**Chapter 4: Image Partitioning**

**4.1 Introduction**

This chapter will try to explain why it is better to vary the block size of a low complexity image transform across the image, rather than keeping it fixed as in the previous chapter. The various methods used in to implement variable block size will be investigated and assessed to produce an acceptable still image compressor that uses a low complexity transform.

In a typical image there are always areas of detail (such as houses) and flat areas (such as sky). It is clear that the detailed areas require more attention that the flat ones. In the previous chapter it was shown repetitively that decreasing the block size improves the level of approximation of the image compressor (this was determined by forming the rate distortion curves). In this chapter the block size that the low complexity transform is applied to will be varied. This is in order to change the level of approximation applied to different areas of the image. Large blocks are used to approximate flat areas and small blocks approximate areas of detail.

Only low complexity transforms were assessed with a variable block size since higher complexity transforms have too many other factors to consider. The coder that is produced using low complexity transforms is still relatively simple and so can be used in software only solutions, unlike high complexity transforms such as the DCT or wavelet, which usually require additional hardware for speed.

There are three basic types of partition methods (although any type of partition is possible) and these are:

1. Quad tree partition.
2. Horizontal/Vertical decomposition (HVD).
3. Triangular partition.

Each method has its own benefits, but since this work is concerned with image compression, triangular partition will only be dealt with in general terms (it is much better for morphology than compression purposes).

There has been little interest in image partition for compression, but it has been examined in the context of motion prediction. Unfortunately work with motion segmentation/partition is not compatible with the goals of image compression so it can largely be discounted. Fractal methods have been applied with an image partition [43], and this is beneficial since the fractal enhancement can be placed where is most effective. However, no consideration of how to partition the image in terms of the best rate distortion performance has been investigated. Simple polynomial functions with complex canonical polygons were examined by X. Wu [12] who implied that simple structures such as Quad-trees [11] where not as effective as more complicated shapes. No real evidence was used to substantiate this claim and only simple polynomials were used to approximate the image blocks (non orthogonal order 4 polynomials) so this evidence will be discounted. Work has been done in the vector quantisation field, using quad-tree structures to expand the effectiveness of VQ [40] but this is again different from the aim of this chapter to use low complexity transforms. In this chapter we concentrate on fast DCT methods, rather than using complicated systems that can not be encoded by fast algorithms and therefore can not be used in video systems.

In this chapter the threshold method will be described and applied to a quad-tree method for image compression. A new method [5], will be described which uses a sorted list of block errors to produce better rate distortion characteristics than the threshold method. The new method will be applied to a quad-tree for comparison with the threshold method and experiments will be performed to find the best combination of quantisation and DCT order to be used in the quad tree, using error sorting. Using the same set-up as the quad tree a HVD method will be tested and compared to the quad-tree method.

A fractal variation of the quad-tree decomposed image will be assessed using a new limiting method (described in chapter 3), to decide when to use the fractal coefficients generated [2, 6]. The fractal method will be compared to the existing quad-tree partitioned order 6 DCT.

**4.2 Existing Methods**

Image partitioning is about dividing the image into sections that are more appropriate for the application to work on. The partition for image compression is performed to improve the rate distortion characteristic of the compressor and this is important to keep in mind. In this section three methods are considered which are relevant to image compression and they are:

* Quad-tree partition [11].
* Horizontal Vertical decomposition(HVD).
* Triangular partition.

These three methods will be discussed in the following sections, but first it is important to consider how the partition is formed. There are two ways to approach the image partition problem. One way is to start with the smallest element/block that will be considered and merge appropriate blocks together, or another way is to split a set of larger shapes into smaller shapes, where appropriate. In either method the merge or split occurs when the block passes a certain threshold that is measured. The metric can be based on anything but is usually reconstruction error, edge image intensity or pixel intensity value.

To implement the above methods an initial set of macro blocks (not necessarily square) are set-up across the image. The metric is calculated for each block (this could involve coding each block and finding the reconstruction error, summing the edge values across the block or any metric that is appropriate). Any block that has the metric value above a certain threshold is then split. The process is then repeated for all the blocks that have been split, until no blocks exist that are above the threshold or they have reached the minimum block size. The same principles apply to a merging method, but blocks below the threshold are merged. There is very little difference between the merge and split methods, but depending on the level of splitting/merging one can be more efficient at reaching a solution for certain images.

The image metric that is used to split the blocks in this work is reconstructed error. This was chosen because the summation of the edge image is only a prediction of the error [44] to increase the speed of the algorithm (the summation of the pixel intensities is an even worse error predictor). Since DCTs are used in this work the reconstruction error can be calculated from the quantised coefficients, without having to reconstruct the coded image.

**4.2.1 Quad-Tree Partition.**

This is the simplest form of image partition and uses blocks which are square, having sides  long. The image in figure 4-1(a) has been partitioned with a quad-tree and produces the typical structure shown in figure 4-1(b). The quad tree structure produced is very easy to encode, since it is a series of split/don’t split decisions, which can be represented as a binary stream.



Figure 4-1. Diagram of quad-tree partition being applied to a simple image.

It can be seen that the quad tree method uses a large quantity of blocks to form the partition and is not very effective at following the image details. Blocks are not placed where they would be most effective and since there is always a 4 way split, some smaller blocks are used, where they are not necessary, as demonstrated in figure 4-2. Only the top left corner of the block has any activity, and all the other blocks only act as fillers.



Figure 4-2. Diagram showing the active quad-tree blocks necessary to code the image shown.

It is possible to modify the way in which the quad-tree works to avoid these problems of overcoding, as shown in figure 4-3. Only the active blocks are coded at the smaller block size, while the other blocks are coded using the larger block size, but ignoring the area that is describes the smaller block during coding. This method requires a more complicated description than the standard quad-tree and although it is possible to implement it, it is not attempted in this work. The standard quad-tree is not overly disadvantaged in comparison to this method since the ‘filler’ blocks are by definition flat and so require very little information to code.



Figure 4-3. Diagram of quad-tree over coding. This prevents ‘filler’ blocks being used to code areas that do not actually require a high detail of coding.

**4.2.2 Horizontal / Vertical Decomposition (HVD)**

Horizontal/Vertical decomposition works by partitioning the image into rectangular blocks of varying size. The effect is that the partition of the blocks produced follows the detail in the image much better than the quad-tree structure. The method starts with an initial set of macro blocks, which fully tiles the image. These blocks are divided into smaller blocks by applying either a horizontal or vertical partition, the process is shown in figure 4-4. The initial macro block (a) is divided by a vertical or horizontal partition to form the new blocks (b). This is continued in (c) by further dividing the blocks, until finally a sensible partition is formed around the image details (d). There are two ways to decide, whether to partition vertical or horizontally:

* Choose the partition that best improves the picture (in rate distortion terms).
* Alternatively partition horizontally and then vertically.



Figure 4-4. Diagram of HVD applied to a square macro block form figure 4-1.

The partition (horizontal or vertical) is placed in a position that equally divides the metric used. The metric is usually proportional to the error and so this effectively divides the block into two new blocks that have roughly the same reconstruction error. Since the approximation functions are the same for both blocks and they produce roughly the same reduction in error, it is best to divide the error equally between the blocks so that no discontinuities occur in the image.

Using HVD without constraints leads to problems. The blocks produced by the error division method can often become too long and very long blocks do not encode very well, since have a large edge to area ratio and hence produce a increased blocking artefact. To prevent this happening it is best to apply the above method of splitting along with an aspect ratio limitation. As soon as the aspect ratio of a new block drops below a certain amount, it is fixed to that minimum by an aspect ratio limiter. It is common to fix the minimum aspect ratio to about 1:4 in either direction. Blocks above this ratio still code reasonably well but allowing higher aspect ratios leads to larger number of limited blocks.

Although the HVD method is much better than quad-trees at partitioning the image shape, it is not necessarily the best for image compression. Each new split requires the splitting point to be encoded and this can be a large overhead, compared to the binary nature of the Quad-tree. Also the HVD blocks are not square and therefore much slower algorithms must be applied to encode them, since there are less symmetries to exploit in rectangular blocks. In some situations HVD is much more effective than the quad-tree, but this depends on the properties of individual images.

**4.2.3 Triangular Partition**

Triangular decomposition splits the image into a collection of triangles that are very effective at representing the image’s shape. An added advantage is that triangles require very simple transformations to change their shape and so a triangular partitioned image can easily be transformed in such tasks as motion prediction. An example of triangular partition is shown in figure 4-5, the triangles are split by placing four new triangles in each old triangle (similar to quad-tree with triangles). It is possible to vary the positions of these triangles after the split, creating an improved triangular split, which requires more information to store.



Figure 4-5. Diagram of triangular decomposition at various stages.

**4.2.4 Experiment: Evaluate the performance of a Quad-Tree order 6 DCT with varying reconstruction error thresholds.**

The image was initially covered with 32x32 pixel blocks. These blocks were coded with an order 6 DCT, the coefficients were quantised and then Huffman coded (in the same way as experiment 3.4.1). The reconstruction errors were calculated for each initial block (using equation 2.10)and any blocks that had errors above the threshold were split into four sub-blocks. This generated another level of blocks that were again compared to the threshold and split if their reconstruction error was above the threshold. This process continued until no blocks were above the threshold reconstruction error or a 2x2 block was reached. The overall effect is to reduce the errors inside the image to the same level or to distribute the error across the image. There is no simple conversion between the threshold value used and the compression produced, so a threshold cannot be easily used to produce an exact compression, without iteration.

The rate distortion curve in figure 4-6 shows the quad-tree order 6 DCT with thresholds compared to a fixed block size order 6 DCT. It can be seen that the quad-tree method does not improve the performance of order 6 DCT at low compressions. In figure 4-7 (where a section of Goldhill is shown), when Goldhill has been compressed to 0.2BPP by fixed block size and quad-tree methods it can be seen that the RMS error is strongly misleading. The section of Goldhill shown is very active (detailed) and so a lot more bits will be available to the quad-tree encoder in this area than in other flatter areas, and this is what effectively improves the quad-tree codec’s performance. Unfortunately the threshold method does not distribute the error effectively and the certain areas of the picture are not encoded to a high enough detail (figure 4-8).



Figure 4-6. Rate distortion curve for an order 6 DCT using quad-tree partitioned blocks compared to a fixed block size.



(a) Original (b) Fixed Block Size (c) Quad-tree

Figure 4-7. Sections of Goldhill compressed to 0.2 BPP using an order 6 DCT, using quad-tree partitioned blocks and using a fixed block size.



Figure 4-8. Goldhill compressed using a quad-tree threshold method to 0.2 BPP.

**4.3 Error Sorting**

Error sorting [5] is a method that removes the need for thresholds in image partitioning and improves the rate distortion performance of the compressor that it is applied to. The error sorting method does not depend on the splitting method, but processes data from the image to decide what is the best order to split blocks.

**4.3.1 Error Sorting Method**

The process of error sorting is very simple, the image is divided into large macro blocks, which are encoded and then their reconstruction error is calculated. These errors from the macro blocks are sorted into a list and the block with the highest error is split. It is very important that the error metric used is the squared error and not the mean squared error. A large block with a small error may well be much more important than a small block with a large error. An example of this is demonstrated in figure 4-9.



Figure 4-9. Diagram showing that the squared error is a better metric than the mean squared error for partitioning.

When the macro block with the highest error is split, the blocks that are produced can be assessed and the process continued as shown in figure 4-10. This process of splitting the block with the most error can be continued until the encoder has reached one of two criterion:

* The error in the image has reached a predetermined level and a compressed image is produced that takes a certain number of bits to store.
* The bits allocated to the encoder have been exhausted and a compressed image is produced with a certain error.

The latter choice is usually used in image compression, but there are a few situations that require a fixed error. The error sorting method has a distinct advantage over using error thresholds, since any of the above criterion can be continued to reach the exact metric required.



Figure 4-10. Diagram of blocks been split and sub blocks been returned to the sorting list.

The error sorting method is close to being the best way to partition an image for compression but it is not optimal. The block with the most error will usually benefit most from being split but this is not always the case. A more complicated solution would be to find the improvement in error caused by splitting a block and then split the blocks that are improved the most. Unfortunately, this method would be computationally expensive, about four time more than the present method. It will not perform much better in rate distortion terms, so in this work the maximum error method is used for error sorting.

**4.3.2 Orthogonal Error**

Error sorting uses the reconstruction error of each block in the partition to chose the best one to split and so it is necessary to have an efficient method to calculate the reconstruction error. Since orthogonal transforms are generally used in image compression, the property described in equation 2.11 can be used to calculate the reconstruction error of a truncated DCT. The maximum error that can be generated, while compressing a block is

 (4.1)

where  is the maximum mean squared error possible,

*f(i,j)* is the image block pixel intensity at *(i,j)* ,

and N is the number of pixels in the *(i,j)* array.

This maximum error can also be calculated from the coefficients necessary to losslessly reconstruct the image (Chapter 2), as shown in equation 4.2. Combining equations 4.1 and 4.2 leads to equation 4.3.

 (4.2)

where *c(k,l)* is the transform coefficient *k,l.*

 (4.3)

The MSE in the truncated DCT, such as the order 6 DCT used in the previous sections, can be represented in two parts, as shown in equation 4.4.

 (4.4)

where  is the quantised DCT coefficient *k,l,*

 is the quantised coefficient error,

and  are the truncated coefficients needed to losslessly reconstruct the image.

The second term in equation 4.4 can be generated by splitting equation 4.3 into two parts as shown in equation 4.5.

 (4.5)

This means that equation 4.5 can be substituted in equation 4.4 and this produces an equation (4.6) that can be evaluated the calculate the error in a truncated DCT.

 (4.6)

Equation 4.6 makes it possible to quickly calculate the error of an image block without decoding (rendering) it, which is vital for an error sorting system to work effectively.

**4.3.3 Experiment: Vary Quantisation factor for error sorting method using order 6 DCT, to find the best quantisation and compare results to fixed blocks and threshold methods.**

An initial partition of 32x32 blocks was applied to the image. The 32x32 macro blocks were encoded with an order 6 DCT, quantised with a quantisation factor (Q) and Huffman coded. The total number of bits required to code the macro blocks was calculated and the macro blocks were sorted into a list based on error. The block with the highest error was split into four equally sized square sub-blocks, which were encoded with same quantisation factor and their error was calculated. The four sub-blocks were then placed back in the sorted list and the number of bits required to describe the image was updated (the number of bits to describe the old block was removed and the bits to describe the four sub-blocks was added). The process of splitting the highest error block was repeated until a full rate distortion curve was produced. The number of bits required to store the quad-tree was updated during the process and was found to be a small overhead by comparison to the block data (less than 1/block compared to ~20bits/block for the DCT coefficients).

The experiment was repeated for Q={1,2,3,4,5,6,7,8} and a set of rate distortion curves was produced, as shown in figure 4-11. Not all the quantisation factors are shown in figure 4-11 for clarity, but the general trend can be observed.



Figure 4-11. Rate distortion curve for varying values of the quantisation factor Q.

It can be seen that Q=4 is the best quantisation factor. There is very little difference between Q=4, 5 or 6, but figure 4-12 does show that Q=5 and 6 are too heavily quantised in visual terms.



(a) Q=1.0 (b) Q=2.0 (c) Q=3.0 (d) Q=4.0



(e) Q=5.0 (f) Q=6.0 (g) Q=7.0 (h) Q=8.0

Figure 4-12. Sections of Goldhill coded to the same compression ratio (0.2 BPP) using different Q values.



Figure 4-13. Rate distortion curves comparing sorting, thresholds and fixed block size order 6 DCTs.

Figure 4-13 shows the Q=4 result compared to the threshold method (using MSE) and fixed block size method from chapter 3. It is clear that the new quad-tree method out-performs the old methods and has the advantage that it can reach any compression exactly. This is true over the whole range of compressions and comes from the fact that the compression is intelligently distributed over the image by the quad-tree structure.

**4.3.4 Experiment: Compute a rate distortion curve for an implementation of HVD and compare the results to quad-trees using error sorting.**

The image was initially divided into 32x32 macro blocks and each block was coded with an order 6 DCT, using a quantisation factor of Q=4 and the quantised coefficients were Huffman coded. The reconstruction error was calculated for each macro block and they were sorted into a list.

The blocks with the largest error were split into two to reduce the error in the sub-blocks. The split position was chosen by scanning the reconstruction error image of the block and placing the split boundary in a position that gives an equal error to both of the sub-blocks. The splits orientation (Horizontal or vertical) was chosen by comparing the two split orientations and choosing the one which best preserved the aspect ratio of the sub-blocks. When the split orientation and position had been chosen, it was then limited so that the aspect ratio never dropped below 1:4. The final split location was then quantised to make it take even numbered block size since this produced a more favourable DCT block. This quantised split position was then Huffman coded along with a bit describing the orientation. The splitting is clarified in figure 4-14.



Figure 4-14. Diagram showing the splitting of a HVD block.

The process of splitting is continued to produce a full rate distortion curve and this is compared to the quad-tree in figure 4-15 for the Goldhill test image.



Figure 4-15. Rate distortion curves comparing order 6 DCT quad-tree with HVD method.

It can be seen that the quad-tree is marginally more effective than the HVD method. This is quite important since the blocks sizes produced by the HVD process are not powers of two and hence the DCTs take longer to compute for the HVD, making the process much less favourable than the quad-tree, for general images.

In the case of Goldhill the quad-trees are better, because the image is highly detailed and because the quad-trees take less overhead to store than the HVD and so the quad-tree has more bits available to encode the image details. It is also partly, because the DCT coefficients produced by the HVD method are not as statistically similar as those produced by the Quad-tree and hence do not losslessly code as well.

However the quad-tree is not as effective as the HVD method (in rate distortion terms), when the image is low in detail or the detail is concentrated in small regions. The mug shots shown in figure 4-16 show that the HVD can out perform the quad-tree at 0.2BPP. The most important detail in the mug shots image is the edge of the face and this shape detail is coded very well by the HVD. Unfortunately, the quad-tree blurs the edges and uses much more information than is necessary to code the image. Figure 4-17 shows the rate distortion performance curves for the mug shots image and it is clear that the HVD is much better at representing these images, than the quad-tree.



(a) Quad-tree (b) HVD

Figure 4-16. Mug shots at 0.1 BPP, comparing quad-tree and HVD methods



Figure 4-17. Rate distortion curves comparing order 6 DCT quad-tree with HVD method for a mug shot.

The HVD can therefore be used effectively, provided that the type of image to be compressed has high detail in small areas of the image (and this can out weight the added speed of the quad-tree method). The quad-tree is generally more applicable, it is probably not the best solution for specific situations, but it is the best overall.

**4.4 Fractal Partitioned Methods**

Fractal methods (previously discussed in section 3.4) can easily be applied to a partitioned image. Fractals are a useful method to clean up edge details that have been lost in image compression, and this still applies in partitioned methods.

In this section a quad-tree order 6 DCT will be used to demonstrate how fractals can improve a partitioned image. Fractals can be applied to other shaped partitions, but they are more effective, when applied to rectangular or square blocks with an integer fractal. Two versions of the fractal will be examined:

* Fractal coefficients will be used to improve every block in the quad-tree partition (except the 2x2 blocks which are lossless) , as discussed in section 3.5.
* Fractal coefficients will only be used on blocks, where they improve the overall rate distortion performance of the system.

The latter method is implemented by finding the fractal coefficients for every block and comparing their rate distortion gain to the average rate distortion of the order 6 DCT at a particular compression. Coefficients that have a higher rate distortion improvement than the order 6 DCT are preserved and the rest are discarded as proposed in equation 4.7 :

 (4.7)

This limits the use of the fractal to important areas of the image and the rest of the blocks have a zero fractal coefficients, which can be entropy coded to reduce its redundancy.

**4.4.1 Experiment: Compute the rate distortion curves for limited and normal fractals in a quad-tree partition**

A quad-tree partition was used to compress the Goldhill test image using an order 6 DCT in each block. The partition was formed by using error sorting (section 4.3) and the quantisation factor was set to *Q=4*. The quantised order 6 DCT coefficients were Huffman coded to reduce their redundancy.

A fractal coefficient was calculated for every child block in the partition. A parent block was centred around each child block and was set to be twice the size (for a 2:1 contraction mapping of the fractal). The fractal coefficients were quantised and Huffman coded in the same way as the DCT coefficients.

For a particular compression the quad-tree partition was calculated, which allowed an estimate of the rate distortion curve gradient to be calculated (equation 4.7). The fractal coefficients that had been calculated from each block were introduced and this produced a rate distortion point for a 100% fractal quad-tree method. The fractal coefficients were then limited, using equation 4.7, and any that did not conform were zeroed. This produced another rate distortion point for the limited fractal method. The above process was repeated for a range of compressions and the rate distortion curves produced are shown in figure 4-18.

It can be seen from figure 4-18 that fractals do not improve the quad-tree method, even when they are only used where they are useful in the image. It is clear that the overhead of a simple fractal method is not sufficient to improve on the quad-tree order 6 DCT. It is interesting to note that fractals are effective at improving the approximation if the DCT is quantised using Q=2, but the overall rate distortion is not as effective as shown in figure 4-18. The rate distortion curves become identical at low compressions since the fractals are virtually never used (due to the limiting equation 4.7) and 2x2 blocks become much more common (which have no fractal enhancement).



Figure 4-18. Rate distortion curves comparing order 6 DCT quad-tree with a fractal version

**4.5 Summary**

In this chapter properties of images partitioned for compression have been investigated. The error sorting method [5] has been developed to effectively compress the image using a particular partition.

The quad-tree method has been shown to be very effective at compressing images and is a more general solution than the HVD. It has been shown that the HVD is a method suited to specialist images such as ‘mug shots’. Since the quad-tree is relatively simple, and errors can be directly calculated from the DCT coefficients, the quad-tree algorithm can be considered to be a fast, efficient image compressor but it does not quite compete with the rate distortion of JPEG.

The fractal methods have been shown to be ineffective when applied to the quad-tree partition method. More complicated fractal enhancements can be effective with the quad-tree as shown by Wakefield [2, 3].