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 Yahoo 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.

<table>
<thead>
<tr>
<th>Uncoded</th>
<th>Bit Planes</th>
<th>Predictive</th>
<th>GIF</th>
</tr>
</thead>
<tbody>
<tr>
<td>27,182</td>
<td>9,266</td>
<td>8,825</td>
<td>6,923</td>
</tr>
</tbody>
</table>

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) is the most commonly used file format for the distribution of palette images. The compression algorithm used for GIF image data is 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 LGC 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 image map, 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[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.
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.

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:

\[
D(p) = (1 - p)^3 + 3p(1 - p)^2.
\]  

1.5
1
0.5
0
0.5
1
2
\( H_T(p) \)
\( H_1(p) \)
\( 2H_1(p) \)
\( p \)

Figure 4
Deterministic Decision Probability

Also, given identical \( p \), the vertical decision entropy is:

\[
H_v(p) = -p \log_2(p) - (1-p) \log_2(1-p).
\]  

The horizontal entropy adjusted for determinism is:

\[
H_h(p) = (1 - D(p))H_v(p),
\]  

and the total edge entropy is:

\[
H_e(p) = H_v(p) + H_h(p).
\]

The maximum of \( H_e \) is 1.607 and it occurs at a full edge probability of 0.632. \( H_e \) is plotted in Figure 5. Note how \( H_e \) approaches the single decision entropy, \( H_1 \), at low \( p \) and approaches \( 2H_1 \) 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:

\[
D(p) = (1-p)^3 + 3p(1-p)^2 + 3p^2(1-p) + p^3 = 1
\]  

Figure 5
Total Edge Map Decision Entropy

The maximum of \( H_e \) is 1.607 and it occurs at a full edge probability of 0.632. \( H_e \) is plotted in Figure 5. Note how \( H_e \) approaches the single decision entropy, \( H_1 \), at low \( p \) and approaches \( 2H_1 \) 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:

\[
D(p) = (1-p)^3 + 3p(1-p)^2 + 3p^2(1-p) + p^3 = 1
\]
giving the expected result of complete determinism for \( D_{1h} \).

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 \( D_1 \) decisions. Since the model for \( D_{1v} \) has eight elements and the model for \( D_{1h} \) has nine, the total number of contexts for conditioning \( D_1 \) decisions is 768. Connectivity constraints reduce the number of active contexts to 512.

![Figure 6 Edge Context Model](image)

C. Color Coding

For typical palette images, neighboring pixels do not have a linear predictable 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 that has occurred previously in the coding process. The size of a guess model is proportional to \( D^2 \) 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 \( D_3 \) 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 \( D^{34} \), 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 7 Diagonal Contexts](image)

Domain color is usually more important than domain shape in conditioning \( D_2 \) decisions. For this reason \( D_2 \) 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 \( D_2 \) 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.

Figure 8

![Guess Contexts](image)

Color guessing, PWC language element \( D_3 \), 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 \( D^2 \) 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 \( D_3 \) 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 \( D^{34} \), 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.
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.
- 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.

"Top Hat"

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
VI. SPARSE IMAGE METHODS

Though significantly more efficient than 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, computational effort is spent encoding uniform image areas. The basic idea is to skip over uniform areas entirely if possible, and to blend two-dimensional modeling and run-length codes. The basic 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.

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.

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.

Table 3 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.

<table>
<thead>
<tr>
<th>$I$</th>
<th>$S=1$</th>
<th>$S=2$</th>
<th>$S=3$</th>
<th>$S=4$</th>
<th>$S=5$</th>
<th>$S=6$</th>
<th>$S=7$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>00</td>
<td>000</td>
<td>0000</td>
<td>00000</td>
<td>00000</td>
<td>00000</td>
<td>00000</td>
</tr>
<tr>
<td>1</td>
<td>01</td>
<td>001</td>
<td>0001</td>
<td>00001</td>
<td>00000</td>
<td>00000</td>
<td>00000</td>
</tr>
<tr>
<td>2</td>
<td>01</td>
<td>010</td>
<td>0010</td>
<td>00100</td>
<td>00100</td>
<td>00100</td>
<td>00100</td>
</tr>
<tr>
<td>3</td>
<td>01</td>
<td>011</td>
<td>0011</td>
<td>00111</td>
<td>00111</td>
<td>00111</td>
<td>00111</td>
</tr>
<tr>
<td>4</td>
<td>01</td>
<td>010</td>
<td>0010</td>
<td>00100</td>
<td>00100</td>
<td>00100</td>
<td>00100</td>
</tr>
<tr>
<td>5</td>
<td>01</td>
<td>011</td>
<td>0011</td>
<td>00111</td>
<td>00111</td>
<td>00111</td>
<td>00111</td>
</tr>
<tr>
<td>6</td>
<td>01</td>
<td>011</td>
<td>0011</td>
<td>00111</td>
<td>00111</td>
<td>00111</td>
<td>00111</td>
</tr>
<tr>
<td>7</td>
<td>01</td>
<td>011</td>
<td>0011</td>
<td>00111</td>
<td>00111</td>
<td>00111</td>
<td>00111</td>
</tr>
</tbody>
</table>

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 $D + 1$. 

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.
IS = bit multiply/divide operation which can be \( \div \), skip forward and run length coding. The inherently embedded in a two-dimensional coding process. The conventional one-dimension run length coding in that it is never codes information in more than one context and therefore useful to double number of contexts used for coding the use.

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 \( D_4 \), PWC augments the basic coder with 4-5-6-7 adaptation where statistics are halved whenever the least probable symbol count reaches eight. For \( D_4 \), 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 \( 16 \times 8 \div 8 \times 16 \) 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 \( D_1 \). In the 16 color model, the nine edge color model is used for \( D_1 \), \( D_2 \) is not made, and the three neighbor model is used for \( D_3 \). The 256 color model uses the nine edge model for \( D_1 \), the orientation/diagonal color model for \( D_2 \), and the single color model for \( D_3 \). The grayscale model does not make \( D_1 \)–\( D_4 \) decisions. The number of model parameters is summarized in Table 4.

<table>
<thead>
<tr>
<th>Palette Depth</th>
<th>Model Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( D_1 )</td>
</tr>
<tr>
<td>1</td>
<td>1024</td>
</tr>
<tr>
<td>4</td>
<td>512</td>
</tr>
<tr>
<td>8</td>
<td>512</td>
</tr>
<tr>
<td>8-gray</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 4: Model Parameters
The number of SI contexts is $2^{\left\lfloor \log_2(W - 1) \right\rfloor + 1}$, 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.

<table>
<thead>
<tr>
<th>Image</th>
<th>GIF</th>
<th>PNG</th>
<th>bzip2</th>
<th>BW-MTF</th>
</tr>
</thead>
<tbody>
<tr>
<td>benjerry</td>
<td>4,401</td>
<td>4,571</td>
<td>3,896</td>
<td>3,412</td>
</tr>
<tr>
<td>books</td>
<td>11,177</td>
<td>10,831</td>
<td>10,310</td>
<td>9,396</td>
</tr>
<tr>
<td>ccitt01</td>
<td>38,862</td>
<td>28,910</td>
<td>24,809</td>
<td>23,412</td>
</tr>
<tr>
<td>cmpndn</td>
<td>62,682</td>
<td>56,397</td>
<td>59,324</td>
<td>57,210</td>
</tr>
<tr>
<td>cmpndu</td>
<td>76,759</td>
<td>69,438</td>
<td>49,146</td>
<td>45,878</td>
</tr>
<tr>
<td>flax</td>
<td>846</td>
<td>318</td>
<td>273</td>
<td>460</td>
</tr>
<tr>
<td>gate</td>
<td>23,313</td>
<td>20,124</td>
<td>18,344</td>
<td>16,991</td>
</tr>
<tr>
<td>music</td>
<td>1,987</td>
<td>1,647</td>
<td>1,729</td>
<td>1,606</td>
</tr>
<tr>
<td>netscape</td>
<td>17,442</td>
<td>15,879</td>
<td>13,842</td>
<td>12,591</td>
</tr>
<tr>
<td>pattern</td>
<td>1,782</td>
<td>1,928</td>
<td>1,537</td>
<td>1,375</td>
</tr>
<tr>
<td>sea_dusk</td>
<td>6,362</td>
<td>2,540</td>
<td>1,866</td>
<td>2,230</td>
</tr>
<tr>
<td>stone</td>
<td>4,753</td>
<td>3,906</td>
<td>4,028</td>
<td>4,361</td>
</tr>
<tr>
<td>sunset</td>
<td>100,186</td>
<td>81,794</td>
<td>76,743</td>
<td>64,783</td>
</tr>
<tr>
<td>winaw</td>
<td>18,559</td>
<td>18,732</td>
<td>16,155</td>
<td>14,995</td>
</tr>
<tr>
<td>yahoo</td>
<td>7,097</td>
<td>6,275</td>
<td>6,212</td>
<td>5,670</td>
</tr>
<tr>
<td>Total</td>
<td>376,208</td>
<td>323,290</td>
<td>288,234</td>
<td>264,370</td>
</tr>
<tr>
<td>Time</td>
<td>2.0/1.5</td>
<td>2.7/1.1</td>
<td>7.7/2.6</td>
<td>—/3</td>
</tr>
</tbody>
</table>

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

Two Dimensional Methods

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.

<table>
<thead>
<tr>
<th>Image</th>
<th>CALICO</th>
<th>EIDAC</th>
<th>RAPP</th>
</tr>
</thead>
<tbody>
<tr>
<td>benjerry</td>
<td>4,193</td>
<td>2,787</td>
<td>2,768</td>
</tr>
<tr>
<td>books</td>
<td>14,033</td>
<td>8,742</td>
<td>9,634</td>
</tr>
<tr>
<td>ccitt01</td>
<td>18,146</td>
<td>15,861</td>
<td>15,895</td>
</tr>
<tr>
<td>cmpndn</td>
<td>56,951</td>
<td>60,033</td>
<td>63,605</td>
</tr>
<tr>
<td>cmpndu</td>
<td>71,109</td>
<td>47,582</td>
<td>46,520</td>
</tr>
<tr>
<td>flax</td>
<td>379</td>
<td>90</td>
<td>124</td>
</tr>
<tr>
<td>gate</td>
<td>20,555</td>
<td>17,891</td>
<td>17,340</td>
</tr>
<tr>
<td>music</td>
<td>1,648</td>
<td>955</td>
<td>831</td>
</tr>
<tr>
<td>netscape</td>
<td>14,302</td>
<td>11,697</td>
<td>12,127</td>
</tr>
<tr>
<td>pattern</td>
<td>1,755</td>
<td>1,123</td>
<td>1,315</td>
</tr>
<tr>
<td>seadusk</td>
<td>1,446</td>
<td>1,208</td>
<td>787</td>
</tr>
<tr>
<td>stone</td>
<td>8,440</td>
<td>4,064</td>
<td>4,665</td>
</tr>
<tr>
<td>sunset</td>
<td>113,710</td>
<td>92,288</td>
<td>62,695</td>
</tr>
<tr>
<td>winaw</td>
<td>21,686</td>
<td>13,384</td>
<td>13,662</td>
</tr>
<tr>
<td>yahoo</td>
<td>6,884</td>
<td>5,079</td>
<td>4,897</td>
</tr>
<tr>
<td>Total</td>
<td>355,237</td>
<td>282,784</td>
<td>256,865</td>
</tr>
<tr>
<td>Time</td>
<td>—/4</td>
<td>43.4/5</td>
<td>30/31</td>
</tr>
</tbody>
</table>

Table 7

Improving PWC Performance

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 lost and more. This result is not yet fully developed and is the topic of a future paper.

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

4 Too cumbersome to measure, but substantial.
5 Timed on a 360MHz SUN SPARC Ultra 5.
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.

<table>
<thead>
<tr>
<th></th>
<th>ccitt</th>
<th>SI</th>
<th>JBIG</th>
<th>JPEG-LS</th>
<th>SI-jls</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>186,513</td>
<td>189,941</td>
<td>577,849</td>
<td>329,828</td>
<td></td>
</tr>
<tr>
<td>Time</td>
<td>3.3</td>
<td>9.5</td>
<td>7.3</td>
<td>2.5</td>
<td></td>
</tr>
</tbody>
</table>

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.

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.

<table>
<thead>
<tr>
<th></th>
<th>Metric</th>
<th>PNG</th>
<th>PWC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comp. Bytes</td>
<td>33,614</td>
<td>13,558</td>
<td></td>
</tr>
<tr>
<td>Comp. Rate</td>
<td>29:2:1</td>
<td>72:4:1</td>
<td></td>
</tr>
<tr>
<td>Encode (sec)</td>
<td>1.1</td>
<td>0.6</td>
<td></td>
</tr>
<tr>
<td>Decode (sec)</td>
<td>0.6</td>
<td>0.6</td>
<td></td>
</tr>
<tr>
<td>Enc/Dec (sec)</td>
<td>1.7</td>
<td>1.2</td>
<td></td>
</tr>
</tbody>
</table>

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.

---

6 Obtained from Nasir Memon.
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 versions 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


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


