AENSI Journals

Australian Journal of Basic and Applied Sciences

ISSN:1991-8178

Journal home page: www.ajbasweb.com

A Robust Compression System Based on Recursive Decomposition Principle for Hierarchical Data Structure

Raad Alwan

Computer Science Dept., Philadelphia University, Jordan

ARTICLE INFO

Article history:
Received 2 February 2014
Received in revised form
8 April 2014
Accepted 28 April 2014
Available online 25 May 2014

Keywords:
Quadtree, Image compression, Linear Quadtree, decomposition.

ABSTRACT

Background: Quadtree is a hierarchical data structure used to represent spatial information which based on the principle of recursive decomposition. **Objective:** This work concentrates upon the use of Quadtree representation of region data (in two-dimensions) in applications of image processing for encodings/decoding images, in order to minimize the storage space required to save the image and to ensure a rather short access time. The Compression system (CS) is used to store and retrieve a two-dimensional coloured and gray-level images using linear Quadtree. Significant improvements are added to the existing algorithms that are used to represent the pointerless quadtree which requires a less space than the pointer-based representation. **Results:** An analytical study to the time and space required for storing and retrieving the image is carried out and the resulted rate of space compression on maps and binary images is found to be 85%. **Conclusion:** In addition to the compression process, the compressed images have the ability to be displayed.

To Cite This Article: Raad Alwan, A Robust Compression System Based on Recursive Decomposition Principle for Hierarchical Data Structure. ***Aust. J. Basic & Appl. Sci.,*** *8(7): 24-31, 2014*

INTRODUCTION

The applications of digital image processing concepts have become widely spread due to the storage capabilities offered by the computer systems (Gonzalez and Woods, 2011). To represent the pictorial information, a number of techniques had appeared, such as array, run-lengths, and chain codes techniques. Hierarchical data structures such as quadtree and octtree have been used widely as a technique in image processing due to their ability to focus on the subsets of data which leads to efficient representation and improve the execution time (Samit, 2006; Zhai *et al*., 2009; Lo and Hu, 2014).

At first sight, the tree encoding appears to be just a compression trick which saves storage space. The savings size is rated as modest to good size, depending on the image properties, and on the general rather better than for *run_lenght* encoding. Manipulating an image is often far easier to do by operating on its tree code than its *run_lenght* code, or even its pixel array (which is called encoded from) (Wu *et al*., 2012; Zhu *et al*., 2013). Keeping images suitable for display with only minor computation needed at display time is also considered in favour of a tree encoding. Moreover, a pointer structure may simplify any operation on the tree and speed up the access to its leaves, where the data stores (Wu *et al*., 2008).

A quadtree is used to describe a class of hierarchical data structures based on the principle of recursive decomposition of space, which can be differentiated on the following bases: (Shusterman and Feder, 1994; Morvan *et al*., 2007; Zhai *et al*., 2009)

- In representing the type of data which are used.
- The decomposition process.
- The resolution.

The decomposition process may be done in equal parts on each level (regular decomposition), or it maybe forced by the input. Resolution decomposition maybe fixed in advance or maybe governed by the properties of the input data. Quadtree uses successive subdivisions into quadrants of equal-size to the bonded image array. If the array does not consist entirely of 1s (black) or entirely of 0s (white), then it is subdivided into quadrants, sub-quadrants, and so on, until each block is entirely contained in the region or entirely disjoint from it.

The subdivision maybe represented by a tree of degree four and is called a regular or pointer-based quadtree. The root node represents the entire image; the four children of a node is four quadrants (labelled in order NW, NE, SW, and SE) and the terminal nodes represent block with single value (Mao and Ban, 2012).

Corresponding Author: Raad Alwan, Computer Science Dept., Philadelphia University, Jordan.
E-mail: Alwanraad@yahoo.com

It is possible to classify the representation of the quadtree into two types; pointer-based representation (traditional or regular quadtree) and a pointerless representation (implicit pointer). Although a pointer structure may simplify any operation on the tree and speed up the access to its leaves, the memory requirements are unacceptable (Bardera *et al*., 2011; Senapati *et al*., 2012; Letellier *et al*., 2013; Suk *et al*., 2013). This is why the pointerless quadtree representation has become of wide interest (Ayedi *et al*., 2012; Francisco *et al*., 2012; De and Chanda, 2013).

Pointerless quadtree representation can be classified into two groups: the first represents the image in the form of traversal of the nodes of its quadtrees, and the second treats the image as a collection of leaf nodes (Gonzalez and Woods, 2011; Shaila and Vadivel, 2012; Yuen *et al*., 2013).

Linear Quadtree (LQ):

Linear quadtree is a pointerless data structure, which saves at least 66 percent of the computer storage used by a regular quadtree and also reduces the time required to perform some operations (such as rotation, scaling, etc.). The linear quadtree differs from the regular quadtree in the following points:

1. Stores the black nodes.
2. The encoding for each node incorporates adjacency properties in the four principle directions.
3. The path from the root to the node is implicitly encoded in the node representation (Manouvrier *et al*., 2002; Chan, 2004).

The linear quadtree represents the position of each leaf node by using a locational code (Mortan Sequence), and the resulting collection of quadtree leaf nodes are usually stored in a list or in an array.

Compression System (CS):

The compression system architecture consists of a number of stages, each one performs a specific operation. The blocked diagram of Fig. 1 illustrates these stages.

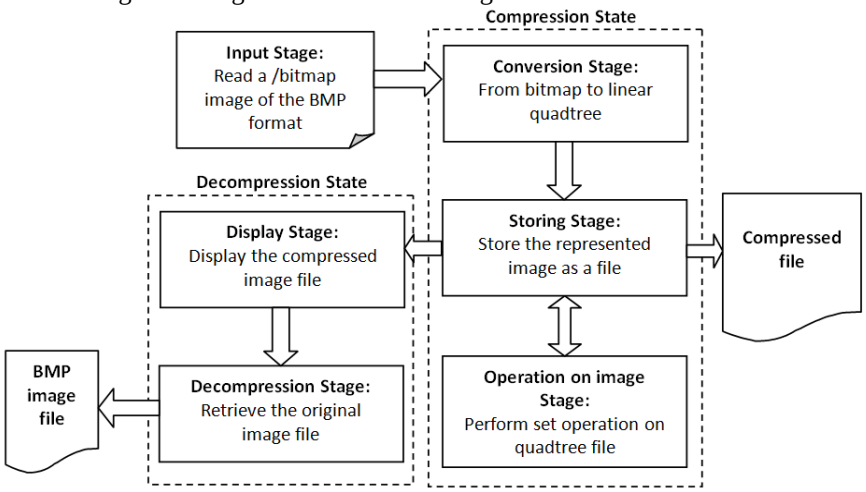

Fig. 1: Input-Stage algorithm.

Input Stage:

The system accepts a bitmap image as an input by reading the file header. Some important information such

Algorithm Input-Stage(BMP image file)
BH=bitmap header
If $BH_{id1} = 'B'$ & $BH_{id2} = 'M'$ *then*

$$byteInRow = \begin{cases} (width+7)/8 & if\ bitPerPel = 1 \\ ((width+7)/8) \ll 2 & if\ bitPerPel = 4 \\ width & if\ bitPerPel = 8 \\ width \times 3 & if\ bitPerPel = 24 \end{cases}$$

Fig. 1: The architecture of CS.

as width, height and bitPerPel must be identified in order to use in the next stages. **Error! Reference source not found.** illustrates the algorithm for this stage.

Conversion Stage:

The input image stored as a raster representation can be converted to other representations such as quadtree. Before the conversion of bitmap image, the data part of the bitmap file must be access which contains the set of pixels defining the image. This stage consists of a number of modules, these are:

Convert the Quadtree Representation module (CQRM):

This module converts a bitmap image structure to one of the pointerless quadtree representations such as linear quadtree. Before that the level of the image must be computed using the depth of the image module.

Compute the Depth of the Image Module (DIM):

This module computes the depth (resolution) of the image. The image's depth represents the number of times that the decomposition process is applied, where the subdivision of the tree, which is recursively decomposed. The process is stopped when either a sufficiently fine resolution is reached (i.e. reached to the depth of the image), or all the nodes become leaves. (see Fig 2)

```
Algorithm DIM (h: hight of the BMP image)
        Depth=0    // initialize the depth
        While h !=0
          h=h div 2
          depth++
```

Fig 2: DIM Algorithm.

Linear Quadtree Representation Module (LQRM):

The raster model is converted directly to a pointerless quadtree representation, which is called a linear quadtree. A linear quadtree stores the image as a collection of leaf nodes. Each leaf node is encoded as a number called locational code which is of base 4, corresponding to a sequence of digits whose values are directional codes that locate the leaf node along the path from the root of the quadtree. The directional code is represented as 0, 1, 2, and 3 according to directions NW, NE, SW, and SE respectively. Fig. 3 shows the detailed algorithm used to build the quadtree.

```
Algorithm Building-Quadtree(BMP image file)
        row= {1, 2, …, byteInRow} of bytes
        ACT= {1, 2, …, n} of pointers of record
        ∀i: i = 1, 2, … number of rows
            Create ACT to keep track of active nodes
            ∀j: j = 1, 2, … byteInRow
                active-node=current active-node ∈ ACT
                if current-pixel.color ≠active-node.color
                    inserted-node=Optimal_Insertion
                    if inserted-node != single pixel then
                        ACT= ACT U inserted-node
```

Fig. 3: Building Quadtree algorithm.

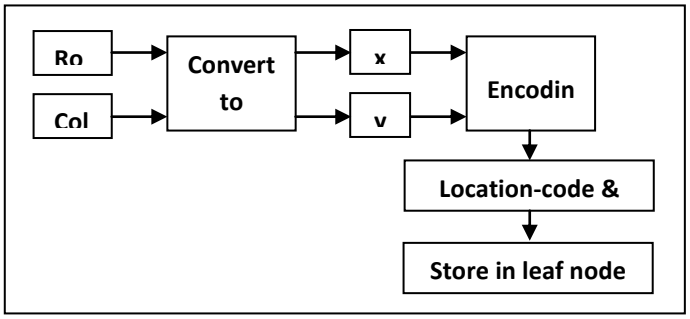

Fig. 4: The modules of conversion stages.

Optimal Insertion Module (OIM):

This module (in the worst case) makes a single insertion for each node in the output quadtree. It is based on processing the image in raster-scanning, as shown in Fig. 4.
The module consists of the following submodules:

A. Convert to Binary submodule:

The input which is an integer value of the coordinate is converted to a corresponding binary value. As illustrated in Fig. 5, this is done in two stages, one for the row coordinate and the other for the column coordinate. The result is assigned to one of the parameters (xc, yc) (one for row and one for column respectively).

```
Algorithm Covert-To-Binary(n: a positive integer)
        B: {bits corresponding to the binary representation of n}
        t=n
        k=0  index of array B
        while (t>0)
            k=k+1
            b[k] = t mod base
            t=t div base
```

Fig. 5: Convert-To-Binary Algorithm.

B. Encoding submodule:

The binary values of (xc) and (yc) which are corresponding to the integer values of the row and column coordinates are considered as input parameters to this submodule. The task of this submodule, is to encode the (xc) and (yc) values to a number called locational code which is of base 4. The output is assigned to a third

Algorithm Encoding(xc, yc: are the binary representation of the rows and columns respectively)
En= {1, 2, …, 10}: En_i *is the locational code*
If xc.length > yc.length then
$\forall i: i = 1, 2, \dots, (xc.length - yc.length)$
yc= 0+yc // append zeros to the left of yc for alignment
else
$\forall i: i = 1, 2, \dots, (yc.length - xc.length)$
xc= 0+xc // append zeros to the left of xc for alignment
$\forall i: i = 1, 2, \dots, (xc.length)$
If (xc_i *&* yc_i *=0) then* $En_i = 0$
If (xc_i *&* yc_i *=1) then* $En_i = 1$
If (xc_i*=1 &* yc_i *=0) then* $En_i = 2$
If (xc_i *=1 &* yc_i *=1) then* $En_i = 3$

Fig. 6: Encoding algorithm.

parameter (En) which is corresponding to locational code. (See Fig. 6).

C. Leaf-Node submodule:

The data structure of the leaf-node consists of a pointer to a record that has two fields; locational code (string type) and color (byte type). This function receives two parameters (locational code and color), and store them in a record.

D. Encoding the locational code submodule:

The total size of locational code and color, which is of 5 bytes size (4 bytes for locational code and one byte for color), is reduced to 3 bytes (2 bytes for locational code and one for color) using the algorithm in Fig. 7.

Algorithm Encoding-Locational-Code (locational-code)
encode=0
$\forall\ ch: ch\ is\ the\ charecter\ in\ the\ location - code$
Shift-left=power of each computed using shift-left operation
num=numerical value in the locational-code
*encod=encod+num*shift-left*

Fig. 7: Encoding-Locational-Code Algorithm.

Storing Stage:

The converted image as a quadtree has to be stored in specific format (Linear QuadTree LQT). The file must have 4 parts, these are:

1. The header 14 bytes long which consists of the following fields
- count : 2 bytes long {number of leaf-node}
- width, height: 2 bytes long {width and height of image}
- filesize : 4 bytes long {file size}
- cbyte, bitperpel: 1 byte long {repetitive color and bit per pixel}
- rep: 2 bytes long {number of repetitive color}
2. RGE table which is the same palette as in the BitMap image file.
3. Path long for each leaf-node, which is 2 bytes long.
4. Color of each leaf-node.

Display Stage:

In this stage and as illustrated in Fig. 8, the linear quadtree image, which is a collection of leaf nodes, is displayed by reading the records that store the locational codes of leaf nodes and their color values. Each locational code is decoded to obtain the coordinates of the upper left corner.

Algorithm Display(linear quadtree image file)
CoC={A_1, A_2, …, A_n}: A_i has three fields: A_i.x, A_i.y, and A_i.color
Read the header of the linear quadtree format
$\forall i \; \exists \; P_{0,i}$: *$P_0$ is the root and P_i are the leaves in the quadtree, and $P_{0,i}$ is a path from the root to a leaf*
Locational-code=decode($P_{0,i}$)
A_i.x and A_i.y =decode-coordinate(Locational-code)
$\forall A_i: i = 1, 2, \dots, n$
A_i.color= color from the image file
Display-linear(CoC)

Function Decode(P_i)
decode= {d_1, d_2, …, d_{10}}
ACT= Building-Quadtree (Image File)
decode= Convert-To-Binary(ACT, 4)
}
Function Decode-Coordinates(En: En is the locational-code)
$\forall En_i: i = 0, 1, \dots, En.length - 1$
Case $\forall En_i$
0: xc=xc+'0' and yc=yc+'0'
1: xc=xc+'0' and yc=yc+'1'
2: xc=xc+'1' and yc=yc+'0'
3: xc=xc+'1' and yc=yc+'1'
En_i.xc= convert to decimal(xc)
En_i.yc= convert to decimal(yc)

Fig. 8: Display algorithm.

Decompression Stage:

The exact original image must be retrieved from the compressed image depending on the retrieval coordinates from the path long for each leaf node in the decompression stage. These coordinates and associated color can produce the missing coordinates and colors among them.

Fig. 9 presents the algorithm used in this stage.

Since the repetitive color and the number of repetition have been stored in the header of the linear quadtree file, we can retrieve the numbers of colors which are eliminated in the first process for encoding can be retrieved.

Operations on Image Stage:

The quadtree is useful for performing set operations such as union and intersection of several images. In order to perform the union or intersection operations, the quadtree for the two compressed images must be obtained first. The quadtrees are traversed in parallel, and an examination of the corresponding nodes is done, then a constructed output quadtree will be outputted. Fig. 10 illustrates the algorithm that is used in the intersection and union operations.

```
Algorithm Decompress (LQT: linear quadtree image file)
        row={1, 2, …, byteInRow}
        ACT={1, 2, …, n}
        BMP.header=LQT.file-size, width, height, bitPerPel
        BMP.palette=LQT.palette
        row=ACT.xc
        col=ACT.xy
        ∀ACT_i: i=1, 2, … n
            If col ≤ yc  then  ∀ACT_j: j=1, 2, …, yc-1
                BMP.color= ACT_j.color
            If row > xc  then  ∀ACT_j: j=1, 2, …, xc
                BMP.color= ACT_j.color
            If node.size ≠ 1 × 1 then add node to ACT
           row=ACT.xc
           col=ACT.xy
```

Fig. 9: Decompress algorithm.

```
Algorithm Set-Operations( A, B: Quadtree images)
        If union operation
          New.RGB=A.RGB ⋁ B.RGB
          ∀ACT_i: i=1,2,…, n
             active-node= ACT_i
             If (A.block.color  ⋁ B.block.color)  ≠ active-node.color
                  New=OptimalInsertion(A)+ OptimalInsertion(B)
                  Storing-Stage
                  Display-Stage
        If intersection operation
          New.RGB=A.RGB ∧ B.RGB
          ∀ACT_i: i=1,2,…, n
              active-node= ACT_i
              If (A.block.color  ∧B.block.color)  ≠ active-node.color
                  New=OptimalInsertion(A)+ OptimalInsertion(B)
                  Storing-Stage
                  Display-Stage
```

Fig. 10: Union and Intersection operations.

Evaluation of (CS):

To test the CS in a real world, different gray-scale images with different sizes have been chosen in order to test its performance. These images are of type (BMP, LQT) and a comparison of the efficiency of CS to process these types is carried out depending on two testing factors. The first is the compression time required to compress the images of type BMP and producing the LQT images, while the second factor is the compression ratio obtained.

This experiment reveals that the compression ratio is inversely proportion with Δg since the compression technique depends on the edges between the differences in the gray scale of the image.

Fig. **12** illustrates an experiment to find out the relationship between the compression ratio and the size of the original image. It is clear that the compression ratio is directly proportion with the size of the tested images.

Conclusion & Future Work:

The following conclusions can be drawn from this work:

1- Using a quadtree can save a big deal of storage and retrieval time by aggregating the data which have identical or similar values.

2- With the pointerless representation and the optimal insertion, the time complexity is proportional to the number of blocks in the output quadtree, whereas in the raster representation it is proportional to the number of pixels in the image.
3- In order to manipulate colour images, the optimal insertion method is found to be better due to encoding technique that reduces the length of the locational code from 4 bytes to 2 bytes.
4- When the quadtree representation in binary images and maps is used, the reduction ratio in space on average is about 85%, while the case is not with the color images.
5- Quadtree encoding falls into group of lossless data when it performs the decompression process.
6- Quadtree is not appropriate to represent the real images due to the many varieties in colors contained in the images.
7- The set operations, such as union and intersection, are easily implemented using quadtree representation.
8- Finally, it is found that the quadtree is quite suitable to compress BitMap images.

Few ideas can be suggested as future work in order to improve the current work. One of these ideas is to mitigate the side effect of the space consumed by the locational information resulted in the worst-case of a quadtree (check-board), it is valuable to use another compression technique, like Huffman coding tree in conjunction with the current technique to achieve this aim.

Another idea is to use the quadtree segmentation in fractal-transform instead of using partition procedure in this encoding.

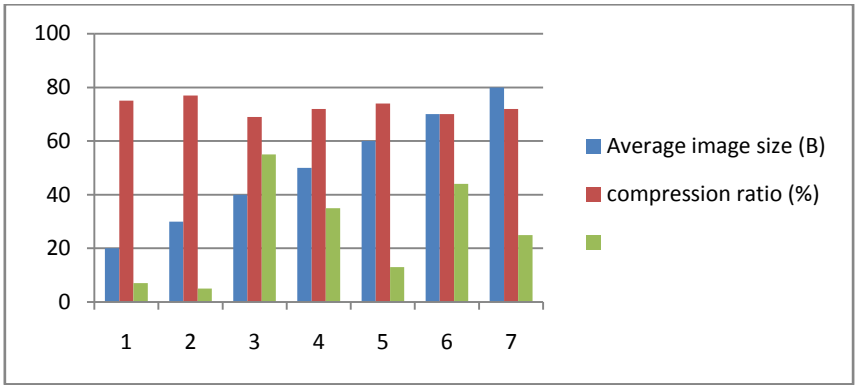

Fig. 11: Compression ratio using CS.

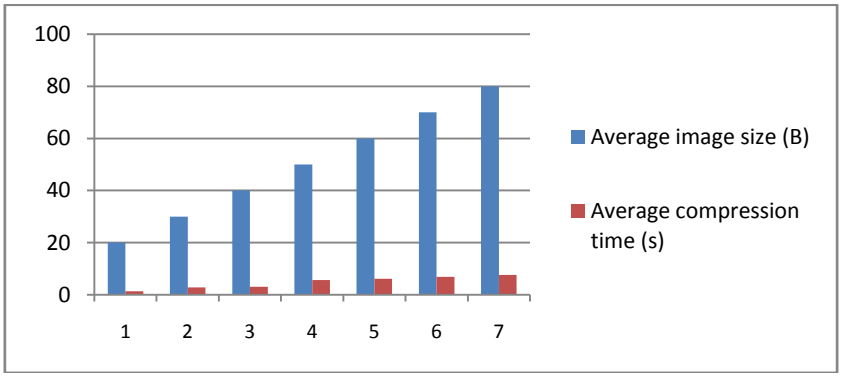

Fig. 12: Compression time compared with image size.

REFERENCES

Ayedi, W., H. Snoussi, M. Abid, 2012. "A fast multi-scale covariance descriptor for object re-identification". *Pattern Recognition Letters*, 33: 1902-1907.

Bardera, A., I. Boada, M. Feixas, J. Rigau, M. Sbert, 2011. "Multiresolution image registration based on tree data structures". *Graphical Models*, 73: 111-126.

Chan, Y.K., 2004. "Block image retrieval based on a compressed linear quadtree". *Image and Vision Computing*, 22: 391-397.

De, I., B. Chanda, 2013. "Multi-focus image fusion using a morphology-based focus measure in a quad-tree structure". *Information Fusion*, 14: 136-146.

Francisco, N.C., N.M.M. Rodrigues, E.A.B. Da Silva, S.R.M.M. De Faria, 2012. "A generic post-deblocking filter for block based image compression algorithms". *Signal Processing: Image Communication*, 27: 985-997.

Gonzalez, R.C., R.E. Woods, 2011. *Digital Image Processing, 3rd*, Pearson Education.

Letellier, V., E. Constant, L. De Marzi, J. Argaud, N. Fournier-Bidoz, A. Mazal, 2013. "AVOQA: Application for voxelization optimized by quadtree algorithm. New CT-scan voxelization tool for MCNPX". *Physica Medica*, 29, Supplement 1**,** e17-e18.

Lo, C.C., Y.C. Hu, 2014. "A novel reversible image authentication scheme for digital images". *Signal Processing*, 98: 174-185.

Manouvrier, M., M. Rukoz, G.V. Jomier, 2002. "Quadtree representations for storage and manipulation of clusters of images". *Image and Vision Computing*, 20: 513-527.

Mao, B., Y. Ban, 2012. "Generalization of 3D building texture using image compression and multiple representation data structure". *ISPRS Journal of Photogrammetry and Remote Sensing*, 79: 68-79.

Morvan, Y., D. Farin, P.H.N. De with, 2007. "Depth-Image Compression Based on an R-D Optimized Quadtree Decomposition for the Transmission of Multiview Images". Image Processing, 2007. ICIP 2007. IEEE International Conference on, V - 105-V - 108.

Samit, H., 2006. *Foundation of Multdimensional and Metric Data Atructure*, Diane D. Cerra.

Senapati, R.K., U.C. Pati, K.K. Mahapatra, 2012. "Listless block-tree set partitioning algorithm for very low bit rate embedded image compression". *AEU - International Journal of Electronics and Communications*, 66: 985-995.

Shaila, S.G., A. Vadivel, 2012. "Block Encoding of Color Histogram for Content based Image Retrieval Applications". *Procedia Technology*, 6: 526-533.

Shusterman, E., M. Feder, 1994. "Image compression via improved quadtree decomposition algorithms". *Image Processing, IEEE Transactions on*, 3: 207-215.

Suk, T.Å., C. HÃSchl Iv, J. Flusser, 2013. "Decomposition of binary imagesâ€"A survey and comparison". *Pattern Recognition*, 45: 4279-4291.

Wu, J., Z. Xu, G. Jeon, X. Zhang, L. Jiao, 2012. "Arithmetic coding for image compression with adaptive weight-context classification". *Signal Processing: Image Communication*, 28: 727-735.

Wu, Q., F.A. Merchant, K.R. Castleman, 2008. *Microscope Image Processing*, Elsevier Inc.

Yuen, C.H., O.Y. Lui, K.W. Wong, 2013. "Hybrid fractal image coding with quadtree-based progressive structure". *Journal of Visual Communication and Image Representation*.

Zhai, G., W. Lin, J. Cai, X. Yang, W. Zhang, 2009. "Efficient quadtree based block-shift filtering for deblocking and deringing". *Journal of Visual Communication and Image Representation*, 20: 595-607.

Zhu, H., C. Zhao, X. Zhang, 2013 "A novel image encryptionâ€"compression scheme using hyper-chaos and Chinese remainder theorem". *Signal Processing: Image Communication*, 28: 670-680.