Overlapping quadtrees for the representation of similar images

M Vassilakopoulos, Y Manolopoulos and K Economou

Overlapping has been proposed for several tree structures. The adaptation of this technique to region quadtrees, along with the necessary algorithms, are presented. The resulting structure is a form of a partial persistent quadtree and can be used to represent sequences of similar binary raster images emerging in various applications such as computer graphics, image processing, geographic information systems, or even satellite pictures. Our motive is to save considerable space, while access time of any of the similar images remains unaffected. Experimentation on random image forms exhibits very promising results in the space reduction achieved.

Keywords: Quadtrees, overlapping data structures, representation of images

The idea of using the same memory region for different purposes to keep space requirements as low as possible is fundamental in the design of efficient algorithms. Overlapping in trees (and generally in linked structures) is a further refinement, where a node is a member of more than one structure: consider a sequence of tree structures representing a gradually changing data set (each tree corresponds to an instance of the set). If we discover common substructures between each tree and its predecessor in the sequence, then we may keep these substructures only once in memory even though they belong to paths of both trees. In case there are no such substructures we have two completely isolated trees. If the two data set instances are identical, so are the trees (the two pointers to the roots of the trees remain distinct, although they point to the same node). The saving of space through common substructures may involve more than two consecutive trees. A substructure may be pointed to by the next tree, the tree after next and so forth, resulting in high space requirement reduction.

It should be clear that this sharing of substructures is transparent to the algorithms accessing each tree. This means that the access time of any node has no time overhead because of overlapping. One might argue that another technique might reduce space requirements even more. However, overlapping succeeds in giving substantial a reduction while keeping access time unaffected.

Mullin[1] suggested using two *B*-trees, the first storing the initial information and remaining unmodified, while the second is used for storing changes. Extending the above idea, overlapping tree substructures have been proposed[2], as an organization for storing similar text files used by an editing system. The technique has been adapted in *B*-trees[3], where the necessary algorithms for the manipulation of the overlapped *B*-trees are presented, as the basis for the representation of a collection of similar files. Moreover, it has been adapted to B^+ trees[4] to efficiently store temporal data. In the latter work experimentation and analysis of overlapped B^+ trees, concerning extra space during updates, was done. Use of common substructures in quadtrees has been proposed[5] as one of the advantages that functional languages (and especially Miranda) provide over procedural languages for graphics programmers.

The structure of the paper is as follows. In the following section we present the structure of quadtrees and introduce the technique of overlapping. The next section gives the necessary Pascal-type definitions and the function performing quadtree overlapping. In the 'experimentation' section the results of a simulation on binary random images are reported. The discussion summarizes our conclusions on the achieved space reduction.

OVERLAPPING QUADTREES

There are numerous versions of quadtrees structures applied in various instances[6]. The region quadtree is a

Division of Electronics and Computer Engineering, Department of Electrical Engineering, Aristotelian University of Thessaloniki, Thessaloniki 54006, Greece

Paper received: 13 May 1992: revised paper received: 28 October 1992

hierarchical data structure that represents binary raster images with $2^n \times 2^n$ pixels, for some integer n. Such a quadtree is a degree four tree of height $\leq n$. Each node corresponds to a square array of pixels. If all of them have the same colour (black or white), the node is a leaf of that colour. Otherwise, the square array is divided in four quadrants (square subarrays), the node is grey and has four children, one for each quadrant, that are labelled *nw* (northwest), *ne* (northeast), *sw* (southwest), and *se* (southeast).

We present this intuitive definition in more formal terms:

Definition *Consider the numbers $n, k, x, y \in \mathbf{N}$, such that $k \leq n$ and $x, y < 2^n$, where x, y are proportional to 2^k. Consider also the binary array $I[0 \ldots 2^n-1, 0 \ldots 2^n-1]$. As $Q_n(I, x, y, k)$ we denote the tree that represents the subarray $I[x \ldots x+2^k-1, y \ldots y+2^k-1]$ and consists of*

- *one external node, called a black (white) node, if every element of its subarray equals 1 (0), or otherwise of*
- *one root node, called a grey node, and its 4 children, the trees*
  - $Q_n(I, x, y, k-1)$,
  - $Q_n(I, x+2^{k-1}, y, k-1)$,
  - $Q_n(I, x, y+2^{k-1}, k-1)$ *and*
  - $Q_n(I, x+2^{k-1}, y+2^{k-1}, k-1)$

The quadtree for I is the tree $Q_n(I, 0, 0, n)$.

The similarity of consecutive image data in many applications such as computer graphics, image processing, geographic information systems (GIS) of satellite pictures, along with the large amount of space needed for representing each image, even by using the popular space saving quadtrees, make an excellent candidate for overlapping. For a good treatment of the above and related issues, see Samet[7,8]. Note that some of these applications involve colour images. The technique presented can be easily transformed to handle colours.

Consider the similar images of Figures 1a and b and the respective quadtrees of Figures 1c and d. If overlapping is applied the resulting structure appears in Figure 1e. Note that when there is a different substructure (which might even be a different pixel for a certain pair of coordinates), the total path to this substructure is stored. This guarantees that the access time for any tree node remains unaffected.

As has already been mentioned, overlapping may be extended to a sequence of similar images. We will be calling this sequence of overlapped quadtrees an *overlapped family*. There, each tree is overlapped with the previous one in the sequence. Under the assumption that consecutive images of this sequence differ slightly to each other, we get high space reduction. Note that there may be overlapped structures that are shared among a large number of consecutive trees.

Recall that a persistent data structure is one that retains its past versions. (A new version is created every time a group of changes is applied to the structure.) There is access ability for any of these versions. Update ability is limited to the latest version if we have partial persistence, or exists for all versions, if we have full persistence. This idea, along with general techniques, has been studied by Driscoll *et al.*[9]. We may view an overlapped family of quadtrees, along with the quadtree of the latest image not yet inserted in the family, as a form of a persistent quadtree, where there is access to any older version of the tree and update ability to its latest version. When a new tree (a new version) appears, this latest version is inserted in the family and may no longer be updated. The difference between this form of persistence and that described by Driscoll *et al.*[9] is that at the expense of some space overhead the access times remain unaffected. The family method could be characterized as partial persistence with time benefit.

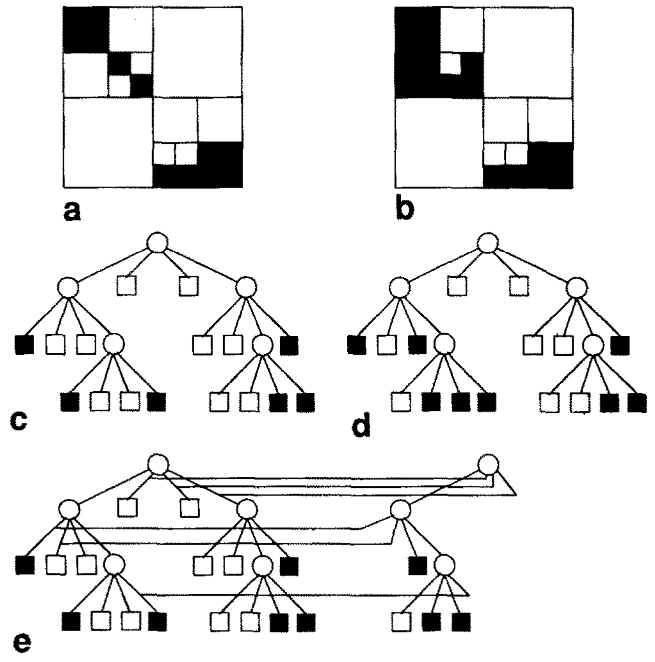

Figure 1. (a), (b) two similar 8×8 images; (c), (d) corresponding quadtrees; (e) the overlapped stucture

ALGORITHM

In this section an algorithm that performs overlapping of the latest quadtree of an overlapped family and a quadtree representing the latest image instance is described. A similar algorithm that performs map overlaying is described by Burton *et al.*[10].

The type of the quadtree node is defined as a self-referenced Pascal record in Figure 2. Each quadtree node has a field which contains the number of pointers currently referencing the node. All nodes with a reference counter greater than 1, together with all

```
QUADTREE = ^QUADREC;
DIR = (nw,ne,sw,se);
COL = (GREY,BLACK,WHITE);
CHILDREN = array [DIR] of QUADTREE;
QUADREC = record
            colour: COL;
            ch: CHILDREN;
            ref: integer;
end;
```

Figure 2. Quadtree node definition

descendants of such nodes, constitute shared information. This counter enables us to perform deletion of a particular quadtree from the overlapped family. We dispose of the node only if it has a zero reference counter. Note that this data structure seems inefficient, since each leaf (black or white node) would have four unused (nil) pointers and a reference counter. It is used here for the sake of simplicity. A minor improvement would be to have only one universal black leaf and one universal white leaf. Every node pointing to black or white leaves would point to the respective universal node. Of course, the universal leaves should never be deleted. Using this improvement, the space overhead induced will only be two leaf nodes. Even more efficient and sophisticated variants are possible, according to the possibilities offered by the programming language used.

We assume that the transformation of the latest raster image to the respective quadtree has already been done. Nevertheless, we must note that an algorithm could be devised that would accept a quadtree and a raster image as input, and produce the overlapped structure as output (without creating an intermediate quadtree for the raster image).

The routine that performs overlapping of two consecutive quadtrees is given in Figure 3. It is called with two node-pointers as parameters, one to each tree. If both the referenced nodes are leaves of the same colour, the second one is disposed (its pointer then points to the first node) and the function returns a true value. If both the referenced nodes are grey, the function recursively calls itself. The call takes place once for every corresponding child pair of the two grey nodes. If all the four return-values are true, the second node is disposed and its pointer is changed to point to the first node. Note also that if the one tree has a leaf node in a position where the other tree has a grey node, no further descending in the latter tree is performed. This means that the algorithm executes in time which is asymptotically bounded upwards by the minimum number of nodes between the number of nodes each tree has.

```
function OverlapQuadtree (var a,b:
                              QUADTREE):Boolean;

begin
  if (a^.colour <> GREY) and
  (b^.colour <> GREY) then
  (* if a^ and b^ are leaves *)
  begin
    if a^.colour = b^.colour then
    (* and of the same colour *)
    begin
      Dispose(b);
      b := a;
      a^.ref := a^.ref + 1;
      OverlapQuadtree := true
    end
    else
      OverlapQuadtree := false;
  end
  else
    if ((a^.colour = b^.colour) and
    (b^.colour = GREY)) then
    (* if both are not leaves *)
    begin
      if (OverlapQuadtree(a^.ch[nw], b^.ch[nw])
      and
      OverlapQuadtree(a^.ch[ne], b^.ch[ne])
      and
      OverlapQuadtree(a^.ch[sw], b^.ch[sw])
      and
      OverlapQuadtree(a^.ch[se], b^.ch[se]))
      then
      begin
        Dispose(b);
        b := a;
        a^.ref := a^.ref + 1;
        OverlapQuadtree := true;
      end
      else
        OverlapQuadtree := false;
    end
    else
      OverlapQuadtree := false;
end;
```

Figure 3. Routine that performs overlapping

EXPERIMENTATION

We denote a class-k image to be a square array of size $2^k \times 2^k$. This image is said to be x% black if it has been created by an algorithm that sets each pixel to black with probability $x/100$, and to white with probability $1-x/100$. Figures 4a–c demonstrate three class-4 images, which are 50%, 25% and 6% black, respectively. Moreover, we denote a class-k image B (for some integer k) to be y% different to a class-k image A if B has been created by an algorithm that takes as input image A and produces image B by flipping A's pixel values with probability $y/100$ and by copying A's pixel values with probability $1-y/100$. The new image, as can be easily seen, is $x+y-2xy$% black and $1-x-y+2xy$% white. Figure 5 demonstrates a sequence of 9 class-3 images where each one is 10% different to its predecessor (the starting image is 25% black). Note that the resulting images tend to reach 50% black. Such a sequence of images might, for example, model an expanding colony of bacteria. Suppose we are given a sequence of n overlapped quadtrees (n is a large positive integer). We define the overlapping percentage, Ov%, as the rate:

$$\frac{\sum_{i=2}^{n} \textit{number of overlapped nodes of tree i and tree } i-1}{\sum_{i=1}^{n} \textit{number of all nodes of tree i (overlapped or not)}}$$

The denominator contains the number of nodes of the first tree in the sequence. Although we cannot talk about space reduction for this tree (it is not overlapped, since there is not a previous tree in the sequence), the overlapping percentage converges to the space reduc-

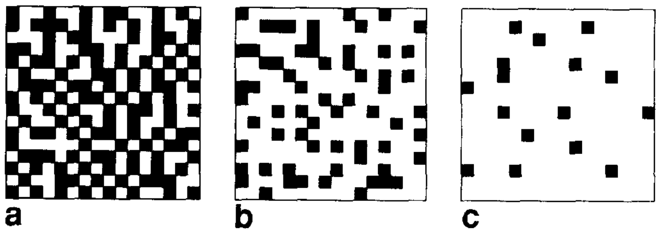

Figure 4. Class-4 images with (a) 50%, (b) 25%, (c) 6% black

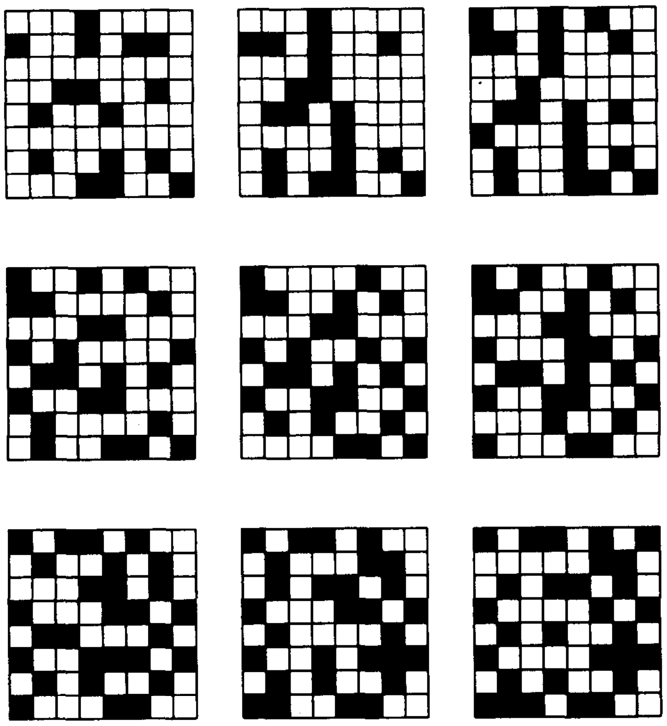

Figure 5. Nine class-3 images differing from each other by 10%. The first one is 25% black

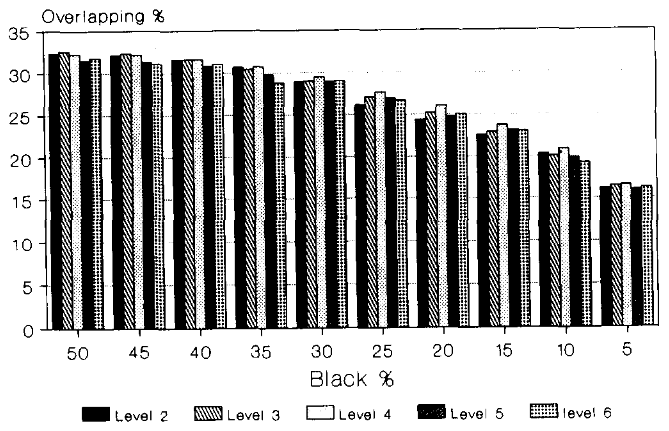

Figure 6. Histogram created from the first experiment

tion percentage for a large enough *n* (then, the number of nodes of the first tree is not important).

We performed two experiments. The first concerned random images of the same class and black percentage that we created statistically independent to each other. Each test was performed 100 times for a family of n = 100 random trees. Every overlapping percentage is the mean value of the 100 executions. We performed tests for the classes 2, 3, 4, 5 and 6, and the black percentages 5% to 50% in steps of 5%. We neglected the rest black percentages because the results are expected to be identical to the results of their complementary percentages (e.g. black 60% would be identical to black 40%).

The aim of this experiment was to investigate the connection of the overlapping percentage to different black-white analogies of statistically independent images. The results of this experiment appear in Table 1 and Figure 6. It is evident that space reduction gets higher for black-white analogies closer to half-by-half. For every class, we reach the maximum overlapping percentage value when black percentage equals 50%. However, it should be noted that we get close to this maximum for images within black range 50 ± 15%.

The second experiment concerned class-3, class-5 and class-6 images that are different to each other by 1, 2, 3, 5, 7, 10, 15 and 20%. Each class-3 and class-5 test was performed 100 times for a family of $n = 100$ random trees, while each class-6 test was performed 10 times for a family of $n = 100$ random trees. The class-6 repetitions were fewer because of the extra processing time required. We performed tests for images that were 10, 20, 30, 40 and 50% black. The aim of this test is to investigate the space reduction in a realistic environment of quadtrees overlapping. The results of this experiment appear in Tables 2, 3 and 4 and in Figure 7 and 8 (we do not present a bar chart for class-6 images since this would be alike the one for class-5 images). It seems that the overlapping percentage is linearly related to the image difference percentage and that it is independent of the black-white analogy.

Note that in both experiments performed the resulting space savings do not change significantly as the class of the images gets higher. This is because overlapping takes place mainly between small image parts (small subtrees), even when our images are very large. We claim that even for higher classes space reduction would be at similar rates.

DISCUSSION AND CONCLUSION

We have seen that overlapping, when applied to quadtrees that represent a gradually changing image, produces high space reduction while retaining the access times of any image instance unaffected. The whole technique is based on a rather simple and computationally efficient recursive algorithm.

The experimentation performed justifies use of

Table 1. Results of the first experiment

Black	*Overlapping percentage*				
percentage	*(Class = 2)*	*(Class = 3)*	*(Class = 4)*	*(Class = 5)*	*(Class = 6)*
50	32.30	32.52	32.23	31.49	31.77
45	32.11	32.28	32.12	31.30	31.05
40	31.58	31.56	31.57	30.81	31.00
35	30.67	30.40	30.69	29.80	28.85
30	28.94	29.04	29.49	28.92	29.01
25	26.16	27.14	27.70	26.98	26.73
20	24.37	25.25	26.06	24.77	25.10
15	22.54	22.89	23.74	23.14	23.05
10	20.33	20.06	20.92	19.86	19.30
5	16.13	16.43	16.56	15.99	16.22

Table 2. Results of the second experiment where class = 3

Difference	*Overlapping percentage*				
percentage	*(Black = 50%)*	*(Black = 40%)*	*(Black = 30%)*	*(Black = 20%)*	*(Black = 10%)*
1	95.84	95.76	95.74	95.67	95.50
2	93.07	93.07	93.04	92.77	92.70
3	90.52	90.44	90.40	90.41	90.25
5	86.01	86.11	86.12	86.12	85.97
7	82.08	82.13	82.21	81.84	81.79
10	76.73	76.72	76.62	76.50	76.72
15	68.89	68.87	69.35	68.92	68.60
20	60.08	62.18	62.32	61.85	62.15

Table 3. Results of the second experiment where class = 5

Difference	*Overlapping percentage*				
percentage	*(Black = 50%)*	*(Black = 40%)*	*(Black = 30%)*	*(Black = 20%)*	*(Black = 10%)*
1	91.54	91.48	91.27	90.70	89.71
2	88.84	88.80	88.53	87.91	86.82
3	86.49	86.39	86.09	85.54	84.68
5	82.22	82.19	81.92	81.47	80.86
7	78.44	78.39	78.20	77.79	77.30
10	73.30	73.26	73.08	72.79	72.43
15	65.85	65.79	65.80	65.43	65.29
20	59.40	59.31	59.20	59.06	58.79

Table 4. Results of the second experiment where class = 6

Difference	*Overlapping percentage*				
percentage	*(Black = 50%)*	*(Black = 40%)*	*(Black = 30%)*	*(Black = 20%)*	*(Black = 10%)*
1	90.74	91.17	91.44	90.27	90.06
2	89.12	88.33	87.95	88.00	86.43
3	86.09	85.88	86.26	86.10	85.34
5	82.20	82.09	81.60	81.14	79.89
7	79.00	78.03	78.02	77.97	77.20
10	74.02	73.24	73.32	72.27	73.22
15	65.35	65.91	66.08	66.00	65.01
20	59.46	59.01	59.02	58.57	59.21

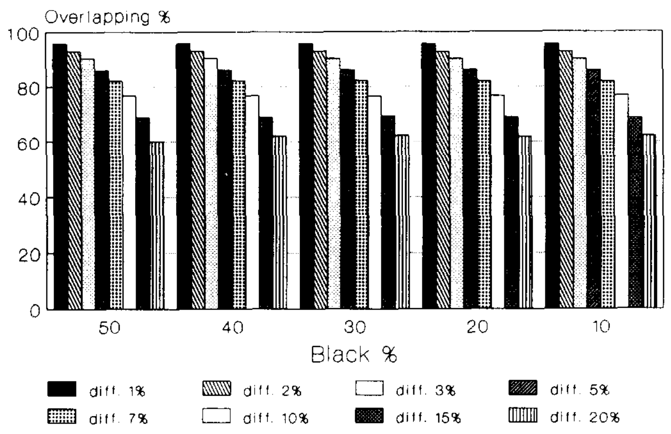

Figure 7. Histogram created from the second experiment (class = 3)

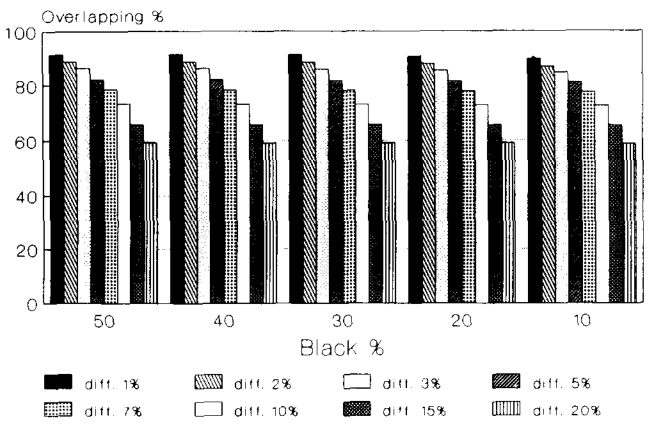

Figure 8. Histogram created from the second experiment (class = 5)

overlapping quadtrees to a broad scope of applications, where a sequence of very similar image data has to be stored and processed. Our experimental conclusion is twofold:

- space reduction of consecutive independent random images, each one having a constant black-white ratio, is higher when the two colours are almost equiprobable and is over 30%
- space reduction of consecutive gradually changing images depends linearly on the difference percentage. More specifically, when this ratio lies in the range 0-10%, this reduction is as high as 80-90%.

Another useful technique for storing gradually changing class-n images would be to temporarily store a sequence of 2^n such images in raster form, creating a $2^n \times 2^n \times 2^n$ pixel cube, and then to build a region octree (the extension of the region quadtree in space), such that one of its three dimensions would be used to discriminate between different images (i.e. one of its three axes would be a time axis).

This method is expected to achieve singnificant compression. Besides, the whole structure is build once for all the represented images, while the technique of overlapping builds a separate quadtree and performs the overlapping algorithm once for every image.

Nevertheless, there are a number of points against the octree method. First, this method works for exactly 2^n images. For a large enough n, even under substantial compression, it is not possible to keep the whole tree in main memory. Besides, there must be enough main memory to hold the 2^n raster images before the tree is built, or complicated operations and time consuming disc accesses must be used. Moreover, single images can not be added or deleted to the sequence, and last but not least, the quadtree access algorithms can not be applied unchanged; then, we cannot guarantee that the access times remain unaffected.

We believe that further experimentation on overlapping quadtrees, concerning a gradually changing contour (found in applications such as animation), might prove at least as promising. This is because we would then deal with large unicolour regions which are slightly shifted. Although the contour would change significantly, most of the unicolour quadrant blocks would remain unchanged. An analysis of the storage efficiency of overlapping quadtrees is presented by Vassilakopoulos *et al.*[11]. Besides, by extending methods presented by Burton *et al.*[12] and Faloutsos[13], further analytical insight to the method could be achieved.

ACKNOWLEDGEMENTS

The first author, who is a postgraduate scholar of the State Scholarship Foundation of Greece, wishes to thank this foundation for its financial assistance.

REFERENCES

1 **Mullin, J K** 'Change area B-trees: a technique to aid error recovery', *Computer J.*, Vol 24 No 4 (1981) pp 367–373

2 **Burton, F W, Huntbach, M M and Kollias, J G** 'Multiple generation text files using overlapping tree structures', *Computer J.*, Vol 28 No 4 (1985) pp 414–416

3 **Burton, F W, Kollias, J G, Kollias, V G and Matsakis, D G** 'Implementation of overlapping B-trees for time and space efficient representation of collections of similar files', *Computer J.*, Vol 33 No 3 (1990) pp 279–280

4 **Manolopoulos, Y and Kapetanakis, G** 'Overlapping B+ trees for temporal data', *Proc. 5th JCIT Conf.*, Jerusalem, Israel (1990) pp 491–498

5 **Burton, F W and Kollias, J G** 'Functional programming with Quadtrees', *IEEE Softw.*, Vol 6 No 1 (1989) pp 90–97

6 **Samet, H** 'The Quadtree and related hierarchical data structures', *ACM Comput. Surv.*, Vol 16 No 2 (1984) pp 187–260

7 **Samet, H** *The Design and Analysis of Spatial Data Structures*, Addison-Wesley, Reading, MA (1990)

8 **Samet, H** *Applications of Spatial Data Structures: Computer Graphics, Image Processing and GIS*, Addison-Wesley, Reading, MA (1990)

9 **Driscoll, J R, Sarnak, N, Sleator, D D and Tarjan, R E** 'Making Data structures persistent', *J. Comput. Syst. Sci.*, Vol 38 No 1 (1989) pp 86–124

10 **Burton, F W, Kollias, J G and Kollias, V G** 'A general PASCAL program for map overlay of Quadtrees and related problems', *Computer J.*, Vol 30 No 4 (1987) pp 355–361

11 **Vassilakopoulos, M and Manolopoulos, Y** 'Efficiency analysis of overlapped Quadtrees', submitted to *CVGIP: Image Understanding*

12 **Burton, F W, Kollias, J G and Kollias, V G** 'Expected and worst-case requirements for Quadtrees', *Patt. Recogn. Lett.*, Vol 3 No 2 (1985) pp 131–135

13 **Faloutsos, C** 'Analytical result on the Quadtree decomposition of arbitrary rectangles', *Patt. Recogn. Lett.*, Vol 13 No 1 (1992) pp 31–40