

A QUAD-TREE DECOMPOSITION APPROACH TO CARTOON IMAGE COMPRESSION

Yi-Chen Tsai, Ming-Sui Lee, Meiyin Shen and C.-C. Jay Kuo

Integrated Media Systems Center and Department of Electrical Engineering
University of Southern California, Los Angeles, CA 90089-2564

Abstract—A quad-tree decomposition approach is proposed for cartoon image compression in this work. The proposed algorithm achieves excellent coding performance by using a unique quad-tree decomposition and shape coding method along with a GIF like color indexing technique to efficiently encode large areas of the same color, which appear in a cartoon-type image commonly. To reduce complexity, the input image is partitioned into small blocks and the quad-tree decomposition is independently applied to each block instead of the entire image. The LZW entropy coding method can be performed as a post-processing step to further reduce the coded file size. It is demonstrated by experimental results that the proposed method outperforms several well-known lossless image compression techniques for cartoon images that contain 256 colors or less.

Keywords—GIF; PNG; JPEG-2000 lossless encoding; Cartoon Image Compression; Quad-tree Decomposition

Topic area—Multimedia Processing.

I. INTRODUCTION

The traditional cartoon contents often contain a sequence of human drawn images to convey a story with simplified curves, colors and motion styles. The simplicity of cartoon contents is however rarely utilized by coding algorithms and standards to achieve a higher coding gain or a lower computational complexity. Cartoon-style images and clip-arts are often entropy-coded using the GIF format [1] that supports 256 colors. However, the GIF format may not be the most efficient way to compress cartoon images. For example, cartoon images often contain large areas of a uniform color. This spatial relationship among colors has not been exploited explicitly in the GIF compression.

In this work, we attempt to improve the performance of cartoon image coding by employing effective color and shape representations based on a quad-tree decomposition scheme. Our method consists of three major steps. First, a color indexing technique similar to that in the GIF compression is used. That is, the 24-bit RGB values are replaced by an 8-bit index. This index color scheme can save as much as two thirds of the original size. Here, the color palette is constructed based on colors in the input image as well as a set of training images. This can be viewed as a pre-processing step. Second, the input image is partitioned into small blocks. For each block, the number of colors is examined. If the block has one or two colors, no further processing is needed and the code for this block will be output; otherwise, the block is subdivided until each subblock has 2 colors or less. After subdivision, the code of each subblock will be output. For a one-color block,

two bytes that represent the block and the color information are output. For a two-color block, in addition to the block and color information, the binary bit pattern that indicates the positions of “0” and “1” in the block also needs to be coded. The bit pattern is encoded using a table look-up technique. A shape palette is created to avoid repeatedly encoding the same bit pattern. Finally, the LZW compression scheme is applied to the data part to yield the final encoded bit stream as a post-processing step.

The proposed method can provide compression as high as 80:1 with excellent preservation of image colors and shapes. As compared to the traditional GIF, or the more recent PNG compression, our new algorithm excels them.

The rest of this paper is organized as follows. Related work is reviewed in Sec. II. The proposed new cartoon image coding algorithm is described in Sec. III. Experimental results are reported in Sec. IV. Finally, concluding comments are given in Sec. V.

II. REVIEW OF RELATED WORK

A. GIF File Format

Cartoon images or graphics are traditionally stored in a file format called GIF (Graphics Interchange Format). GIF offers a protocol intended for on-line transmission and interchange of raster graphic data. A GIF data stream is a sequence of protocol blocks and sub-blocks representing a collection of graphics. The graphics in a data stream are often related to each other in some way, and some control information is shared among them. It is recommended that encoders should group related graphics together to minimize hardware changes in the processing and the control information overhead. On the other hand, unrelated graphics or graphics that require resetting hardware parameters should be encoded separately.

The GIF data stream first gets the color palette that contains colors in an image. The color of every pixel can be easily represented by a color palette index. Then, the size of an image is reduced to one third of the original since only one byte is needed to record the color index rather than three bytes used to represent the RGB color components. Afterwards, these indices will further be entropy-encoded by the famous Lempel-Ziv-Welch (LZW) [2] algorithm, which will be described in Sec. II.C. The GIF compression is lossless. The maximum compression performance achieved by GIF depends on the amount of color repetition in an image. For instance, if an image contains only few flat-colored areas, it could be encoded down to one tenth of the original file size. In contrast, if an image has complex and non-repetitive colors, the file size

reduction can be as low as about 20%. A general rule is that the fewer the number of colors in the original image, the smaller the corresponding GIF file size. The idea of the GIF compression is simple and straightforward. The major drawback of GIF is that the number of colors in an image is limited to 256 or less.

B. PNG File Format

Portable Network Graphics (PNG) [3] is one of the newest and modern image formats for lossless image compression. PNG is a patent-free replacement for the popular GIF file format. The PNG file format supports grayscale, indexed color, and true colors as well as other features, such as transparency, interlacing and gamma correction. The compression performance of PNG is slightly better than GIF.

C. JPEG 2000-lossless Mode

JPEG 2000 [4] is a wavelet-based image compression standard. It was intended to replace the original block DCT-based JPEG standard. JPEG 2000 not only improves coding efficiency over existing JPEG, but also includes more features such as scalability, region-of-interest coding, etc. Moreover, it provides efficient lossy and lossless compression in a single framework. The lossless mode of JPEG 2000 uses Le Gall 5/3 integer filter and a quantization step size of one. In the lossless mode, all bit-planes must be encoded by its core coding algorithm, EBCOT [5], and no bit-plane can be dropped. Although JPEG 2000 offers the lossless coding option, it is not intended to replace the commonly used lossless image file formats. Actually, JPEG 2000 lossless coding is more suitable for photograph-type images with lower contrast of edges, such as medical images, while PNG is more efficient for cartoon-type images that contain large areas of the same color and sharp edges.

D. LZW Lossless Compression

LZW is a dictionary-based lossless compression algorithm. The basic idea of LZW coding is to replace a string of characters with a reference to a previous occurrence of that string. A data dictionary (also called a translation table or string table) is built to record data occurring in an uncompressed data stream. For example, a string of characters in a data stream will be added into the dictionary if it is not presented previously. Then, if the reoccurrence of that particular string is found, the index of the dictionary will be output instead of the string itself. The compression is achieved because the size of the index is usually smaller than the string it represents. There are several dictionary-based encoding algorithms, and they are different mainly in the way to create a dictionary.

III. PROPOSED CARTOON CODING ALGORITHM

The main difference between our proposed algorithm and GIF coding lies in that we exploit the simple shape properties appearing in a cartoon image using the quad-tree decomposition scheme.

A. Quad-tree Representation and Coding

Instead of using 2-D arrays, an image can be represented by a data structure known as the quad-tree [6-8]. A quad tree is a tree whose nodes either are leaves or with 4 children. To represent an image by a quad-tree representation, the image is first divided into 4 quadrants of equal size. Then, each quadrant will be further sub-divided if it has more than one color. The process continues until each quadrant or sub-quadrant (possibly a single pixel) contains only one color. In terms of the tree representation, the root node corresponds to the entire image. Each child node represents a quadrant. The leaf node of the tree represents an area of uniform color in the image. To keep the advantages of block-based image processing, we apply the quad-tree decomposition to blocks of size 16x16, rather than the entire image. We have tried different block sizes and choose the size of 16x16 due to the best tradeoff of implementation convenience and coding performance.

An example of the quad-tree representation is illustrated in Fig. 1, where the illustration is a block of size 8x8. There are a few ways to encode the quad-tree representation as shown in Fig. 1(b). For example, we may assign 1 to the internal node that needs to be further divided and 0 to a leaf node where no more subdivision is required. In the case of a leaf node, the color of the corresponding array should be encoded immediately after 0. Let 0 and 1 represent color green and blue, respectively. Then, if we encode the quad-tree from the root to leaves in a depth-first order, the final binary code of the quad-tree reads as

“1101101000000101010100001100010000010010101000100100010101”.

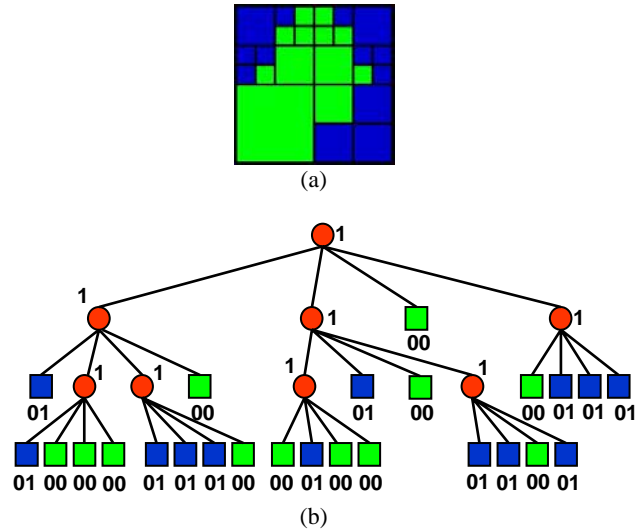


Fig. 1: An example of the quad-tree representation applied to an 8x8 block: (a) block decomposition and (b) quad-tree representation of the block decomposition.

B. Detailed Description of Proposed Algorithm

The proposed algorithm includes 3 major steps. First, each 24-bit RGB value is replaced with an 8-bit index number as done in the GIF compression. An 8-bit color palette is created to represent 256 colors in an image. This can be viewed as a pre-processing step. Second, we adopt a block-based encoding scheme. Within each block, a quad-tree decomposition

scheme is used to encode the color and the shape of that block. This step is our main contribution, which makes our algorithm outperform GIF. Finally, the encoded bit stream is further compressed by the LZW coder, which can be viewed as a post-processing. These three steps are described in detail below.

Step 1: Pre-Processing -- Color Palette Generation

We define the color palette in a way similar to GIF. A color palette with 256 entries is generated based on the image to be coded. Each color in the image is originally represented by 24-bit RGB values and stored in the color palette. The color of each pixel can be referred to by 8-bit index instead of 24-bit RGB values. Note that our algorithm aims at the compression of cartoon images that usually have very few colors. However, if the number of colors in an image exceeds 256, dithering may be applied before encoding so that the color number can be restricted to 256 while preserving high fidelity of the original image.

Step 2: Quad-tree Decomposition and Shape Encoding

After the color palette is generated, the input image is uniformly divided into blocks of size N by N , where N is a power of 2. The choice of block size depends on the size of the input image. Currently, we set $N=32$ for images with size 256×256 or larger. When input images are small, smaller N can be used. The smallest N used is 8. Blocks are processed in a raster scan order. For each block, the number of colors within this block will be examined. There are 3 possible cases.

1. If a block contains only one color, the one-byte block information will be output followed by one-byte code to specify the corresponding color of that block.
2. If a block contains exactly two colors, 4 bytes will be output. The first byte represents the block information. The next two bytes represent the two colors within that block. The last byte is used to encode the binary bit pattern (0's and 1's) to represent the two colors inside that block, which is the shape information. Here, we do not encode the binary pattern directly but use a look-up table called the "shape palette" to record the pattern of 0's and 1's. The value of the last byte is an index indicating the position in the shape palette. The construction of the shape palette will be detailed later.
3. If there are more than 2 colors in a block, the block will be further divided into 4 subblocks of equal size. The sub-division continues until one of the above two cases is met and the corresponding codes are output. We stop block splitting when the block is of size 2×2 , even it has more than 2 colors. In this case, a code that indicates colors of each pixel in the block is output. That is, one byte of block information will be output to a stream followed by a 4-byte code to indicate the color of each pixel. Fig. 2 (a) shows an example of the above three cases.

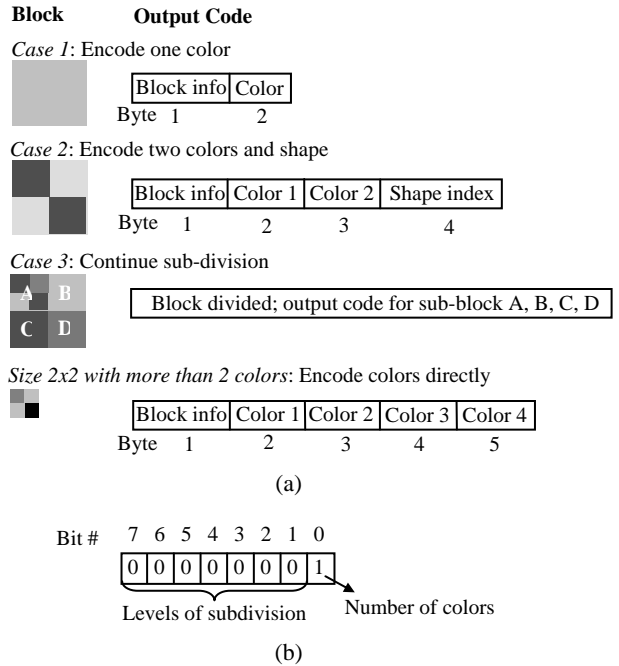


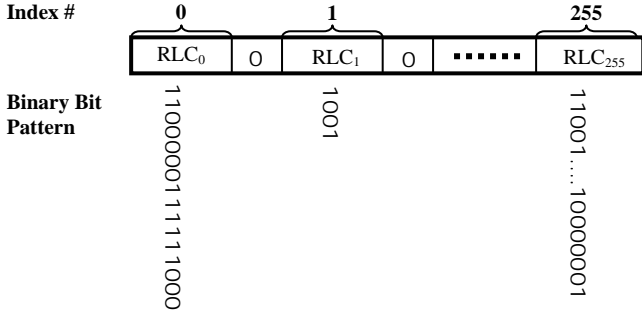
Fig. 2: (a) output codes for 3 possible cases and (b) an example of the block information byte.

The block information byte is organized in the following way. Bit 7 to Bit 1 indicates levels of subdivision. Bit 1 is set to "1", when the block is subdivided once. The "1" is shifted to the left if there are more levels of subdivision, *i.e.* when Bits 7-1 are:

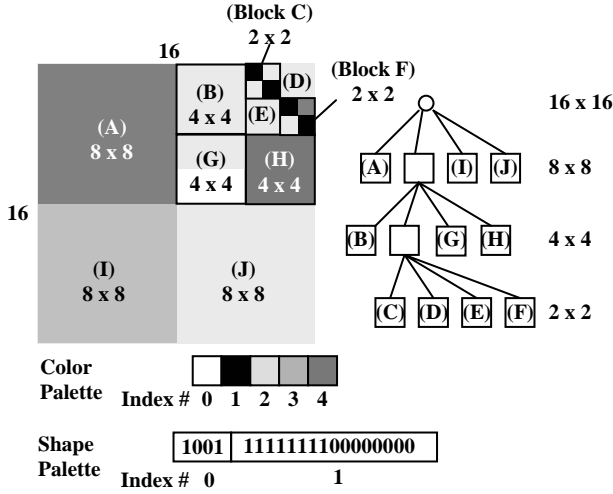
- "0000000" – undivided
- "0000001" – one level of subdivision
- "0000010" – two levels of subdivision
-
- "1000000" – seven levels of subdivision

If the block has one color (*i.e.*, Case 1), bit 0 is set to "0". If the block contains exactly two colors (*i.e.*, Case 2), bit 0 is set to "1". In the case of a 2×2 block with more than 2 colors, colors in this block will be encoded directly. Bits 7-1 will indicate if the subdivision already reaches the single-pixel level. If it does, bit 0 becomes a "Don't Care" bit. An example of the block information byte is shown in Fig. 2 (b).

As mentioned previously, each color is coded using one byte. The value of that byte is the index of the color in the palette. At any stage of subdivision, if a block or a sub-block contains exactly two colors, the binary bit pattern showing the position of each color needs to be coded as well. If we set the color at top-left corner as "1" and the other color in the block as "0", the binary bit pattern is the pattern of "1" and "0" in the two-color block scanned in a raster-scan order.



(a)



Let $\langle N \rangle$ denote output code for block N
 $\langle A \rangle = "00000010\ 00000100"$
 $\langle B \rangle = "00000100\ 00000010"$
 $\langle C \rangle = "00001001\ 00000001\ 00000010\ 00000001"$
 $\langle D \rangle = "00001000\ 00000010"$
 $\langle F \rangle = "0001000x\ 00000001\ 00000100\ 00000010\ 00000001"$
 $\langle G \rangle = "00000101\ 00000010\ 00000000\ 00000001"$
 $\langle I \rangle = "00000010\ 00000011"$
 The final code output of the 16x16 block is:
 $"\langle A \rangle \langle B \rangle \langle C \rangle \langle D \rangle \langle E \rangle \langle F \rangle \langle G \rangle \langle H \rangle \langle I \rangle \langle J \rangle"$

Fig. 3. Encoding examples (a) an example of the shape palette and (b) an example of encoding a size 16 x 16 block.

Similar to color coding, the bit pattern is not encoded directly, but based on a look-up table indexing technique. We use a table of size 256 to record all bit patterns appearing in the image. We call the look-up table the "shape palette", and it is placed at the beginning of the output file right after the color palette. The binary bit pattern of two-color blocks is encoded using an 8-bit index, and the value of index locates the position of the pattern in the shape palette. Therefore, whenever we encounter a two-color block, we first check if its bit pattern is already in the shape palette. If yes, the index of the bit pattern in the shape palette will be output; otherwise, it is a new pattern and should be added to the shape palette. The newly created index for that pattern will be output when encoding this block. The number of entries in the shape palette increase as the coding process proceeds. The shape palette is completed when the table is full or no more new

binary pattern is found. It is possible that the number of binary patterns exceeds 256. In such a case, a larger-sized table can be used. We observe that a table of size 256 is enough to record all binary patterns in most cases for cartoon images with simple drawing and large regions of flat colors.

Fig. 3 (a) illustrates the shape palette coding idea with an example of $N = 16$. Each entry in the shape palette records the binary bit pattern of a two-color block (size of 16x16 or smaller) and it is encoded losslessly using the run-length code (RLC). Since RLC results are of variable length, two entries are separated by a single byte of zeros. An example of encoding a 16 x 16 block is shown in Fig. 3 (b).

Step 3: Post-processing -- LZW Compression

To further reduce the size of the encoded bit stream, lossless LZW coding is applied to the data part of the output bit stream. The final compressed bit stream will be organized in the format as shown in Fig. 4.

Color Palette	Shape Palette	Compressed Data
---------------	---------------	-----------------

Fig. 4. The format of the output bit stream.

IV. EXPERIMENTAL RESULTS

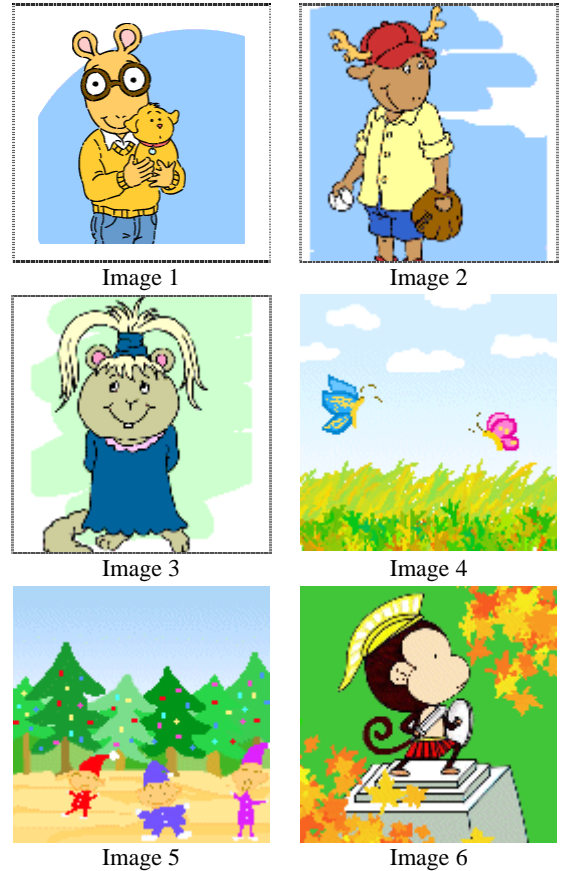


Fig. 5. Test images.

We applied the proposed algorithm to six test images as shown in Fig. 5. All test images are of size 256 x 256. In all experiments, the default block size is set to 32 x 32 and both color and shape palettes are implemented by a table of 256

entries (*i.e.*, 8-bit index). We tested our scheme with and without LZW post-processing. The results are also compared with other lossless compression methods, including GIF, PNG, and JPEG 2000 lossless. The PNG files were generated by JASC Paint Shop Pro. The results of JPEG 2000 lossless coding were obtained using a shareware called Morgan JPEG toolbox developed by Morgan Multimedia. All obtained results are shown in Table 1.

Table 1. Coding performance comparison of several algorithms.

	Method	Size (bytes)
Image 1	Raw	196608
	J2K-lossless	14426
	GIF	6616
	PNG	6355
	Proposed (step 1&2)	4620
	Proposed + LZW	2446
Image 2	Raw	196608
	J2K-lossless	15507
	GIF	6362
	PNG	6059
	Proposed (step 1&2)	5489
	Proposed + LZW	2916
Image 3	Raw	196608
	J2K-lossless	18570
	GIF	6617
	PNG	6602
	Proposed (step 1&2)	5393
	Proposed + LZW	2863
Image 4	Raw	196608
	J2K-lossless	85218
	GIF	21866
	PNG	18577
	Proposed (step 1&2)	24794
	Proposed + LZW	11780
Image 5	Raw	196608
	J2K-lossless	68739
	GIF	15349
	PNG	12699
	Proposed (step 1&2)	11422
	Proposed + LZW	5608
Image 6	Raw	196608
	J2K-lossless	79425
	GIF	20348
	PNG	18212
	Proposed (step 1&2)	10774
	Proposed + LZW	5433

We see from the results in Table 1 that the proposed method outperforms all other methods in all test images. Besides, the use of the LZW entropy coder as a post-processing step does help substantially. The performance of the proposed algorithm varies according to the image content. An image is easier to compress when it contains more flat areas, less line drawing and fewer colors. For example, the compression ratio of Image 1 by the proposed method is as high as 80:1. The substantial bit saving is mainly due to the large blue background area, which is flat and contains only few colors. Even in the worse case, which is Image 4 with more details in

the grass area, a compression ration of around 16:1 can still be achieved. The proposed method performs much better than PNG. The sizes of files generated by the proposed method are about 1/2 to 1/3 of that generated by PNG on the average.

There are several reasons to explain the superior performance of the proposed method over other existing color platted-based techniques. First, the quad-tree decomposition is better adaptive to local image contents and more efficiently exploit large flat areas. Second, shape coding is able to reduce the cost of encoding line drawing and some edge structures. Experimental results also demonstrate that JPEG 2000 lossless coding is actually not suitable for cartoon or clipart-type images that have sharp edges and limited colors. Its performance is even worse than GIF. This is because sharp edges and line drawings in cartoon images result in a lot of significant coefficients that need to be encoded in the JPEG 2000 lossless mode. Consequently, the coding performance is substantially degraded. In terms of complexity, since a block-based approach is adopted to avoid a large quad tree, the proposed method only increases complexity slightly as compared with GIF and PNG.

V. CONCLUSION

The simple color and shape properties in a block of cartoon image were exploited using quad-tree decomposition and color and shape palettes for efficient coding. The resultant bit stream can be further encoded by the LZW coder as a post-processing step. It was demonstrated by experimental results that the proposed method gives significantly better performance than other well known lossless image compression methods such as GIF, PNG, J2K-lossless.

REFERENCES

- [1] Graphics Interchange Format(sm) Version 89a (c) 1987, 1988, 1989, 1990, Copyright CompuServe Incorporated Columbus, Ohio, 31 July 1990.
- [2] Mark Nelson, "LZW data compression," *Dr. Dobb's Journal*, Oct. 1989.
- [3] ISO/IEC 15948:2003 (E), *Portable Network Graphics (PNG) Specification (Second Edition) Information technology - Computer graphics and image processing - Portable Network Graphics (PNG): Functional specification*, November 2003.
- [4] ISO/IEC 15444-1:2004, *Information technology -- JPEG 2000 image coding system -- Part 1: Code coding system*, Sept. 2004.
- [5] D. Taubman, "High performance scalable image compression with EBCOT," *IEEE Trans. on Image Processing*, 9 (7), pp 1158-1170, 2000.
- [6] G. M. Hunter and K. Steiglitz,, "Operations on images using quad trees," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. PAMI-1, no. 2, pp. 145-153, April 1979.
- [7] H. Samet, "Region representation: quad trees from binary arrays", *Computer Graphics & Image Processing*, vol. 13, no. 1, pp. 88-93, May 1980.
- [8] T. Markas and J. Reif, "Quad tree structures for image compression applications", *Information Processing & Management*, vol. 28, no. 6, pp. 707-721, 1992.