The Piecewise-Constant Image Model

Paul J. Ausbeck Jr., *Member IEEE*

***Abstract*—The piecewise-constant image model (PWC) is a new technique for lossless compression of palette images. PWC is a blend of traditional scanline oriented and newer object based methods. Remarkably, PWC delivers the best known compression across a wide variety palette image types while delivering translation speeds comparable to highly tuned one dimensional methods. This paper introduces the topic of palette image coding and traces the development of the piecewise-constant model from a completely object oriented code requiring two image passes to a high performance scanline oriented code.**

***Index Terms*—Lossless image compression, palette image compression, PWC.**

I. INTRODUCTION

A palette image is composed of two components: color information contained in a lookup table or *palette*, and image information composed of a series of palette indices. Palette images are ubiquitous in modern computer systems. The user interface elements of most windowing operating systems are composed of palette images. Black and white documents are a simple form of palette image. Almost every page on the Worldwide Web contains one or more palette images. Figure 1 is a grayscale version of the palette image serving as the banner of the Web site http://www.yahoo.com in October of 1997.

Figure 1
A Typical Palette Image

In spite of their widespread use, a good model for palette images has yet to be devised. Palette images generally contain too few colors to make effective use of linear predictive models such as used in JPEG–LS[1] and contain too many colors to avoid the sparse context problem that arises when using neighborhood color models such as those of JBIG[2]. Table 1 shows the results of applying various coding methods to the grayscale image of Figure 1. The first method is to individually code the image bitplanes with JBIG, the second method is JPEG-LS predictive coding, and the last method is dictionary coding via GIF. Perhaps surprisingly, the one dimensional model used in GIF performs better than either of the alternate two dimensional models. For palette images that have not been converted to grayscale, the superior performance of dictionary coding methods continues to hold true.

Uncoded	Bit Planes	Predictive	GIF
27,182	9,266	8,825	6,923

Table 1
Motivational Coding Example

The author is with Caravian Software Designs, 74 Carlyn Ave, Campbell, CA 95008. He can be reached via electronic mail at: paula@alumni.cse.ucsc.edu.

II. BACKGROUND

The CompuServe Graphics Interchange Format (GIF)[1] is the most commonly used file format for the distribution of palette images. The compression algorithm used for GIF image data is the LZC[3] improvement to the LZW[4] form of the LZ78[5] universal dictionary compressor. LZC is particularly straightforward to implement and is used in both GIF and the UNIX "compress" utility.

Patent claims against the LZ78 class of algorithms were the impetus for development of the relatively new Portable Network Graphic (PNG) format. PNG compresses image data using the DEFLATE[6] algorithm, a combination of LZ77[7] dictionary compression and Huffman[8] coding. This combination is a particularly effective adaptation strategy for palette images, yielding compression significantly better than LZC. Also introduced in PNG is the option of using a predictive "filter" as a preprocessor to DEFLATE. While filters improve compression significantly on natural images, they unfortunately degrade compression of palette images.

The LZC algorithm is fairly symmetric, requiring slightly more computation on encode than decode. DEFLATE is more flexible, providing options for trading off compression and speed. Interestingly, at comparable speeds both methods produce roughly equivalent compression. An extended discussion of the relative merits of the various forms of dictionary compression can be found in Bell et al[9].

Since existing palette image standards are based upon universal one-dimensional compression algorithms, there should be significant room for improvement. One avenue of exploration is an improved universal algorithm. The Burrows-Wheeler Transformation[10] (BWT) has gained recent attention because it provides compression comparable to sophisticated context models at speeds closer to LZ methods. The best known BWT method is the BZIP implementation of the method of Fenwick[11]. BZIP performs a block sort transformation, followed by move to front (MTF) coding, followed by zero order arithmetic entropy coding. Interestingly, while MTF coding is useful for text, it may be less so for images. Arnavut[12] has recently shown that removing MTF from BZIP improves its compression of palette images.

One drawback of the BWT method is that optimal compression is attained by sorting large blocks of data. This both requires large amounts of memory and precludes standard image domain techniques such as streaming or progressive transmission. Secondly, it should be possible to achieve better compression through the use of a proper two-dimensional image model. Surprisingly, this second goal has been remarkably difficult to achieve.

The two dominant lossless image coding techniques are linear predictive coding as used in JPEG-LS and neighborhood context modeling as used in JBIG. Predictive coding works well on natural images where spatially adjacent pixels tend to have similar values. Context modeling works well on black/white images where the limited number of colors allows the use of a reasonably sized neighborhood model. Since palette images lack both of these properties, neither method is suitable.

[1] June 1990, copyright by Compuserve Inc., available at various locations on the internet.

One possible avenue for matching predictive codes to palette images is to reorder the palette to increase the correlation of spatial and value adjacency. Memon[13] has shown that palette reordering significantly improves CALIC[14] compression of palette images. Arithmetic CALIC is appropriate for use in this role since it performs significantly better than JPEG-LS on the uniform pixel runs that typically appear in palette images.

A standard technique used to apply JBIG style neighborhood context models to grayscale images is to separately code each bitplane as a black/white image. Because there may be significant correlation between the planes, improved compression can be achieved by using pixels from previous (inter) planes in the context model for subsequent planes. The Embedded Image-Domain Adaptive Compressor (EIDAC)[15] uses this approach to produce an embedded description of "simple" grayscale images. The original version of EIDAC uses a single pixel from each available inter bitplane in the coding context for the current (intra) plane. A second version[16] using multiple pixels from the immediately preceding inter plane shows improved results.

Neither palette ordering combined with predictive coding nor bitplane coding compress as well as the better universal methods and a more customized approach is clearly in order. One such approach is Runs of Adaptive Pixel Patterns[17] (RAPP). The basic structure of RAPP is similar to a predictive coder with the prediction being formed from the closest four causal neighbors. In RAPP the prediction is always one of the neighboring values. A neighborhood map coloring is used to form 15 contexts for conditioning the prediction decision. Failed predictions fall into a decision scheme where remaining unpredicted neighboring values are considered. Finally, anomalous values are encoded. Neighborhood decisions are made arithmetically, anomalous information is encoded using DEFLATE.

RAPP has been combined with EIDAC-like methods in a content-progressive representation of street maps[18]. Excellent compression is achieved by making use of layered composition information provided by map publishers. Instead of coding each bitplane of a composite map image, each color of the map composition is coded separately using inter and intra pixels in the coding context. Since typical maps contain upwards of 15 colors, this approach is computationally expensive. An acceleration scheme codes only the most important map layers such as text and street outlines individually. The residue formed by subtracting the initial bitplanes from the final image is then coded using a variation of RAPP.

As a replacement for current palette image coding practice, none of the previously discussed methods has the appropriate properties. Though it compresses better than LZ methods, the BWT approach is computationally more expensive, requires much more memory and precludes streaming or progressive presentation. Even with palette ordering, standard predictive models are really mismatched to the material. Bitplane techniques are computationally expensive and require side information such as composition information to achieve the best compression. RAPP is not yet a complete method, requiring both context conditioned arithmetic coding and dictionary compression. The subject of this paper is a new palette image compression method that is computationally efficient, has a scanline proportional memory footprint, and provides the best known compression of a wide variety of palette image material.

III. THE PIECEWISE CONSTANT MODEL

Whether synthetically produced or derived from continuous tone pictures, palette images are distinguished by three characteristics:

- They tend to contain far fewer colors than pixels.
- Pixels of the same color tend to be contiguous.
- The color of a pixel is statistically related to surrounding colors.

The original Piecewise-Constant Image Model (PWC)[19] captures these characteristics with a two pass object-based model. In a first image pass, boundaries between constant color pieces or *domains* are established. A second pass then determines domain colors. Remarkably, this object-based approach can also be accomplished within a framework that differs little from a standard scanline oriented image code. Further, performance can be comparable to commercially mature one-dimensional methods. The remainder of this paper traces PWC's evolution in a way that is hopefully insightful.

IV. OBJECT BASED CODING

An important objective in designing an object based model is to assure that boundary and color information can be coded under a common framework. The framework used by PWC is that of a multiple context binary arithmetic coder. This framework was selected for two reasons. First, composing the model from binary decisions minimizes granularity and maximizes opportunity for compression. Second, arithmetic coding can take the hard edges off of a model, allowing it to be used effectively on a wider variety of source material.

A. The PWC Language

The PWC coding language is composed of the four decisions shown in Table 2. **D1** decisions are used to establish the boundaries between constant color domains. Decisions **D2**-**D4** are used to establish domain color.

D1	Is the current pixel's color identical to that of a specified rectilinearly connected neighbor?
D2	Is the current pixel's color identical to that of a specified diagonally–connected neighbor?
D3	Is the current pixel's color identical to a guessed value?
D4	What is the current pixel's color?

Table 2
Piecewise–Constant Language

D1-**D3** are naturally binary. To maintain compatibility with PWC's coding framework, **D4** is accomplished through a composition of binary decisions. The following discussion describes how the PWC language is used in an object-based model.

B. Boundary Coding

One way of viewing the constant color domains of a palette image is as countries on a geographical map. In fact, one possible way to code boundary information would be to recolor domains using just enough colors to maintain different colors for adjacent domains. Some images, such as black/white documents, require only two colors. However, it has long been known that at least four colors are necessary to color an arbitrary map[2][20]. Figure 2 is an example of a very simple map that cannot be colored with fewer than four colors.

[2] Recently. (at least relatively) it has been proven that four colors are sufficient.

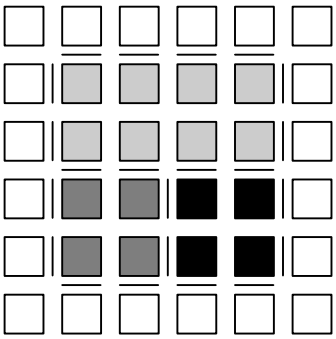

Figure 2
The Simplest Four Color Map

Minimal map colorings can be difficult to obtain and as such are not very useful for coding[21]. However, the *chromaticity* of a map does give a proportional indication of how much information is necessary to code it. A two-color map requires only one decision at each pixel location: is the pixel black or white? A four color map requires two decisions to choose among the four possible colors. A desirable goal is to find a single representation that can efficiently represent both types of map.

The edge map, introduced by Tilton[22], represents boundary information via the introduction of imaginary edges between pixels. Each pixel is assigned one vertical and one horizontal edge in a *separator lattice*. In the edge map representation, binary decisions can be naturally used to determine whether or not a particular lattice site is *full*. The toy image of Figure 2 has twenty two full lattice sites. The remainder are *empty*.

A remarkable property of edge maps is that connectivity constraints prevent them from being arbitrarily populated. When fully exploited, the connectivity property allows a boundary coder to adapt to local chromaticity.

1. Connectivity Constraints

PWC populates its edge map boundary model in raster order. At each pixel location, **L**, the state of vertical separator site is determined first, followed by the horizontal site. Population decisions are made using **D1** decisions from the PWC language. On Figure 3 the two rectilinear separator decisions are labeled **D1v** and **D1h** respectively.

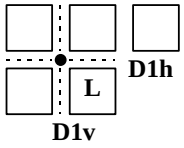

Figure 3
Connectivity Constraints

Due to connectivity constraints, **D1h** can often be made deterministically. For example if none of the three causal edges touching the left end of separator site **D1h** is full, then **D1h** is deterministically empty. If only one of the causal edges is full, then **D1h** is deterministically full.

Using the idea of deterministic decisions, an upper bound on the maximum rectilinear decision entropy can be developed. Given a zero order probability, p, that a separator lattice site is full, the probability that a horizontal separator is deterministically determined is:

$$D(p) = (1-p)^3 + 3p(1-p)^2 . \quad (1)$$

$D(p)$ is plotted in Figure 4.

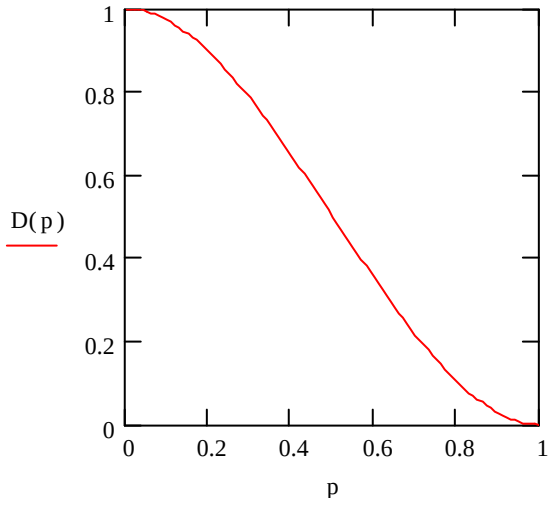

Figure 4
Deterministic Decision Probability

Also, given identical p, the vertical decision entropy is:

$$H_1(p) = -p\log_2(p) - (1-p)\log_2(1-p) . \quad (2)$$

The horizontal entropy adjusted for determinism is:

$$H_2(p) = (1-D(p))H_v(p) , \quad (3)$$

and the total edge entropy is:

$$H_T(p) = H_1(p) + H_2(p) . \quad (4)$$

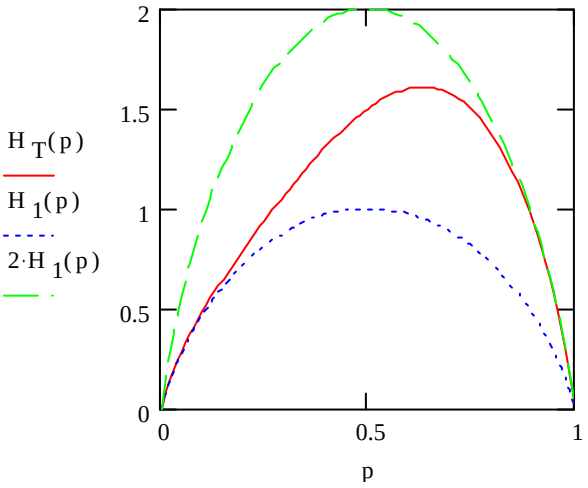

Figure 5
Total Edge Map Decision Entropy

The maximum of H_T is 1.607 and it occurs at a full edge probability of 0.632. H_T is plotted in Figure 5. Note how H_T approaches the single decision entropy, H_1, at low p and approaches $2 \cdot H_1$ at high p. On sparse images only one **D1** decision need typically be made at each pixel location.

A completely two color map has the additional connectivity property that each separator lattice intersection can have only 0, 2, or 4 adjoining edges. $D(p)$ therefore further simplifies:

$$D(p) = (1-p)^3 + 3p(1-p)^2 + 3p^2(1-p) + p^3 = 1 \quad (5)$$

giving the expected result of complete determinism for **D1h**.

2. Edge Decision Context Models

For conditioning separator site population decisions, PWC uses the edge model proposed by Tate[23] and shown in Figure 6. Tate used a ternary alphabet in his work but as described previously PWC uses two binary **D1** decisions. Since the model for **D1v** has eight elements and the model for **D1h** has nine, the total number of contexts for conditioning **D1** decisions is 768. Connectivity constraints reduce the number of active contexts to 512.

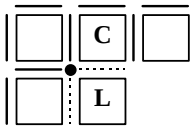

Figure 6
Edge Context Model

C. Color Coding

For typical palette images, neighboring pixels do not have a linear predictive relationship. Further, the sparse context problem makes it prohibitive to keep track of complete neighborhood color statistics. PWC avoids these problems and takes advantage of the strengths of both methods through a novel multi-stage color determination process.

When establishing the color of a domain, PWC first tries to establish diagonal connectivity. Failing that, a more general process called color guessing is attempted. Finally, when color guessing fails, the color is established via predictive coding. A predictive model is used for the final stage because some color reduced natural images, especially those of predominantly one tint, can exhibit significant correlation in spatial and value adjacency.

1. Diagonal Connectivity

D2 decisions are used to establish diagonal connectivity in PWC. Diagonal connectivity is only defined at lattice intersections where there is no rectilinear connectivity. Figure 7 shows the two causal orientations of diagonal connectivity at a lattice intersection, **L**, that has four impinging boundary segments. Each potential diagonal connection requires one **D2** decision. The number of diagonal connections considered by the model drops quite rapidly as the edge density decreases and is typically less than 0.5 decisions per pixel.

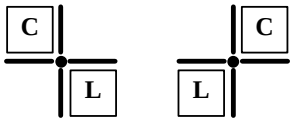

Figure 7
Diagonal Contexts

Domain color is usually more important than domain shape in conditioning **D2** decisions. For this reason **D2** decisions are only made once the color to be propagated across an diagonal connection is known. Connection orientation is also an important conditioning criteria in many images. Using both orientation and color in a context model for **D2** decisions requires two model parameters for every color used by an image. On Figure 7 the two orientations are represented by the left and right glyphs and the propagating color is labeled **C.**

2. Color Guessing

Given a causal color context, typically only one or a few following colors predominate. This leads to the idea of using previously determined colors in the same context as *guesses*. To maintain one guess for each context of a 256 color first order model, requires only 256 model parameters.

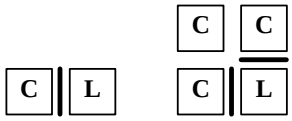

Figure 8
Guess Contexts

Color guessing, PWC language element **D3,** is designed to model the neighboring color relationships in an image while using a controlled number of model parameters. A guess is simply some color that has occurred previously in the coding process. The size of a guess model is proportional to D^S where D is the palette depth of the image and S is the number of neighboring colors used in the model. To maintain a reasonably sized model, the number of neighboring domain colors used to condition **D3** must be limited. For 256 color images it is usually only profitable to include one neighboring color in the coding context. The left glyph of Figure 8 shows a known pixel, **C**, used as a guess context for the unknown pixel, **L**, to its east. The right glyph of the figure shows three neighboring colors used as a guess context. The three color configuration is normally only useful for palette images of depth four (sixteen possible colors).

The exact size of a guess model is determined by the number of guesses for which statistics are maintained. One possibility is to maintain statistics for every possible color occurring in each context. The size of this straightforward guess model is D^{S+1}, no different from a complete neighborhood color model. With such a large model, many guesses are not very useful in determining color. Compression suffers because of the large number of mostly useless parameters to be learned. Coding speed suffers because a large number of largely irrelevant decisions are made.

One way to solve both of these problems is to limit the number of guesses maintained simultaneously by the model to some fixed number. When limiting guesses, a mechanism is needed to maintain only *good* guesses: guesses that are mostly correct. One way to achieve this is through *guess competition* within a *guess pool*.

The competitive mechanism used by PWC a least recently used (LRU) chain. In this application a context is moved to the front of the LRU any time its associated guess is correct. When a new guess is added to the pool and the pool has reached its maximum size, the guess at the end of the LRU chain is sacrificed. Figure 9 shows the guesses of a guess pool chained from lists determined by a context identifier. The average number of guesses per context is the size of the guess pool divided by the number of guess contexts. It should be emphasized that the guess pool is global. Competition occurs both between guesses in the same context and guesses in other contexts.

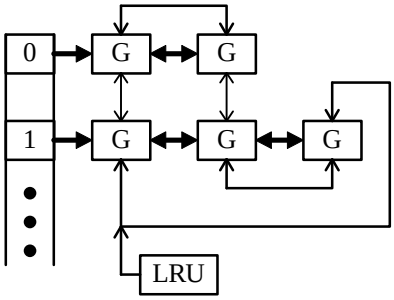

Figure 9
Guess Pool Structure

The complete guess pool operation is as follows:

- When **D4** decision is made, the result is added to the head of the guess pool LRU and to the tail of the appropriate decision context's guess chain.
- When a **D3** decision is correct, its associated context is moved to the head of its guess chain and to the head of the guess pool LRU.
- When the guess pool is full and a guess must be sacrificed, it is removed from both its associated guess chain and the end of the LRU.
- The statistics of a **D3** context are only updated when they are actually used to make a decision. No attempt is made to keep accurate conditional probabilities.
- The statistics of a sacrificed **D3** context are decayed and kept as a prior for its new role.
- Guesses that are known to be impossible from prior **D1** and **D2** decisions are ignored.

3. Guess Failures

When diagonal connectivity and color guessing both fail, PWC makes **D4** decisions to introduce innovative colors to its model. Depending upon the number of colors in the image, two different procedures are used for **D4**. When the number of colors is sixteen or fewer, **D4** is made using zero order color statistics. Because all decisions must fit into PWC's binary arithmetic coding framework, color statistics are kept in a binary tree and each bit of the coded symbol's binary description is coded separately.

When the number of image colors is greater than sixteen, **D4** decisions are made via predictive coding. The predictor used is the hybrid planar/edge predictor of JPEG-LS[1]. Prediction residuals are coded using Don Speck's Activity Level Classification Model[24]. ALCM uses Rice[25] mapping followed by Golomb[26] coding. The Golomb parameter is selected using the logarithm of the absolute maximum difference in the causal neighborhood. Again in keeping with PWC's coding framework, each Rice-Golomb bit is coded individually.

D. Object PWC

The PWC language is used in an object-bases coder in two image passes. In the first pass, boundaries between constant color rectilinear domains are established via one or two **D1** decisions at each pixel location. In a second pass, domain colors are established through a sequence of decisions **D2-D4**. The operation is summarized as follows:

- Scan the image in raster order and at each pixel location:
  - Populate the vertical separator site (**D1**).
  - Populate the horizontal separator site using connectivity constraints or failing that by making a **D1** decision.
- Mark all pixel as uncolored.
- Scan the image in raster order and color each uncolored domain as it is encountered:
  - Determine if the topmost domain stripe can be diagonally connected to an already colored pixel by making zero, one or two **D2** decisions.
  - Failing that, using a sequence of **D3** decisions to determine if the unknown color is in the guess pool.
  - Failing that, establish the domain color using **D4**.
  - Flood the uncolored domain with the established color.

V.SCANLINE ORIENTED CODING

The flooding process of object PWC is the only aspect of the algorithm that is not raster local. Even though flooding always commences from the "highest" pixel in a domain, it is not a completely top down process. For domains that are concave on their upper periphery, the flooding process must descend into the body of the domain and then re-ascend into upper extremities. For example, the "H" on Figure 10 is first encountered at the top of its left vertical segment. In order to reach the top of the right vertical segment, the flooding process must re-ascend from the connecting horizontal segment.

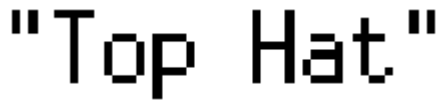

Figure 10
Excerpt from JPEG-LS Compound Document #1

To convert object PWC to a single pass algorithm, the flooding process must be altered to be entirely top down. One fortunate characteristic of palette images is that upper concave domains are relatively rare in practice. Further, when they do occur they often have significant extent, so the cost of color acquisition relative to shape description is relatively low. One exception is small-font two-color text like that of Figure 10. However, in this context the cost of color acquisition itself is relatively low. Taken together these considerations lead to the idea of top down flooding augmented with limited color reacquisition.

A. Streaming PWC

Instead of making two complete image passes as in object PWC, the basic strategy of streaming PWC[27] is to make two passes through each image scanline. The first pass makes **D1** decisions to establish rectilinear connectivity within the scanline and to the previous scanline. The second pass determines the color of each domain *stripe* by either *propagating* the color from the previous scanline or failing that by making decisions **D2-D4.**

The requirements for the color determination pass are somewhat subtle. The key point is to avoid making unnecessary color decisions. For example, once the color for the roof of the "**T**" in Figure 10 is established on the first scanline the color can be propagated down the trunk without making further color decisions. Similarly, though the color of the "**H**" must be determined twice on the first scanline, color propagation is possible on subsequent scanlines. Perhaps not as obvious is that color propagation is also possible for the cross of the "**t**".

Color determination is accomplished via two passes over each domain stripe. On the first pass, the boundary model built by the connectivity pass is consulted to determine whether or not a color can be propagated from the previous scanline. Propagation is possible if at least one pixel on the previous scanline is not

separated from the current scanline stripe by horizontal separators in the boundary model.

If color propagation is not possible, the stripe color is determined via decisions **D2**-**D4**. The stripe is filled in a second pass. The streaming algorithm is summarized thusly:

- For each image scanline
  - Make a first pass and make **D1** decisions to determine rectilinear connectivity.
  - On a second pass, determine the color of each domain stripe:
    - Make a first pass to determine the possibility of color propagation.
    - If color propagation is not possible, determine the stripe color via a **D2**-**D4** sequence.
    - Make a second pass to update the color model.

VI. SPARSE IMAGE METHODS

Though significantly more efficient that object PWC, streaming PWC suffers from a problem common to all neighborhood context models: a disproportionate amount of computational effort is spent encoding uniform image areas. Because large uniform areas are quite common in palette images, the average compression speed of streaming PWC is significantly slower than competing one dimensional methods.

To ameliorate this problem a recent version of PWC uses a new method called the skip-innovation model[28] to efficiently blend two-dimensional modeling and run-length codes. The basic idea is to skip over uniform areas entirely if possible, and to partially skip them if not.

A. The Skip-Innovation Model

Sparse images often consist of a relatively large number of *features* embedded in a smaller number of relatively uniform *seas*. When a two-dimensional model is used to code such images, acquisition of image features largely takes place in uniform contexts and coding of previously acquired features largely takes place in non-uniform contexts.

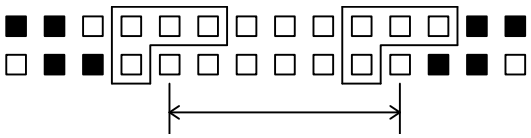

Figure 11
A Skip

To take advantage of this characteristic, a new decision is introduced into the PWC coding model. When a uniform context is encountered, its length is determined and a decision is coded as to whether or not it can be skipped entirely. Runs that cannot be skipped are coded in the normal unary fashion. For example, the two black features on Figure 11 are separated by a run of seven uniformly white coding contexts. Using the skip model this run is coded as a single affirmative skip decision.

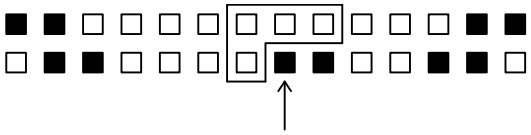

Figure 12
An Innovation

Skip failures are the result of new features needing introduction to the model. The position of these new features, or *innovations*, is the information that must be conveyed by the coder. The example of Figure 12 shows an innovative feature located four pixels into a contiguous context of length seven. In the skip-innovation model, innovative locations are encoded with special binary codes.

B. Skip-Innovation Codes

Skip innovation codes are related to Golomb codes. The skip decision, *S*, can be viewed as the magnitude or unary portion of the code, the innovation, *I*, as the binary portion. The length of the unary portion of the code is always one. The basic length of the binary portion is the ceiling of the base two logarithm of *S*. As with Golomb codes, the basic length of *I* can be significantly reduced if *S* is not a power of two. The procedure used for constructing skip-innovation codes is as follows:

- Count the number, *S*, of uniform contexts that occur before the next occurrence of a non-uniform context.
- Counting no more than *S*, count the number of pixels, *I*, to be coded whose value is identical to that populating the coding context.
- If $I = S$, encode a one, otherwise encode a zero and:
- Determine $D = \lceil \log_2(S) \rceil$, the number of binary digits required for a maximal *I*.
- Form a *D* digit binary representation of *I*.
- For each digit of the binary representation of *I* starting from the most significant:
  - Determine the minimum value, *T*, that would result if that digit took on a value of one and previous digits took on their previously encoded values.
  - If $T < S$ encode the digit.

SI codes for values of *S* and *I* up to seven are shown in Table 3.

I	*S*=1	*S*=2	*S*=3	*S*=4	*S*=5	*S*=6	*S*=7
0	0	00	000	000	0000	0000	0000
1	1	01	001	001	0001	0001	0001
2		1	01	010	0010	0010	0010
3			1	011	0011	0011	0011
4				1	01	010	0100
5					1	011	0101
6						1	011
7							1

Table 3
Skip-Innovation Codes for *S* = 1–7

For *m* not equal to a power of two, Golomb assigned the shorter binary sequences to shorter run lengths. Perhaps somewhat counterintuitively, the shorter skip-interval codes are assigned to longer runs. The first reason for this is obvious. The most frequent value for *I* is *S*, representing the lack of innovation. The second reason is more subtle.

Certain irregular or low slope features may be untrackable by a reasonably sized context model. The smaller the context model, the more likely a feature is to fall into this category of *pseudo-innovations*. For example, on Figure 13 the pixel labeled with a arrow is a pseudo-innovation. It is connected to a larger feature already known by the model but the model is too small to make a local determination. In this case, *S* is six and *I* is five resulting in the *SI* code of 011, one bit shorter than the basic code length of $D+1$.

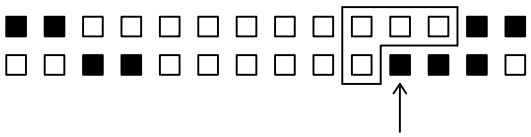

Figure 13
Pseudo-Innovation

Note that the *SI* code generation mechanism is extremely simple and efficient. In fact, it was originally chosen just for these properties. Only after several failed attempts at improvement was it recognized that skewing the code distribution towards *S* was a natural way to take advantage of the increased likelihood of pseudo-innovations appearing near *S*.

C. SI Context Models

Since skip-interval codes are part the image model, adaptive arithmetic coding can be used to further match them to the source material. A convenient context for coding *S* is *D*, previously calculated to determine the maximum possible number of binary code digits. To the extent that skips of various sizes are not uniformly distributed, using *D* as a context model for *S* can reduce the overall code string length.

Often, multiple innovations are located in a failed skip. Due to the structure of the *SI* codes, *I* will contain multiple leading zeros when the distance between innovations is substantially less than the skip length. The following context model can be used to capture this structure:

- Allocate one context for each bit of the maximum possible skip length.
- Designate one additional context the lumped context.
- Initialize a variable, ONE_SEEN to zero.
- From the most significant bit position of *I* to the lowest:
  - Code the bit under its positional context if ONE_SEEN is zero and under the lumped context otherwise.
  - If the coded bit is a one, set ONE_SEEN to one.

On black and white documents, the average black run often differs substantially from the average white run. Therefore it is useful to double number of contexts used for coding the *SI* bits.

D. Mixing SI *Codes and Unary Codes*

SI codes are designed for use as an alphabet extension mechanism in line oriented image codes. Such extensions allow a coder to switch between normal pixel at a time, or *unary*, coding and run length coding. The *SI* mechanism differs from conventional one-dimension run length coding in that it is inherently embedded in a two-dimensional coding process. *SI* never codes information in more than one context and therefore does not lose any of the benefit of the two-dimensional model in use.

The two-dimensional nature of *SI* creates one subtlety that may not be immediately apparent. A typical one-dimensional run length code, in addition to encoding a run of identical pixels, also imparts some information about the pixel immediately following a coded run. The additional information imparted is that the following pixel is different from the coded run of pixels.

The *SI* mechanism is slightly different in that it only imparts information about the following pixel when a skip failure has occurred. The pixel immediately after a successful skip may or may not be of the same color as the just skipped run. The follow procedure shows how a decoder intermixes *SI* and unary codes on a single image scanline:

- For each pixel location on the scanline
  - If the current pixel's coding context is non-uniform
    - Decode the pixel in the conventional unary manner.
    - Advance the current pixel pointer one location.
  - Otherwise
    - Decode a skip-innovation code
    - If $I = S$, skip forward *S* pixel locations filling skipped pixels with the current color.
    - Otherwise:
      - Skip forward *I* locations filling skipped pixels with the current color.
      - Fill the current pixel using the information that it is different from the current color.
      - Advance the current pixel pointer by one.

VII. ARITHMETIC CODING

The binary arithmetic coder used in PWC is the carry-free coder[29] written by Don Speck. The carry-free coder goes back to the roots of arithmetic coding to avoid IP issues associated with more modern techniques. Statistics are kept as counts and the coding interval is split with a multiply/divide operation. For all decisions other than **D4**, PWC augments the basic coder with 4-5-6-7 adaptation where statistics are halved whenever the least probable symbol count reaches eight. For **D4**, statistics are only halved when the maximum count value is attained.

Because the skip-innovation model eliminates the need for the arithmetic coder to accommodate large symbol skews, the maximum count value used in PWC is 255. This allows for a $16 \times 8 \div 8 = 16$ bit multiply/divide operation which can be performed relatively quickly on modern CPU's. As a further refinement, the most recent version of PWC approximates the arithmetic multiply/divide operation with a table lookup and multiply. Since the precision of the counts is only eight bits, the lookup table requires 64KB of memory.

VIII.THE PWC CODEC

The PWC codec uses four different models depending upon the characteristics of the source material. The first model is tailored for two color images, the second for images up to 16 colors, the third for color images up to 256 colors, and the last for grayscale images up to 256 colors. In the B/W model the default JBIG ten color model is used for **D1**. In the 16 color model, the nine edge model is used for **D1**, **D2** is not made, and the three neighbor model is used for **D3**. The 256 color model uses the nine edge model for **D1**, the orientation/diagonal color model for **D2**, and the single color model for **D3**. The grayscale model does not make **D1**–**D3** decisions. The number of model parameters is summarized in Table 4.

Palette	Model Parameters			
Depth	**D1**	**D2**	**D3**	**D4**
1	1024	0	0	0
4	512	0	256	16
8	512	512	1024	760
8-gray	0	0	0	760

Table 4
Model Parameters

The number of *SI* contexts is $2(\lceil \log_2(W-1) \rceil + 1)$, where *W* is the image width. Half of the contexts are used to condition *S* and half for *I*. In the black/white model, the number of *SI* contexts is doubled.

IX.EXPERIMENTS

A. The PWC Corpus

During the initial development of PWC, a group of images intended to serve as a benchmark palette image corpus was assembled. The PWC corpus contains completely synthetic images, nearest color quantized images, quantized images with error diffusion, and compound images containing both synthetic and natural elements. An attempt was made to balance the number of bits of each source type. Some particular emphasis was placed on obtaining palette images from popular sites on the World-Wide Web. The last image in the corpus, yahoo, is shown in grayscale form in Figure 1.

Image	GIF	PNG	bzip2	BW-MTF
benjerry	4,401	4,571	3,896	3,412
books	11,177	10,831	10,310	9,396
ccitt01	38,862	28,910	24,809	23,412
cmpndn	62,682	56,397	59,324	57,210
cmpndu	76,759	69,438	49,146	45,878
flax	846	318	273	460
gate	23,313	20,124	18,344	16,991
music	1,987	1,647	1,729	1,606
netscape	17,442	15,879	13,842	12,591
pattern	1,782	1,928	1,537	1,375
sea_dusk	6,362	2,540	1,886	2,230
stone	4,753	3,906	4,028	4,361
sunset	100,186	81,794	76,743	64,783
winaw	18,559	18,732	16,155	14,995
yahoo	7,097	6,275	6,212	5,670
Total	**376,208**	**323,290**	**288,234**	**264,370**
Time	**2.0/1.5**	**2.7/1.1**	**7.7/2.6**	**—[3]**

Table 5
One Dimensional Methods on the Palette Image Corpus

Table 5 shows the results of applying various one-dimensional compression methods to the PWC corpus. PNG compresses better than GIF but both are outperformed by bzip2. The column labeled BW-MTF is bzip2 without move to front coding.

Table 6 shows the results of several two-dimensional methods applied to the PWC corpus. The first column is CALIC augmented with palette ordering, the second column is the second version of EIDAC, and the last column is RAPP. RAPP is the only one of these methods designed expressly for palette images and it is the only one that compresses better than bzip2.

Image	CALICO	EIDAC	RAPP
benjerry	4,193	2787	2,768
books	14,033	8742	9,634
ccitt01	18,146	15861	15,895
cmpndn	56,951	60033	63,605
cmpndu	71,109	47582	46,520
flax	379	90	124
gate	20,555	17891	17,340
music	1,648	955	831
netscape	14,302	11697	12,127
pattern	1,755	1123	1,315
seadusk	1,446	1208	787
stone	8,440	4064	4,665
sunset	113,710	92288	62,695
winaw	21,686	13384	13,662
yahoo	6,884	5079	4,897
Total	**355,237**	**282,784**	**256,865**
Time	**—[4]**	**43.4[5]**	**30/31**

Table 6
Two Dimensional Methods

Table 7 shows compression results from three different versions of PWC. PWC-O is object PWC, PWC-S is streaming PWC and PWC-SI is streaming PWC plus SI codes. Encode and decode times are symmetric so only one timing result is shown for each method. Color reacquisition accounts for the slightly worse compression performance of PWC-S relative to PWC-O. Interestingly, PWC-SI recaptures this loss and more. This result not yet fully developed and is the topic of a future paper.

Image	PWC-O	PWC-S	PWC-SI
benjerry	2,387	2,418	2,399
books	8,630	8,616	8,153
ccitt01	12,890	12,881	12,683
cmpndn	40,390	54,780	53,021
cmpndu	53,951	40,917	39,556
flax	149	107	142
gate	15,530	15,784	15,282
music	735	755	696
netscape	10,649	10,786	10,533
pattern	1,174	1,178	1,099
seadusk	657	646	678
stone	4,001	4,268	3,637
sunset	52,341	52,923	51,623
winaw	11,440	11,459	10,853
yahoo	4,374	4,443	4,350
Total	**219,218**	**221,961**	**214,705**
Time	**22.8**	**8.0**	**2.4**

Table 7
Improving PWC Performance

Table 8 shows in system performance of PWC, PNG, and GIF in an Internet browser environment. The first column shows local hard-drive performance under Netscape Navigator 4.5. The second column shows local hard-drive performance under Microsoft Internet Explorer 4.01. The last column shows IE performance over a 28.8Kbit/sec dialup connection.

[3] Unknown but probably better than bzip2.

[4] Too cumbersome to measure, but substantial.

[5] Timed on a 360MHz SUN SPARC Ultra 5.

Format	Navigator	Explorer	28.8Kb
GIF	2.5	2.0	2:03
PNG	6.0	2.0	1:51
PWC	2.5	2.0	1:13

Table 8
In-Browser Performance

B. An Expanded Palette Image Corpus

The PWC corpus is designed for benchmarking the average behavior of palette image algorithms. It is less useful for focusing on various palette image subclasses. Therefore an expanded corpus was assembled to better map the characteristics of the higher performance methods.

The expanded corpus of Table 9 contains five image groups, each designed to cover a specific performance regime. The CCITT fax documents represent the class of two-color images. Representing images with relatively few colors is the JPEG-LS test image, pc. The third group is the PWC corpus, representing images with an average number of colors. The dither group contains two highly dithered images with a full color complement that also retain some residual predictive structure in the palette[6]. The last group is the well known grayscale image of lena.

	PWC-SI	bzip2	PNG	GIF	JPEG-LS
ccitt	186,513	372,640	418,490	464,437	577,849
pc	98,831	198,723	225,936	376,482	361,570
corpus	214,705	288,363	323,290	376,208	415,734
dither	477,564	485,321	544,694	645,732	559,652
lena	141,944	173,645	223,051	230,921	138,883
Total	**1,119,557**	**1,518,692**	**1,735,461**	**2,093,780**	**2,053,688**
Time	**12.5**	**36.3/12.6**	**16.0/3.6**	**9.9/6.7**	**15.6/12.9**

Table 9
Expanded Corpus Results

The four highest performance reference coding methods, bzip, PNG, GIF, and JPEG-LS, were compared against PWC-SI on the expanded corpus. In the table bzip, PNG, and GIF encode and decode times are shown separated by a slash. Two encode times are shown for JPEG-LS. The first is compression time elapsed. The second is time reported by the program.

PWC is the only method that is robust across all the image classes. PWC operates similarly to JBIG on b/w images and to JPEG-LS on predictive material. It blends smoothly between the two and even does well on images that can perhaps be better described with a one dimensional model.

C. The SI Mechanism

In PWC, *SI* codes are used within an arithmetic coding framework. However, because it cleanly separates one and two dimensional modeling, *SI* may have general utility as a model blending tool. Table 10 uses the CCITT Fax documents to show how *SI* can be effectively used in both arithmetic and non-arithmetic coding frameworks.

[6] Obtained from Nasir Memon.

ccitt#	SI	JBIG	JPEG-LS	SI-jls
1	12,675	12,788	35,840	21,829
2	7,726	7,938	30,439	13,221
3	19,494	19,950	71,211	40,110
4	48,461	48,942	126,450	84,595
5	22,647	23,187	73,769	42,589
6	11,493	11,689	51,664	24,983
7	51,085	52,227	133,423	77,349
8	12,932	13,220	55,053	25,152
Total	**186,513**	**189,941**	**577,849**	**329,828**
Time	**3.3**	**9.5**	**7.3**	**2.5**

Table 10
CCITT Fax Reference Documents

Columns PWC-SI and JBIG of Table 10 are PWC-SI and sequential JBIG respectively. Remarkably, PWC-SI compression is almost 2% better than JBIG. This despite the fact that the JBIG probability estimator is tuned for these images The source of PWC-SI's improvement is that by using *SI*, more contexts are available for modeling uniform image areas.

The speed up achieved by PWC-SI over JBIG (with typical prediction) is ~3:1. To the author's knowledge this is the highest performance arithmetic result on the CCITT documents yet published.

The final two columns of Table 10 compare JPEG-LS and a matched non-arithmetic SI variant. The results show that for structured material SI is a more robust alphabet extension mechanism than the *block*-Melcode of JPEG-LS. The reason is that once in run mode JPEG-LS pays no attention to the surrounding context. This results in context mixing and loss of compression efficiency. The *SI* mechanism differs from conventional one-dimension run length coding in that it is inherently embedded in a two-dimensional coding process. *SI* never codes information in more than one context and therefore does not lose any of the benefit of the encompassing two-dimensional model.

D. Dynamically Created Content

Because the PWC-SI model is symmetric it lends itself to compression of dynamically created content of the type commonly used on the Internet. Such content is often synthetic or composite and as such is often both sparse and highly structured. Examples of such material include charts, figures, maps, clip art, page backgrounds, and user-interface elements.

Metric	PNG	PWC
Comp. Bytes	33,614	13,558
Comp. Rate	29.2:1	72.4:1
Encode (sec)	1.1	0.6
Decode (sec)	0.6	0.6
Enc/Dec (sec)	1.7	1.2

Table 11
Dynamic Content Examples

In Table 11 an example from each of these classes was compressed using PWC-SI and PNG, its closest competitor in terms of compression rate and efficiency. PWC-SI's compression rate is about two and a half times better than that of PNG. Remarkably, PWC-SI matches PNG's extremely fast decode speed on encode as well.

E. Experimental Notes

All compression times were obtained using a 200 MHz Intel Processor running Microsoft Windows. The *SI* augmented PWC codec and browser plugins for the two major internet browsers are available at http://www.caravian.com. The SI-jls codec and older veisions of PWC are obtainable via email request from the author. PWC compression times were obtained using the "-flip" option of the codec.

JBIG results were obtained using the JBIGKIT[30]. JPEG-LS results were obtained using the HP Labs LOCO-I implementation[31]. BWT results were obtained using bzip2[32].

Unless otherwise noted,. all times are time elapsed. GIF compression times were simulated using compress[33], and PNG compression times were simulated using command line versions of zip[34] and unzip[35].

CALICO, EIDAC, and RAPP results were obtained using programs obtained from their respective authors. Ziya Arnavut supplied the BWT-MTF data.

X.SUMMARY

From its object based inception, PWC has exhibited the best known lossless compression of palette images. Over time it has evolved into a high performance scanline oriented code that handles image structure particularly well. Since it is symmetrical PWC may find its best use in compression of dynamically created synthetic content.

PWC has also introduced a new philosophy for using binary arithmetic coding to blend widely disparate image models. Along this line several new ideas have been introduced including context competition and SI run-length codes. PWC's development has opened many avenues for further investigation and it promises to improve further in the future.

XI.REFERENCES

[1]M. Weinberger, G. Seroussi, G. Sapiro, and M. W. Marcellin, "The LOCO-I Lossless Image Compression Algorithm: Principles and Standardization into JPEG-LS," *HPL98-193*, HP Labs, 1998.

[2]JBIG, "Progressive Bi-level Image compression", International Standard ISO/IEC 11544, ITU-T Recommendation T.82, 1993.

[3] S. W. Thomas, J. McKie, S. Davies, K. Turkowski, J. A. Woods, and J. W. Orost "Compress (version 4.0) program and documentation," 1985.

[4]T.A Welch., "A Technique for High-performance Data Compression," *Computer* 17(6) pp. 8-19, June 1984.

[5]J. Ziv and A. Lempel, "Compression of Individual Sequences via Variable-Rate Coding," *IEEE Transactions on Information Theory*, 24(5) pp. 530-536, September 1978.

[6]Peter Deutsch, "DEFLATE Compressed Data Format Specification," *rfc1951*, http://www.cis.ohio-state.edu/htbin/rfc/rfc1951.html, May 1996,

[7]J. Ziv and A. Lempel, "A Universal Algorithm for Sequential Data Compression," *IEEE Transactions on Information Theory*, 23(3) pp. 337-343, May 1977.

[8]D. A. Huffman, "A method for the construction of minimum-redundancy codes," *Proc. IRE*, 40(9) pp. 1098-1101, September 1952.

[9]Timothy C. Bell, John G. Cleary, and Ian H. Witten, *Text Compression*, Prentice Hall, Englewood Cliffs, NJ, 1990.

[10] M. Burrows and D.J. Wheeler, "A Block-sorting Lossless Data Compression Algorithm", SRC Research Report, Digital Systems Research Center, 130 Lytton Avenue, Palo Alto, California 94301.

[11] Peter Fenwick, "The Burrows-Wheeler Transform for Block Sorting Text Compression: Principles and Improvements," *The Computer Journal*, 39(9), September 1996.

[12] Ziya Arnavut and Hasan H. Otu, "Lossless Compression of Compound and Psuedo-Color Images with Burrows-Wheeler Transformation," publication pending.

[13] N. D. Memon and A. Venkateswaran, "On Ordering Color Maps for Lossless Predictive Coding," *IEEE Transactions on Image Processing*, 5(11) pp. 1522-1527, November 1996.

[14] Xiaolin Wu, "An Algorithmic Study on Lossless Image Compression," Proc. Data Compression Conference, March 1996.

[15] Y. Yoo, Y. Kwon, and A. Ortega, "Embedded Image-Domain Adaptive Compression of Simple Images," *32nd Asilomar Conference on Signals, Systems and Computers*, November 1998.

[16] Y. Yoo, Y. Kwon, and A. Ortega, "Embeded Image Domain Compression," *International Conference on Image Processing*, Kobe, Japan, Oct.1999.

[17] Viresh Ratnakar, "RAPP: Lossless Image Compression with Runs of Adaptive Pixel Patterns," *32nd Asilomar Conference on Signals, Systems and Computers*, pp. 1251-1255, November 1998.

[18] O.R. Jensen and S. Forchhammer, "Content Progressive Coding of Limited Bits/Pixel Images," *Proc. 3rd IEEE Workshop on Multimedia Sig. Processing*, pp. 419-424, Elsinore, Denmark, 1999.

[19] Paul J. Ausbeck Jr, "Context Models for Palette Images," *Proceedings Data Compression Conference*, March 1998.

[20] K. Appel and W. Haken, "Every planar map is four colorable," *Illinois J. Mathematics*, Vol. 21, pp. 429-567, 1977.

[21] Paul J. Ausbeck Jr., "Image Partition Boundary Coding," *Applications of Digital Image Processing XXI*, SPIE, July 1998.

[22] D. A. Novik and J. C. Tilton, "Adjustable Lossless Image Compression Based on a Natural Splitting of an Image into Drawing, Shading, and Fine-Grained Components," *Space and Earth Science Data Compression Workshop*, 1992.

[23] Stephen R. Tate, "Lossless Compression of Region Edge Maps," *CS–1992–9*, Computer Science Deptartment., Duke University, Durham, NC, 1992.

[24] Don Speck, "Local Activity Level Classification Model for Continuous–tone Coding", *Document N198*, Submitted to ISO/IEC JTC1/SC29/WG1, June 1995.

[25] Robert F. Rice, "Lossless Coding Standards for Space Data Systems," *Thirtieth Asilomar Conference on Signals, Systems and Computers*, pp. 577-585, November 1996.

[26] S. W. Golomb, "Run-Length Encodings," *IEEE Transactions on Information Theory*, Vol. IT-12, pp. 399-401, July 1966.

[27] Paul J. Ausbeck Jr., "A Streaming Piecewise-Constant Model", *Proceedings Data Compression Conference*, March 1999.

[28] Paul J. Ausbeck Jr., "The Skip-Innovation Model for Sparse Images," *Proceedings Data Compression Conference*, March 2000.

[29] Don Speck, *Carry-free Arithmetic Coder*, personal communication, April, 1996.

[30] Markus Kuhn, *JBIG-KIT*, Version 0.9, available at ftp.informatik.uni–erlangen.de/pub/doc/ISO/JBIG.

[31] HP Labs, *LOCO-I/JPEG-LS Page*, http://www.hpl.hp.com/loco.

[32] Julian Seward, *bzip2*, http://sourceware.cygnus.com/bzip2/.

[33] *WinXs Unix Tools for Windows*, available at the ZDNet Software Library, http://www.hotfiles.com.

[34] Mark Adler, Richard B. Wales, Jean-loup Gailly, Onno van der Linden and Kai Uwe Rommel, *Zip*, http://www.cdrom.com/.

[35] Greg Roelofs, *Unzip*, http://www.cdrom.com/pub/infozip/.