CONGRESSUS NUMERANTIUM

VOLUME 81

DECEMBER, 1991

WINNIPEG, CANADA

A Persistent Quadtree to Store Graphic Images

M. K. Zand, H. Saiedian, and H. Farhat

Department of Mathematics and Computer Science
University of Nebraska at Omaha
Omaha, Nebraska 68182-0243
email:zand@unocss.unomaha.edu

May 1, 1991

Abstract

This paper introduce the idea of persistent quadtree. Persistent quadtrees could be used to store those images in which the sum of differences of points between two consecutive images is not more than a fraction of all points. Three data structures are examined for implementation of persistent quadtress. Comparative analysis of space requirement and time performance of these structures is provided.

1 Introduction

Quadtrees are hierarchal structures designed to store two dimensional graphic images [3, 6, and 8]. An ephemeral quadtree represents one frame (instance) of a graphic image at a time and each new frame of object is stored separately. Suppose there are a sequence of k frames, such that the sum of differences of points (pixels) between adjacent frames is not more than a fraction of all points in a frame of the object. We propose a modified quadtree to efficiently store these frames and to maintain direct access to frames for the following operations: Retrieve frame f_n, store new frame f_n, or delete frame f_n. A quadtree structure in which could efficiently store and support such operations is called Persistent Quadtree (PQ-tree). There are two major ways to make a hierarchal structure like quadtree persistent: node copy [1] and path copy [5 and 7]. We have extended the idea of persistent B-tree [9] to quadtrees. To store frame i in this approach only differences between frames i and $i-1$ (if frame $i-1$ exists and modification on $i-1$ is not substantial) and side effects of those modifications are stored.

Our proposed approach doesn't need an *ad hoc* structure or methods to keep the log of modifications. Each frame f_n is accessible individually and there is no need to access previous frames to construct frame f_n. All operations on an ephemeral quadtree are supported in PQ-tree.

2 Quadtree Structure

There are two major data formats to represent graphic images. *Raster Format* first one, models images as a collection of square cells of uniform size called *pixel*. The second format, *Vector Format*, models images as ideal geometric spaces such as *points, lines, segments, polygons*, etc [4].

The term *quadtree* is used to describe a class of hierarchical structures whose common property is that they are based on the principle of recursive decomposition of space. Quadtrees are used to represent point data, regions, curves, surfaces, and volumes. In

CONGRESSUS NUMERANTIUM **81**(1991), pp.173-182

this paper we focus on quadtree representation of graphic images utilizing raster format. This representation is based on the successive subdivision of the image array into four equal-sized quadrants. If a quadrant is not covered by a unique element (such as color or point) it is further partitioned into four equal-sized quadrants. The subdivision applies recursively until all regions are covered by a uniform element. These regions are either numbered as 0, 1, 2, and 3, or labeled as NorthWest (NW), NorthEast (NE), SouthWest (SW), and SouthEast (SE) [2].

Figure 1 illustrates a quadtree representing a black and white graphic image shown in the same figure. There are three types of nodes in the figure. Gray nodes are nonleaf node and represent a partition in the region. There are two types of leaf nodes: black and white. Black nodes represent an entirely black region and white nodes correspond to an entirely white (or background) region [3].

There are three main approaches to implementing a quadtree. The following section briefly describes these data structures.

2.1 Quadtree implementation

The first method is encoding the quadtree as tree structure that uses pointers. In this approach each internal node has four pointers for four subdivisions. Also, a bit is required to indicate whether the node is internal or external. This approach is considered to be not space efficient because of the amount of space used for pointer fields.

The second approach is storing the reorder traverse of the tree. For example the image shown in Figure 3 is represented by GWGWWBBGWBWBB. Although this approach is space efficient, makes it causes difficult to efficiently implement some of the operations on graph.

The third approach utilizes locational codes to store external nodes. Each region is represented by a pair of numbers. The first number is called the locational code and the second is the level of the tree at which the node is located. For example (03,2) represents region NW SE and level 2.

3 Bintree Structure

This approach was originally proposed to reduce the number of external nodes to represent a region of bintree. In this approach each region is partitioned into two regions. At each level of tree a region splits against one plain. For example at the first level a region splits into two regions against the x plane and at the second level new regions split against the y plane into four regions and so on. each region is checked to see if it is covered by unique elements before splits happen. The process of check and split is repeated until all regions of image are represented [4].

To represent a region two external nodes and one internal nodes could be enough in the best case. This is a 50 percent reduction compared to four external and one internal nodes required by quadtree representation. However, in the worst case bintree needs three internal nodes and four external nodes while quadtree needs one internal node and four external nodes. Average case analysis is more difficult than best and worst case analysis. In average case a pointer-based implementation of bintree may or may not be more compact than quadtree. An analysis of the worst case situation is provided in a later section of this paper.

4 Persistent Hierarchical Structures

A typical data structure supports two types of operations: queries and updates. When a data structure is updated, the state of the structure is changed and the previous state of the structure is not recoverable. Sometimes, however, it is useful to retain the past states of a data structure. Ordinary data structures are called ephemeral while a data structure in which past states are accessible is called persistent. Persistent data structures are useful in the implementation of very high level programming languages [5], in text editing where it is necessary to retain past versions of the text [5] and in computational geometry [7].

To be more precise we provide following definition:

Suppose there is a sequence of k lists, such that the sum of differences of items between adjacent lists is not more than a fraction of all items in the lists. This set of lists is called persistent.

We are interested in the following operations on the lists:

- retrieve at time t to time tt ($t <= tt$), item i or items i_j for all j, where $n > j > 1$ inclusively. (n is all items in those lists)
- Insert at time t.
- Delete at time t.

There are two major approaches to make a tree structure persistent. The first idea which is to copy only the nodes in which changes are made. Any node which contains a pointer to a node that is copied must itself be copied. This means copying a node causes a ripple of copying all the way up to the root, and this is called the *path copy* method.

In the second approach, node copying, each node in the tree has p auxiliary pointers in addition to its original set of pointers. For each update the key information and last pointers of the node are copied into a new node. Also a pointer in the parent pointer list is set to point to the new node. If all pointers in a parent node are used, recursievly access the parent node until a node with an unused pointer or the root is reached. The auxiliary pointers in the node require a time stamp to indicate the 'time' at which they are set. Also a root array is required to provide access to the root of the appropriate version of the tree. More details on these two approaches can be found in [1, 7, and 9].

The Path copying method is versatile in the application it supports, i.e. it can update any version of the tree, provided that an update is assumed to create an entirely new version. But the space requirement for path copying is the major draw back of this approach. For example, in a B-tree of L levels, for each update L nodes must be copied. The larger size of persistent B-tree (PB-tree), stored in secondary memory, increases access time and the paging requirement.

For the node copying method (NCM) relative to the path copying method (PCM) the number of node copy operations is less for NCM, but the size of node for NCM is greater than the node size for PCM. However, for PCM there is a side effect which is defined as: the increased probability of performing node copy on each of the nodes on the path from parent of the copied node to the root node as a result of filling the parent's pointers. This side effect is equal to:

$$c = (1 - (1/k)^L)/(k - 1)$$

where k is number of pointers per field and L is level of the copied node. In estimating number of nodes in an NCM persistent tree this side effect should be considered.

5 Persistent Quadtrees

An ephemeral quadtree represents one frame/instance of a graphic image at a time and each new frame of object is stored separately. Suppose there is a sequence of k frames, such that the sum of differences of points (pixels) between adjacent frames is not more than a fraction of all points in a frame of the object. We may wish to maintain the following operations on the frames:

Retrieve frame(s) f_k (or f_i to f_j , where $i \leq k \leq j$),

store a new frame f_k,

modify frame f_k.

A quadtree structure in which could support such operations is called persistent Quad-tree (PQ-tree).

Both linear and hierarchical encoding of quadtrees could be used to implement PQ-tree. To make the linear encoding of quadtree persistent the *delta* approach is used. In this approach each frame is assigned a time stamp. All external nodes of the first frame (which are represented and stored by locational code) are stored as a whole. To store the next frame, only differences between the first frame and the second frame are stored and time stamped. Similarly, to store frame f_i only the differences between frames f_i and f_{i-1}, if f_{i-1} exist are stored. The list of all differences between two consecutive frames is called delta. In this method only the first frame is accessible individually. To construct frame f_i, the first frame, f_0 must be accessed, and all modifications according to sequence of deltas between f_0 and f_i should be incorporated. To retain the corrected frames (or previous frames) backward correction is possible. For example, to go back from frame f_i to frame f_j all deltas between theses two frames need to be incorporated into frame f_i to retain frame f_j. Figure 2 illustrates the sequence of delta representing modification on the first frame shown in Figure 3. " < " symbol denotes insertion of a black region, and " > " denotes removal of a black region.

5.1 Hierarchical Presentation of PQ-trees

To provide direct access to each frame of quadtree we propose hierarchical PQ-tree structure. In this section both bintree and quadtree representations of graphic images are examined for implementation of hierarchical persistent structure. We show that space and time performance of persistent quadtree structures is better than persistent bintree.

To make these structures persistent we propose the use of more than one pointer for each region field and a time stamp for each of those pointers. Structure of an external node of a PQ-tree is shown as follows:

```
E-node
    region-field[4]
        pointer[k]
        timestamp[k]
        colorbit[k]
```

For simplicity this structure assumes black and white graphs. Bintree representation has the same structure but has only two region-fields.

The colorbit field is used to indicate the color of region. If the pointer field is set the color field is not used. Adding this field to the quadtree or the bintree structure eliminates the need for external nodes to represent black and white graphic images.

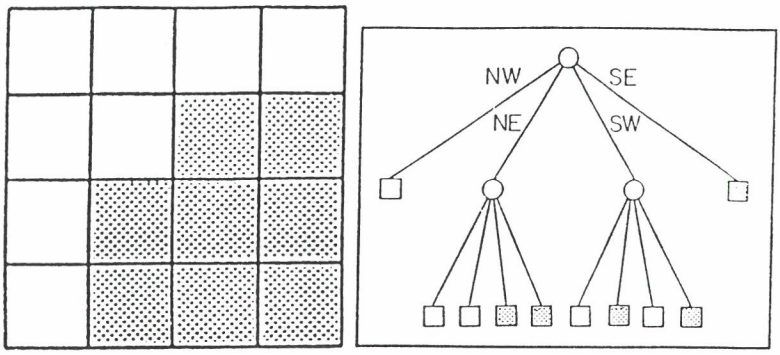
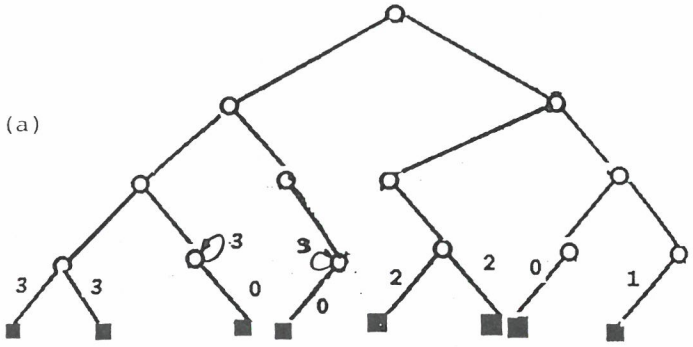


Figure 1. Pointer encoding of Quadtree of image.

(a)



(b)

t= 0 : 0011, 0110, 1100

t= 1 : < 1110

t= 2 : <1010, <1011

t= 3 : >0011, >0110, <0000, <0011

Figure 2. (a) Persistent bintree representing the first four frames shown in Figure 3. (b) Sequence of delt for same frames.

Since the number of external nodes is three times more than the number of internal nodes, this results in a significant space saving. We also use the node copying approach to make a quadtree persistent to reduce the number of internal nodes in.

5.2 Updating PQ-trees

Using the modified structure of quadtree and bintree nodes, to store frame f_i only differences between frames f_{i-1} (if f_{i-1} exists and modifications on f_{i-1} are not substantial) and side effects of those modifications are stored. Each frame f_i is accessible individually, there is no need to access previous frames to construct frame f_i. *Ad hoc* structure is not needed to keep the log of modifications. To retain the corrected frames or go back to one of the previous frames the root of the tree at search time is accessed (using the root header). And the tree is traversed following those pointers with the largest time stamp equal to or smaller than frame time stamp.

Modification of a region is actually insertion or deletion of one or more nodes. Deletion of node(s) results in creation of a new version of tree. Deletion is preformed by setting a colorbit to white, in related parent nodes, and stamping it to the deletion time. Thus, no node is deleted and all "deleted" nodes (regions) are preserved. Insertion happens either on the last frame, called *present* time, or one of the previous frames called *past* frames. To insert in one of the past frames, f_i, the root of the tree related to f_i is accessed and a copy of $f - i$ is created. This new copy is stripped off from all pointer fields with time stamp larger than i, insertion takes place in the newly created tree and a new time stamp is assigned to this tree. To insert a new node in PQ-tree at present time, first the root of the tree at insertion time is accessed. The given region is found following region location directions and pointer fields, on the path from root to the node representing region, with the largest time stamp.

Figure 2 illustrates an example of four frames of a persistent bintree. This tree represents four frames shown in Figure 3. Numbers on the edges represent time stamp of the pointer shown by the edge.

Figures 3-5 illustrate the PQ-tree related to the graph frames in the same figure. Figures imply that pointer field and colorbit are the same. This might be misleading, nevertheless, both fields are combined in in one to make these figures less crowded. At time "0" (or the first frame) PQ-tree has three nodes. The NE region doesn't need a node because it is all white. The node which represents NW region contains a black colorbit in the SE region, shown by black color in the figure. This black colorbit is time stamped to "0" to indicate a black region in the SE region. All other colorbit fields are left empty to indicate that those regions are all white. Accordingly, nodes representing SW and SE each have one colorbit to indicate a black in NE and NW, respectively, and both are time stamped to "0" to indicate insertion time of black region.

At time 1, NE of the SE region is modified to black, therefore the NE field of that node shows a black colorbit with time stamp 1. At time 2 a new node is created to represent two black regions in NE. At time 3, region SW is all white, therefore, there is no need for a node to represent that region. A new colorbit in the SW field of the root is set and is time stamped to "3" to indicate the all white region at time 3.

Modifying NW and NE of the NE region, at time 4, results in consumption of all color bits in this field, and another change in the same regions forces node copy action. Figure 4 illustrates this case. At time 7, NE of NE region is split into two regions, therefore a new level is added to the tree. Also, NW region is changed to black and a new colorbit is needed. Since all color bits in this field are consumed node copy is required. This node copy is followed by another node copy in the root (all pointers in NE field are consumed). Root header is updated to show the new root. And finally, at

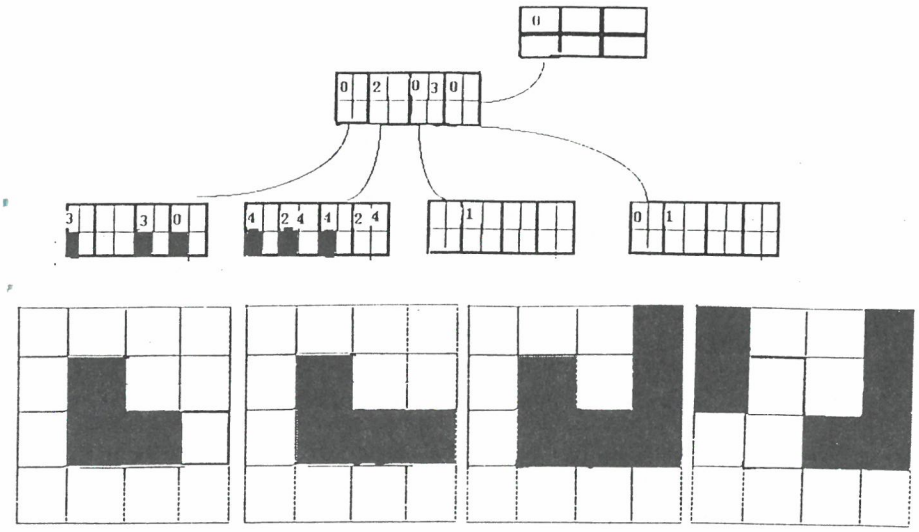


Figure 3, Frames 0-3 and PQ-tree representing frames.

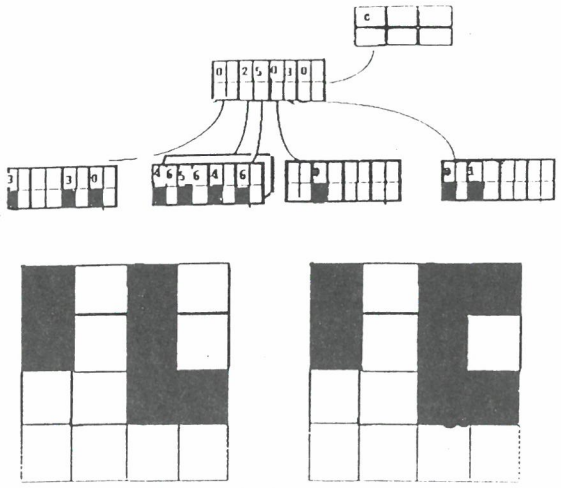


Figure 4, Frames 5 and 6 and PQ-tree representing frames.

Table I. Space requirement of PQtree and ephemeral quadtree.

	Ephemeral Quadtree	Persistent
number of bytes to store f frames of an n*n image	$\frac{1}{3} * n^2 * f * 4 * p$	$\frac{4}{3} n^2 * (1 + f * d * (1 + c) * k * p$
n=256, k=3, p=3 f=100, d=.05	$26 * 10^6$	$3.8 * (1 + c) * 10^6$

Table II. Space and time performance of PQ-tree and Persistent bintree at worst case analysis.

	Bintree	Quadtree
Number of Internal nodes	$2^{(n-1)}(2 + d * f * (\frac{1}{k} + c)) * (2 * p * k)$	$4^{(\log^* - 1)}(4 + d * f(\frac{3}{k} + c)) * (4 * p * k)$
Tree Level	$\log_2 n^2$	$\log_4 n^2$
possibility of Node- copy (Equal Distribution)	1/(2*k)	1/(4*k)
Possibility of Node-copy (Worst Case)	1/k	1/k

time 8 the NE region is all black and the tree represents this change by setting a new colorbit in the root to black at time 8.

At the end of this sequence of modifications the related PQ-tree contains two roots and six nodes. All frames of the graph are directly accessible. For example to access to the frame at time 3, we access the root via the pointer field of the root header time stamped to "0" (the next time stamp in the header root is larger than the search time). Then the tree is traversed through following all pointer fields with the largest time stamp less than or equal to 3.

5.3 Space and Time Performance

In this section we examine space efficiency from two different aspects. First we compare space requirement for the ephemeral quadtree to the PQ-tree requirement. Then space requirements for PQ-tree and persistent bintree are examined.

Table I illustrates space requirements for ephemeral quadtrees and PQ-trees. Symbols used in this table and Table II are defined as follows:

f denotes number of frames,
p denotes size of pointers (in bytes)
d denotes average percentage of modifications between any two consecutive frames
c denotes probability of node-copy side effect after modification in a node
k denotes number of pointers per region field

An example provided in Table I shows an approximately 75 percent reduction in required storage to store 100 frames in a PQ-tree comparing to storing those frames in 100 ephemeral quadtrees. It should be noted that to store an $n * n$ frame only $1/3n^2$ node is needed.

Table II illustrates a comparison between persistent bintree and PQ-tree. Possibility of node copy in persistent bintree is higher than possibility of node copy in PQ-tree. This is due to the fact that an $m * m$ region is represented by $1/3m^2$ nodes in a PQ-tree and by m^2 nodes in a bintree. Considering the higher possibility of node copy in bintree, for stroing a given number of frames is more in a persistent bintree more than a PQ-tree. The size of nodes in PQ-tree structure is almost twice the size of nodes in persistent bintree; nevertheless, as is shown in Table II the storage requirement is approximately twice as large in the persistent bintree approach.

Table II also shows the quadtree is shorter than bintree. This makes search and traverse time shorter in the case of PQ-tree.

6 Summaries and Conclusions

In this paper we have introduced the idea of persistent structures for storing graphic images. This idea is applicable to those images in which the sum of differences of points between two consecutive images is not more than a fraction of all points in a frame. It is shown that persistent structures for storing graphic images are much more efficient than ephemeral structures. Hierarchical persistent structures are superior over linear persistent structure because they provide direct access to each frame. We have shown that persistent quadtrees provide faster access to a given frame than persistent Bintree and use about one-half the storage compared to persistent bintree in the worst case analysis. However, persistent bintrees could be more compact than PQ-trees in the best case analysis.

References

[1] Cole, R., "Searching and Sorting Similar Lists," *J. of Algorithms* 1986, 202-220.

[2] Hunter, G. M., and Steiglitz, K. "Operation on Images Using Quadtrees", *IEEE Transaction Pattern Anal. Mach. Intel.* April 1979, 145-153.

[3] Klinger, A., "Pattern and Search Statistics," Rustagi, J., ed *Optimizing methods in Statistics* Academic Press, New York, 1971, 303-307.

[4] Knowlton, M., "Progressive Transmission of Grey-Scale and Binary Pictures by Simple, Efficient, and Lossless Encoding Schemes," *Proc. IEEE*, July 1980, 885-896.

[5] Reps, T. and Teitelbaum, T., "An Incremental Context Dependency Analysis for Language-Based Editors," *ACM Transaction on Programming Systems and Languages, 5*, 1983, 449-474.

[6] Samet, H., *Application of Spatial Data Structure: Computer Graphics, Image Processing and GIS*, Addison-Wesley, Reading MA, 1990.

[7] Sarnak, N. and Tarjan, R. E., "Planning Point Location Using Search Trees," *Communication of ACM, 29*, 1986, 669-679.

[8] Waronock, J., "A Hidden Surface Algorithm for Computer Generated Half-Tone Pictures", *Technical Report TR 4-15, Computer Science Department, University of Utah*, Salt Lake City, UT, June 1969.

[9] Zand, M. and Fisher, D. D., "Deletion on a Persistent B-tree," *Proceeding on Applied Computing, ACM*, 1989, 90-96.

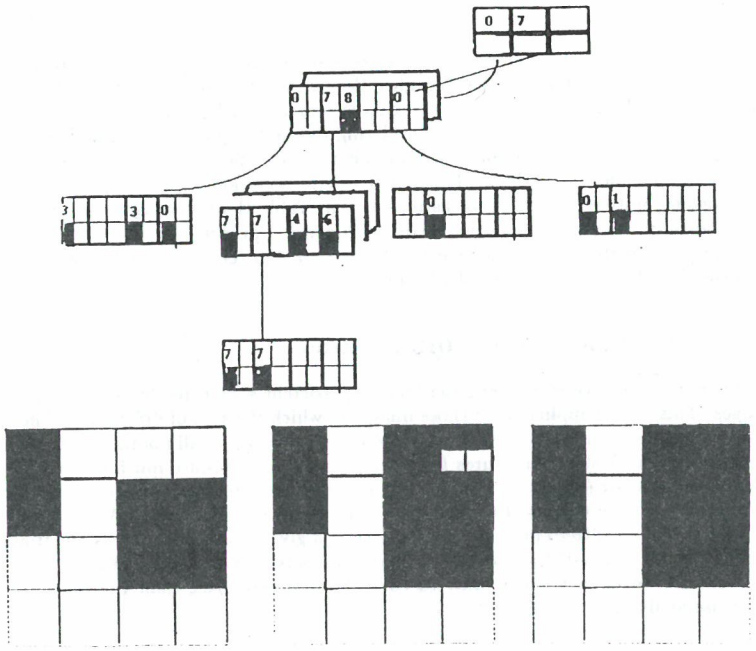


Figure 5. The last three frames and related PQ-tree.