as a tree and has several advantages over other types of QTs. This representation is a tree structure with pointers to refer to nodes and for descriptor details. Figure 1a and b shows this kind of representation; Fig. 1a shows the structure of the node descriptor pointing to the four sons and Fig. 1b shows the representation of the tree. This type of representation provides an efficient means for navigating within the QT, so the set of operations (union, intersection and logic difference), or connected component labelling can be performed efficiently. Although a pointer structure may simplify any operation on the tree and speed up the access to its leaves, the memory requirements are unacceptable. As a result, the pointer-less QT representation got a considerable amount of interest.

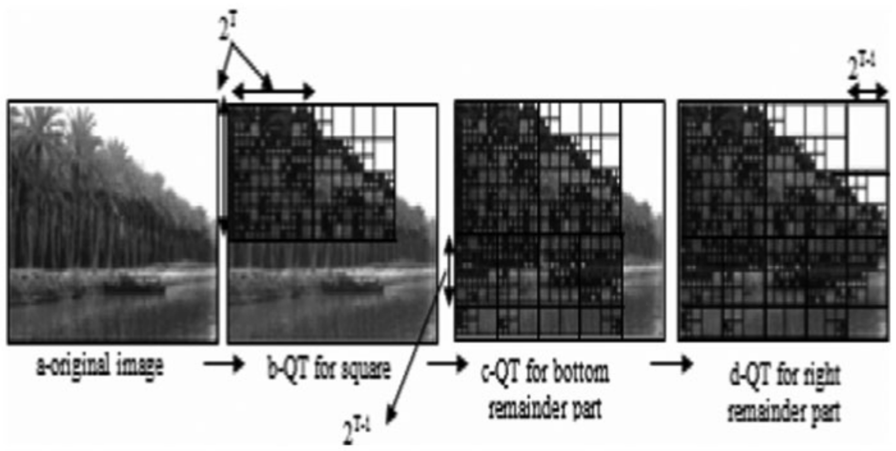
Pointer-less representation (linear representation)

Although it is the most natural way to represent a QT, the regular representation is inefficient as described by Shukla *et al.*[10] This data structure is in the form of a linear list consisting of the QT nodes in some traversed order. In this form of representation, the nodes are ordered so that the QT is in son's nodes order. Each son immediately follows its parent node. Figure 1 shows the linear list of the tree. Information within the nodes is the same as that stored in the node descriptor of the regular QT. From this ordered list, it is shown that certain operations could be efficiently carried out upon the represented image. The space requirements for such a data structure are obviously much less than for the regular QT. The nodes in quadrant can only be visited in the order in which they are initially stored. The pointer-less QT representation has gained an increasing use since it partially overcomes the previous two problems in many new researches. This representation can be classified into two groups: initial step represents the image's nodes traversed in the QT and next step represents desired image as a group of terminal nodes.

Implementation of QT

Non-squared images are the main problem in image segmentation by the QT,[11] so Figs. 2 and 3 show the proposed pre-partitioning process to solve this problem. In this new process, the image is partitioned into tiles, which involves the handling of margin regions (i.e. some remainder parts existed on the right and bottom sides of the partitioned image). To solve this problem, the horizontal and vertical remainders are handled by implementing separate partitioning procedure. Tiles help the user to decode the part of the image (region of interest). After segmenting the image into tiles, the subdividing process is repeated recursively with each tile until the minimum size of the block is reached.[12,16]

Figure 3 illustrates the division process of QT: the process starts with the squared part of the image, then the vertical/horizontal parts are divided into $2^{T-1}\times 2^{T-1}$, $2^{T-2}\times 2^{T-2}$, …, $2^2\times 2^2$, where 2^T is the size of partitioned blocks. The flowchart of the QT procedure (QThomogenity) is exhibited in Fig. 4, where the input represents the image, and the output, which is the structure tree, represents the dynamic segmentation of the image. This segmentation depends on the uniformity of the image. The main steps for the QT procedure in our programming are:



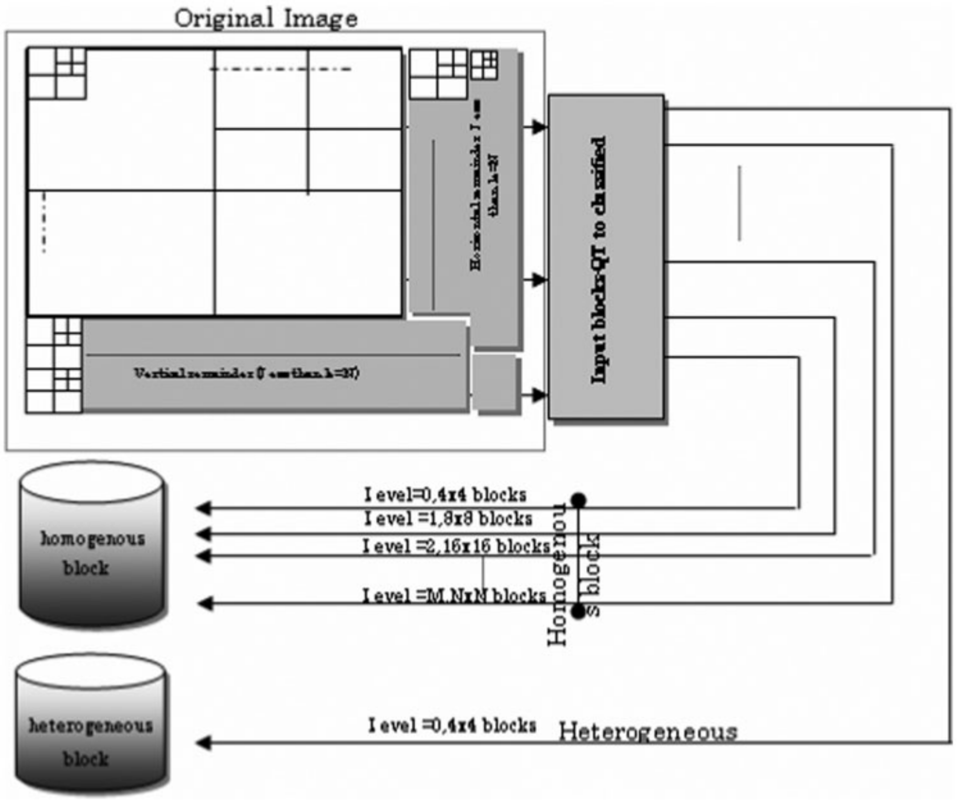
2 Quadtree pre-partitioning process for the pre-image coding process

1. Check-homogeneity: the region of the leaf block is checked pixel by pixel to find the region uniformity.
2. QThomogeneity: this procedure is a recursive technique, which divides the heterogeneous region into four equal size sub-regions.
3. Set P (homogenous/heterogeneous): a pointer is set to indicate the uniformity (homogenous/heterogeneous) to the input region.

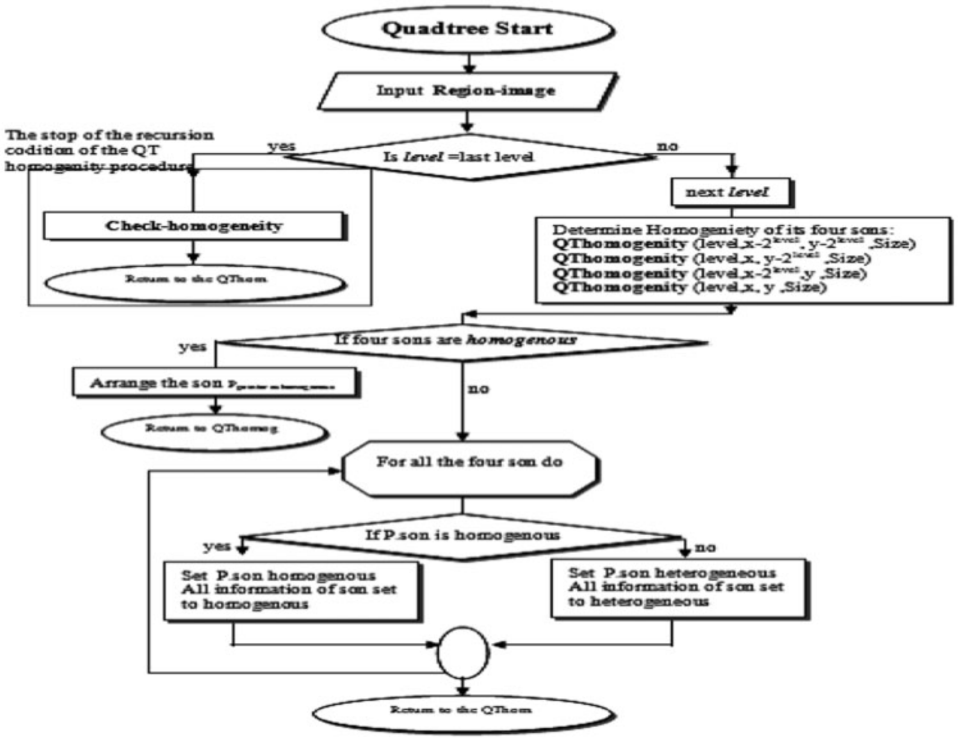
The flowchart in Fig. 4 is started with partitioning the input image into blocks of max.-size, then for each block, the homogeneity level is determined, where the program tests the four sub-blocks, and checks whether the block is a leaf region (no more subdivision, level=0), or not. When the block is not a leaf, then it is divided into four sub-blocks by applying QT procedure (recursion techniques). If any of the four sons is homogenous, then it will set the information of the subdivided region as homogenous point; otherwise, it proceeds by dividing it into sub-blocks. Building the link list structure with this process saves the information of these sons during the recursion process. Each son should point to the homogeneity type of its corresponding region. The process of subdividing a block is illustrated in Fig. 5. The first block is ($x-2^{\text{level}}$,$y-2^{\text{level}}$,x,y), which is called the NW, the second (x,$y-2^{\text{level}}$,$x+2^{\text{level}}$,y) represents the NE; the third is ($x-2^{\text{level}}$,y,x,$y+2^{\text{level}}$) which corresponds to SW; and the last sub-block is (x,y,$x+2^{\text{level}}$,$y+2^{\text{level}}$) which is equal to SE.

The previous process is a bottom-up construction algorithm. The algorithm begins with the sub-block (son block) size and if all relevant sub-blocks have common homogeneity, then they should be combined to produce parent (larger) block. Figure 6 shows the bottom-up process; the bottom-up uses the homogeneity criteria to merge son blocks to construct the parent block.

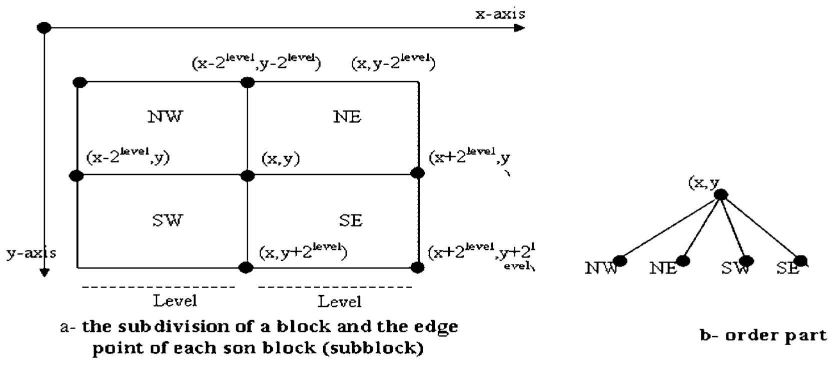
Tables 1 and 2 show the data structure that has been used to save the attributes of the subdivided blocks. The



3 Solving the problem of unsquared image


Quadtree Start
Input Region-image
Is Level =last level
yes
no
The stop of the recursion codition of the QT homogenity procedure
Check-homogeneity
Return to the QThom
next Level
Determine Homogeniety of its four sons:
QThomogenity (level,x-2^level, y-2^level ,Size)
QThomogenity (level,x, y-2^level ,Size)
QThomogenity (level,x-2^level,y ,Size)
QThomogenity (level,x, y ,Size)
If four sons are homogenous
Arrange the son
Return to QThomog
For all the four son do
If P.son is homogenous
Set P.son homogenous All information of son set to homogenous
Set P.son heterogeneous All information of son set to heterogeneous
Return to the QThom

4 The flowchart of QThomogenity procedure

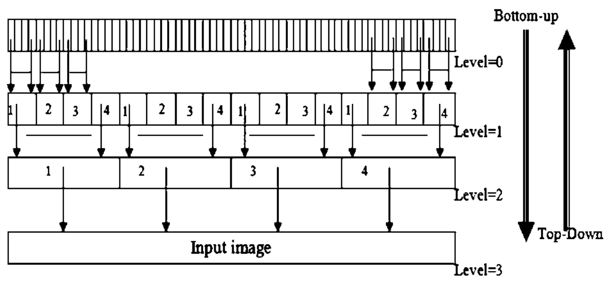

x-axis
y-axis
NW
NE
SW
SE
(x,y)
Level
Level
a- the subdivision of a block and the edge point of each son block (subblock)
b- order part

5 Quadtree block

data structure in Table 2 is used to register the information of each son (sub-block) and father (parent block). This structure is then transmitted to the decoder. The first structure (Table 1) is used in the second structure (Table 2). Algorithm 1 illustrates the use of these two structures. When the check-homogeneity procedure is used by the algorithm, it returns the D.type (D is structure-pair), then the D.type is referred to the colour of the block (D.Grey set to zero for homogeneity, and D.Po is NULL), since it represents only this block.

Algorithm (1) QT homogeneity procedure

Input the region of the image; Max-depth: maximum depth of the partitioning, level: determine the depth division



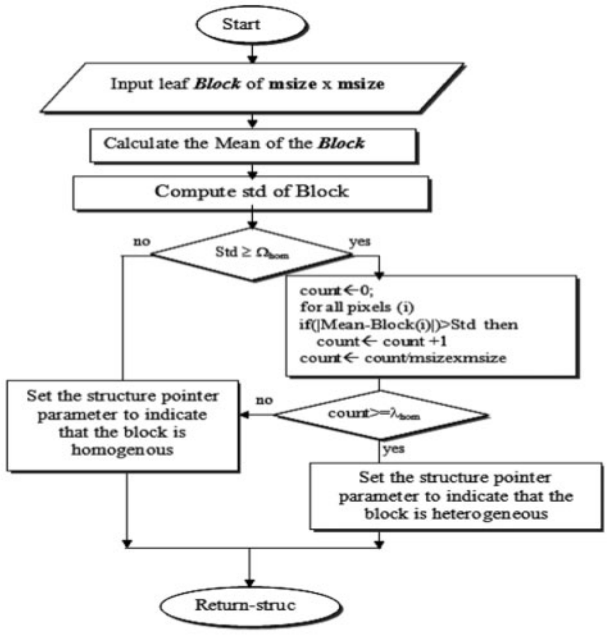
6 Bottom-up and top-down structures

Table 1 Structure L-node

N	Name	Comment
1	Type	Buffer for grey value
2	Grey	Homogeneity
3	Po	Pointer structure of L-node

Table 2 Structure pair

N	Name	Comment
1	ATT	Sonblock=0, parentblock=1
2	Father	Pointer structure of L-node
3	Son	Array of pointer structure of L-node
4	Colour	Buffer for grey value

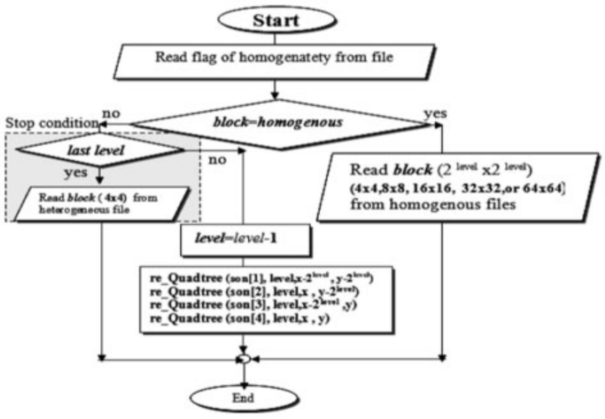

Start
Input leaf Block of msize x msize
Calculate the Mean of the Block
Compute std of Block
no
Std ≥ Ωhom
yes
count←0;
for all pixels (i)
if(|Mean-Block(i)|)>Std then
count← count +1
count← count/msizexmsize
Set the structure pointer parameter to indicate that the block is homogenous
no
count>=λhom
yes
Set the structure pointer parameter to indicate that the block is heterogeneous
Return-struc

7 Changes of Ω and λ in homogenous function

Output pointer that save the constructed tree (mapped QT), the stream of homogenous and heterogeneous blocks

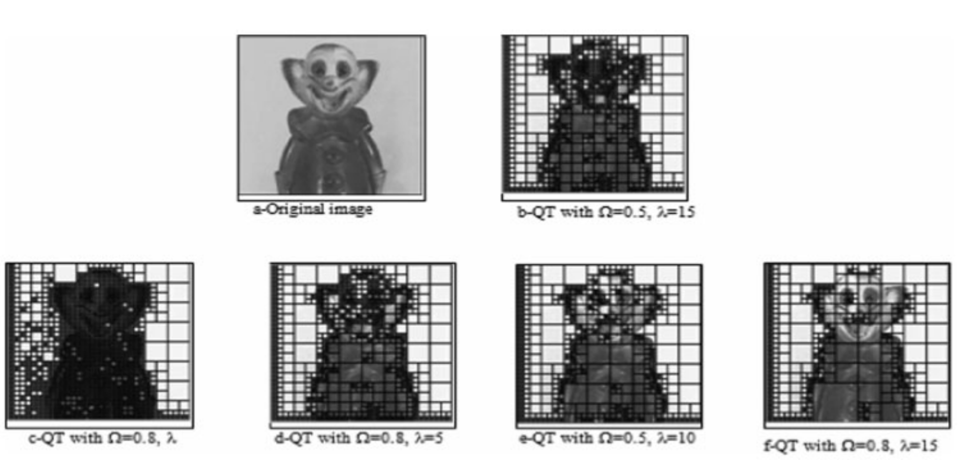
Variables x,y,i: index; P: pointer structure of pair for four node save; D,q,r,m: temporary structure pair

```
Procedure Begin Input the Region-image(x,y)
    If level is the last level=Min_depth then D←Homogenous (x,y,bsize), Return (D)
    Else Level←level-1
      P[0]←QThomogenity(level,x-$2^{level}$,y-$2^{level}$,size);
      P[1]←QThomogenity (level,x, y-$2^{level}$, size)
      P[2]←QThomogenity (level, x-$2^{level}$,y,size);
      P[3]←QThomogenity (level,x,y,size)
        If the four sons have the same color and all are homogenous then return one of the four son
        Else q←Create-node-pointer {*create a new position of type pointer structure *}
          Let q.att←NULL
          for all four children do (i←1 … 3) do
            if p[i] is homogenous then r←Create-node-pointer, Let r.color←p[i].type, All four sons of r is NULL (empty),
                  Let r.att←1, r.father←q, q.son[i]←r
            Else {*not homogenous node*}
                If p[i] is not a leaf node then {*not leaf node*} p[i].father←q
                  Else m←Create-node-pointer, Let m.color←0, and m.father←q
                    For all four sons of m is NULL Let p[i].po←m
                endelse
                Let q.son[i]←p[i].po
            endelse
          endfor
      D.po=q, return(D)
  endprocedure
```

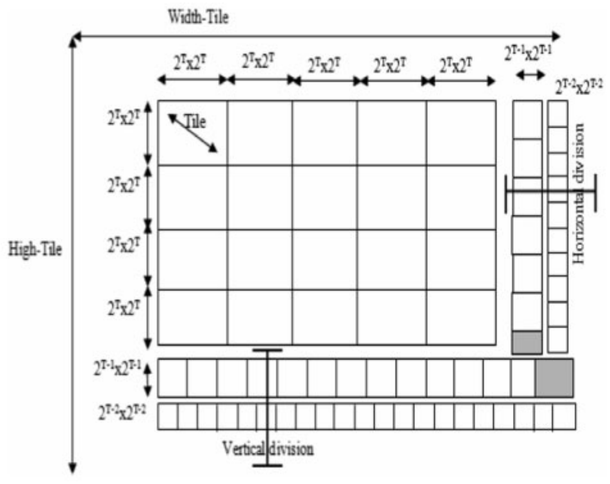

Start
Read flag of homogenatety from file
no
Stop condition
block=homogenous
yes
last level
no
yes
Read block (4x4) from heterogeneous file
Read block (2 level x2 level) (4x4,8x8,16x16, 32x32,or 64x64) from homogenous files
level=level-1
re_Quadtree (son[1], level,x-2level , y-2level)
re_Quadtree (son[2], level,x , y-2level)
re_Quadtree (son[3], level,x-2level ,y)
re_Quadtree (son[4], level,x , y)
End

9 Re-quadtree for the inverse of quadtree partitioning

The homogeneity check is shown in Fig. 7. The first step is to calculate the variance of the input block. Then, variance is compared with threshold (Ωhom) for homogeneity. When the variance exceeds the Ωhom, it means that the block is homogenous; otherwise, it is not. The other criterion that is applied to determine the homogeneity of the block, is by counting the number of block elements (pixels) whose gray values is far from the mean of the tested block by a distance less than variance. Then the process finds the normalisation of the count. The λhom represents the door step to the homogenous block. Figure 8 shows the QT partitioning method applied on the 'clown' image, for different sets of (Ωhom and λhom). Figure 9 shows the flowchart of the inverse QT procedure. In this flowchart, the input comes from different streams (mapping, heterogeneous blocks and homogenous blocks), while the output is represented by one image.



8 The flowchart of QThomogenity procedure



10 Remainder partitioning

It is started with checking a 1-bit flag from the mapping stream (0 for heterogeneous block, 1 for homogenous). If it is homogenous, then it will read the components of the specific block ($2^{\text{level}} \times 2^{\text{level}}$) from the homogenous stream; otherwise, the block is read from the heterogeneous stream. Figure 10 shows the solution for the problem of remainder (horizontal/vertical) part, after dividing the image into a region of $2^{\text{level}-1} \times 2^{\text{level}-1}$ and then constructing the QT. Algorithm (2) exhibits the steps of dividing the remanding bottom part, and the same procedure is followed to handle the right remaining blocks. Figure 11 declares the flowchart wrt (Ppointer, Level, xposition, yposition), which converts the QT pointers into file. The output map file has 0 for homogenous blocks and 1 for heterogeneous blocks.

Algorithm (2): Vertical division for remainder part

Input input image; maximum size of the block: minimum size of the block; width, height: the dimension of input image; Xsize: tile size;

Output mapped contained the map of the image partitioning by the QT, it contains only 0 and 1 to represent the leaf node.

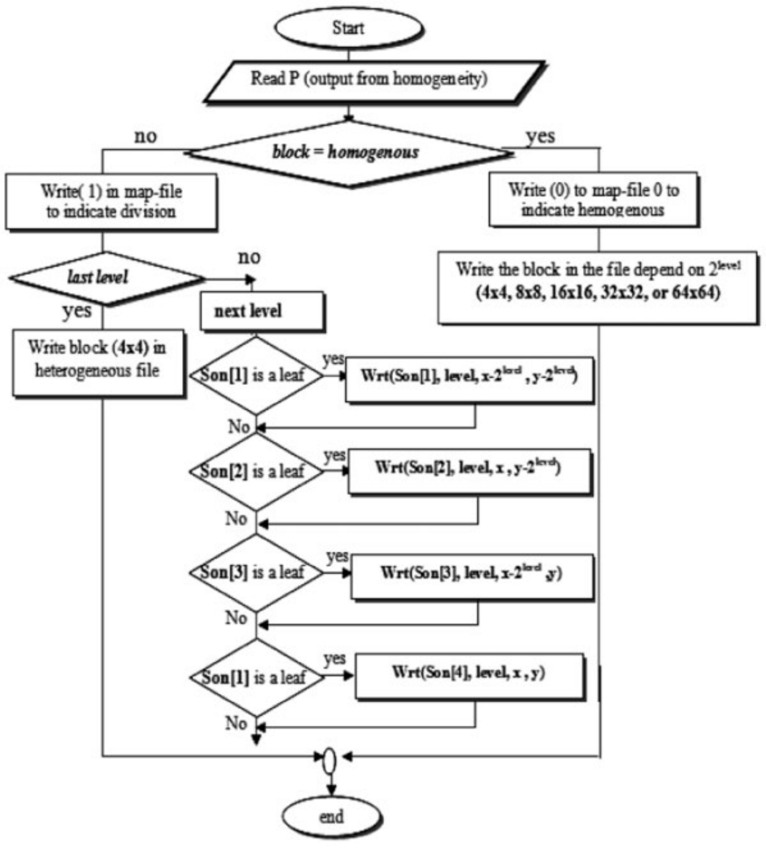
the homogenous blocks; the heterogeneous blocks

Variables noW: number of blocks in the vertical coordinate; restW: remained width from the vertical coordinate

noH: number of blocks in the horizontal coordinate; G: pointer for input data image

P: pointer to save the output map from QT partitioning

```
Procedure Begin
  Let noW←width/Xsize; Let restW←width-(noW*Xsize)
    If there is block to be quad in remainder then
      For all the blocks in remained remainder do
        If restW <> msize-1 then restW←restW-$2^i$; noH←high/$2^i$ {find the number of blocks in the horizontal to be square}
          For all the blocks in the horizontal (j=0 to noH) do
            G=read_block (newW, $2^i$ *j,$2^i$); P=QThomogeneity(i,$2^i$,$2^i$,$2^i$); Wrt (P,i,$2^i$,$2^i$)
          endfor
        endif
      endfor
    endif
endprocedure
```


Start
Read P (output from homogeneity)
block = homogenous
no
yes
Write(1) in map-file to indicate division
Write (0) to map-file 0 to indicate hemogenous
Write the block in the file depend on 2^level (4x4, 8x8, 16x16, 32x32, or 64x64)
last level
no
yes
next level
Write block (4x4) in heterogeneous file
Son[1] is a leaf
Wrt(Son[1], level, x-2^level, y-2^level)
Son[2] is a leaf
Wrt(Son[2], level, x, y-2^level)
Son[3] is a leaf
Wrt(Son[3], level, x-2^level, y)
Son[1] is a leaf
Wrt(Son[4], level, x, y)
No
end

11 'Wrt' recursive procedure

Table 3 The important parameters in the research work

N	Parameters	Comments
1	MaxQT partitioning	Different values (8 × 8–128 × 128) have been analysed
2	MinQT partitioning	Only one vale 4 × 4
3	QT threshold	Two thresholds have been used to have more control to find out homogenous and heterogeneous regions

Conclusion

This paper has presented QT-based efficient approach to region desired information in image segmentation process. The proposed approach explores texture features that present in an image and provides a general framework to discriminate features for image segmentation. The weakness of the existing fixed size partitioning approaches lies in the fact that there are regions which are difficult to discriminate perfectly; however, the proposed QT approach does not need to be of uniform depth as well as it can region the desired information in a mixed image accurately. The selection of parameters values used in the segmentation should be chosen precisely, as these values has a great effect on the segmentation process. Accordingly, Table 3 exhibits the important parameters used in the representation process. As a result, the pointer-less QT representation (B type) gets a considerable amount of interest than the other ones. The A type of representation (regular QT) provides an efficient mean for navigating within the QT and the B type of representation gets a considerable amount of interest. The space requirements for such a data structure are obviously much less than for the A type representation, but the nodes for the quadrant in type B are only visited according to their initial order. Finally, the significance of the proposed approach is elaborated with the help of flowchart and diagrams in the paper.

Acknowledgements

We express our deepest thanks and appreciation to the Deanship of Scientific Research at King Saud University Riyadh KSA for funding this research. We are also thankful to our colleague researchers for their valuable suggestions to improve this work.

References

1. Rad, A. E., Rahim, M. S. M., Rehman, A., Altameem, A. and Saba, T. Evaluation of current dental radiographs segmentation approaches in computer-aided applications. *IETE Tech. Rev.*, 2013, **30**, 210–222.
2. Rehman, A., Saba, T. and Sulong, G. An intelligent approach to image denoising. *J. Theor. Appl. Inf. Technol.*, 2010, **17**, 32–36.
3. Saba, T., Rehman, A. and Elarbi-Boudihir, M. Methods and strategies on off-line cursive touched characters segmentation: a directional review. *Artif. Intell. Rev.*, 2011, doi: 10·1007/s10462-011-9271-5.
4. Rehman, A. and Saba, T. Neural network for document image preprocessing. *Artif. Intell. Rev.*, 2012, doi: 10·1007/s10462-012-9337-z.
5. Saba, T. and Rehman, A. Effects of artificially intelligent tools on pattern recognition. *Int. J. Mach. Learn. Cybern.*, 2013, **4**, 155–162.
6. Rehman, A. and Saba, T. Document skew estimation and correction: analysis of techniques, common problems and possible solutions. *Appl. Artif. Intell.*, 2011, **25**, 769–787.
7. T. Saba, Al-Zorani, S. and Rehman, A. Expert system for offline clinical guidelines and treatment. *Life Sci. J.*, 2012, **9**, 2639–2658.
8. Balmelli, L. Kovacevic, J. and Vetterli, M. Quadtree for embedded surface visualization: constraints and efficient data structures, Proc. IEEE Int. Conf. on *Image processing: ICIP 2009*, Cairo, Egypt, November 2009, IEEE, Vol. **2**, pp. 487–491.
9. Shukla, R., Dragotti, L., Do, M. and Vetterli, M. Improved quadtree algorithm based on joint coding for piecewise smooth image compression, Proc. IEEE Int. Conf. on *Multimedia and expo: ICME 2002*, Lausanne, Switzerland, August 2002, Swiss Federal Institute of Technology.
10. Shukla, R., Dragotti, P., Do, M. and Vetterli, M. Rate-distortion optimized tree-structure compression algorithms for piecewise polynomial images. *IEEE Trans. Image Process.*, 2005, **14**, 343–359.
11. Rehman, A. and Saba, T. Performance analysis of segmentation approach for cursive handwritten word recognition on benchmark database. *Digit Signal Process.*, 2011, **21**, 486–490.
12. Saba, T. Sulong, G. and Rehman, A. Document image analysis: issues, comparison of methods and remaining problems. *Artif. Intell. Rev.*, 2011, **35**, 101–118.
13. Rehman, A and Saba, T. Features extraction for soccer video semantic analysis: current achievements and remaining issues. *Artif. Intell. Rev.*, 2012, doi: 10·1007/s10462-012-9319-1.
14. Rehman, A. and Saba, T. Document skew estimation and correction: analysis of techniques, common problems and possible solutions. *Appl. Artif. Intell.*, 2011, **25**, 769–787.
15. Saba, T. and Altameem, A. Analysis of vision based systems to detect real time goal events in soccer videos. *Appl. Artif. Intell.*, 2013, **27**, 656–667.
16. Saba, T. and Alqahtani, F. A. Semantic analysis based forms information retrieval and classification. *3D Res.*, 2013, **4**, 4.