Algorithm for hierarchical vector quantization of video data

Paul Cockshott, Dept Computer Science University of Glasgow,

Robert Lambert, Dept Computer Science University of Strathclyde

March 24, 1999

Abstract

We present a recursive algorithm designed to construct quantization tables and codebooks for the hierarchical vector quantization of images. The algorithm is computationally inexpensive and yields high quality codebooks.

Introduction

> Vector quantization is an image compression algorithm that is applied to vectors rather than scalars. Scalar quantization maps a large set of numbers to a smaller one and includes such operations as "rounding to the nearest integer," ... Vector quantization rounds off (or *quantizes*) groups of numbers together instead of one at a time. (*Cosman 93*)

Vector Quantization (VQ) has been found to be an efficient technique for the lossey compression of images [11][6]. VQ represents a mapping from a k-dimensional space R_k to a finite subset Y of R_k. This finite set Y is called a codebook. To code a data sequence, an encoder assigns to each data vector $x \in R_k$ an index corresponding to a vector in Y, that in turn is mapped to a code-word y in the set Y by a decoder.

In its simplest implementation for image compression, VQ requires that an image is divided into fixed sized non-overlapping sub-image blocks, each represented by a data vector $x \in R_k$. The pixels in the sub-image block correspond to the elements in the data vector x. Each image data vector

is compared with the entries of a suitable codebook Y and the index i of the codebook entry y_i most similar under some metric to the source data vector is transmitted to a receiver. At the receiver, the index accesses the corresponding entry from an identical codebook, allowing reconstruction of an approximation to the original image.

While simple, this implementation can only compress at a single compression ratio determined by the size of the block used to partition the image and the number of entries in the codebook. Increasing the size of the blocks can in principle raise degree of compression. However, to maintain quality the size of the codebook must be increased as the block size is increased. This limits the practicality of codebooks for large block sizes because of the memory required for storing them. An alternative, variable dimension vector quantization, uses several distinct block sizes with corresponding codebooks. Large blocks are used to encode areas of low detail, while small blocks encode areas of high detail[7]. In this way, an image can be compressed to a pre-determined compression ratio set by the total number of blocks of all sizes used to encode the image.

Codebook size can be reduced by setting the mean intensity of each coding block to zero before it is vector-quantized[1]. Normalising block variance (or contrast) can achieve a further reduction in codebook size[8]. Using this approach, for each block the encoder sends a triple $[i, c, b]$ consisting of the codebook index i, a contrast adjustment c and brightness offset b.

If we perform a linear search of the codebook of size $n = 2^c$(Fig 1), with codebook entries of edge 2^l pixels the complexity of the search operation will be 2^{2l+c}, more generally, a time proportional to the number of codebook entries. It will be appreciated that for real time video encoding this can be prohibitively slow.

What is needed is a lower complexity operation, analogous for example to tree search or hashing, whose time order is a small constant independent of the number of codebook entries. Tree structured VQ [5] provides a lookup technique that is of order $M \log n$ where $M = 2^{2l}$ is the number of pixels in a patch. However even faster lookup is possible using techniques analogous to hashing.

Hierarchical vector quantization [2][3] is a technique used in image and sound compression for using table lookup operations to find the nearest entry B_i from a codebook B of vectors to a source vector v. It is of complexity order M. We can illustrate how it works with two examples.

1. Suppose that we are dealing with two-dimensional vectors, whose members are byte-integers. Suppose that our codebook B_0 contains 256 entries or less. We construct a two-dimensional array of byte-integers T_0 such that $B_0[T_0[i, j]]$ is the closest entry in B_0 to the vector $[i, j]$. Finding the closest entry in the codebook to a given input vector can then be determined simply by looking up the table T_0.

2. Suppose that we are dealing with 4 dimensional vectors $[i, j, k, l]$, again expressed as byte-integers. Using table T_0 we can map these into two-dimensional vectors $[T_0(i, j), T_0(k, l)]$. Suppose that our codebook B_1 contains 256 entries or less, each of which is a 4 dimensional vector $[p, q, r, s]$. We construct a two-dimensional array of byte-integers T_1 such that

$$B_1(T_1(T_0(i, j), T_0(k, l)))$$

is the closest entry in B_1 to the vector

$$[B_0(T_0(i, j)), B_0(T_0(k, l))]$$

Finding the closest entry in the code-book to an given input vector can then be determined by looking up the table T_0, on adjacent pairs of vector elements, and then applying T_1 to the resultant pair.

It is clear that the method can be applied recursively to match vectors of any power of two dimensions with entries in a codebook. By using a hierarchy of lookup operations, on 2-dimensional arrays each holding 2^{16} entries, then a 4x4 patch is encoded in 15 lookup operations as shown in figure 2. This allows very fast video compression to be performed - we find that we can compress CIF sized video sequences at 12.5 frames per second on a 300Mhz Pentium.

Whilst this shows that it is possible to reduce the codebook search to a series of lookup operations, it leaves the not inconsiderable problem of deriving the lookup tables and associated codebooks. Chou and his co-workers use the approach of training a set of codebooks using the GLA algorithm and from these derive a set of encoding tables. We take the inverse approach, construct encoding tables and from these derive codebooks.

Recursive decomposition

It is clear from figure 3 that the lookup process is essentially recursive. Pairs of 8-bit values[1] are combined to form 16-bit table indices, resulting in more 8-bit values, which are themselves combined to word length indices, etc. This gives us a good hint that the process of constructing the tables is itself likely to be a recursive process.

At each level of the recursion during lookup we are reducing two dimensions to one dimension. Let us, for a moment, allow ourselves to assume that what we are dealing with at each input level are two-dimensional metric spaces. Since the overall aim of the whole lookup process is to find which

[1] We use the example of 8-bit values as inputs to the tables for explanatory purposes, the precision of the arithmetic can of course be varied.

16 dimensional vector from our codebook is nearest to a given input vector, let us assume that this overall aim can be reflected at the two dimensional level. In that case the problem of constructing the individual lookup tables can be considered that of tabulating for each point in the two dimensional space the index of the output vector that is closest to it.

Consider the first level of tables. These take as indices two pixels. The pixels can be considered as orthogonal axes in a two dimensional space. We want to generate as output an 8 bit index number that identifies one of 256 vectors, each of two dimensions and for the two dimensional vector so identified to be the one that is most similar to the pair of input pixels. We view the process as having a codebook of 256 pixel pairs, and for all possible pairs of 8 bit pixels determining which codebook entry best approximates it.

If we use the brightness values of a pair of adjacent pixels to define the x and y axes of a graph, then the putative 2-pixel codebook consists of a scatter of points that are to serve as codewords for sample points. Around each codeword there will be a partition of the space within which all sample points are mapped to the codeword. The partition of attraction of an codeword α consists of all points π such that there exists no codeword β for which $d(\beta, \pi) < d(\alpha, \pi)$, for some metric function $d()$. We would expect these domains to polygonal. This is illustrated in Figure 4.

There is an obvious weakness with the scheme outlined above - we do not start out with a set of codewords at the 2-pixel level. We must somehow derive these small-scale codewords. There is also a subtler problem. The cascaded table lookup process shown in figure 2 is essentially recursive . The mapping tables shown in figure 2, actually implement the partitions illustrated in figure 4 by establishing a mapping between points in the plane and their attractors. This implies that there must be analogous diagrams to figure 4 that can be constructed to represent the level 2, level 3 etc mapping tables. However when we come to consider the level 2 mapping table, the pair of numbers that will be fed into it are no longer pixels, but codebook index numbers for a 2-pixel codebook.

When it is a question of looking up a table, this is of no import, from the standpoint of the computer's indexing hardware. But when it is a question of constructing a map of the partitions like figure 4, it does matter what sort of numbers are used to construct the axes. Using bytes to determine co-ordinate positions according to some axes implies that the set of possible bit patterns within the bytes are fully ordered. It implies that the pattern 1000001 is adjacent to 10000010, etc. This is a prerequisite for the construction partitions as per figure 4. Such a partition groups neighbouring points, but the notion of neigbouring is only defined if the set of co-ordinate pairs constitute a metric space. This implies, in its turn, that the axes themselves must be metric spaces - one-dimensional in this case. Thus the codes that we use to label codewords should ideally be metric spaces.

Let T be a mapping table as shown in figure 2. Let $\alpha\beta\gamma\delta$ be points in a two-dimensional space as shown in figure 4. We want neighbouring points $\alpha\beta$, to have codes $T(\alpha), T(\beta)$ that are themselves close together. We want points in the plane $\gamma\delta$ that are far apart to have codes $T(\gamma), T(\delta)$ that are themselves far apart. That is to say, we want $|T(i) - T(j)|$ to be positively correlated with $d(i, j)$ for all ij in the plane, and for some appropriate metric d.

We have deduced that the construction of the mapping tables must meet the following constraints:

1. the construction process must be recursive
2. at each level it must tabulate partitions
3. at each level it must derive the codewords of the partitions
4. it must assign index numbers to the codewords such that distances in index number space are strongly correlated with distances in 2-D space.

To use it effectively we need to construct a series of tables $T_{0,1,...,m}$ and codebooks $B_{0,1,...,m}$ where m is $\log_2$ of the number of pixels in the largest patch that we wish to use for compression.

Chang [2] proposed using the recursive application of the generalised Lloyd algorithm[7][10] to the construction of a hierarchy of codebooks and lookup tables. We present below an alternative approach.

The Construction algorithm

Our algorithm [4] will construct a series of mapping tables $T_1, T_2, ...$ such that $T_1(i, j)$ specifies the codeword index to be output at level 1 when given a pair of horizontally adjacent pixels i, j. At the next level of the hierarchy, T_2 will map vertically adjacent pairs of T_1-mapped horizontally adjacent pairs of pixels. The process is shown schematically in figure 3, where the pixel square

```
i j k l

m n o p

q r s t

u v w x
```

is shown being passed through 4 levels of mapping to give a single output from the last table.

The first phase in the construction of a table at any given level in the hierarchy is to construct a co-occurrence matrix F. This is a square matrix of rank 256 which tabulates how often pairs of inputs occur. Hence F_{ij} will be the frequency with which inputs i, joccur in some sample set. We will term the first level frequency table F^1, the second F^2 etc.

The frequency tables F follow a hierarchy exactly corresponding to that of the mapping tables T. Whilst we are constructing tables at level 1 in the hierarchy, these 'inputs' are horizontally adjacent pixels. Whilst we are constructing tables at level 2 in the hierarchy the inputs i, j are derived from the outputs of applying T_1 mappings to the level below etc. It follows that we can only construct the frequency matrix for level n after the mapping table for level $n-1$ has been built.

The frequency tables are built up by scanning a series of images and taking sample patches from all possible patch origins. An example of such a co-occurrence matrix is shown in Figure 5. It will be observed that at level 1 there is a strong tendency for occurrences to be clustered along the diagonal. This expresses the fact that adjacent pixels are positively correlated. Within the diagonal cluster, there is a further clustering towards the mean brightness.

Given the co-occurrence matrix, the algorithm proceeds as follows:

1. Determine the mean of the distribution.

2. Determine whether the distribution of the data in the x or y direction has the greatest variance.

3. Partition the frequency map into two, using a line normal to the axis of greatest variance passing through the mean of the distribution.

4. For each of the resulting sub distributions, recursively apply steps 1 to 4 until the depth of recursion reaches the number of bits in the index codes (typically 8).

The effect of this partitioning algorithm is illustrated in figure 6. It might seem that it would be mode appropriate to split the distribution on its median, to ensure that each output code would occur with equal frequency but:

> The goal in lossy compression, however is to choose long or short codewords to minimise average distortion for a given bit rate, not to match improbable or probable vectors as in lossless coding. *(Cosman 93)*

Each split represents a bit of information about the vector. We wish these bits to be used in a way that will minimise some error metric, in our case Mean Squared Error(MSE). The mean of a one dimensional Probability

Density Function(PDF) is the single estimator of the distribution which minimises expected MSE. If a one dimensional PDF is split through its mean, the means of the resulting two partitions constitute the pair of estimators which minimise the MSE. If a two dimensional PDF is to be split by a line parallel to one or Other of the axes then, the selection of the axis with greatest variance as the one to split, ensures that the split contributes the greatest amount to the minimisation of estimation errors.

The sequence in which these splits were performed can now be used to assign code numbers to the resultant rectangles. The first split is used to determine the most significant bit of the code number, the second split the next most significant etc. The bit is set to one for the right hand side rectangle of a horizontal split or the upper rectangle of a vertical split. Data points that are widely separated in 2-space will tend to have codes that differ in their more significant bits. We thus arrive at codes that both preserve locality, and recursively ensure that the highest code is assigned to the top right and the lowest code to the bottom left rectangle. Codes thus correlate to mean brightness.

Alongside the partitions of the co-occurrence matrix F, a parallel partitioning is performed on the mapping tables T . Each time F is subdivided, an additional bit is set or cleared for all entries in the corresponding partition of T . Thus the first partition to be carried out determines the most significant bit of all entries. The second level of recursion determines the second most significant bit etc. This process is illustrated in Figure 6. In each case, the partition whose elements have higher values in terms of the axis of greater variance has its bits set to 1 and the other has its bits set to 0. After eight levels of recursion all of the bits of all the bytes in the table T will have been determined.

It should be noted that the training algorithm is fast. Segmentation of a frequency table with 2^{16} entries takes about 10 seconds on a 75Mhz Pentium. For experimental work, this is a considerable advantage over other codebook training algorithms.

Forming the codebooks

What we have described so far constitutes a categorisation system for groups of 16 pixels, or, if the algorithm is extended further, for groups of 64 etc pixels. Given such categories, we have to derive the codebooks that correspond to it. Once we have an appropriately partitioned frequency map, this can be used to form a codebook of the relevant scale. For the codebooks with patch size 4x4 the procedure is straightforward. We scan an image and use the HVQ tables to assign code numbers to sample patches within the image. For each code number we average all of the sample patches assigned to it, and the resulting set of intensity values forms the codebook entry.

Complexity analysis

Assume that we have to train a codebook with 2^c entries using a sequence P training pictures each of dimension x, y. We will consider the complexity of training codebooks for square patches of edge size 2^l pixels. It is evident that for $l << x$ and $l << y$ there are of the order of Pxy training vectors in the picture sequence.

Using the Generalised Lloyd Algorithm [10] we have the following steps:

1. Select an initial set of vectors.

2. For i iterations, (with i determined by the degree of error on the final codebook)

   (a) Compare each training example to each code book entry to obtain a distance measure. This step is of order $Pxy.2^{2l+c}$.

   (b) For the codebook entry closest to the training vector add the training vector to an accumulator and increment a frequency count. This step is of order $Pxy.2^{2l}$.

   At the end of each of the i iterations divide the accumulators by the frequencies to obtain the new codebook vectors. This step is of order 2^{2l+c}

The overall order of the algorithm is thus $iPxy.2^{2l+c}$.

Using the algorithm described in this paper, we have to initially train $2l$ encoding tables.

1. For $j = 1..2l$, do

   (a) For each training vector increment a weight in a 2 dimensional probability density table. Acessing the position in the table will require $2^j - 1$ table lookups, thus the cost of this step is $Pxy(2^j - 1)$.

   (b) Partition the probability density table. For codebooks of size 2^c this implies a spliting tree of depth c so that each point in the table is allocated to a partition c times. Since a partition involves sampling each point twice, once to compute means, once to compute standard deviations, this step is of cost $2^{2c+1}c$.

   The cost of producing the tables is thus made up of two terms, the series $[1, 3, 7, ...2^{2l} - 1]Pxy$ and the partitioning cost of $lc2^{2(c+1)}$.

2. We then form a codebook which involves adding each of the Pxy training vectors to one of 2^c accumulators (a cost of 2^{2l} adds, with $2^{2l} - 1$ array lookups to determine which accumulator, an overall complexity of $Pxy[2^{2l+1} - 1]$.

The overall complexity is thus given by

$$[1, 3, 7, ...2^{2l+1} - 1]Pxy + lc2^{2(c+1)}$$

.

Comparing this with the GLA for a 256 entry codebook using 4x4 patches and a training set of 64 images each 256 pixels square, this works out as the GLA algorithm being about 300 times slower than the one reported here.

Normalisation

It should be emphasised that both when training and when forming the codebook, we use normalisation of the training data. That is, if we are building a set of HVQ tables up to patch size 4 x 4, we take 4 x 4 tiles, normalise their contrast and brightness. During encoding we will take 4 x 4 samples and normalise them prior to using the tables for indexing, thus the tables must be trained under the same conditions. Figure 7 shows a codebook of 4 by 4 patches formed in this way. Patches are allocated code numbers starting with 0 in the top left and 255 in the bottom right in column major order. Observe that patches that are vertically adjacent to one another, and thus differ by 1 in code, are similar in appearance.

When using the HVQ tables to compress an image patch whose size is greater than 4x4 we first shrink the patch to size 4x4, averaging source pixels as we go, normalise the resulting patch and then use the tables to determine the codebook index of the required patch. A consequence of this is that the large scale codebook entries should look like enlarged versions of the smaller ones. This is illustrated in Figure 9. We have experimented with two methods for constructing codebooks for patches of sizes 8x8, 16x16 etc.

1. The 4x4 codebook is simply scaled up by repeatedly doubling the pixels and applying a smoothing filter. This was how the codebook in figure 9 was generated from that in figure 7.

2. For each size an independent run through the source data is made, the source patches are categorised using the same method as in the compressor, and the codebook entries formed as the averages of the source patches that fall into the category. This is shown in figure 8.

The first of these approaches is preferred as it leads to smoother codebook patches with less high frequency noise. Using the second approach one gets a speckled effect on the largest patches, unless the training set is very large.

Results

The Strathclyde Compression Transform (SCT) [9] is based on *variable dimension* VQ. That is, regions of low spatial detail are approximated by vectors with a high dimensionality, while regions of high spatial detail are approximated by vectors of low dimensionality. Using vectors with a dimensionality of 16, 64, 256 and 1024 provides high quality compression from 1 bit per pel down to 0.05 bits per pel, but at a significant computational cost. The HVQ lookup described in this paper represents a mechanism for significantly reducing the encode time.

The codebooks used by the SCT are both non-adaptive and near universal. They are generated from a large selection of natural and man-made images, excluding the Lena image, and are normalized with respect to intensity and contrast.

Table 1 illustrates the compression of the Lena image where the source is monochrome eight bits per pel, 512×512 pels in size. The table shows PSNR and timing results for the SCT using full search for VQ lookup and for the SCT using the HVQ search process described in this paper. For comparison, the results from *Chou & Vishwanath* [3] are also. Note that PSNR is defined as;

$$\mathrm{PSNR} = 10 \log_{10} \frac{255^2}{\mathrm{MSE}}$$

Table 1: Low compression ratio PSNR results for monochrome Lena at 512×512 pels.

Ratio	**SCT**		**Chou & Vishwanath**
bpp	**Full Search**	**HVQ Lookup**	**HVQ Lookup**
	PSNR	PSNR	PSNR
1	36.6	31.9	31.8
0.5	35.3	31.4	29.7

It must be understood that the SCT uses vectors with a very high dimensionality in order to achieve very high compression ratios. Further the SCT is optimized for high compression ratios with regards to both quality and encode time. This can be seen in figure 10 which illustrates the rate distortion curves for the colour Lena image (512×512 at 24 bpp). These curves represent a compression range of 0.05 bits per pel (480:1) up to 1.2 bits per pel (20:1) and show the PSNR for the SCT with full VQ search, the

PSNR for the SCT with the HVQ lookup and for comparison, the PSNR for the JPEG codec. PSNR for the colour Lena image is calculated from,

$$\text{PSNR} = \frac{\text{PSNR}_{red} + \text{PSNR}_{green} + \text{PSNR}_{blue}}{3}$$

For higher compression ratios, the SCT is clearly superior to JPEG in terms of compression quality. This is particularly important when applied to low bit rate video coding where each video frame must be encoded with a very limited number of bits, regardless of the temporal change between frames.

When applied to the SCT, the HVQ lookup process gives a speed up of approximately two orders of magnitude over full search, as shown in figure 11, but at the expense of reproduction quality.

The quality from the HVQ lookup does however give better arithmetic quality than the JPEG codec for compression ratios below 0.25 bits per pel for Lena, and better subjective quality for compression ratios below 0.35 bits per pel. The perceptual qualities from these techniques are shown in figure , which shows a sub-area of the Lena image following the encoding of the full 512×512 Lena image to 0.25 bpp.

Practical application

Whilst landline videophone systems are experiencing a continuous increase in available bandwidths, mobile systems constrained by the scarce resource of the RF spectrum, have to use much lower bandwidths. For example, GSM digital channels have an effective throughput of under 8Kbps. The techniques described here perform well at these very low bandwidths using the restricted processing power available on mobile computers. Codebooks and compression tables derived using the algorithms described here have been incorporated in a demonstration video telephone that is capable of transmitting 128 by 128 pixel video at 10fps along with audio over a single GSM channel.

We acknowledge the financial assistance of Orange, the UK telecoms company, for allowing us to take our ideas to the stage of a working prototype.

References

[1] R. L. Baker and R. M. Gray. Differential Vector Quantization of Achromatic Imagery. In Proceedings of the International Picture Coding Symposium, March 1983.

[2] P.C. Chang, J. May and R. M. Gray, "Hierarchical Vector Quantisation with Table Lookup Encoders", Proc. Int. Conf. on Communications, Chicago, IL, June 1985, pp. 1252-55

[3] P.A. Chou, M. Vishwanath, N. Chaddha, Hierarchical Vector Quantization of Perceptually Weighted Block Transforms, in Proceedings Data Compression Conference, Pages 3-12, March 1995.

[4] Paul Cockshott, Robert Lambert, Vector Quantization, British Patent Application 9622055.3, 1996.

[5] Pamela C. Cosman, Karen Oehler, Eve A. Riskin, and Robert M. Gray, "Using Vector Quantizers for Image Processing." Proceedings of the IEEE, 81(9): 1326-1341, September 1993.

[6] A. Gersho, B. Ramamurthi. Image coding using vector quantization. In International Conference on Acoustics, Speech and Signal Processing, Volume 1 pages 428-431, Paris, April 1982.

[7] A. Gersho and R. M. Gray. Vector Quantization and Signal Compression. Kluwer Academic Publishers, 1992.

[8] R. B. Lambert, R. J. Fryer, W. P. Cockshott and D.R. McGregor, Low Bandwidth Video Compression with Variable Dimension Vector Quantization. In Proceedings of the First Advanced Digital Video Compression Engineering Conference, Pages 53-57, Cambridge 1996.

[9] R. B. Lambert, R. J. Fryer, W. P. Cockshott and D. R. McGregor, "A Comparison of Variable Dimension Vector Quantization Techniques for Image Compression", Proceedings of ECMAST'96, Part II, pp. 655-670, Louvain-la-Neuve, Belgium, May 1996.

[10] Y. Linde , A. Buzo, R.M. Gray, An algorithm for Vector Quantizer Design, IEEE Transaction on Communications, COM-28:84-95, January 1980.

[11] Y. Yamada, K. Fujita and S. Tazaki, Vector quantization of video signals, In Proceedings of Annual Conference of IECE, page 1031, Japan, 1980.

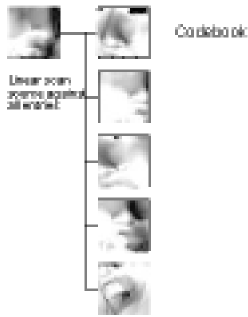


Figure 1: Linear scanning, this is conceptually the easiest way to find a matching entry from the codebook

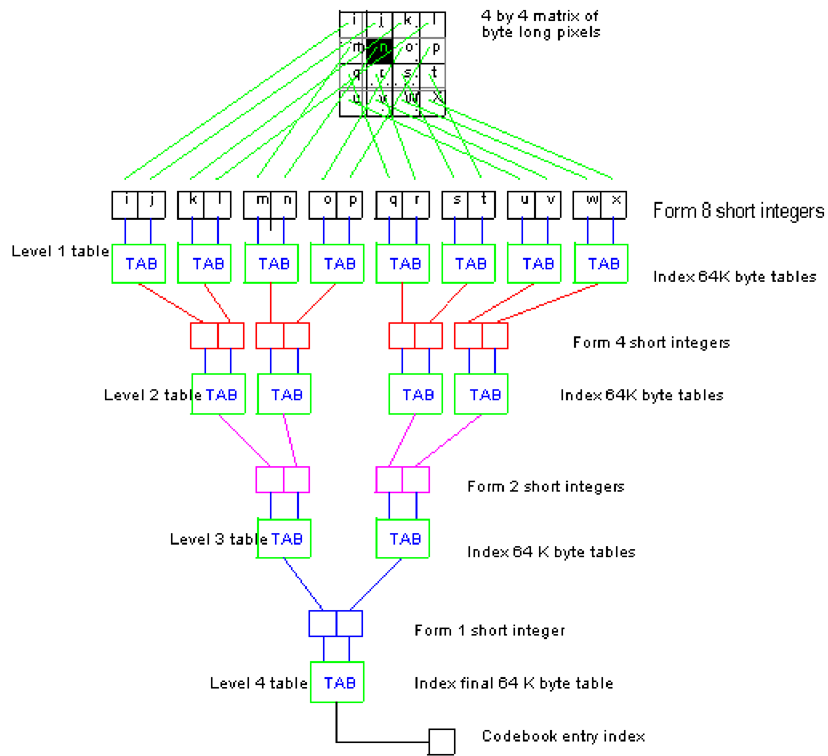


Figure 2: How 15 table lookup operations can map 16 pixels to a codebook entry.

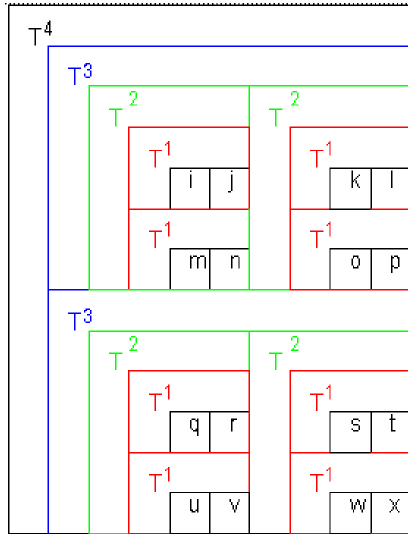
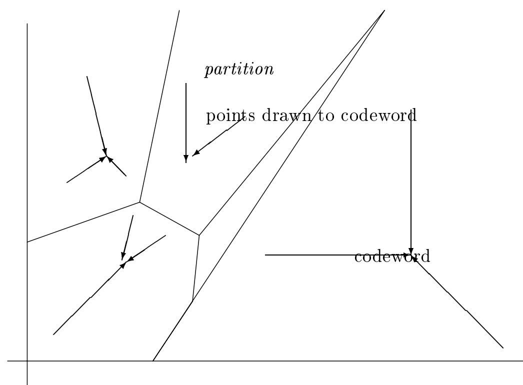


Figure 3: Hierarchy of table lookup operations used in the compression process, showing how horizontal and vertical groupings alternate at different levels.



1

Figure 4: Approximation of pairs with 2 dimensional codewords.

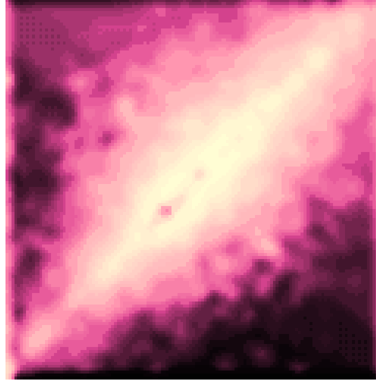


Figure 5: A false colour representation of a frequency co-occurrence map of adjacent pixel values

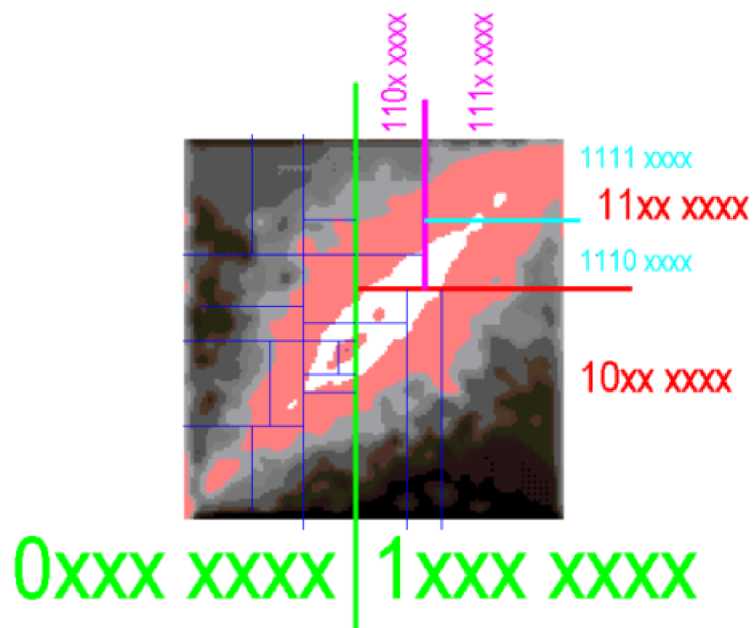


Figure 6: An orthogonal-axis recursive partitioning of a frequency map, with resulting codes Allocating codes

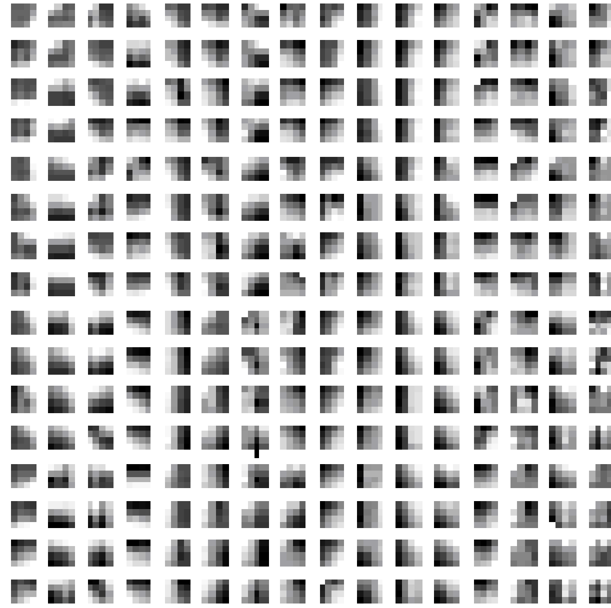


Figure 7: 4x4 codebook with 256 entries generated by the algorithm.

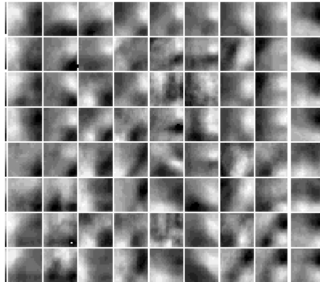


Figure 8: A portion of a 32x32 codebook formed by source patch averaging.

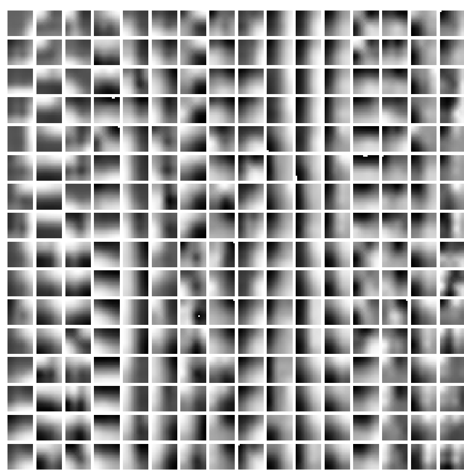


Figure 9: 16x16 pixel codebook formed by expanding and smoothing that in Figure 7

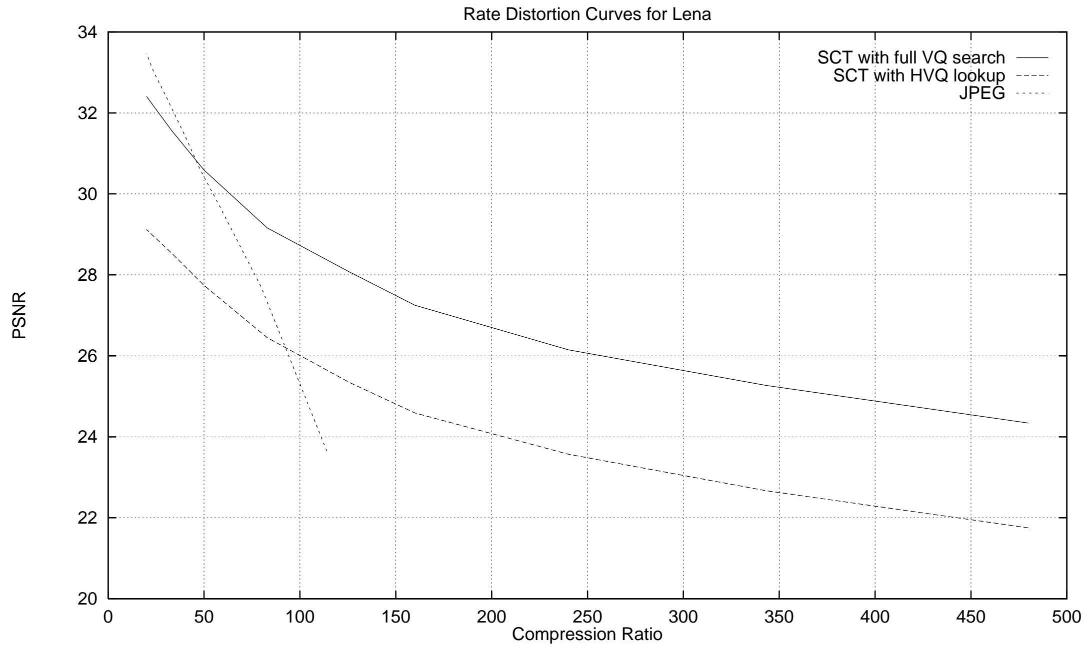


Figure 10: Rate distortion curves for Lena (512×512 at 24 bpp).

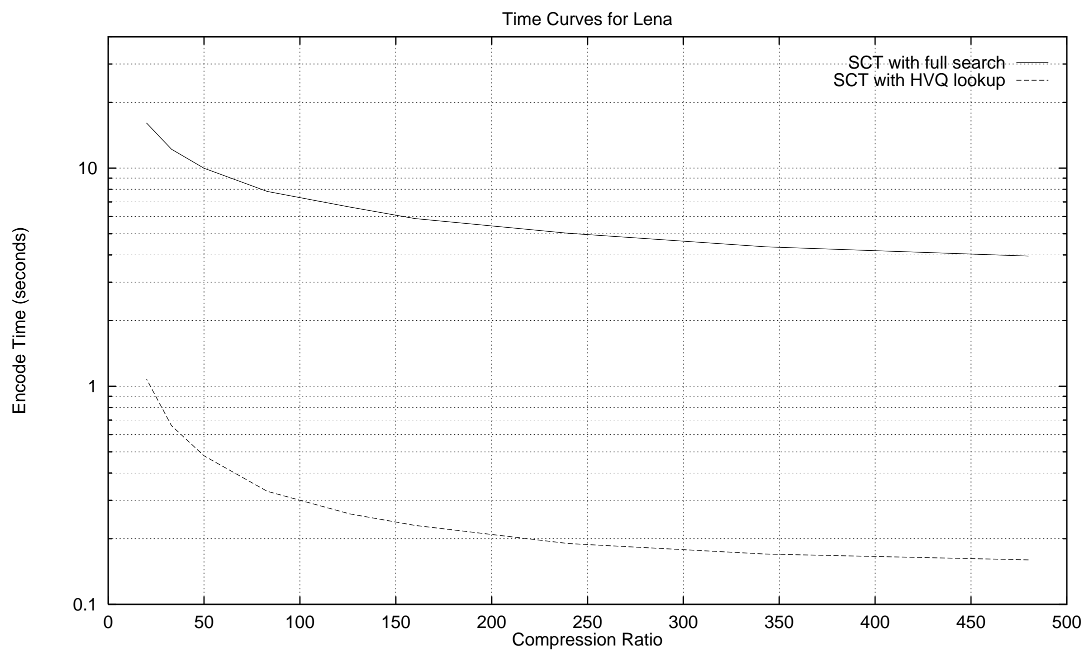


Figure 11: Compression time for Lena (512×512 at 24 bpp). The timing results are for a P2-400Mhz with 128Mbytes of ram, 32k level1 cache and 512k of level 2 cache. MMX is not used by either the off-line or HVQ system.

(a)

(b)

(c)

(d)

Figure 12: Lena (512×512 at 24 bpp) encoded to 0.25 bpp. Picture (a) is the original, (b) is for JPEG, (c) is for the SCT with full VQ search, and (d) is the SCT using the HVQ lookup. Note that only a 256×256 sub-area is shown.