Compression of Compound Images and Video for Enabling Rich Media in Embedded Systems

Amir Said
Imaging Systems Laboratory
HP Laboratories Palo Alto
HPL-2004-89
May 11, 2004*

E-mail: said@hpl.hp.com, said@ieee.org

compound, document, video, compression

It is possible to improve the features supported by devices with embedded systems by increasing the processor computing power, but this always results in higher costs, complexity, and power consumption. An interesting alternative is to use the growing networking infrastructures to do remote processing and visualization, with the embedded system mainly responsible for communications and user interaction. This enables devices to behave as if much more "intelligent" to users, at very low costs and power. In this article we explain how compression can make some of these solutions more bandwidth-efficient, enabling devices to simply decompress very rich graphical information and user interfaces that had been rendered elsewhere. The mixture of natural images and video with text, graphics, and animations simultaneously in the same frame is called compound video. We present a new method for compression of compound images and video, which is able to efficiently identify the different components during compression, and use an appropriate coding method. Our system uses lossless compression for graphics and text, and, on natural images and highly detailed parts, it uses lossy compression with dynamically varying quality. In addition, since it was designed for embedded systems with very limited resources, its program has small executable size, and has low complexity for classification, compression and decompression. Other compression methods (e.g., MPEG) can do the same, but are very inefficient for compound content. High-level graphics languages can be bandwidth-efficient, but are much less reliable (e.g., supporting Asian fonts), and are many orders of magnitude more complex. Numerical tests show the very significant gains in compression achieved by these systems.

Compression of Compound Images and Video for Enabling Rich Media in Embedded Systems

Amir Said*
Hewlett-Packard Laboratories, Palo Alto, CA 95014

ABSTRACT

It is possible to improve the features supported by devices with embedded systems by increasing the processor computing power, but this always results in higher costs, complexity, and power consumption. An interesting alternative is to use the growing networking infrastructures to do remote processing and visualization, with the embedded system mainly responsible for communications and user interaction. This enables devices to behave as if much more "intelligent" to users, at very low costs and power. In this article we explain how compression can make some of these solutions more bandwidth-efficient, enabling devices to simply decompress very rich graphical information and user interfaces that had been rendered elsewhere. The mixture of natural images and video with text, graphics, and animations simultaneously in the same frame is called compound video. We present a new method for compression of compound images and video, which is able to efficiently identify the different components during compression, and use an appropriate coding method. Our system uses lossless compression for graphics and text, and, on natural images and highly detailed parts, it uses lossy compression with dynamically varying quality. In addition, since it was designed for embedded systems with very limited resources, its program has small executable size, and has low complexity for classification, compression and decompression. Other compression methods (e.g., MPEG) can do the same, but are very inefficient for compound content. High-level graphics languages can be bandwidth-efficient, but are much less reliable (e.g., supporting Asian fonts), and are many orders of magnitude more complex. Numerical tests show the very significant gains in compression achieved by these systems.

Keywords: embedded systems, image and video coding, mixed raster content, remote visualization.

1. INTRODUCTION

It is an inevitable trend to make consumer devices more "intelligent" and loaded with features, since—whenever this makes sense—it increases the perceived device value to customers. This goal can be achieved in two ways. First, it is possible to embed powerful processors to these devices, plus the necessary software, GUI, real-time operating system, etc. However, to achieve the state of the art in user interfaces, this approach can be become quite expensive due to the price of hardware and software development, licensing fees and royalties. In addition, the processing power translates in a great increase in power consumption, with quite serious cooling and battery-life problems for portable devices.

A second way to achieve this goal depends on the reliable availability of wireless or wired networks. In this case, the embedded processors can have much less computing power (and be quite cool), because the applications can run remotely, and the resources in the device can be used to improve the user experience with better displays, sound, and interaction. It is interesting that a similar approach had long been in use by mainframes and supercomputers, with varying degrees of success. Presently any desktop personal computer has the computation power that only supercomputers used to have, and they can similarly be used to process jobs requests from many different users. However, we can predict that in the future the "users" will be the systems embedded in a variety of devices. Figure 1 show a diagram of such a system.

We are currently seeing both approaches being adopted. The first is the most straightforward, and obviously the one initially preferred by the hardware and software vendors. The second solution started being considered seriously when designers realized the widespread availability of low-cost wireless LANs. They also realized that, before implementation of this approach it is necessary to decide if the system should use a high-level graphics language and protocol[1] (e.g., X-11), or the radically simple approach of just "capturing" in real time the remotely rendered screen and transmitting it as a video stream, as shown in the upper part of Figure 1.

* E-mail: said@hpl.hp.com, said@ieee.org.

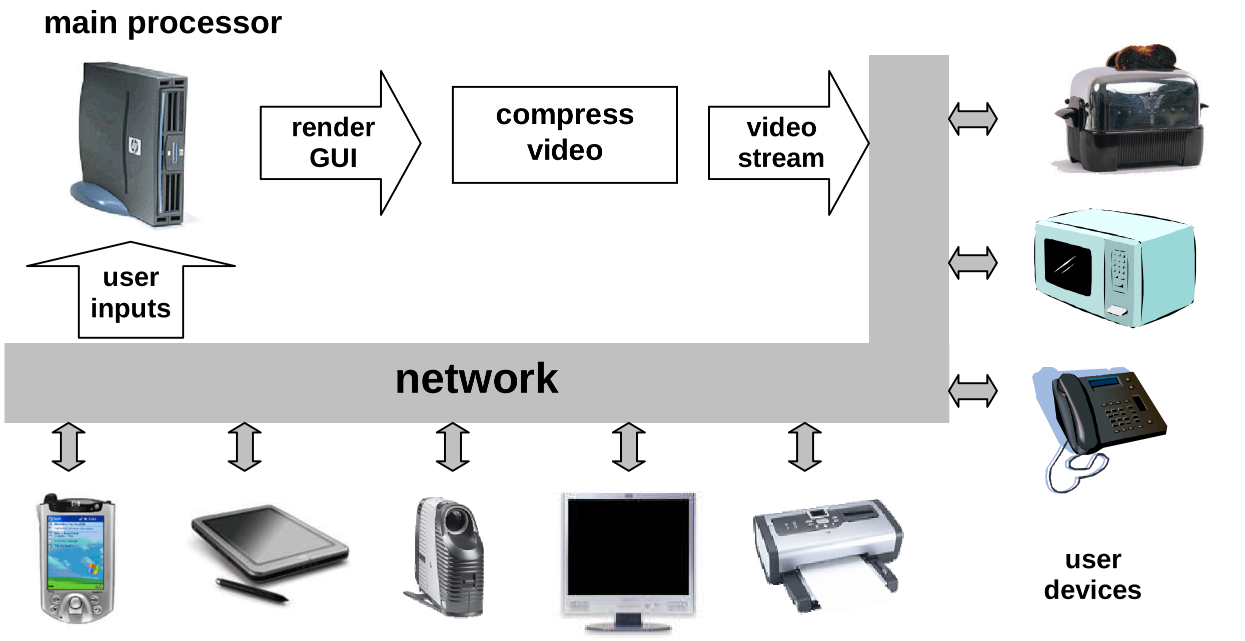


Figure 1 – System based on screen-capture for allowing a main processor to render demanding programs, GUIs, and graphics that are used by devices with embedded systems.

Below we present some characteristics of these two approaches

a) High-level graphics languages and protocols.
   - Data can be parsed to obtain high-level semantic information.
   - May work only if supported from software application.
   - Unreliable, can break down completely. Needs careful installation.
   - Yields inconsistent results with different fonts, languages, and on different platforms.
   - Important legal issues with licensing and Copyrights (from fonts to content format).
   - Rendering (visualization) programs with large size, and greater power consumption.
   - Lowest bandwidth on text and graphics. Can actually use more bandwidth for images, video, some graphics and animations.

b) Compressed screen video.
   - Semantic information is lost during rendering.
   - Can work with practically any application.
   - Much more reliable. Decoder can be unbreakable.
   - Small and inexpensive programs, low power consumption.
   - Bandwidth and quality depends on effectiveness of the compression method.

The fact that high-level language *may* keep the semantic information, which is lost after rendering, has usually created a strong initial preference for high-level approaches. The other problems of the high-level languages were frequently considered transitory, and it was assumed they could be solved because of the dominance of a single company on a certain platform, or that that the industry would soon agree on a unique reliable standard. However, we can see now that these problems had not been solved satisfactorily, and there is little confidence that they will be in the near future.

What the second approach lacks in versatility is compensated by its reliability and low cost, but it is believed that it also requires very large bandwidth usage. However, this belief is based on completely obsolete ideas of the type of graphic content most used today. The assumption that the graphic languages are bandwidth-efficient is commonly based on very simple examples, like the fact that a single graphics command can "paint" a large screen area. In reality, high-level graphics languages may need comparable or even higher bandwidth due to the following facts.

a) Current user interfaces uses shading and other tricks extensively to create a 3-D illusion, greatly increasing the number of graphics commands for even simple parts, while many visual elements are composed of several texture components stored as images[1];

b) Future interfaces will use more realistic 3-D graphics components—the type currently used by computer games—producing an even richer and more complex visual, but also requiring great amount of graphics information[1];

c) Graphics languages or protocols may not support the best image and video compression methods, or they may incorrectly chosen by the user (e.g., see common use of 256-color GIF images);

d) While graphics languages tend to use very low bandwidth for text on a single font, current interfaces may use many fonts, and there is a bandwidth increase resulting from embedding fonts.

What had been missing is the knowledge that the bandwidth needed for compound video could be very significantly reduced with specialized compression methods,[5,7] and that it is proportional to the video resolution and complexity of the visual scene, but not to the amount of data needed to create it. For instance, in realistic 3-D applications each frame may be defined by models with millions of polygons, using lots of texture information, and rendered using very sophisticated shading (ray-tracing) algorithms, but the result is always one screen of video. It is also interesting to note that the two solutions discussed above are not exclusive. The solutions based on compressed video are so inexpensive that they can be easily supported together with other techniques.

In this document we present a set of efficient compression methods for compression of compound video and images. In Section 2 we review the features of compound video and images, and present the main compression techniques developed for them. In Section 3 we introduce our new coding system, and explain the choices we made on its different components. Section 4 contains some compression results demonstrating how the specialized coding method can be much more efficient than those designed for lossless compression or natural images.

2. COMPOUND IMAGES AND VIDEO

The components that compose a compound video or image can be classified in many different ways, depending on the type of analysis that is intended. Just to have a good intuition of the nature of the problem we can use one of the simplest classifications.

a) Compound still-image components
    - Text, background, graphics.
    - Natural images (includes synthetic image rendered to look natural).

b) Compound moving-image components:
    - Moving text and graphics animations.
    - Natural video (includes synthetic video rendered to look natural).

For compression applications we have particular ways of defining the different parts in terms of the compression method needed. The first bullet in each class—text and graphics—corresponds to content that is quite "clean", i.e., there is little or no noise, and it has well defined patterns and color gradients, and commonly has crisp edges. The second bullet—natural-looking content—is definitely noisy, with crisp and blurred edges, and complex random textures. Note that these properties are the ones that really matter for compression applications, i.e., if an image region has all the compression properties of a text/graphics region, we must classify it as "text/graphics" for proper compression, independently of the actual classification of the region by human observers.

Using this classification we have to decide the strategy for compressing each of the components. First, we must note that the psychological requirements for text are quite strict, and are based on our reading skills and our experience with books. For instance, the quality of text should not be evaluated in a static manner, but instead by how much it affects our reading speed. Artifacts around text give the impression that the text is "dirty," while blurring and aliasing can reduce the legibility. Extensive experience on compression of natural images and video, on the other hand, has shown that this content type is quite resilient to compression, and that the visual compression artifacts become objectionable only when the image pixels had been modified by very significant amounts.

The observations above indicate that for compound image and video coding we need, at the very least, to separate these two types of components, and use different coding and quality setting for each. Below we present the characteristics of the three main approaches for compound image compression.[5]

(a) **Full image analysis and segmentation with arbitrary region boundaries.** In this approach a full segmentation of the image is done to separate the different image components.[3,11] Next, the boundaries of all the regions are coded, and finally different coding methods are applied to each region, depending on its type. This method is best suited for document compression when the classification is also used for document understanding, including stages like character recognition (OCR) and translation. In this case the computational effort is quite high, but is indispensable to obtain high-level semantic information. There are many difficulties involved in the use of this approach for compression only. The computational complexity of the morphological analysis and coding arbitrarily shaped regions is relatively quite high. Furthermore, it commonly requires visiting the same image area several times, which increases the memory access times considerably, making real-time video compression very expensive or impossible.

(b) **Multi-layer image decomposition.** The main appeal of this approach is that it is the shortest path to supporting compound compression with existing standards. The idea is to decompose the images in different "layers", some containing the color information, and some defining transparency (alpha channel). The most common is the MRC standard (ITU T.44),[7] which supports two color layers, called background and foreground, plus one binary layer, called mask, which signals if a pixel of the decompressed image is to be copied from the background layer or the foreground layer. This approach is extended in Part 6 of the JPEG 2000 standard, which allows for more layers and more transparency bits. For simple image parts, like those with a single-color text on a single-color background the decomposition is quite straightforward and reasonably efficient. For more complex parts it is not easy to define a decomposition that simultaneously optimizes bit rates and visual quality.[8,10,12,14] In practically all cases there is a substantial redundancy in this approach, since three images have to be encoded, with a corresponding increase in encoding and decoding speed. It is very easy to extend this multi-layer approach to video content and to video standards like MPEG, but again the problem resides in finding the best decomposition, and the techniques used for still images have to be adapted accordingly.

(c) **Image block classification.** This technique is by far the simplest. The compound image or video is divided in blocks of a certain size (like 8×8), and a classification method is applied to the block to decide which compression method is to be used to code its pixels. The classification information is also coded before coding the block pixels, and usually it depends only on the values of the pixels inside a block, and possibly the decision on the neighboring blocks. Experimental results found out that while this approach is definitely too simple for document analysis and understanding, it is quite effective for compression purposes.[5,6] The important fact is that misclassification results only in an increased number of bits, or excessive quality degradation in a block, while the latter can be always avoided with sufficiently conservative classification rules.

The image and video coding system that we propose here use block classification because of its low computational complexity—essential for real-time compound video applications—that results from the following properties.

- There is no redundancy in the representation (as in the multi-layer decomposition), so each pixel is classified and coded once;
- Two passes on block pixels, one for classification and one for coding, minimizes the bandwidth from memory to the CPU (pixels loaded to cache memory for classification can stay in the cache until coding);
- Working with rectangular blocks allows coding the classification much more efficiently than coding arbitrarily-shaped regions;

- Because the number of blocks can be much smaller then the number of pixels, more complex classification can be applied to groups of blocks;
- Since the decoder receives the data with classification, its speed is comparable to other standard image and video compression methods.

In the next section we present the implementation details of our coding method.

3. COMPONENTS OF THE NEW CODING SYSTEM

The method that we present here is similar to that reported in a previous paper.[5] Only the encoder performs classification, and includes its decisions in the compressed data stream. This minimizes the computational effort of the decoder, but can add a substantial overhead to the compressed data. For this reason, in this new version of the coding method we developed aggregation techniques to minimize this overhead, coding the classification information about preferably large frame areas. Figure 2 shows a diagram of this process. In this framework the encoder, instead of adding information about the classification of each block, first encodes some information about the number of blocks that have the same classification (group), then codes the classification, and finally codes the whole group of blocks.

The advantages of this approach are:

- Blocks of the same type are frequently neatly together, so coding the group size can be more effective than using context-based entropy coding for every block.
- Regions with low entropy (e.g., text) are more efficiently coded if the block is not too small. This is particularly important when not using arithmetic coding, and binary data need to be coded as run-lengths.
- Pixels inside an 8×8 block are tested only once, but the information about a whole block can be processed more than once, in the block neighborhood analysis, to improve the classification reliability.

In the following subsections we provide the details about the classification and grouping techniques, and in Subsection 3.4 we explain how they are combined.

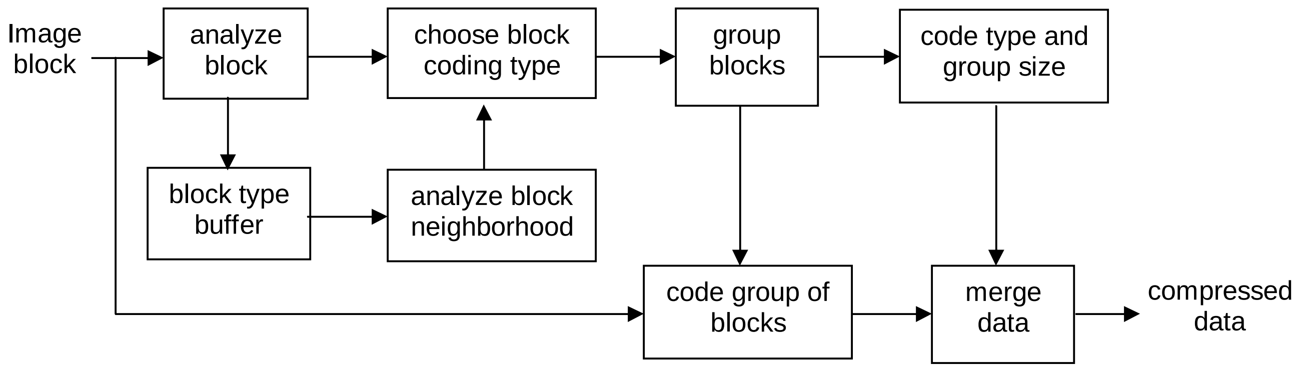


Figure 2 – Method for classification of the frame blocks using block-neighborhood analysis and aggregation of blocks with same classification in groups.

3.1. Block classes

Different classification of the blocks of pixels is defined for video and still images, with basically two different coding methods and four classes.

1. Interframe coding
   a. No change (i.e., no pixel information needs to be coded);
   b. Code 8×8 block difference using DCT coding dynamic quality settings.
2. Intraframe coding
   a. Lossless compression of text and graphics;
   b. Code 8×8 blocks using DCT with dynamic quality settings.

Note that for still-image coding the choices are restricted to intraframe coding methods 2a and 2b. Our video coding currently does not support motion compensation, because it would increase the complexity of the encoder substantially, but the changes required to support it are straightforward. What we found necessary to support is a class for blocks that do not change at all from one frame to the next. For natural video (coded at reasonably high quality) this situation is quite rare. For compound video it is quite common to have large areas do not change from one frame to the next, and even the situation in which not a single bit changes in the whole frame is not uncommon.

With this classification we have basically only two coding methods that need to be implemented. A lossy method based on the discrete cosine transform (DCT),[3] and a lossless compression method that is described in the next subsection because it is related to a particular classification technique. There are many reasons for choosing the DCT. First, the fact that it works in blocks is here a clear advantage since it matches the block classification strategy perfectly. Second, it is quite fast, with well-known optimizations, which is essential for video. Next, it is fair to say that with proper entropy coding its compression performance is quite good for high-quality applications, as normally found in the situations where compound video is required. Finally, with DCTs it is easy to dynamically change quality setting for each block, which is essential for compound video, not only for rate control, but also to avoid artifacts created by block misclassification. If there is a chance that the block to be coded with DCT contains text and graphics, then it is best to increase quality settings for that block. For instance, higher quality settings can be used on all blocks in the regions that transition from lossy to lossless compression.

We found that the mechanisms available for changing quality in standard DCT coding methods, i.e., scaling the whole quantization matrix by different factors, may not be the best choice for compound video and images. The quantization matrices optimized for natural images and video have quantization steps that grow quite fast with the order of the DCT coefficient, especially for the chrominance components.[3] On the other hand, to avoid "mosquito" and color artifacts around text and graphics it is better to use a matrix with a flatter distribution of quantization factors, and relatively more bits assigned to chrominance. A small number of matrices can be used for that purpose. In our system we use only four, which are chosen depending on an edge-detection technique (Subsection 3.3.), and are scaled depending on the quality settings.

3.2. Color-based classification

While a variety of sophisticated techniques can be used to identify text and graphics, for coding purposes we found that a remarkably simple technique, based on counting the number of different colors in a block, is simultaneously very effective in terms of classification accuracy and compression, and also very efficient in terms of speed and memory usage. Because of the noise that is inherent to natural images and video, their pixels' red-green-blue (RGB) values may be similar, but not identical. Thus, blocks that have a small number of colors (e.g., less than 16 different RGB vectors) are almost certainly composed of text and graphics. At the same time, we have the fact that regions with small number of colors normally have small entropy and can be more effectively coded with lossless methods. Since quality cannot degrade with lossless compression, for these blocks we have an unusual situation in which, in a certain way, we can improve both compression and quality simultaneously. In other words, even if the region does not contain text and graphics, it is still best to code it with a lossless method.

Thus, the first classification criterion uses in each block is counting the number of different colors. If the number is less than a threshold C_t (we used 9), then we use a lossless method to code first what are the colors in the block, and then what is the distribution (location) of those colors in the block. Fig. 3 shows an example of a block with three colors. Note that, following the scheme in Fig. 1, that information is not coded immediately when an 8×8 block with less than C_t colors is found. Commonly several blocks with the same set of colors are together in the image, so we merge blocks with the same colors if the shape of the merged block is rectangular (Subsection 3.5). Runs of pixels with the same color can be coded using methods like adaptive Rice-Golomb coding.[15]

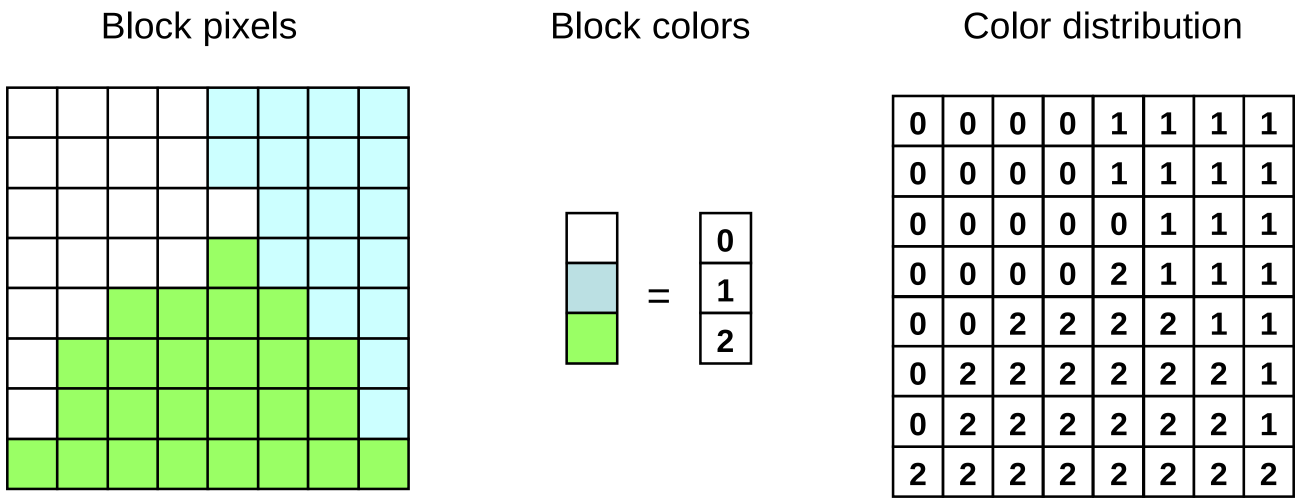


Figure 3 – Scheme for coding blocks that have a small number of different colors: first the RGB values of each color are coded followed by coding the distribution of the colors inside the block.

The classification based on counting different colors can be quite fast. When applied to natural images the average number of pixels tested is slightly larger than C_t, i.e., it quickly finds that those blocks have too many different RGB vectors. Graphics and text blocks commonly have a number of colors smaller than four. Even regions that have color gradients have few colors because, for reasonable video and image resolutions, the number of colors in a region is changing but in each block it is small.

An analysis of the complexity in the worst case, which corresponds to blocks with $C_t - 1$ different colors, demonstrates how the complexity of this classification technique can be managed. (We disregard the complexity of initializations to simplify the analysis.)

a) If the RGB colors are set in an unordered list as they are found, it is necessary to compare each new RGB vector to an average of $C_t/2$ list entries, which means an average of $C_t/2$ comparisons per pixel.

b) If the 24-bit RGB values are hashed[2] to b bins, each containing its own list of RGB vectors previously hashed to that bin, then the complexity is reduced to $C_t/2b$ comparisons per pixel.

c) Saving the RGB vectors in an ordered list, and using bisection to search for colors in the list, we need $\log_2(C_t)$ comparisons per pixel.

Since the worst-case is not very common, the best strategy depends on the type of video commonly found and on the value of C_t (which is normally small). For instance case (c) is best for larger values of C_t, but is relatively inefficient in its initial stages when the ordered list is being formed. It is also interesting to note that our lossless compression method is faster than our lossy compression (which needs to compute a color transform and DCT), and in consequence, if more time is spent testing the blocks with small number of colors, it is later saved while coding.

3.3. Identification of blocks that need higher quality

The classification technique described in the previous section was chosen because of its speed and its suitability to our chosen lossless compression method. Due to its simplicity it cannot identify all types of text and graphics. For instance, text can be rendered with its edges slightly blurred—meaning that it is not bi-level—to avoid aliasing if it is later rescaled. Smoother transitions in the edges makes the DCT coding more efficient and the artifacts less pronounced, but depending on the resolution, block content, and quality settings, we still can have visible artifacts. However, if we can identify these conditions *a priori* in a block, then we can eliminate those artifacts by increasing the compression quality settings in that block.

We have to observe that the most visible DCT artifacts on text and graphics occur around their edges, but at the same time we can have complex textures with many edges that can be highly compressed because those DCT artifacts are hard to perceive in the complex background. Thus, we need a method to detect blocks that have few edges in a simple pattern, and reject blocks with edges plus complex textures and noise. Furthermore, we care only about the presence of edges, and not their location or shape.

Our detection method is based on the fact that regions with text and graphics tend to have large changes in luminance, but in simple graphical patterns. Since these features are unrelated to color representations, in this subsection we consider only a grayscale representation of the image pixels. In other words, the classification techniques presented next should be used on the gray or Y color component (color plane).

Let us represent the luminance of each pixel in block as $p_{i,j}$. We can get information about the block activity and the presence of edges by computing the horizontal and vertical absolute differences

$$d^h_{i,j} = | p_{i,j} - p_{i,j-1} |, \qquad d^v_{i,j} = | p_{i,j} - p_{i-1,j} |, \tag{1}$$

and then analyzing the histogram of these absolute differences. Fig. 4 shows the type of distribution normally expected for each type of block content. Blocks that are part of photos or complex graphic textures tend to have histograms that peaked at zero with a somewhat smooth decay. Blocks with text and graphics normally have histograms containing a few isolated peaks.

We wan to use this property of the histograms to identify the blocks that have edges, but we also want to minimize the computational effort analyzing the histogram. The ideal is to have a single number that, when compared to an absolute threshold, indicates if the block has the right structure or not. One function that possesses some of the desired features is the entropy function.

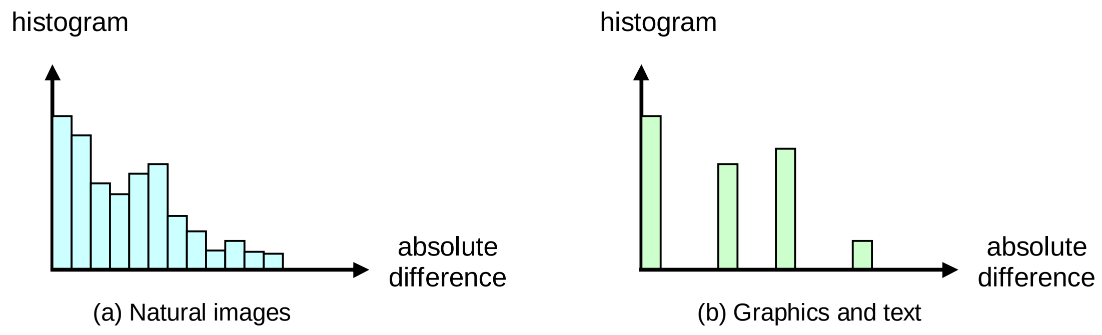


Figure 4 – Examples of typical absolute difference histogram for image or video blocks with different visual content.

Let h_n, $n = 0, \ldots, S–1$, be a set of non-negative numbers corresponding to a histogram of absolute differences (both vertical and horizontal), and define

$$T = \sum_{n=0}^{S-1} h_n \; . \tag{2}$$

The entropy function of the histogram is defined by

$$\mathrm{y}\,(\mathbf{h}) = \sum_{h_n \neq 0} \frac{h_n \log(T/h_n)}{T} = \log(T) - \frac{1}{T} \sum_{h_n \neq 0} h_n \log(h_n) \tag{3}$$

In addition, we define the maximum argument in the histogram as

$$\mathrm{m}(\mathbf{h}) = \max_{h_n \neq 0} \{n\} \tag{4}$$

An advantage of the entropy function is that its value does not depend on the exact location of the peaks on the histogram, but only on their shape. Since it is defined using logarithms and multiplication, it appears to be computationally *very* costly. Instead, it is quite the opposite. Note that in a histogram obtained from an 8×8 block we always have $0 \le h_n \le T = 128$. Thus, all the function factors in the left-hand-side of (3) can be pre-computed, scaled and rounded to integers, meaning that all demanding computations can be replaced by fast table look-up.

For our purposes a peak at value zero does have a special significance: it represents flat areas, which should be disregarded by the edge detection. We use the maximum difference to quickly know if there is only one peak at zero, or a different distribution. Table I shows the expected magnitude of these two functions in different image block types. We can see that on blocks that contain text we can expect the entropy to be low, and the maximum difference to be high. A function that can be used to identify this condition is

$$\Phi(\mathbf{h}) = \frac{a[\mathrm{y}\,(\mathbf{h}) + b]}{\mathrm{m}(\mathbf{h}) + c} \tag{5}$$

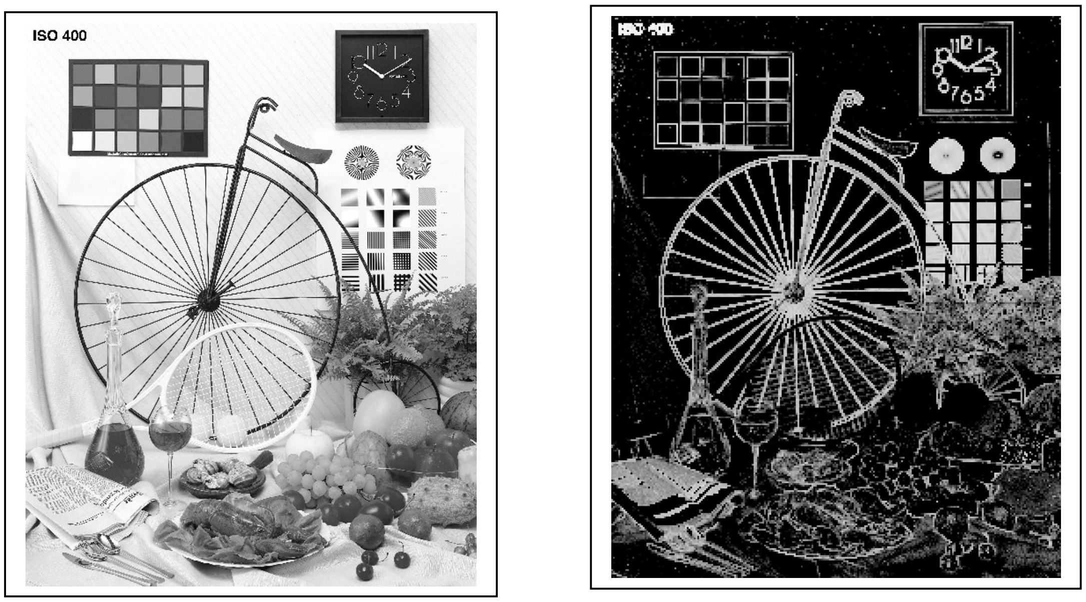
where a, b, and c are constants that normalize the entropy, and change the sensitivity of the classification. They should be chosen so that $\Phi(\mathbf{h})$ is small when the image block contains text, and $\Psi(h)$ is large in all other cases. Under those conditions, a block is classified as one that needs higher quality if $\Phi\ (h)$ is smaller than a given threshold, like $\Phi(\mathbf{h}) < 1$. Alternatively, we can map the values of $\Phi(\mathbf{h})$ to several settings, for a continuous gradation of quality.

Fig. 5b shows the result of the computation of this type of function to the 2048×2560 image ‘Bike’, with lighter areas indicating small values of $\Phi(\mathbf{h})$, i.e., the parts that may need higher quality settings. Note on the top left that the blocks around the text “ISO 400” are very clearly identified.[*] Normally we would choose a threshold such that only this area would be classified as text/graphics. However, we can also see in Fig. 5b that our classification function scales quite well, i.e., other text/graphics parts, or parts that look like graphics, like the bicycle’s spokes, or the clock numbers, are also identified, but at a different level.

Table I – Properties of the entropy and maximum difference functions.

Block features	**entropy**	**maximum difference**
Smooth block	small	small
Texture block	large	large
Graphics and text	small	large

[*] The grayscale gradations of this image may be easier to see in the electronic version of this document, but the 8×8 blocks cannot be seen individually.



(a) Original 'Bike' image

(b) Grayscale map of function Φ(**h**)

Figure 5 – Result of the computation of function Φ(**h**) to 8×8 blocks of image 'Bike': lighter areas represent blocks that need to be compressed with higher quality setting to minimize DCT artifacts.

We show in Fig. 2 that we can perform an analysis of the blocks in the neighborhood to improve classification decisions. To simplify the notation we use Φ(**h**) to identify quality settings in general, including decisions based on adjacent blocks. For instance, we can set Φ(**h**) = 0 whenever one of the adjacent blocks had been coded with the lossless method, to avoid significant jumps in quality from one block to the next.

3.4. Classification and coding

Putting together the techniques explained in the previous subsections, we present the coding algorithm we used for still images. For each 8×8 block in the image we use the following algorithm:

1. If the block has less than $C_t = 9$ different RGB colors then use lossless coding (see Subsection 3.2).
2. Otherwise compute function Φ(**h**) (see Subsection 3.3)
   a. If Φ(**h**) < threshold then code the block using DCT, with higher-quality quantization matrices;
   b. Otherwise code the block using DCT, with lower-quality quantization matrices.

The coding of the side information is done for groups of 8×8 blocks, which is explained in the next subsection. As explained in Subsection 3.1, the higher-quality DCT quantization matrices should set smaller quantization steps for all coefficients, and should also avoid large errors in the high-frequency DCT coefficients. Note that step 2 of the algorithm can be easily modified to allow for more choices of quantization matrices depending on Φ(**h**).

For video coding we have a similar algorithm. For each block in video-frame we have

1. No information to code if the block is *identical* to previous block in the same position of the previous frame.
2. If the block has less than C_t different RGB colors then use lossless coding;
3. Otherwise compute difference between block frames.
   a. If sum of absolute differences less than a threshold then compute difference of frame blocks to code;
   b. Otherwise set current block as information to code;

4. Compute function Φ(**h**) (Subsection 3.3)
    a. If Φ(**h**) < threshold then code the block using DCT, with higher-quality quantization matrices;
    b. Otherwise code the block using DCT, with lower-quality quantization matrices.

For video coding the quantization matrices and the entropy coding can be different depending if the DCT is applied to a block or a difference of blocks.

3.5. Technique for grouping blocks

Our design objective was to add versatility to the coding of all the side information about the decisions taken for each block, but at the same time keeping it simple and fast. We used the following strategy

1. Find the type of all 8×8 blocks in a 32×32 region.
2. If the blocks do not have the same type then code the type of each of the 8×8 blocks.
3. Otherwise code the number of following 32×32 regions that have all blocks with same type as the current 32×32 region.

In the algorithm above we use the term type to indicate if the block is to be coded using DCT, or, when it is to be coded with the lossless method, if the blocks have the same number of different colors. For video coding the type of block can also mean blocks that do not change from one frame to the next. With this approach the overhead of coding the type of blocks increases only on complex regions, which have to be subdivided. Simple regions, on the other hand, have large number of blocks of the same type together, and with this approach we reduce the overhead by coding the run length of 32×32 regions of the same type.

4. COMPRESSION RESULTS

Coding methods are normally compared against each other using some standard measure of the image and video quality, like mean squared error (PSNR). It is a well-known fact that PSNR is not an exact measure of subjective quality, but it is at least a good indicator when applied to natural content. Unfortunately, the PSNR is much less suited for the evaluation of the visual quality of text and graphics. In consequence it is very difficult to compare different compression methods applied to compound content, since they may use different quality factors, in different parts of the image, depending on their segmentation and classification, or their image layer decomposition.

One comparison criterion would be largest compression ratio that would produce a “visually lossless” decompressed video or image, but this condition is quite hard to define, depending on a full characterization of the viewing conditions, and its evaluation is quite expensive. However, we maintain that these difficulties should not be an obstacle for us to show that methods especially designed for compound content can be in many ways much better than those best suited for text or natural images only. Thus, here we do not try to provide exact comparative results, but we aim to show that the gains can be really substantial, and to demonstrate the advantages of our approach.

In Table II we show the compression ratios, and the encoding and decoding times of some well-known lossy and lossless compression methods, compared to our new method. The image used, ‘Toy Store’ is shown in Fig. 6a: it is a typical compound page at rendered at 600 dpi (4800×6300 = 30 Mpel), meant to test compound document compression and print quality (this image was not created by the author). Details of the other methods are as follows.

- The IJPEG implementation of the JPEG standard, with minimum acceptable text quality, default settings.
- Commercial version 8.0 of WinZip, with settings for maximum compression.
- JPEG-LS implementation by the University of British Columbia, default modes.
- The HP Labs implementation of MRC-JPEG, using arithmetic coding (see reference[12]).
- The new coding method, without using arithmetic coding.

Table II – Results of different coding methods applied to compound image 'Toy Store' (Figure 6a).

Coding method	**Compression ratio**	**Encoding/decoding time**
JPEG	24.6:1	5 s / 4 s
WinZip	21.0:1	18 s / 7 s
JPEG LS	21.5:1	4 s / 7 s
MRC-JPEG	166.3:1	16 s / 11 s
New method	170.7:1	3 s / 4 s

Fig. 6 shows how we can evaluate the visual quality by showing the error after decompression (all results correspond to the compression ratios in Table II). Fig. 6b shows how lossy methods like JPEG produce large errors around the text. Fig. 6b shows how the MRC decomposition and coding can improve both compression ratios and quality significantly. However, it still can have artifacts on the most complex parts. The new method, on the other hand, has large error only where they are least visible, i.e., in the photos. In the rest of the page it uses only lossless compression. Table II shows that its compression is also significantly better than the other lossy and lossless compression methods, and at the same time it is quite fast.

Comparisons of compound video coding with standard video coding are also very problematic. While the bandwidth required for natural video can change significantly in time, the changes for compound video can be much more dramatic. For instance, even high-resolution compound video can be compressed to only a few bytes per frame if nothing is changing. Even with motion, the bandwidth required for lossless compression of graphics animations can be many times smaller than that required for natural video.

Our coding method had been tested for remote visualization of programs for industrial computer-aided design, 3-D modeling and animation, and other demanding graphical applications on high-end graphics workstations and supercomputers. The average bandwidth measured during significant user activity, with visually lossless compression on 1280×1024 progressive video resolution, 15 frames/s, is around 8-12 Mbits/s.

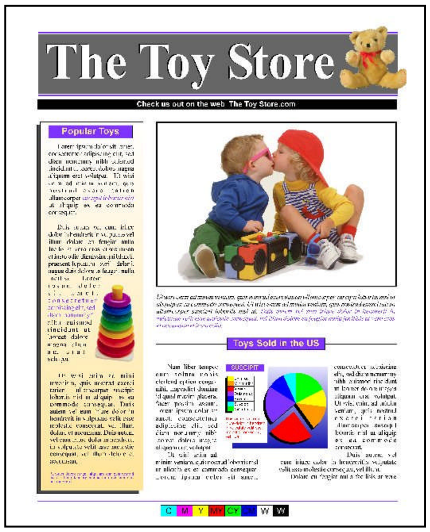
5. CONCLUSIONS

Computer-generated images and video can have a mixture of several visual components, like text, graphics, photos, animations and video. In this paper we present a new method for compression of this type of visual content, called compound images and compound video. The new method is able to efficiently identify the different components during compression, and use the appropriate coding method. Whenever possible, it uses lossless compression for graphics and text, and on natural images it uses lossy compression. It was designed for embedded systems with very limited resources, and thus has very small footprint (can be downloaded with media), and low complexity for classification, compression and decompression.

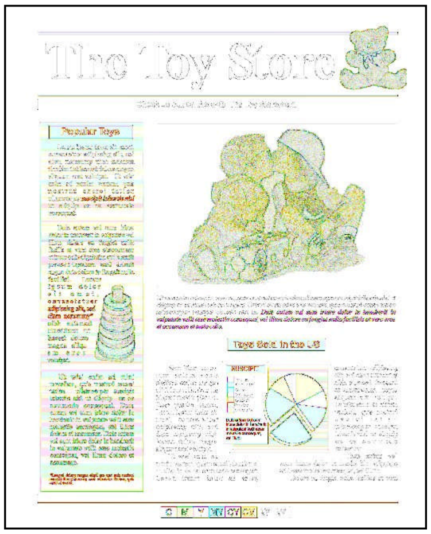
We also explain why this type of compression method can greatly reduce costs and increase usability of devices with embedded systems, since they enable them to simply decompress rich graphical contents and GUIs that has been rendered elsewhere. Other compression methods (e.g., JPEG/MPEG) can do the same, but are very inefficient for compound content. High-level graphics languages can be more efficient, but are much less reliable (e.g., supporting Asian fonts), and can be orders of magnitude more complex.

Our system is based on a fast classification of 8×8 blocks, developed to be combined with the coding. One important feature of the new method is the use of color for fast classification. All regions that have colors fitting in a small set ("palette regions") can be efficiently compressed with lossless compression, simultaneously improving compression and maintaining perfect quality. All regions that cannot fit in this reduced-color representation are compressed with a lossy method based on DCT, with dynamically varying quality for both rate control, and to avoid visual quality degradation if blocks are misclassified. We also developed aggregation techniques to minimize this amount of classification side-information, coding the block-type information of preferably large frame areas.

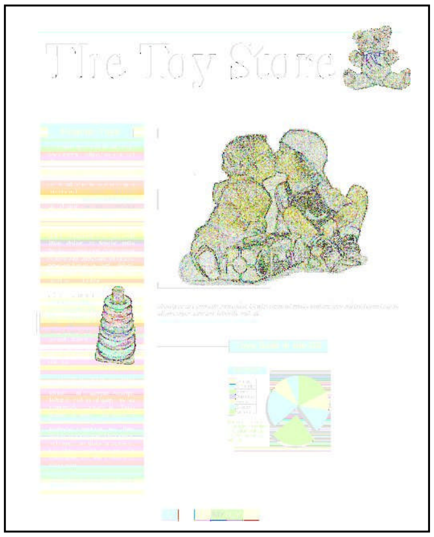
In the results section we compare the compression and decompression speed, and the image quality of the new method with other lossless and lossy coding methods. We demonstrate how lossy coding methods like JPEG produce large errors around text, while the new method allows large errors only where they are least visible. We show that it yields compression ratios and quality that are significantly better than that achieved coding methods designed for natural video and images, with similar or faster compression and decompression.



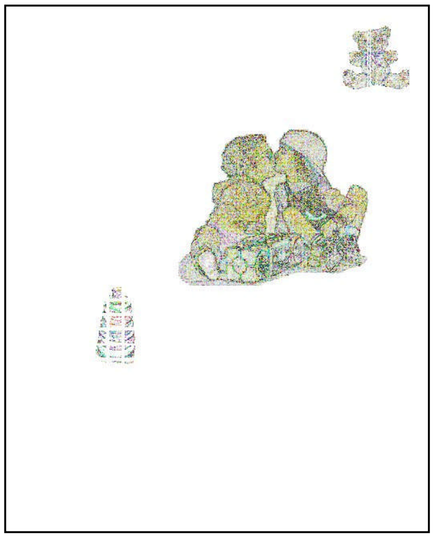
(a)



(b)



(c)



(d)

Figure 6 – Images representing the magnitude of the error after decompression (scaled $\times$ 30, white = zero): (a) original compound image (600 dpi page); (b) after JPEG compression; (c) after MRC-JPEG compression; (d) after compression with proposed coding method.

REFERENCES

1. J.D. Foley, A. van Dam, S.K. Feiner, and J.F. Hughes, *Computer Graphics, Principles and Practices*, Addison-Wesley Pub. Co., Reading, MA, 1990.
2. T.H. Cormen, C.E. Leiserson, and R.L. Rivest, *Introduction to Algorithms*, MIT Press, Cambridge, MA, 1990.
3. R.C. Gonzalez and R.E. Woods, *Digital Image Processing*, Addison-Wesley Pub. Co., Reading, MA, 1992.
4. J.M. Gilbert and R.W. Brodersen, “A lossless 2-D image compression technique for synthetic discrete-tone images,” *Proc. Data Compression Conf.,* pp. 359–368, March 1998.
5. A. Said and A.I. Drukarev, “Simplified segmentation for compound image compression,” *Proc. IEEE Int. Conf. Image Processing,* vol. 1, pp. 24–28, Oct. 1999.
6. H. Cheng and C.A. Bouman, “Multilayer document compression algorithm,” *Proc. IEEE Int. Conf. Image Processing,* vol. 1, pp. 244–248, Oct. 1999.
7. R.L. de Queiroz, “Compression of Compound Documents,” *Proc. IEEE Int. Conf. Image Processing,* vol. 1, pp. 24–28, Oct. 1999.
8. P. Haffner, Y. LeCun, L. Bottou, P. Howard, P. Vincent, and B. Riemers, “Color documents on the Web with DjVu,” *Proc. IEEE Int. Conf. Image Processing,* Oct. 1999.
9. D.A. Tompkins and F. Kossentini, “A fast segmentation algorithm for bi-level image compression using JBIG2,” *Proc. IEEE Int. Conf. Image Processing,* Oct. 1999.
10. R.L. de Queiroz, Z. Fan, and T.D. Tran, “ Optimizing block-threshold segmentation for MRC compression,” *Proc. IEEE Int. Conf. Image Processing,* vol. 2, pp. 597–600, Sept. 2000.
11. H. Cheng and C.A. Bouman, “Multiscale Bayesian segmentation using a trainable context model” *IEEE Trans. Image Processing,* vol. 10, pp. 511–525, April 2001.
12. D. Mukherjee, N. Memon, and A. Said, “JPEG-matched MRC compression of compound documents,” *Proc. IEEE Int. Conf. Image Processing,* vol. 3, pp. 434–437, Oct. 2001.
13. D. Mukherjee, C. Chrysafis, and A. Said, “Low complexity guaranteed fit compound document compression,” *Proc. IEEE Int. Conf. Image Processing,* vol. 1, pp.225–228, Sept. 2002.
14. D. Mukherjee, C. Chrysafis, and A. Said, “JPEG2000-matched MRC compression of compound documents,” *Proc. IEEE Int. Conf. Image Processing,* vol. 3, pp. 73–76, Sept. 2002.
15. M.J. Weinberger, G. Seroussi, and G. Sapiro, “The LOCO-I lossless image compression algorithm: principles and standardization into JPEG-LS,” IEEE Trans. Image Proc., vol. 9, pp. 1309–1324, Aug. 2000.