Proc. VIIth Digital Image Computing: Techniques and Applications, Sun C., Talbot H., Ourselin S. and Adriaansen T. (Eds.), 10-12 Dec. 2003, Sydney

An Efficient Implementation of Max Tree with Linked List and Hash Table

Xiaoqiang Huang, Mark Fisher, Dan Smith

School of Computing Sciences, University of East Anglia, Norwich, NR4 7TJ, UK
(xq.huang@uea.ac.uk, mhf@cmp.uea.ac.uk, djs@cmp,uea.ac.uk)

Abstract. The max tree is a multi-scale image representation based in mathematical morphology which has been applied to image filtering, segmentation, tracking and information retrieval. This paper considers the problem of efficiently building max tree structures from images and retrieving information from them. Our aim is to find an economical data structure that provides fast direct access to the max tree nodes while keeping the memory usage for the tree to a minimum. For this we combine a linked list data structure which allows for dynamic allocation of computer memory and flexible management of tree nodes together with a hash table to give direct access to each tree node as the underlying data structure. Experimental results confirm that using this approach max tree image descriptions can be built in linear time $O(n)$.
Indexing terms: max tree, tree based image representation, multi-scale image decomposition.

1 Introduction

Although a rectangular array is the most intuitive way to represent and store digital image data this traditional representation has major drawbacks. For example Salembier [7] identifies two problems with the pixel based image representation. Firstly, its elementary unit, the pixel, provides extremely local information and secondly, most of the time, algorithms working at the pixel level need to be very simple because they have to deal with a very large number of pixels. These problems have motivated researchers to search for image representations more suited to high level computer vision applications. Region based image representations potentially offer attractive solutions to the problems encountered by pixel based approaches and could be considered as a first level of abstraction with regard to the raw information.

Two data structures supporting region based image representations have been identified. One is the Region Adjacency Graph (RAG) and the other is the Tree [10, 8, 3]. The RAG has two main drawbacks for practical applications, firstly it just defines the image at one scale only and secondly the connectivity between regions is not space invariant. Compared to the RAG, the Tree is a much more flexible image representation approach, which provides for a scale hierarchy. This means that one pixel does not have to be classified into one region at one

scale. Instead, one pixel could belong to different regions at different scales. This important property allows an image to be viewed at different levels of detail.

A tree could be viewed as a collection of elements called nodes along with a relation that establishes a hierarchical structure on the nodes. To provide a universal definition for tree based image representations is not trivial due to the diversity of the types of trees. However, all image trees should satisfy the following graph theoretic fundamentals [10]:

1. A Tree is a directed acyclic graph and each vertex of the graph is called node.
   (a) There is only one node, called the root N_{root}, to which no edges enter.
   (b) All nodes except the root have exactly one entering edge.
   (c) There are a number of nodes, called the leaf nodes N_{leaf}, to which no edges leave.
   (d) All nodes except the leaf nodes have at least one leaving edge.
   (e) If there exists an edge leaving from node N_1 to N_2, node N_1 is called the father node of node N_2 and node N_2 is called the child node of node N_1
   (f) There is a unique path from the root node to each node.
2. A Tree is also a hierarchical data structure collecting possible regions of pixels
   (a) Each node N_i is associated with a region of support R_i.
   (b) The region of support of the root R_{root} is the entire image.
   (c) $\forall i, j, k,\ i \neq j$ such that $father(N_i) = father(N_j) = N_k \Rightarrow R_i \subset R_k,\ R_j \subset R_k,\ R_i \bigcap R_j = \emptyset$.

Two similar tree based image descriptions satisfying the above criteria and founded in mathematical morphology have been recently proposed. The max tree proposed by Salembier et. al. [2, 10] is a multi-scale image representation formed by considering a hierarchy of connected level-sets of pixels in an image. The max tree can be regarded as an instance of the more generalised opening tree [11] from L. Vincent. The sieve tree from Bangham et. al. [3] is also based on a level-set image decomposition and shares the same topology as the max tree structure but maps regions in a slightly different way. The implementation issues addressed in this paper are equally applicable to both structures. Both Salimbier and Bangham have achieved some success in applying the max tree and sieve tree a number of classical computer vision problems including stereo matching [6], image filtering, segmentation [3] and information retrieval [10, 4].

Three sub-problems need to be addressed before either 'scale' tree can be successfully constructed:

1. Find all possible tree nodes from the original image
2. Create the father-child relationships (links) between each possible pair of tree nodes
3. Create an efficient data structure to store information (region attributes) at nodes in the scale tree.

The first sub-problem has been successfully addressed by a recursive flooding algorithm [10]. The focus of this paper is on discussing issues related to the second and third sub-problems. This work is motivated by the need to find a data structure for implementing the scale tree that is efficient both in terms of CPU execution time and computer memory usage. An efficient scale tree data structure should allow for direct access and flexible manipulation of the tree nodes. Here, the main contribution of this paper is in combining a linked list (for dynamic allocation of computer memory and flexible management of nodes) with a hash table (for direct access to nodes) as the underlying data structure to accommodate the scale tree.

The rest of this paper is organised as follows. The second section explains how the linked list and the hash table are used in detail. In the third section we discuss the important issues regarding the evaluation of scale tree image descriptions and present some experimental results which confirm the algorithm we propose operates efficiently in space and time. Finally conclusion are drawn and some suggestion for further work are made

2 Proposed Data Structure for the Max Tree

2.1 Requirements for efficient hierarchical storage of image data

A suitable data structure for the scale tree should meet the following three criteria:

1. The size of the data structure should be expansible.
2. The time cost of creating an new empty node should be constant.
3. All tree nodes should be accessible directly.

In considering these criteria we need to emphasise that building the scale tree is not our primary objective and once available the image description will be used to address problems in the fields of image processing and computer vision. Direct access to the pixel data stored in the tree and the flexible manipulation of tree nodes (e. g. to reconstruct the image at the required scale) are very important.

2.2 Linked List VS Array

Many classical data structures such as the queue, stack and table may be implemented either as an array or a linked list. The array and the linked list are two basic ways of allocating computer memory that could be used by a program. The array is normally used when the data type and the number of elements is known before the memory is allocated while the linked list is normally used when the number of elements is unknown and the computer memory needs to be dynamically allocated at run time. The size of the scale tree (number of nodes) is image dependent and unknown until the tree has been built. For this reason the linked list seems a better option than the array as the underlying data structure for the scale tree.

Inherently, the nodes in a linked list must be accessed serially so the complexity of visiting a random node in a linked list is $O(N)$. In contrast, the complexity of visiting a random element in an array is $O(1)$. This probably explains why the array is adopted as the underlying data structure in recent implementations of the max tree [9] and sieve tree [5]. This approach is not memory efficient and in most cases the scale tree requires far more memory than it actually needs. Another disadvantage of using the array is that when an element is deleted, maintaining the structure becomes cumbersome. The linked list does not have this problem as all one needs to do is to remove the unwanted element from the linked list and link its preceding and successive elements together. Therefore, the linked list provides more flexible and economical way of deleting (managing) tree nodes.

To summarise, the linked list provides dynamic allocation, efficient use of memory and flexible management of tree nodes while the array provides direct and fast access to a memory unit. Using a hash table within the data structure for the scale tree leads to a more efficient implementation which combines the advantages of the linked list and the array. It is well known that hash table is prone to becoming inefficient when there is a need for re-hashing of the table. Fortunately, in our implementation of the max tree, there is no need for re-hashing as the size of the hash table is pre-known before the hash table is created.

2.3 Elements Stored in a Max Tree Node

Once the underlying data structure of the scale tree has been decided, one needs to consider what information should be stored at a tree node. A minimal set of attributes should include

1. ID: node identification number
2. father's ID
3. a list of children IDs
4. a list of pixel coordinates
5. area: number of pixels belonging to a node

Obviously, each node in a tree needs a unique identification number. As a tree is a hierarchical data structure, each node is linked to its father and children nodes. The area attribute records the total number of pixels belonging to a node (i.e. the size of a granule or region). A node would also need to store information of pixels belonging to its support region so that an image could be reconstructed from a tree. Note that the list of children's ID and the list of pixel coordinates are also implemented by a linked list.

2.4 Technical Details for Creating the Max Tree

Salembier [2]has proposed a very efficient way of finding scale tree nodes using a recursive flooding algorithm. The flooding algorithm begins at the root node of the tree (this is, the lowest gray value of the image) and recursively constructs

each of the branches of the tree. They use hierarchical first-in-first-out queues of NG levels, with NG the possible number of gray levels (usually $NG = 255$). These queues are used to define the scanning and processing order of pixels comprising the image. An array $STATUS$ of the same size as the image is used to determine to which node a pixel belongs to. A pixel p with gray-level h belongs to the node C_h^k if $STATUS[p] = k$. Initially, all elements in $STATUS$ are set to $NOTPROCESSED(< 0)$. One array of G integers called $Number - Nodes(h)$ is used to store the number of max tree nodes detected so far at each gray level h. The values of Number-Nodes is updated whenever new nodes are created at gray-level h. A further array of G Boolean $Node - at - Level(h)$ stores at which levels below the current gray-level nodes have been detected in the path from current node to the root.

A node in a max tree could be uniquely presented by C_h^k. For instance, a node presented by C_{20}^3 means this node's gray value is 20 and it is the third node found in the level of 20 during the recursive flooding algorithm. Note that a node C_h^k corresponds to a connected component P_h^k. However, C_h^k contains only those pixels in P_h^k which have gray-level h for the purpose of memory efficiency. The hash table of the max tree is created in the end of the recursive flooding algorithm. The key for an element in the hash table relating a node in the max tree would be $\{h, k\}$.

Our implementation of the max tree starts with applying the recursive flooding algorithm to recover scale tree nodes from the original image. Each flooding operation recovers a new node (and associated image pixels) and a father-child relationship between two nodes. These nodes are all appended to a linked list but due to the complexity ($O(n)$) of visiting elements in a linked list we do not visit tree nodes to establish relationships in this stage. Instead, a temporary linked list $Temp_L$ is created to store the information about the relationships and when the whole recursive flooding algorithm is finished, a hash table is created to index the linked list representing nodes of the created max tree. With the help of the hash table, the complexity of visiting a random scale tree node now becomes $O(1)$ (although the tree is still stored in the linked list). Finally, the information stored in the linked list $Temp_L$ is eventually retrieved and the father-child relationships are added as attributes at the scale tree nodes. The complexity of the whole process is $O(n)$.

3 Experiments

In this section we evaluate the scale tree building algorithm and demonstrate that with the help of the use of a hash table, the efficiency of the algorithm is significantly improved.

Our experiments are based on the max tree implementation defined by pseudo code in [10] but the approach is equally applicable to the sieve tree variant proposed by Bangham [3]. The algorithm is implemented in C++ by the Microsoft Visual C++ IDE (version 6). The specification of the computer in which all

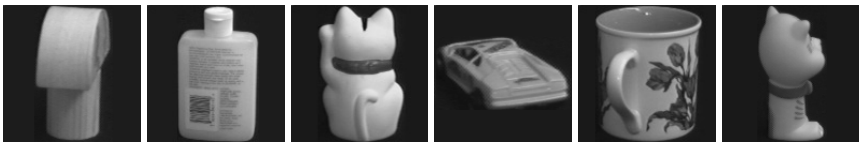
experiments are carried out is as follows: OS: Windows 2000, CPU: Pentium 1.7GHZ, Memory: DDR 1.0G Bytes.

It is common to evaluate a tree construction algorithm against image size [10]. However, we think this evaluation method is not fair. We believe that the complexity of the image content is a more important factor than the size of the image in the evaluation. If an evaluation of a tree construction algorithm against image size is not avoidable, it is much fairer to choose test images of different size but of the same image complexity. The term of 'image complexity' is defined as follows:

$$image\ complexity = \frac{total\ number\ of\ flat\ zones}{image\ size} \tag{1}$$

We test our algorithm using three groups of gray level images. The first group of images, shown in figure 1, is chosen randomly from the COIL-100 image data set from columbia university image library (http://www.cs.columbia.edu/). These images have the same size (128 × 128) and background but represent different foreground objects. The test results (see figure 2) using the first image group show that for our implementation the time taken to build a max tree is slightly dependent on the image complexity. They also show that when the image size is small, using a hash table would increase the execution time because the time taken to build the hash table is more significant than the computing gain achieved by using it. However, the following two experiments show that when the image size is larger or the scale tree is more complex, using the hash table achieves a gain in overall performance.

(a) 1 (b) 2 (c) 3 (d) 4 (e) 5 (f) 6



(g) 7 (h) 8 (i) 9 (j) 10 (k) 11 (l) 12

Fig. 1. The first group of test images (128 × 128)

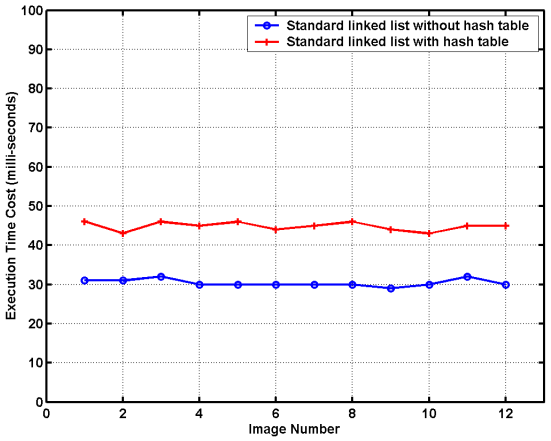


Fig. 2. Test results using images from figure 1

The second image group is shown in figure 3. These image are of different size but of the same image complexity. The test results are shown in figure 4. Figure 4 shows that the complexity of our implementation of the max tree using a hash table is linear with respect to image size provided the test images have the same image complexity. However, the implementation of max tree by linked list without the use of a hash table does not work in linear time.

The third image group, shown in figure 5 comprises a typical image at different resolutions. Here, image 5 in figure 5 is the original image and the others are the resized versions of it. Figure 6 shows that even for this group of images our max tree implementation of also works linearly.

4 Conclusion and Further Work

An efficient data structure for the max tree is proposed in this paper. This data structure is based on the use of a linked list along with a hash table. The linked list provides dynamic allocation of the memory ensuring that the data structure for the tree is memory efficient. This method outperforms the previously published approach using an array [9]. The hash table provides direct access to a tree node so that a node could be reached without visiting its preceding nodes first. The use of hash table turns the complexity of visiting an element in a linked list from $O(n)$ to $O(1)$ so that the time cost of building the max tree and applying the max tree to solve real tasks is linear with repect to image size.

The performance of this new implementation of the max tree by linked list and hash table is not fully analysed. Current performance analysis is focused on the computer execution time. Our further work will include a more detailed analysis of memory usage. Additionally, we hope to extend the approach to

(a) image 1: 128 × 128

(b) image 2: 128 × 256

(c) image 3: 256 × 256

(d) image 4: 256 × 512

(e) image 5: 512 × 512

Fig. 3. The second group of test images

produce efficient implementations of other scale tree structures, e.g. the Inclusion Tree [8] and Critical Lake Tree [1].

References

1. J. Cichosz, F. Meyer: Morphological multiscale image segmentation. Proceedings of Workshop of Image Analysis for Multimedia Interactive Services, (1997) 161–166
2. P. Salembier, A. Oliveras, L. Garrido: Anti-extensive connected operators for image and sequence processing. IEEE Transactions on Image Processing, No. **4**, Vol. **7**, (1998) 555–570
3. J. A. Bangham, J. R. Hidalgo, R. Harvey, G. Cawley: The segmentation of images via scale-space trees. Proceedings of the Ninth British Machine Vision Conference, (1998) 33–43
4. J. A. Bangham, K. Moravec, R. Harvey, M. Fisher: Scale-space trees and applications as filters, for stereo vision and image retrieval. Proceedings of the Tenth British Machine Vision Conference, (1999) 113-122

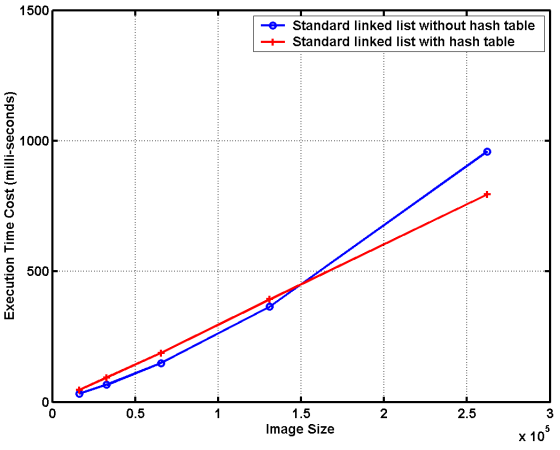


Fig. 4. Test results using images from figure 3

5. Javier Ruiz Hidalgo: The representation of image using scale trees. Master theis, School of Information Systems, University of East Anglia, Norwich, Norfolk, United Kingdom", October, 1999
6. K.Moravec, R.Harvey, J.A.Bangham: Scale trees for stereo vision. Proceedings of IEE on Visual Image Signal Processing, No. **4**, Vol **147**, (2000), 363–370
7. Philippe Salembier, Luis Garrido: Binary partition tree as an efficent representation for image processing, segmentation, and information retrieval. IEEE Transactions on Image Processing, No. **4**, Vol. **9**, (2000) 561–576
8. Pascal Monasse, Frédéric Guichard: Scale-space from a level lines tree. Journal of Visual Communication and Image Representation, No. **2**, Vol. **11**, (2000) 224–236
9. A. Meijster, M. Wilkinson: A comparison of algorithms for connected set openings and closings. IEEE Transactions on Pattern Analysis and Machine Intelligence, No. **4**, Vol. **24**, (2002) 484–494
10. Luis Garrido Ostermann: Hierarchical region based processing of image and video sequences: application to filtering, segmentation and information retrieval. PhD thesis, Department of Signal Theory and Communications, Universitat Politecnica de Catalunya, Barcelona, Spain, 2002
11. L. Vincent: Fast grayscale granulometry algorithms. In Proceedings of the International Symposium on Mathematical Morphology and its Applications to Image Processing, (1994) 265-272

(a) image 1: (40 × 32) (b) image 2: (80 × 64) (c) image 3: (160 × 128) (d) image 4: (320 × 256)

(e) image 5: (640 × 512)

Fig. 5. The third group of test images

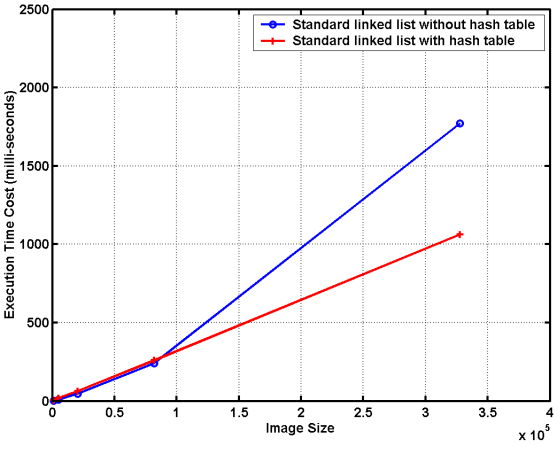


Fig. 6. Test results using images from figure 5