Compressed Quad Trees

J. R. Woodwark
School of Engineering, University of Bath, Claverton Down, Bath BA2 7AY, UK

A compact representation for digital images is described which is based on a traversal of the quad tree form. It is designed to store coloured pictures of the type which are displayed on a raster scan device using a colour lookup table. Two experiments are reported which show that the code appears to occupy 65% of the space of a previous traversal scheme, and 105% of that of a run length encoding technique.

INTRODUCTION

Quad trees are formed by the recursive four-way division of a digital picture of side 2^n pixels into equal square areas of side 2^m pixels, where $0 \leq m < n$. These quads are arranged in a tree the leaves of which are squares of a single colour. As a representation, the quad tree is able to exploit the coherence of many images of interest to reduce storage requirements. In this respect it is similar to run-length encoding, and has comparable efficiency. The area basis of the quad tree, however, makes it a much more computable form in which to have an image available. In particular, the quad tree can act as many versions of the same image with different degrees of resolution, which allows programs, especially in computer vision applications, to focus easily onto areas of detail. The author has also found the quad tree a very serviceable structure in a task diametrically opposed to vision: the construction of digital images from solid models of objects.

Attention has recently been drawn[1,2] to quad tree representations, based on tree traversal, which emphasize the compactness of the quad tree form. This paper presents an enlargement of this recent work, and defines a traversal code which is particularly efficient for certain common classes of coloured image.

QUAD TREE REPRESENTATIONS

Before proceeding with the main topic, the author feels that it is worth mentioning the three common quad tree representations other than traversal codes.

Pointers

The pioneers of the quad tree[3–5] were mainly concerned with the flexibility of the quad tree representation. They therefore developed storage techniques using pointers between the quads on the tree: links from father to son, backward pointers, and even 'ropes', further links between adjacent leaf quads. These approaches allow the editing of a quad tree with maximum facility. They are not very efficient in their use of storage. For many images the colour word length is much smaller than the address length required for the number of quads, and the storage required for links is several times that for colour information.

Explicit

The explicit quad tree[6,7] avoids the necessity for links by reserving a storage location for every possible quad. It is a structure that the author has found very useful in image generation. It is however the least space-efficient of all quad tree codes, requiring more storage than a pixel plane representation of the image. Its advantage lies in very rapid access to large areas of picture.

Leaf codes

There are a number of codes which represent the leaves of a quad tree alone. In these representations each leaf quad is stored as a colour and a size. The positions of the leaves are determined either by attached co-ordinate information, which is rather cumbersome, or by an implicit ordering. This may take the form of an agreed traversal of the tree,[2] and in this case has a similarity to the traversals presented below. The leaf quad sizes are often inconvenient quantities to store, and storage requirements are relatively large: furthermore, leaf codes lack the structure which is preserved in the true traversals.

An alternative leaf code, that has been used by the author and his colleagues,[8] involves storing the leaves in scan order. This is a form from which quad trees can be displayed directly by a special raster scan display.

TRAVERSAL CODES

Traversal codes are simply lists of quads generated during a defined traversal of the tree. This is performed depth-first, to restrict the depth of stack needed for both encoding and decoding. The position of each quad is then completely defined, knowing the order in which every quad is visited. This simple but effective idea has only recently reached the literature in simultaneous papers by Gargantini[1] and Oliver and Wiseman.[2] The former paper deals with binary images only, but a positional code is attached to each quad which is used to support movement between adjacent pixels, to fulfill the same goals as the 'roped' quad tree mentioned earlier. The latter paper, on the other hand, uses codes applicable to 16-level greyscale images, and also stores average intensity values at non-leaf nodes, making the picture directly available at lower resolutions. The main reason for using traversal codes must be data compression, but

CCC–0010–4620/84/0027–0225 $02.50

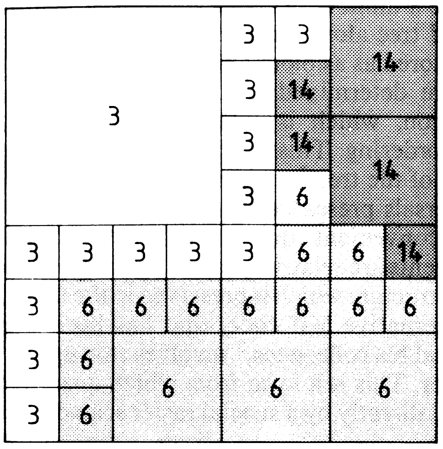
both papers indicate ways in which operations on images may be performed without conversion to other representations. Such activities are necessarily limited to the processing or combination of whole trees, and changes to small areas of a picture would be very inefficient with these structures. If, on the other hand, the requirement is for whole image work, then the compactness of the data will enhance processing speed.

This paper is concerned mainly with deriving maximum benefit from the compression obtainable from traversal codes, and therefore Gargantini's scheme will not be considered further. Oliver and Wiseman's code, on the other hand, is a traversal pure and simple. Their data items consist of five-bit numbers, of which the last four bits constitute the colour value. The first bit is cleared when the quad is a leaf, and set when it is not. In non-leaf quads the colour value refers to an average of the quads beneath. Figure 1(a) provides an example of a simple quad tree in their coding.

A very common type of raster scan display device stores an image in from four to twelve pixel planes, using a colour lookup table to transform the values stored in the planes into intensity values for the red, green and blue components of each pixel. This scheme makes available a much larger palette of colours than would be possible if the electron guns in the monitor were driven directly from small groups of planes, and avoids an awkward factor of three as the number of planes. One or two useful 'special effects' are also available from this type of device. A popular number of pixel planes is eight, which allows full use to be made of an eight-bit byte to store pixel values within the host computer. How can Oliver and Wiseman's code be adapted for such a device?

There is now little point in retaining the averaged pixel values stored at non-leaf quads. Unless the lookup table happens to contain an ordered sequence of shades of a single hue, averaging the numerical values of pixels will not yield an average of their colours. The whole point of the lookup table arrangement is to allow many unrelated colours on the screen at once. In this case, there is a huge range of possible colours which could be generated as averages of the original ones. The average colours that would actually be required depend on which different coloured regions of the picture are adjacent. For a picture of reasonable complexity, it may be expected that the number of free entries in the colour table left after the original colours have been represented will be insufficient by far to hold the extra average colour values that would be required for an accurately averaged lower resolution picture. The same problem occurs when considering anti-aliasing on devices of this type.

If average colour values are abandoned, then, instead of using the first bit of each code to represent whether that quad is a leaf or not, we may simply withdraw a single colour value, say 255 in an eight-plane system, and use this code to indicate the presence of a non-leaf quad in the traversal code. A reduction of a single colour in the picture bandwidth may not be very important: it is often convenient in any case to 'reserve' one or two colours for graphics cursors, and so forth. This code contains the same number of records for a picture of a given complexity as that of Oliver and Wiseman: we simply lose the lower resolution representations which we have already said cannot normally be used with a lookup table driven device.



a) Oliver and Wiseman's treecode

$\underline{22}$ $\underline{20}$ $\underline{20}$ 3 3 6 6 $\underline{20}$ 3 3 6 3 6 $\underline{20}$
6 3 6 3 3 $\underline{22}$ 6 $\underline{21}$ 6 3 6 6 6 $\underline{24}$ 6
6 6 14 $\underline{26}$ $\underline{22}$ 3 3 6 14 $\underline{22}$ 3 3 14
3 14 14

Underlined numbers are average values (+16 for identification) at non-leaf quads

b) Compressed traversal code

$\underline{9}$ 3 $\underline{6}$ 14 $\underline{29}$ 3 14 $\underline{46}$ 3 14 6 $\underline{2}$ 6
$\underline{29}$ 3 6 $\underline{32}$ 3 6 $\underline{38}$ 3 6 $\underline{4}$ 6 $\underline{41}$ 3 6
$\underline{37}$ 6 14

Underlined numbers are type codes (see figure 2)

Figure 1. Traversal and compressed traversal coding of a simple quad tree.

COMPRESSED TRAVERSAL CODE

In a picture coded using the modified Oliver and Wiseman scheme above, the value which indicates a non-leaf quad is repeated many times: intuitively this does not look like the most efficient code possible. How can a more compact form be obtained? Examine any quad tree representation of a reasonably coherent image, and notice that it is very common for a division to produce two or more subquads which are the same colour. A boundary between two different coloured regions provides many examples of this effect. Now, using the modified Oliver and Wiseman code, we must repeat the colour values for every leaf quad which is that colour. The compressed traversal code proposed in this paper avoids this repeated information.

Consider a code that represents the non-leaf quads of the tree traversal only. Leaf quads are represented only as attributes of a given non-leaf quad. A non-leaf quad consists of a record, comprising a type code, followed by the colours of any leaf quads immediately below that non-leaf quad. If more than one of the dependent leaf quads is the same colour, this colour value nevertheless appears only once. The allocation to the leaves of the colour values following the code is determined by the

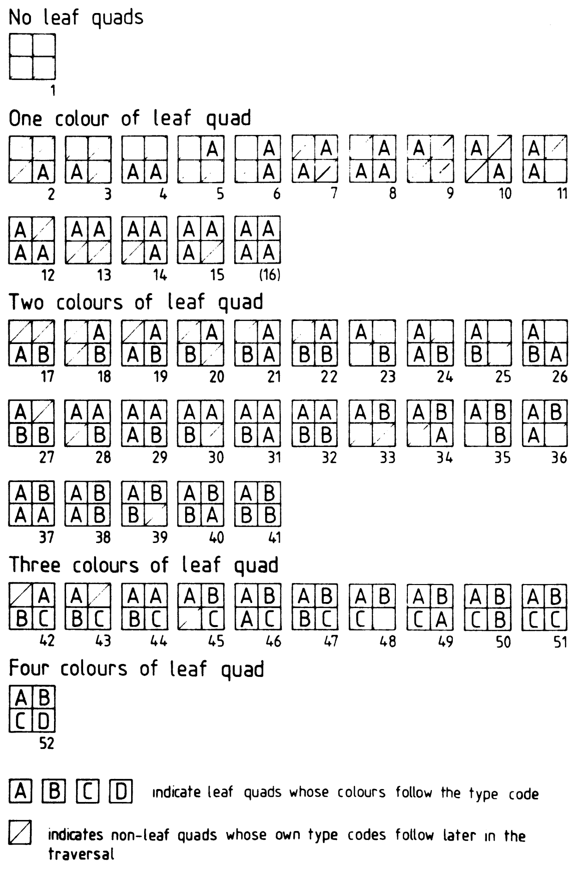

No leaf quads
One colour of leaf quad
Two colours of leaf quad
Three colours of leaf quad
Four colours of leaf quad
A B C D indicate leaf quads whose colours follow the type code
indicates non-leaf quads whose own type codes follow later in the traversal

Figure 2. Type codes for compressed traversal coding.

value of the type code for that quad. There are 52 possible configurations for the sons of any non-leaf quad (Fig. 2), including the degenerate code 16, which will be discussed shortly. These allow for any or all of the sons of that quad to be defined further down the tree, or for them to be the same or different coloured leaves. If, for example, a non-leaf quad has dependent leaves of two colours, A and B, the record representing that quad will consist of the appropriate type code, followed by the colour values of A and B in the order defined by the type code. Any dependent non-leaf quads will follow in traversal order, using the convention that sub-trees are visited in scan order. Figure 1(b) shows a simple quad tree represented using this compressed traversal code.

Before going on to see how this code behaves in practice, it is worth making four brief points.

(a) Because there are 52 type codes, they would be most efficient in a 6-bit application. The results presented in the next section assume the use of more generally acceptable 8-bit colour values, as already discussed.
(b) Even in the worst case, the storage requirement for this code never exceeds that of the unmodified traversal code.
(c) Unlike the unmodified code, none of the 256 colour values need be sacrificed.
(d) The code (16) which indicates that all the leaf codes dependent from a given non-leaf quad are the same colour is not generally useful. It is included to allow imperfect quad trees to be represented, perhaps in an interim stage in some processing operation, and to permit degenerate images, of uniform colour overall, to be stored.

EXPERIMENTS ON TWO IMAGES

The compressed traversal code we have just described was tested experimentally on two digital images, shown in Figs 3 and 4, generated on the author's VOLE solid modelling system.[9] The models from which the images were generated were selected because they are common objects rather than specialized engineering components. They are a square pin U.K. mains plug, with its top removed, and a bungalow with its roof lifted to show interior detail. The particular views were chosen with the aim of filling the image area fairly well. The pictures are at 256 x 256 pixel resolution. This is not the full resolution of the author's display, but was used to facilitate scan line encoding, as will be mentioned shortly. It is important to remember that these are coloured images, but reproduced here in black and white. It is likely that many adjacent areas of the image will appear

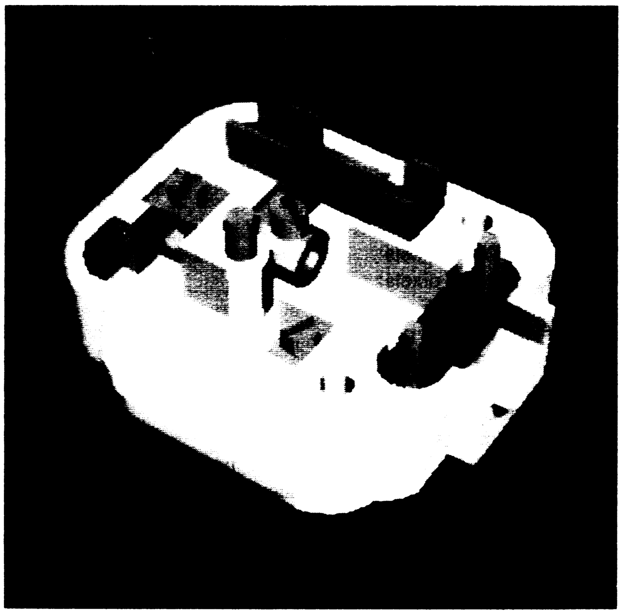


Figure 3. Plug image (256 x 256 pixels).

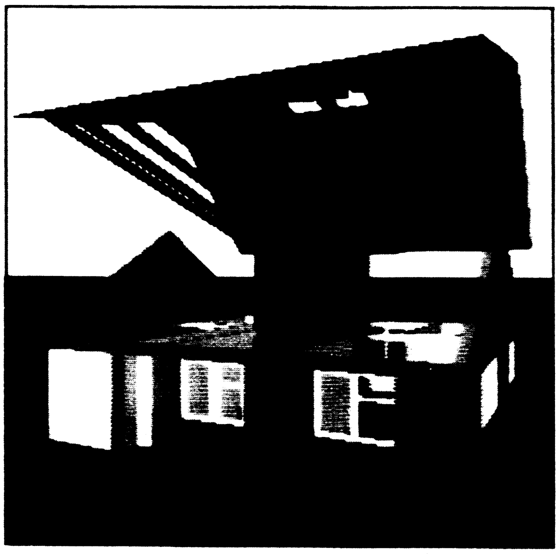


Figure 4. Bungalow image (256 x 256 pixels).

to be the same shade in the figures, whereas they were originally composed of two or more different pixel values. The quad tree is constructed from these colour values, of course, and not from the images as they appear here.

The following sections discuss the image data tabulated in Table 1. Although these pictures are typical of those generated in the modelling work at Bath, there are only two, and the results presented are not intended as a full justification of the proposed code.

Table 1. Data corresponding to the images in Figs 3 and 4

	Plug Image (Fig. 3)	Bungalow Image (Fig. 4)
(a) Quad tree data		
Number of non-leaf quads		
of size 2 × 2 pixels:	1462	2569
of size 4 × 4 pixels:	945	1380
of size 8 × 8 pixels:	428	507
of size 16 × 16 pixels:	154	173
of size 32 × 32 pixels:	45	56
of size 64 × 64 pixels:	14	16
of size 128 × 128 pixels:	4	4
of size 256 × 256 pixels:	1	1
Total number of non-leaf quads:	3053	4706
Number of leaf quads		
of pixel size:	5848	10 276
of size 2 × 2 pixels:	2318	2951
of size 4 × 4 pixels:	767	648
of size 8 × 8 pixels:	188	185
of size 16 × 16 pixels:	26	51
of size 32 × 32 pixels:	11	8
of size 64 × 64 pixels:	2	0
of size 128 × 128 pixels:	0	0
of size 256 × 256 pixels:	0	0
Total number of leaf quads:	9160	14 119
Total number of quads of both types, corresponding to the total number of bytes in an Oliver and Wiseman traversal code representation:	12 213	18 825
(b) Compressed traversal encoding		
Occurrences of type codes		
with no leaf quads:	155	359
with one colour of leaf quad:	1105	1428
with two colours of leaf quad:	1718	2722
with three colours of leaf quad:	73	197
with four colours of leaf quad:	2	0
Total number of bytes in a compressed traversal code representation:	7821	12 169
(c) Run-length encoding		
Total number of bytes in a horizontally run-length encoded representation:	7290	11 780
Total number of bytes in a vertically run-length encoded representation:	6230	11 106

Quad tree data

Table 1(a) shows the numbers of non-leaf and leaf quads of all sizes. From the totals we see that direct traversal code representation of the pictures would occupy 12 213 and 18 825 bytes for the plug and bungalow models, respectively. Note that these figures are dominated by leaf node data.

Compressed traversal encoding

Table 1(b) shows the numbers of each length of record required to construct the compressed traversal representation of these pictures. The author had expected the case where a non-leaf quad had four leaves of different colours to be rare, but the actual figures are quite surprising. The storage requirements for the plug and bungalow images are 7821 and 12 169 bytes, respectively. These figures are both slightly less than 65% of the storage requirements for the straight traversal codes. Note that in both cases the compressed code occupies less space than the colour values of the leaf nodes alone, when these are all separate.

Run-length encoding

Table 1(c) concludes by indicating the number of bytes that would be needed to run-length encode the images. This assumes that a coding scheme of a 'length of run' byte followed by a colour byte is used. This simplicity is permitted with the present image size of 256 × 256 pixels which is the reason for the choice of this resolution. The plug and bungalow require 7290 and 11,780 bytes, respectively. These figures are lower than either quad tree scheme, 60% and 63% of the traversal code, and 93% and 97% of the compressed traversal code. Of course, the efficiency of scan line encoding depends on image size, and these images were specially arranged to be convenient.

Equivalent figures for encoding in a vertical direction are also given in Table 1(c), to check for any anisotropy in the image, which would be reflected in the efficiency of the run-length encoding. The figures are somewhat lower, but not sufficiently to condemn the images in this respect.

CONCLUSIONS

This paper has presented a quad tree traversal based image encoding scheme that appears to be one third more compact than a similar scheme proposed by Oliver and Wiseman,[2] and more suitable for storing images which are displayed using a colour table. The efficiency of the coding is similar to, but does not exceed, that of simple run-length encoding. These figures refer to the sorts of image with which the author is working.

In using this compressed form, some more programming will be required, but most of the work of interpreting the codes can be performed by simple lookup table operations. The author sees no reason why those of the image operations mentioned by Oliver and Wiseman which do not use average colour values should not be performed using the compressed code directly. Where the images are stored on a slow data medium, such as disc, the space saving should produce a corresponding saving in execution times.

Acknowledgements

The author would like to thank his research students Mr. K. M. Quinlan and Mr. A. F. Wallis, who originally prepared the plug model, and his colleagues Mr. D. J. Milford and Dr. P. J. Willis for their comments.

REFERENCES

1. I. Gargantini, An effective way to represent quadtrees. *Communications of the ACM* **25**(12), 905–910 (1982).
2. M. A. Oliver and N. E. Wiseman, Operations on quadtree encoded images. *The Computer Journal* **26**(1), 83–91 (1983).
3. A. Klinger and C. R. Dyer, Experiments on picture representation using regular decomposition. *Computer Graphics and Image Processing* **5**, 68–105 (1976).
4. G. M. Hunter and K. Steiglitz, Operations on images using quad trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence,* **PAMI-1**(2), 145–153 (1979).
5. H. Samet, Region representation: Quadtrees from binary arrays. *University of Maryland Computer Science Centre Report TR-767* (1979).
6. J. R. Woodwark, The explicit quad tree as a structure for computer graphics. *The Computer Journal* **25**(2), 235–238 (1982).
7. F. W. Burton and J. G. Kollias, *Short note in the Computer Journal* **26**(2), 188 (1983).
8. D. J. Milford, P. J. Willis and J. R. Woodwark, Exploiting coherence in raster scan displays. *Proceedings of the Electronic Displays 81 Conference,* Network, 34–46 (1981).
9. J. R. Woodwark and K. M. Quinlan, Reducing the effect of complexity on volume model evaluation. *Computer Aided Design Journal* **4**(2), 89–95 (1982).

Received July 1983