Image Partition Boundary Coding

Paul J. Ausbeck Jr. [a]

Department of Computer Engineering
University of California
Santa Cruz, CA 95060

ABSTRACT

This paper introduces two image partition boundary coding models that are composed solely of binary decisions. Because of their simplified decision structure, the models can take advantage of various accelerating schemes for binary arithmetic coding. The number of decisions necessary to describe a partition using either model varies between one and two per pixel location and is proportional to partition complexity.

The first model is a binary decomposition of Steve Tate's neighboring edge model. The decomposition employs boundary connectivity constraints to reduce the number of model parameters. The constraints also reduce the number of descriptive decisions to just over one per pixel for typical partitions. A theoretical zero order entropy bound of 1.6 bits per pixel also results.

The second model represents a partition as a sequence of strokes. A stroke consists of one or two three-way chains. Chain termination is accomplished without redundant boundary traversal by using a special termination decision at encounters with previously drawn chains. Chain initiation decisions are also conditioned on previously drawn edge patterns. Chain direction decisions are conditioned via a boundary state machine.

The paper compares object based boundary coding and pixel based coding, placing the new coders into the latter category. A technique for determining the appropriate application domain of pixel based codes is developed. The new coding models are placed into context with previous pixel based work by the development a new categorization of image partition representations. Four representations are defined, the map coloring, the edge map, the outline map, and the perimeter map. Experiments compare the new methods with other pixel based methods and with a canonical object based method.

Keywords: contour coding, boundary coding, image segmentation, map coloring, image partition, chain coding, edge map, outline map, perimeter map

1. INTRODUCTION

The image partition has traditionally been closely linking with the idea of image segmentation. More recently, the partition has been separated from any particular segmentation method and found use in its own right. For example, the new MPEG-4 video coding standard[1] uses a simple partition coding method for foreground/background separation. Another new use of partitioning is for compactly representing palette images[2]. The purpose of this paper is to help define just what image partition coding is about, and to disclose two new efficient coders.

In the first section, the image partition is defined and related to the long-standing mathematical problem of map coloring. The partition coding problem is then connected to the image coding problem by the definition of several pixel based partition representations. Next, a technique for comparing pixel based partition codes with object/vertex codes is developed. Once the motivation for pixel based coding schemes is clear, one of the pixel-based representations, the edge map, is used as the basis for two new boundary codes. The first code uses a raster neighborhood decision conditioning template. The second uses a chain code model. The two new codes are then compared to other image based codes on a synthetic geometric image rendered at various resolutions. Interpretation of the experiments and conclusions follow.

[a]The author is an adjunct researcher at UCSC and cofounder of Netcelerate Software, Inc (www.netcelerate.com). His email address is paul@netcelerate.com or alternately paula@cse.ucsc.edu.

2. PROPERTIES OF IMAGE PARTITIONS

An image *partition* is a segregation of the image pixels into related groups. A *domain* is defined as a related group of pixels in an image partition. Partitions may be classified according to the shape and topological properties of their domains. A domain is *contiguous* if for all pairs of its pixels there exists a path between the pair that traverses only other pixels within the domain. A domain is *rectilinearl- connected* if all such paths start at the center of one pixel, make only rectilinear movements, and make intermediate stops only at the center of other pixels. The algorithms of this paper are designed for partitions containing only contiguous rectilinearly-connected domains.

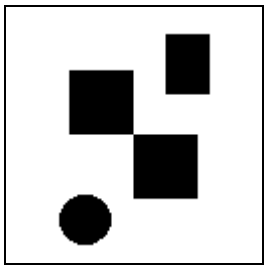

Figure 1

One particularly useful method of representing partitions is the map-coloring. A map-coloring of a partition is an image of the same size as the partitioned image whose pixels have two properties. First, all the pixels in the same domain have the same color. Second, all adjacent pairs of domains have different colors. Because all domains must be contiguous and rectilinearly-connected, a map-coloring is a complete representation of a partition. For example, on the map coloring of Figure 1 there are five domains: two squares, a rectangle, a circle, and the surround. Since all domains must be contiguous, even though the circle and rectangle have the same color they must comprise separate domains. Similarly, since all domains must be rectilinearly connected, the two squares represent separate domains even though they touch at a corner.

Figure 2
Three Color Partition

The *chromaticity* of a partition is the minimum number of colors necessary to map-color its domains. Figure 1 is an example of a two-color partition. Two-color partitions allow *separation* of foreground from background. Further, if the color assignment is known it is possible to *identify* foreground and background given a two-coloring.

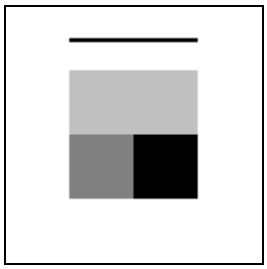

Figure 3
Four Color Partition

Some partitions require three colors and it has long been known that the general case requires four. That four colors is also sufficient was not proven until 1977[3]. Figure 2 is a three color example and Figure 3 is a simple four color example.

The two color partition coding problem is equivalent to the black/white image coding problem and as such has significant literature coverage. One of the more recent applications of two-color coding is for foreground/background separation in MPEG–4[1]. The three color coding problem does not yet appear to have been addressed, but it may not have much commercial significance. The focus of this paper is coding four-color partitions.

3. PARTITION REPRESENTATION

While the map coloring is an important theoretical tool and an excellent medium for partition visualization and distribution, it is not particularly suited to coding. One problem is that four-colorings are relatively difficult to obtain. Another is that any code based upon a map coloring will necessarily reproduce the exact original coloring. Since any valid coloring is all that is necessary to reconstruct a partition, coding a particular coloring is inefficient.

Another representation called the *edge map* avoids these problems and is often better suited as a base for coding a partition. In an edge map, pixels are modeled as small squares embedded in a *separator lattice*. Each pixel has four rectilinearly adjacent separator lattice *sites*. A separator lattice site is *full* if the two adjacent pixels lie in different domains and *empty* otherwise. A full separator site is simply called a *separator*. Since every separator lattice site is adjacent to two pixels, it is convenient to assign each site to a particular pixel. In this paper, the convention is to assign to a pixel the separator sites to its west and north. After assignment, two bits per pixel suffice to store an edge map. The low order of these bits is arbitrarily assigned to the *vertical* (western) separator site and the remaining bit to the *horizontal* (northern) separator site. Interestingly, the edge map has the same storage requirement as the four-coloring, but stores no particular color information. Even more interesting, as shown later, is that the uncoded storage requirement for typical edge maps is actually significantly less than two bits per pixel.

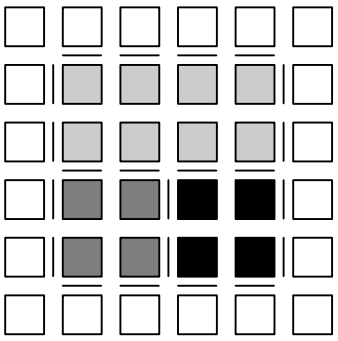

Figure 4
Edge Map Detail

Figure 4 is a detailed rendering of an example edge map. The 22 full separator lattice sites are modeling with short line segments. For typically sized images, the detailed visualization method of Figure 4 is unusable. Further, it is desirable for an edge map visualization to fit into the same size image as the source partition.

Figure 5 is a same size edge map rendering of the 128x128 partition shown in the map-coloring of Figure 3. The possible edge values of 0, 1, 2, and 3 are rendered with luminance values of 255, 224, 128, and 0 respectively. The author believes that this particular method is the best possible for rendering edge maps at the resolution of the source partition. Still, it's not an ideal visualization method.

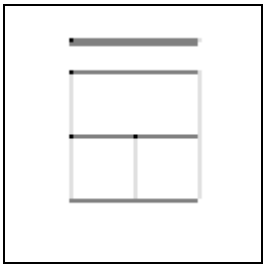

Figure 5
Compact Edge Map
Rendering

A true outline rendering of a partition cannot be done at the source resolution. This is because two pieces of information must be displayed for each pixel location. Resolution expansion in a single direction is not effective because it introduces distortion. Figure 5 is an *outline* rendering of the map coloring of Figure 3. An outline rendering is performed at 2x resolution in both the vertical and horizontal direction. For this example, a 128x128 map-coloring produces a 256x256 outline rendering.

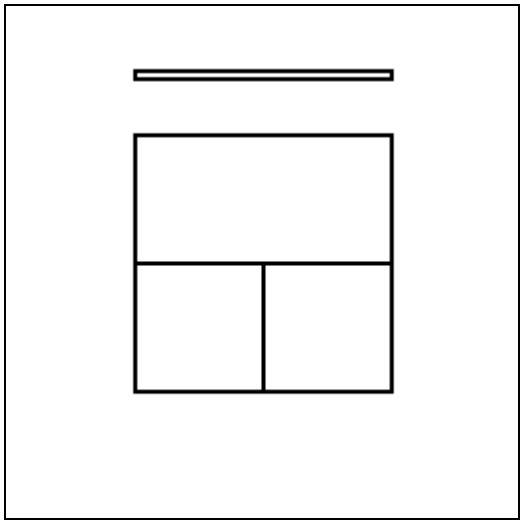

Figure 6
Outline Map

An outline rendering is produced from a map-coloring with the following algorithm.

- Replace each pixel of the map coloring with a square block of four white pixels.
- If the map colored pixel is different from its western neighbor, darken the two leftmost pixels of the corresponding outline block.
- If the map colored pixel is different from its northern neighbor, darken the two topmost pixels of the corresponding outline block.
- If the map colored pixel is different from the northwestern neighbor, darken the upper leftmost pixel of the corresponding outline block.

An outline rendering is also a valid partition representation since a partition can be recovered from it. It's not optimal for coding because its size before coding is twice that of both the map-coloring and the edge map. Even if a coding method could be designed that produced good compression efficiency, the coding speed would be no better than ½ that for the more appropriate representations.

The *perimeter* map representation, shown in Figure 7, is produced by darkening all pixels that are rectilinear neighbors of pixels in other domains. The perimeter map is not a valid representation for three-color or four-color partitions. For example, the single line segment near the top of the partition cannot be distinguished from several smaller collinear line segments if those segments are separated by fewer than three pixels. The perimeter map can be used, however, for building up a partition from separate descriptions of its domains. This process is called partition *compositing*.

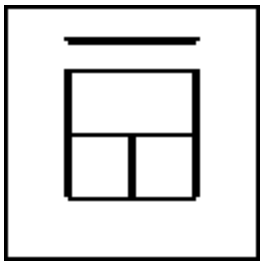

Figure 7
Perimeter Map

When used as a based for coding, a composited representation is inefficient for partitions requiring more than two colors. The reason is that all contours that are not foreground/background separations must be coded twice. For example, The boundary between the two squares of Figure 7 must be coded once for the left square and again when coding the right square. The coding methods reported in this paper operate on complete partition representations and are designed to avoid the redundant coding problem.

4. OBJECT VS PIXEL BASED CODING

Boundary descriptive codes can be classified as either object/vertex based or pixel based. In object based codes boundaries are described with a relatively small number of fairly complex descriptors. In pixel based codes a relatively large number of simple descriptors are used. Because object based methods use roughly the same number of bits to describe a feature regardless of its size, one can expect object based methods to work better for partitions with large features and pixel based methods to be superior for small features.

R(0,0,256,256)
R(0,160,72,256)
R(56,160,72,256)
R(0,160,56,176)
T(56,160,72,160,56,176)
R(144,0,256,256)
T(152,256,255,134,256,255)
R(192,64,256,208)
I((x-64)^2+(y-96)^2-1024)
R(128,32,160,96)
I(2.5*(x-142)^2+3*(x-142)*(y-132)+ 2.5*(y-132)^2-4096)
I(1.0e8*(0.155*(x-128)-0.985*(y-66))^8+ (0.985*(x-128)+0.155*(y-66))^8-1.667e15)

Figure 8
Geometric Image Synthetic Elements

The following procedure can be used to empirically determine an average feature size below which any particular object based method becomes inefficient:

- Define an object based partition description language that is canonical for the type of partition being coded.
- Construct a resolution independent description of a reference partition.
- Using the resolution independent description, synthesize image partitions at multiple resolutions.
- Compress the resolution independent description with an appropriate object based method.
- Compress the various resolution image partitions with an appropriate pixel based method.
- Find the resolution, α, where the object based and pixel based compression methods produce comparable results.

If F is the number of full separator sites in a particular resolution partition, and N is the number of objects in the resolution independent description then define:

$$\Gamma = \frac{F}{N}. \tag{1}$$

to be the average feature size of a particular resolution partition. If Γ_α is the Γ of the α partition then partitions for which $\Gamma <= \Gamma_\alpha$ should be better compressed with pixel based codes. Such partitions are designated *freeform*.

An example application of this procedure should illuminate it further. Figure 8 shows an object based partition description using the RTI geometrical descriptive language. The RTI language has three elements: R, T, and I. R draws a rectilinear rectangle given two opposing corners. T draws an arbitrary triangle from its vertices. I draws the contour where the given algebraic expression is zero or greater on one side and less than zero on the other. Each element specifies a complete closed contour. The drawing order is important in that any boundaries enclosed by a later drawn contour are lost.

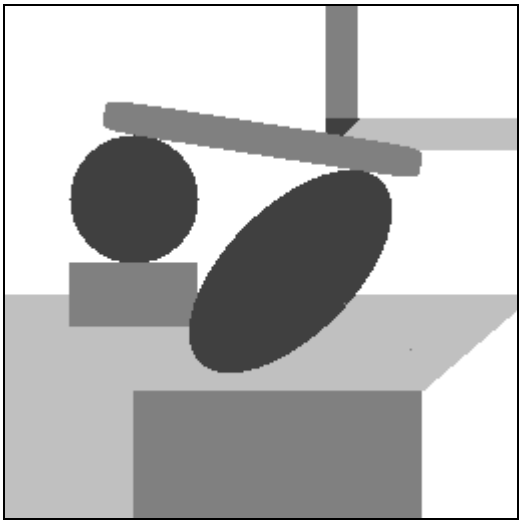

Figure 9
Map Coloring of a Geometric Image

When applied to a 256x256 integer grid, the description of Figure 8 produces the partition whose map coloring is shown in Figure 9. The RTI descriptive language is arguably canonical for this particular reference image. RTI is one dimensional and is effectively compressed with LZ methods. The ZIP utility compresses the 351 bytes in the description of Figure 8 to 163 bytes.

Size	Separators	Γ
32x32	236	19.7
64x64	481	40.1
128x128	964	80.3
256x256	1938	162
512x512	3899	325
1024x1024	7828	652

Table 1
Object Model Instantiation

Instantiating the RTI description at six resolutions produces the separator counts and Γ values shown in Table 1. Note that Γ roughly doubles for each 4x increase partition pixel count. The is because Γ is proportional to the partition's total perimeter, not to its total area. Since determination of Γ_α requires actual data, its calculation of deferred to the experiments.

5. RASTER NEIGHBORHOOD CODES

The first use of a context model in the coding of edge maps was proposed by Tate[4]. The Tate model consists of the eight causal separator sites shown in Figure 10. This model is extremely simple and produces excellent coding results.

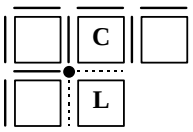

Figure 10

One drawback to Tate's method is the use of a four symbol alphabet to encode the separator state at each lattice location. While this strategy successfully brings all relevant information to bear on each four-way decision, it does not take advantage of advanced acceleration techniques that have been developed for coding binary alphabets. This section proposes an efficient technique for decomposing Tate's four-way decisions into the fewest possible binary decisions.

6. A FAST BINARIZED RASTER NEIGHBOR CODE

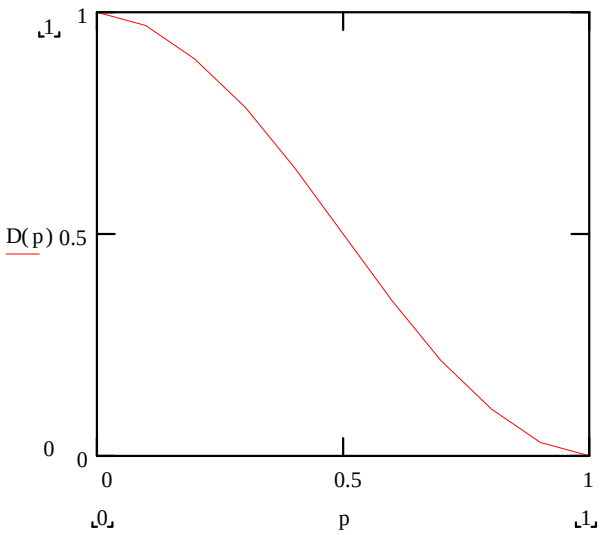

Figure 11
Deterministic Horizontal Decision Probability

A straightforward decomposition of the four symbol edge map produces two binary decisions for each raster location. In this discussion, for each raster location the vertical decision is made first followed by the horizontal decision. After decomposition, 256 binary contexts are required to hold vertical decision statistics. In order to bring a previous vertical decision to bear on its associated horizontal decision, the number of horizontal decision contexts must be double the number of vertical contexts.

Since image partitions are not simply random edge maps, separator lattice connectivity constraints can be used to reduce both the number of decisions and the number of conditioning contexts. The fundamental constraint imposed by separator lattice connectivity is that each end of a separator must connect with some other separator. For a raster scan where horizontal decisions are secondary, this means that many horizontal decisions are completely determined by previously known separator patterns and need not be coded.

Two types of previously known separator patterns are relevant. The first of these is the completely empty separator lattice intersection. At such an intersection three of the separator sites are known to be empty and therefore the fourth must also be empty. The second relevant pattern is the separator lattice intersection with a single full site. There are three possible singleton patterns, N, S, and E, each forcing the presence of a second separator to maintain connectivity.

One consequence of the application of connectivity constraints is that only 256 binary contexts are required to hold horizontal decision statistics. The other 256 states are deterministic. The second consequence is that for typical partitions, substantially fewer coded decisions are necessary. If p is the zero order probability that a separator lattice site is full, then the zero order entropy associated with vertical decisions is

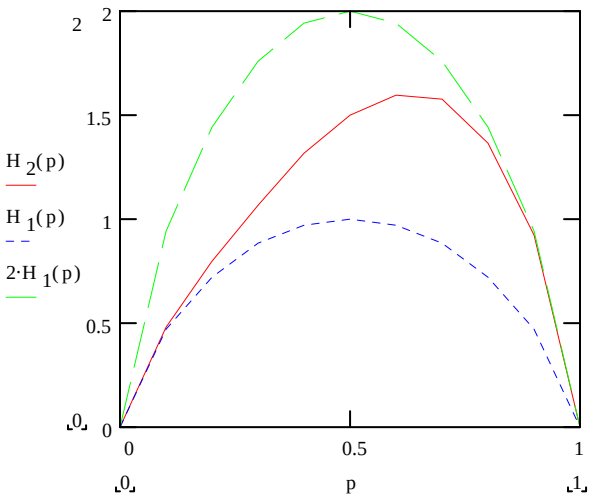

Figure 12
Total Edge Map Decision Entropy

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

Given the same p, the deterministic horizontal decision probability is

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

The horizontal decision entropy is therefore

$$H_2(p) = (1-D(p))H_v(p), \quad (4)$$

and the total entropy is

$$H_T(p) = H_1(p) + H_2(p). \quad (5)$$

The maximum of H_T is 1.607 and it occurs at a full edge probability of 0.632. $D(p)$ is plotted in Figure 11 and H_T is plotted in Figure 12. Note that H_T approaches the single decision entropy, H_1, at low p and approaches the unconstrained two decision entropy at high p.

7. CHAIN CODES

Chain codes can be divided into two major families, the distinguishing feature being the number of possible directions of movement at each point in the chain. With the recent application of context dependent probability estimation to four-way chains[5], the desirability of eight-way chains is somewhat attenuated. We focus exclusively on four-way chains.

Three types of information must be provided for a general-purpose boundary chain code: chain starting points, chain direction information, and chain termination indicators. When coding two-color partitions[5], each chain is guaranteed to return to its starting point. Obviously, no two black features can touch each other or they would be the same feature. This characteristic allows *self-terminating chains*, where termination information is implicitly delivered by return to the starting point.

Reliable termination conditions without backtracking allow for simplified direction information. Only three decisions are possible at each point on the boundary, turn left, right or go straight. The base information per chain event is reduced from two to $\log_2 3$ bits. Unfortunately, when chain coding the boundaries of a three or four-color partition, things get more complicated and some form of redundancy must be introduced.

One way to deal with the termination problem is to composit the partition from individual objects[6]. If the edge map representation is used, this results in the overhead of traversing most or all boundaries twice. If the perimeter representation is used, double traversal *and* backtracking are necessary. One possible way to avoid these problems is to traverse only right or left sides of each domain[7]. This method, however, results in one starting point for each convex boundary section. For irregular domains, this produces multiple chain starting points per domain.

The ideal boundary chain code should take advantage of connectivity constraints to make between one and two decisions per pixel location. It should use approximately one starting point per partition domain and should not incur significant chain termination overhead. The stroke code described next is one such code.

8. THE STROKE CODE FOR IMAGE PARTITIONS

The stroke code develops a partition via a series of *strokes*. Each stroke consists of a start point, and one or two boundary chains. Each chain of a stroke is terminated upon encountering a previous stroke or the image boundary. The key idea is that *encounters* with previous strokes may or may not terminate a stroke. Further information disambiguates each encounter.

Strokes are decoded in turn: start, chain and termination information is interleaved in the code stream. Once the boundary is fully specified via strokes, domains may be grown by recursively joining groups of pixels that do not have a separator between them. Figure 8.1 has seven domains of 1, 1, 2, 5, 6, 8 and 10 pixels plus the surround. As can be seen, only six strokes are necessary to completely specify the boundary between these domains.

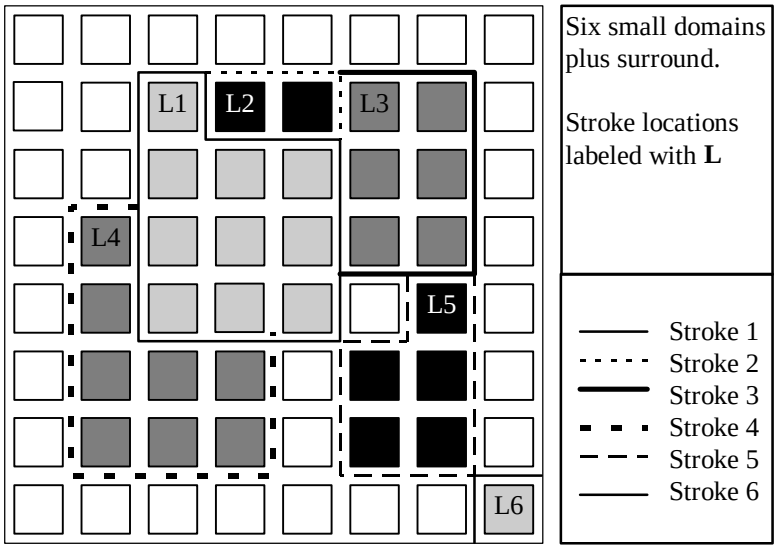

Figure 8.1
Stroke Example

We cover the major elements of the stroke code syntax from the perspective of the decoder. We first cover stroke starting points or *locations*. We then apply context dependent probability estimation to reduce the information content of stroke chains. Finally, we develop a termination mechanism for stroke chains that almost completely retains their three-way decision character.

8.1. Stroke Locations

Each pixel in the image is a possible stroke location. The decoder scans the image in raster order and determines whether or not a pixel is a stroke location. The information associated with this process is called the *stroke location information*. Each of the pixels labeled with **L** in Figure 8.1 are stroke locations.

Each stroke location can have one or two boundary separators. The possible separators are along the northern and western edges of the associated pixel. Deciding that a pixel is a stroke location conveys additional information about the associated boundary separator sites. The additional information conveyed depends upon what other separators abutting the stroke location are already known. The previously known separators associated with a possible stroke location determine the *stroke location context.*

If a pixel site is a stroke location and it already has one of its separators known, then the other separator becomes known. If the separator to the north is already known, the separator to the west becomes known. A stroke location context of this type is called a *top* location. Similarly, a *side* location has its western separator previously known.

Name	Determining information
le	Two previously known separators (enclosed)
lt	One known separator to the north (top)
ls	One known separator to the west (side)
lb	Zero known separators (bare)
lc	Zero known separators (SE corner)
lc2	SE corner, Decide one or two separators
lcw	SE corner, One separator, Decide N or W

Table 8.1
Stroke Location Contexts

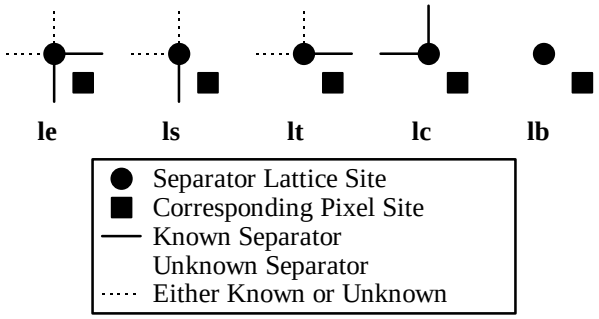

Figure 8.2
Stroke Location Contexts

If a stroke location has neither separator known, it can be either *bare* or a *southeast corner*. A bare location has no edges impinging on its northwest corner. It must be both northernmost and westernmost since it has no other edges with which to connect.

Since it can connect to the boundary corner to its northwest, a southeast corner location can have a separator along its western edge, its northern edge or both. Further *corner location disambiguation information* is supplied by the encoder to differentiate between the three possibilities. Disambiguation takes at most two binary decisions. The first decision is both chains or one. If one, the second decision is north or west.

Table 8.1 summarizes the stroke location contexts and defines abbreviated names. There is no information associated with the ***le*** context; these pixels are not possible stroke locations. The skew of the ***lb*** context is typically much higher than that of the ***ls***, ***lc***, or ***lt*** contexts. On Figure 8.1, strokes one and four have ***lb*** locations. Strokes two and three have ***ls*** locations. Stroke five is ***lt*** and stroke six is ***lc***. Stroke six is also ***lc2*** since it is a corner location and has two associated separators. One additional decision is made for this stroke location.

Figure 8.2 is a graphical representation of the stroke location contexts. A filled square represents the pixel associated with a possible stroke location. A filled circle represents the associated *separator lattice* intersection. Separator sites are shown as solid lines, dashed lines, or invisible lines. Solid sites are known to contain a separator and are designated *full*. Invisible sites are not yet known to contain or not contain a separator. Dashed sites are irrelevant ("don't care").

8.2. Stroke Chains

Every stroke location has a separator along either its northern edge or western edge or both. If a stroke location already has one of its separators known from a previously decoded stroke, it has only one stroke chain that starts in the direction of the just decided edge.

Stroke locations that are ***lb*** or ***lc2*** with two separators can have one or two chains. The chain starting along the stroke location's western edge and heading south is decoded first. If that chain is not *closed*, a second chain is decoded starting along the northern edge and heading east. A closed chain returns to its start location heading in the opposite direction from its outset. The chain starting to the south is also called the *clockwise* chain and the eastern heading chain is correspondingly the *counterclockwise* chain.

Stroke chains are three direction chains: left, right straight. Typically, each chain follows the periphery of a single domain. Therefore, each counterclockwise chain has slightly more left turns and each clockwise chain has slightly more right turns. For example, a closed counterclockwise chain has four more left turns than right turns.

uccw	first decision of a ccw chain
ucw	first decision of a cw chain
is	previous straight, in prior to that
os	previous straight, out prior to that
is*i	in, zero or more straights, in prior to that
os*o	out, zero or more straights, out prior to that
$***is***^{+}***o***	out after at least one straight, in prior to that
$***os***^{+}***i***	in after at least one straight, out prior to that
$***iss***^{+}	at least two previous straights, last turn in
$***oss***^{+}	at least two previous straights, last turn out
oi	previous in, out prior to that
io	previous out, in prior to that
$***oi(oi)***^{+}	***oi*** repeated more than once
$***io(io)***^{+}	***io*** repeated more than once
$***(io)***^{+}***i***	***ii*** with at least one included ***oi***
$***(oi)***^{+}***o***	***oo*** with at least one included ***io***
$***(oi)***^{+}***s***	special case of ***is***
$***(io)***^{+}***s***	special case of ***os***
$***(oi)***^{+}***ss***^{+}	special case of $***iss***^{+}
$***(io)***^{+}***ss***^{+}	special case of $***oss***^{+}

Table 8.2
Stroke Chain Contexts

To take advantage of this statistical disparity, one option would be to keep track of separate statistics for clockwise and counterclockwise chains. However, since this halves the number of decisions in each context, sparse contexts become even sparser and may never develop good probability estimates. A small trick solves this problem. Instead of turning left and right, each chain turns inward or outward. An inward decision is a left turn for a counterclockwise chain and a right turn for a clockwise chain. Outward decisions correspondingly equate to right and left.

Our three way decision is now: in, out, straight. This three-way decision is transformed to two binary decisions via a two level coding tree. The first decision differentiates between straight and turn. If the first decision indicates a turn, the second decides between in and out.

The probability estimation contexts used to reduce the information content of stroke chains are shown in Table 8.2 (actually, each line in the table corresponds to two decision contexts, one for each level of the coding tree). A simple state machine determines the context used to code each chain direction. Where appropriate, the states are named using regular expression notation with the letters ***s***, ***i***, and ***o,*** corresponding to straight, inward, and outward respectively.

These contexts are designed to capture the statistics of raster drawn lines and curves. The main point to note is that they are not simply encodings of the last few chain directions. They are full boundary states as the regular expression notation suggests. Arbitrary length prefixes with the proper structure can specify a context. For example, several contexts end in $\boldsymbol{ss}^{+}$ and their purpose is to differentiate straight sections that are known to be *long* (two or more straight events) from those that are not yet known to be long. Regardless, of the length of the straight section, the preceding turn structure is part of the boundary state.

8.3. Stroke Chain Termination

Perhaps the most distinguishing feature of the stroke code is its mechanism for terminating stroke chains. The key idea is that *encounters* with previously decoded boundary separators or with the image boundary are the only way to terminate stroke chains. A decision is made at every encounter as to whether or not to continue the chain. The different types of possible encounters are illustrated in Figure 8.3 for an example northward heading counterclockwise chain

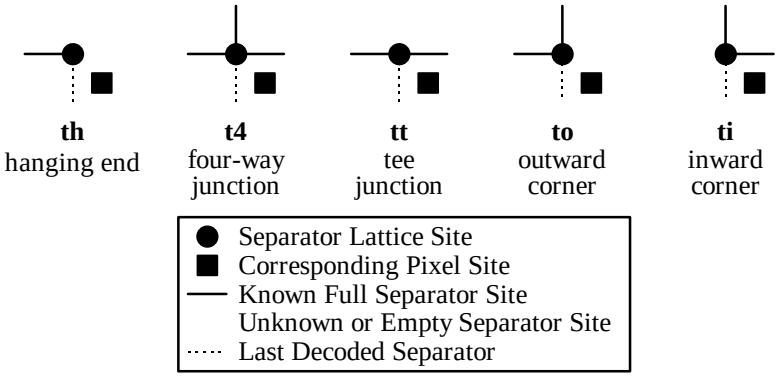

Figure 8.3
Stroke Chain Termination Contexts

We now make some observations about each type of encounter. Since each chain must terminate by encountering a previously decoded boundary, only one hanging end encounter is possible: an encounter with the beginning of the current chain. Four-way junctions are *unambiguous* encounters. There is no empty exit path and a chain must terminate. A tee junction has one possible exit path, but depending upon the method used by the encoder to generate strokes, it may never be taken. A corner junction may be constrained inward or outward. An inwardly constrained corner has a single possible exit in the inward direction. An outwardly constrained corner has an analogous exit in the outward direction. Again, depending upon the method used to generate strokes, the outwardly constrained exit may never be taken.

Looking back to Figure 8.1, we can see some example encounters. Stroke 1 is closed and therefore makes a hanging end encounter with itself. Stroke 2 makes a constrained outward encounter with stroke 1 and terminates. Strokes 3 and 4 have terminating tee encounters with stroke 1. Stroke 5 makes a constrained inward encounter with stroke 1 and a constrained outward encounter with stroke 3. Stroke 6 has two encounters with the image boundary. Encounters with the image boundary are always terminating.

A probability estimation context is kept for each type of boundary encounter. On sparse partitions, boundary encounters are fairly infrequent and those that do occur are frequently terminating. Further, the encoder can arrange for higher skew statistics in some contexts. For example, if certain rules are followed by the encoder, tee and constrained outward encounters may never be continued. Taken together, these characteristics make stroke termination information a negligible portion of the total code string for sparse and even substantially detailed partitions.

9. EXPERIMENTS

To place the new codes in context with other pixel based codes, several experiments were performed with color-map, edge map and outline map representations. Because of the redundancy introduced by composition it was deemed unnecessary to include codes relying on it. This decision completely excluded codes based upon the perimeter map from the results. The experiments were run on the synthetic geometric partitions documented in section 4. These partitions both cover a wide range of detail levels and allow comparison with object based codes.

Three different representations of each partition resolution were coded using LZ'77[8] dictionary coding and JBIG bilevel image compression[9]. The results of these reference experiments are summarized in Table 3. All data are byte counts and exclude any file header overhead.

Several things can be noted from the data. First, given a large enough partition, context conditioned arithmetic coding performs far better than dictionary coding. In fact, the asymptotic ZIP performance is proportional to the number of partition pixels, whereas the asymptotic JBIG performance is proportional to the number of full separator sites.

One somewhat surprising result is that outline JBIG is the best performer at two intermediate resolutions. This despite the fact that this representation requires twice the number of binary decisions as the other two. This is because providing all information in a single context model is vastly superior to separate horizontal and vertical models.

Another interesting result is that at the highest resolution, map colorings are compressed significantly better by JBIG but worse by ZIP. The ZIP difference should not be surprising in that the number of dictionary entries necessary to encode runs of a single color is less than the number of dictionary entries necessary to encode runs of more than one color. The JBIG difference is due to the separate coding of each bitplane. The edge map does not hold up well when separated, especially for non-diagonal but sloped linear features.

Comparing these results with the 163 bytes necessary to code the object based representation of the geometrical image shows that Γ_α is approximately 20 for all of the reference compression methods. This means that for partitions with an average feature size much greater than 20, object based coding would outperform any of the reference methods.

Size	Color Zip	Edge Zip	Outline Zip	Color JBIG	Edge JBIG	Outline JBIG
32x32	**127**	**147**	**176**	**131**	**152**	**165**
64x64	224	294	344	239	293	232
128x128	452	604	738	340	436	300
256x256	1013	1456	1729	399	604	446
512x512	2762	3073	3987	562	982	718
1024x1024	10467	7751	12969	838	1711	1,275

Table 3
Reference Compression Methods

The coding results for the new raster neighborhood and stroke codes are shown in columns 2 and 4 of Table 4. The experimental coders are based upon Don Speck's carry-free binary arithmetic coder[10] augmented with LPS 3–4–5 adaptation[11].

Both codes perform far better than the reference methods. The stroke code is the superior of the two especially at higher resolutions. This is because the model used by the stroke code is more effective for large features. Note Γ_α is approximately 80 for the raster neighborhood code and roughly 160 for the stroke code. The associated byte count at the α resolution is boldfaced in the table. The results are remarkable considering the canonical nature of the competing object description.

Size	Raster	Stroke	Decisions	
32x32	50	46	1118	1119
64x64	86	68	4264	4286
128x128	**141**	99	16707	16762
256x256	250	**158**	66161	66283
512x512	475	253	263408	263644
1024x1024	971	429	1051095	1051573

Table 4
New Methods

Note that the number of decisions (columns 3 and 5 of the table) made by both codes is roughly equivalent to the number of pixels plus the number of full separator sites. This is the main reason for their overall success.

The experimental data given here do not give a complete picture of the relative performance of the raster neighborhood and stroke codes. The model used by the raster neighborhood code exhibits very good performance on partitions that are quite dense. This is because all local shape information is effectively used to make coding decisions. The performance of both codes on a corpus of image partitions derived from multiple sources is the subject of a future paper.

10. CONCLUSION

One of the main contributions of this paper is to show the relationship of image partitions to map-coloring. The map-coloring and three other partition representations, the edge map, the outline map, and the perimeter map are developed. These representations are used to show the equivalence of partition and image coding. Partition coding schemes are classified into object and pixel based methods. A technique for comparing the performance of the two families is developed. The technique is used to motivate the development of two new pixel based partition coding algorithms.

The new methods are based upon binary arithmetic coding and context modeling. The first method uses a raster neighborhood edge model. The second method uses chain coding and a boundary state model. Both models minimize the required number of binary decisions. The methods are compared with each other and with several reference compression schemes. The new methods exhibit excellent compression performance. Remarkably, the average feature size at which these codes should be expected to outperform object based codes is at least 40-80 pixels.

11. REFERENCES

[1] Jorn Ostermann, Euee S. Jang, Jae-Seob Shin and Tsuhan Chen, "Coding of Arbitrarily Shaped Video Objects in MPEG-4", ICIP 98, Santa Barbara, 1998.

[2] Paul Ausbeck, "Context Models for Palette Images", Proceedings of the Data Compression Conference, Snowbird, Utah, 1998, pp. 309-318.

[3] K. Appel and W. Haken, "Every planar map is four colorable", Contemporary Math. 98 (1989).

[4] Stephen R. Tate, Lossless Compression of Region Edge Maps, CS-1992-9, Department of Computer Science, Duke University, Durham, NC, 1992.

[5] Robert R. Ester, Jr. and V. Ralph Algazi, "Efficient error free chain coding of binary documents", Proc. Data Compression Conference, Snowbird, Utah, March 28, 1995, pp. 122-131.

[6] Martin J. Turner, "Entropy Reduction via Simplified Image Contourization", NASA Space and Earth Science Data Compression Workshop, Snowbird, Utah, March 27, 1992, pp. 27-42.

[7] Christophe Oddou , "Device for Encoding One Contour Side of Segmented Images, and Decoder Therefor", U.S. Patent 5,459,513, Oct. 17, 1995.

[8] PKZIP for Windows, SHAREWARE Version 2.01, PKWARE, Inc.

[9] Markus Kuhn, Version 0.9 of the JBIG–KIT, available via anonymous ftp at ftp.informatik.uni–erlangen.de/pub/doc/ISO/JBIG/jbigkit-0.8.tar.gz.

[10] Don Speck, "Local Activity Level Classification Model for Continuous–tone Coding", document N198 submitted to ISO/IEC JTC1/SC29/WG1 June 29, 1995.

[11] Glen G. Langdon, Jr. and Jorma Rissanen, "Compression of Black–White Images with Arithmetic Coding", IEEE Transactions on Communications, Vol. COM–29(6), pp. 858–867 (June 1981).