

Region Boundary Generation and Compression

Bryan J. Mealy

UCSC-CRL-02-41
December 12, 2002

Computer Engineering Department
University of California
Santa Cruz, CA 95064

Abstract

A new technique for generating and losslessly compressing region boundaries in digital images is presented. Region boundaries are generated by finding a minimal set of pixels required to differentiate each region from neighboring regions. An enhanced, pixel-based, differential chain code is used to encode the generated boundaries. We utilized run-length encoding of links and introduce the techniques of *dynamic template switching* and *extended vector templates*. These features allow accurate region boundary representation using a minimal number of links while retaining the inherent simplicity of traditional chain codes.

The results show that the boundary generation and encoding approach is able to encode all possible region boundary conditions when pixels are used as region boundaries. The results compare well to other pixel-based encoders (in bits-per-symbol) despite the fact our encoder contains extra information embedded in the chain code. The bit-encoding rates of the boundaries are relatively low compared to the number of pixels in the images over a test image set. Despite the addition of multiple features to the boundary encoder, the chain coder remains relatively simple. The technique of dynamic template switching was shown to improve the bit-encoding rate of the generated region boundaries.

Keywords: chain coding, contour coding, region boundary compression.

Contents

1	Introduction	2
2	Overview	3
2.1	Keywords and Definitions	4
3	Border Image Extraction	6
4	Region Boundary Mapping	6
4.1	Searching for Bobs	9
4.2	Next-direction Searching	9
4.3	Bixel Types	10
4.3.1	Non-standard Bixels	10
4.4	The Region Boundary Mapping Algorithm	13
4.5	Hole Region Mapping	14
5	Region Boundary Encoding	17
5.1	Encoding Phase Non-Standard Bixels	17
5.2	Image Partition Coding Example	19
5.3	Dynamic Template Switching (DTS)	19
5.4	Template Realignment and Extended Vector Templates	19
5.5	Region Image Reconstruction	22
6	Results and Comments	22
6.1	Pixel-based Boundary Encoding	25
6.2	Two-pass Image Partition Encoding	25
6.3	Other Pixel-based Encoding Schemes	26
7	Conclusion	26
A	Image Partition Coding Language Specification	27
	References	28

1 Introduction

The goal of image partition coding is to represent the partitioned image using minimal information, while allowing it to be accurately reconstructed. The general approach is to encode the boundaries of each region in order to delineate any one region from neighboring regions. Efficient representation of region boundaries reduces the number of bits required to encode them.

Boundaries lines, or contours, can be represented with polynomials of varying degree but are more often represented by chain codes [6]. The direction toward the next “entity” in the chain forms a chain link. Chain codes are classified as either *edge-based*¹ or *pixel-based*². The chains in edge-based codes are formed from the notion of boundaries between pixels. By their nature, edge-based chain codes are constrained to using a 4-connected pixel model for link directions. The chains in pixel-based chain codes are formed from pixels and can utilize an 8-connected pixel model. The functional difference between edge and pixel-based chain codes is that pixel-based codes are able to use an 8-connected pixel model for link directions while link directions in edge-based codes are constrained to using a 4-connected pixel model.

The numerous published works describing or utilizing chain codes can be classified into three categories: line coders, region boundary coders, or image partition coders. Line coders [7, 12] make up the least complex of chain coding applications since they are concerned only with encoding simple chains. Region boundary coders [8, 3, 16] build upon the techniques associated with line coders. The requirement of encoding the boundaries of regions, or closed contours, as opposed to encoding only contour lines, adds complexity to standard line coders. Image partition encoders are the most complex applications that utilize chain codes. This is mainly because of the added challenges appearing when the encoding of adjacent regions is required. Image partition coders apply chain coding methods and various techniques to locate and represent all possible region characteristics. Our discussion is focused on image partition coders since representing every region in the partition is required.

Relatively few published works deal with region boundary coding of partitioned images. Unfortunately, the common trend in these works is to mention the boundary coding methods only in passing [10]. Attention instead is focused on overall compression results, rather than to details of boundary encoding methods. The impressive compression results stated in these works possibly attenuates the need for efficient boundary representation. On the other hand, several works dealing with image partition coding become exceedingly complex [4, 18, 21]. These works are of little relevance to our discussion since developing techniques that preserve the original simplicity of chain codes is preferred. A common trend in works requiring the encoding of region boundaries is exemplified in [13]. This work assumes a bit-per-pixel value as a cost for encoding region boundaries but fails to provide details as to how region boundaries are generated or encoded.

The need to efficiently represent region boundaries prevents the most basic approach from

¹Edge-based chain codes are also known as NESW (North-East-South-West) codes [5], crack codes [9], and MAE (move along edge) codes [20].

²Pixel-based chain codes are also known as MTC (move through center) codes [20].

consideration. A simple method to represent region boundaries is to encode the boundary “entities” of each region in the partition [19]. When pixels are used as region boundaries, this encoding method requires the coding of pixels on each side of the edge that separates regions. In the case where pixel edges are used as boundaries, each boundary edge requires two traversals for proper region representation. Unfortunately, the simplicity of this approach introduces unnecessary redundancy to the image partition coder.

A straight-forward method occasionally used to encode pixel boundaries is a raster neighborhood coding [11]. These approaches are inherently pixel-based in that region boundaries are considered *black* pixels on a *white* background. The image is treated as a binary image and chain coding is not required. Several published applications use this JBIG-type approach in their implementations [2, 17]. The major drawback of these works is that the representation of certain region characteristics is either impossible or unduly complex.

2 Overview

The image partition coder presented by Ausbeck [1] is possibly the best described work regarding image partition coding. Ausbeck’s image partition coder is able to code any type of region³ without introducing unreasonable complexity to the coder. This coder, however, uses edge-based chain coding with its notion of *separators* to represent region boundaries, thus constraining the coder to using a 4-connected pixel model.

Fig. 1 shows that edge-based chain codes are not always the most efficient approach for boundary representation. Fig. 1(a) shows an image fragment with two regions indicated by shaded pixels. Fig. 1(b) and (c) show how the boundary of Fig. 1(a) is encoded using edge and pixel-based chain codes with arrows indicating the “entities” representing region boundaries. The edge-based approach of Fig. 1(b) requires eleven chain links to encode while the pixel-based approach of Fig. 1(c) requires six links.

The main purpose of this chapter is to describe a image partition coder as robust as Ausbeck’s work while using a pixel-based chain code. This approach to image partition coding is known to be more complex than image partition coders utilizing edge-based codes [5], primarily because seemingly complex region boundary characteristics are created when pixels are used to delineate regions. This chapter describes these characteristics and presents methods to simplify their encoding. An introduction to several of the topics described in this chapter appears in [14].

The image partition coder we present incorporates the simplicity and efficiency of chain coding in the context of encoding a complete image partition. The image partition coder is centered around a lossless, two-pass, pixel-based, differential chain code. The chain code retains the inherent simplicity of chain coding, while adding extended features to optimally represent region boundary characteristics. The chain code introduces *dynamic template switching*, *extended vector templates*, and *template realignment* to represent chain link statistics more accurately. The chain code also provides a solution to the classic backtracking problem associated with chain

³This coder, however, must divide single regions that contain two pixels connected by an 8-connection only into two separate regions.

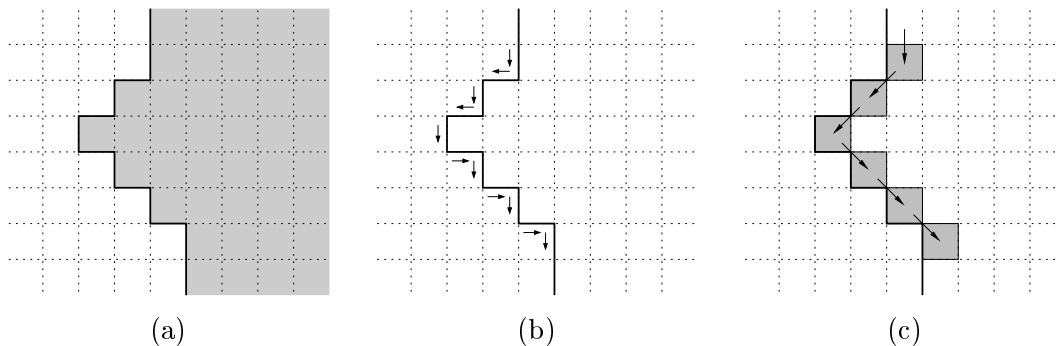


Figure 1: An example showing the advantage of pixel-based chain coding over edge-based chain coding. (a) shows a portion of an image with two regions differentiated by shaded pixels and a darkened boundary line. (b) and (c) use arrows to show the links required to encode the boundary of (a) using an edge-based and pixel-based chain code, respectively.

codes. All chain start and termination information is embedded within the chain code and relative addressing is used extensively to improve coding efficiency. The chain code's extended features are general enough for use in any chain coding application. The new chain code is well structured and is modeled with a context-free grammar in Appendix A.

Fig. 2 shows the process of image partition coding is divided into three areas: 1) border image extraction, 2) region boundary mapping, and 3) region boundary encoding. Border image extraction generates region boundaries based on the region image. Region boundary mapping locates and records boundary pixels in the border image. Region boundary encoding generates symbols used to describe the mapped region boundary pixels. Each of these areas are described in the following sections.

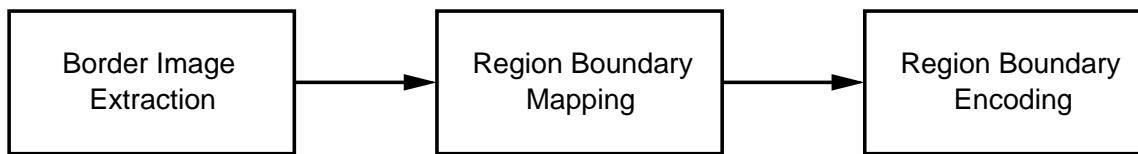


Figure 2: The three steps in image partition coding.

2.1 Keywords and Definitions

The following terms represent a short list of terms used to clarify the description of the image partition coder. A complete list of terms used in this work appears in the glossary.

region image: an image partitioned into a set of non-overlapping regions. Each region is defined by the set of pixels belonging to that region and each pixel belongs to only one region.

hole: a region that is adjacent to only one other region.

border image: an image where every pixel is either a boundary pixel or a non-boundary pixel. The set of boundary pixels that collectively delineate the regions in the partition. Boundary and non-boundary pixels are equivalent to the terms *on* and *off* or *black* and *white* pixels used by other contour encoding schemes. The boundary pixels in the border image are used to reconstruct the original region image.

boundary pixels: or *bixels*, are pixels used to delineate each region from other regions in the image.

boundary objects: or *bobs*, are sets of connected bixels in the border image. Each bixel in a bob is able to reach any other bixel in the bob by following a path of contiguous bixels.

hole boundary objects: or *hole-bobs*, are boundary objects delineated with hole regions.

backtracking: a condition associated with contour coding which causes the coder to associate multiple links with a single pixel. Fig. 3 shows the inherent differences between backtracking for pixel and edge-based chain codes. Fig. 3(a) is a region image fragment containing two regions with boundary lines indicated by darkened pixel edges. Fig. 3(b) and Fig. 3(c) use arrows to indicate required links for a pixel-based and edge-based chain code, respectively. This example demonstrates that several boundary “entities” are encoded more than once for both edge and pixel-based codes.

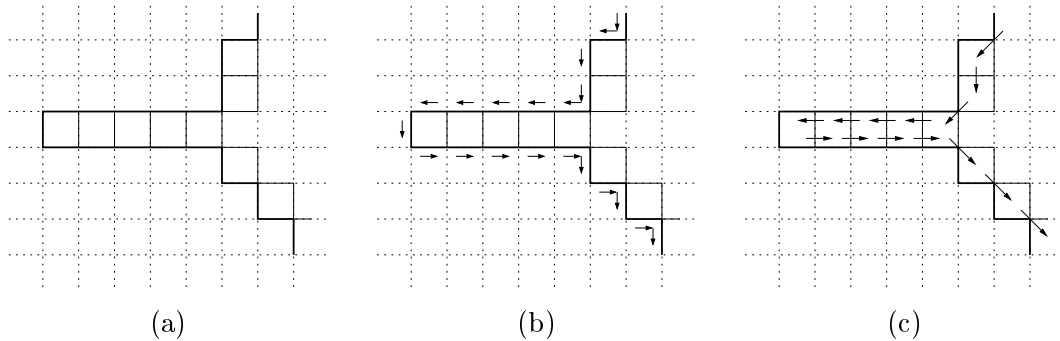


Figure 3: An example of the backtracking problem associated with chain codes. (a) shows a portion of a region image with boundary lines indicated with darkened edges. (b) and (c) show the inefficiencies inherent in encoding region boundaries with both edge and pixel-based codes, respectively.

3 Border Image Extraction

Border image extraction is used to generate a border image containing a minimal number of bixels. Moreover, bobs in the border image contain as few bixels as possible, but are able to reconstruct the regions they delineate.

Border image extraction is driven by the border image creation algorithm. This algorithm finds the set of pixels bounding each region in the region image. These boundary pixels, or bixels, delineate each region from its neighboring regions. The border image creation algorithm searches for member pixels of the current region representing the last line of pixels before entering into another region. The algorithm recursively *grows* each region until a region's boundary is discovered. Discovering a region boundary halts the recursion. Bixels are assigned only if the pixels halting the recursion are members of the region being processed. Previously discovered bixels are necessarily members of previously processed regions. Regions are processed in the order they are encountered using raster scan order. Fig. 4 shows pseudo-code for the border image creation algorithm.

Fig. 5(a) and (b) show a region image and its associated border image, respectively. Shaded pixels in the border image of Fig. 5(b) are bixels for the various regions. The two bobs in Fig. 5(b) are comprised of all the bixels in the border image. The numbers appearing in the unshaded pixel locations of Fig. 5(b) represent the first pixels encountered in each region and are used as starting points for the region growing process. The numbered pixels show the order that the algorithm encounters and grows the regions using raster scan order. We refer to this order as region-scan order.

The border image creation algorithm is applied to hole regions separately from non-hole regions. Separate consideration of hole regions exploits the fact that chains representing holes are shorter if the bixels describing hole regions are members of that region. Using an interior bixel as a starting point for region border creation forces the hole to be bounded by pixels belonging to the hole. Fig. 6 demonstrates that bounding regions with non-member bixels requires more bixels than bounding the same region with member bixels. The region image in Fig. 6(a) shows an image containing one hole region. Fig. 6(b) and Fig. 6(c) demonstrate the number of bixels required to bound the hole if the hole is bounded using inside and outside bixels, respectively. In this example, it requires four less bixels to bound the hole from inside bixels as opposed to the outside bixels.

4 Region Boundary Mapping

After the border image is extracted from the region image, each bixel in every bob in the border image is located and recorded. *Mapping* denotes the process of locating bixels and storing pertinent information associated with them. Region boundary mapping and region boundary coding are independent processes, which makes this a two-pass technique. The first pass processes bobs in the border image and constructs a list of mapped bixels. The second pass converts the list of bixels to actual code. Region boundary encoding and the advantages of using a two-pass approach are described in Section 5.

```

/*  $N_G(p)$  represents every pixel ( $D_4 = 1$ ) in the neighborhood of pixel  $p$  */
CREATE_BORDER_IMAGE(region_image,border_image)
{
  for each (pixel  $p$  in region_image) {
    if (region_image( $p$ )  $\neq$  PROCESSED); {
      EXTRACT_REGION_BIXELS( $p$ , label( $p$ ));
    }
  }
}

EXTRACT_REGION_BIXELS( $p$ , label)
{
  region_image( $p$ ) $\leftarrow$ PROCESSED;
  for each ( $p' \in N_G(p)$ ) {
    if (border_image( $p'$ )  $\neq$  PROCESSED)
    {
      if (region_image( $p'$ )  $\neq$  label) {
        border_image( $p'$ ) $\leftarrow$ BIXEL;
      }
      else {
        EXTRACT_REGION_BIXELS( $p'$ , label);
      }
    }
  }
}

```

Figure 4: The border image creation algorithm.

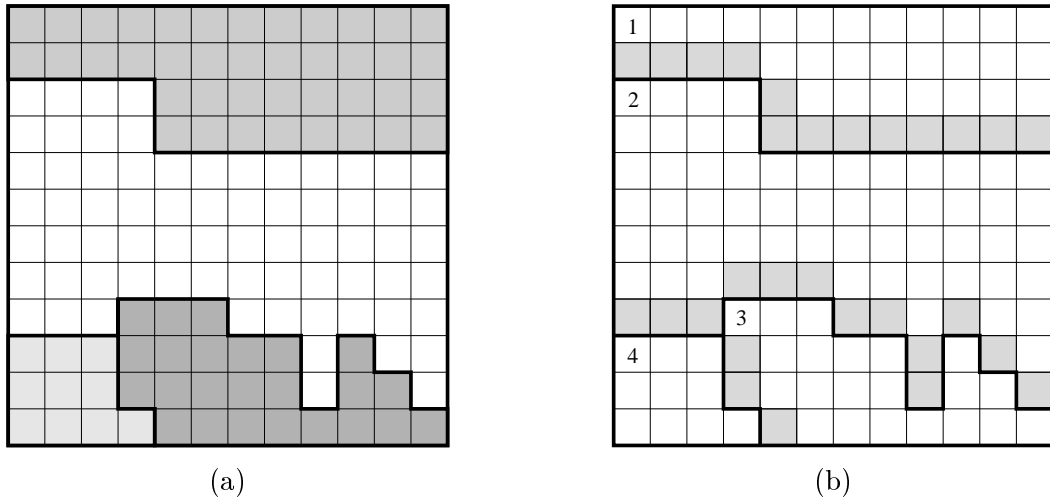


Figure 5: An example of a region image and its associated border image. (a) is a region image containing four regions and (b) shows the associated border image containing two bobs. The shaded pixels of (b) are the bixels of the regions in (a).

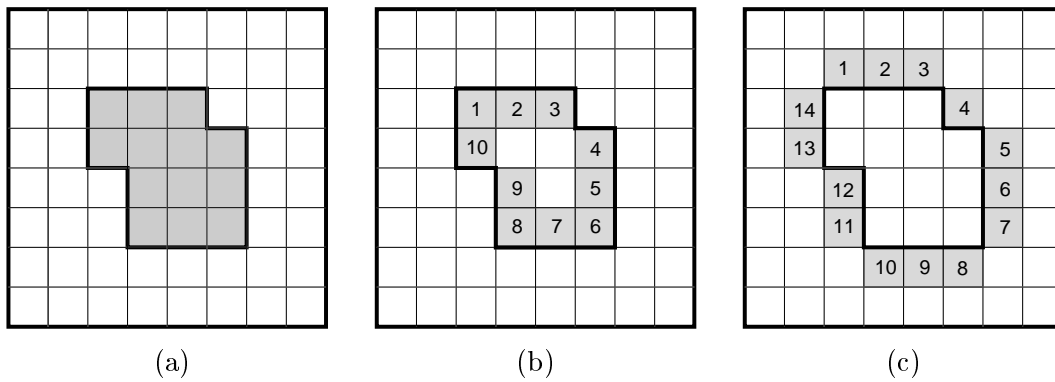


Figure 6: An example showing the benefits of using bixels belonging to the hole to represent the hole region. (a) shows a region image containing one hole. (b) and (c) shows the bixels required to represent the hole if hole region bixels and if bixels from the surrounding region are used to represent the hole region, respectively.

4.1 Searching for Bobs

Region boundary mapping begins by scanning the border image searching for bixels. The first bixel found is the starting bixel for the current bob. Bob processing continues until all bixels in the bob are processed. Border image scanning then continues searching for unprocessed bixels. Region boundary mapping is complete when every bixel in the border image is mapped.

4.2 Next-direction Searching

Two different modes are required for finding the direction of the next bixel in the chain⁴. The *search sequence* for the bixels associated with each region is established after finding the initial bixel of a region and before searching for the next bixel in that region. The search sequence is either a clockwise (CW) or counter-clockwise (CCW) search and is assigned based on the region labels of local pixels. Search sequences are used by the *next-direction finders* to locate the next bixel in the bob being processed. Next-direction finders operate by searching as *left* or *right* as possible for the CW and CCW next-direction finders, respectively. This border following technique is an extension of the border following algorithm of Morrin [15]. Morrin's approach is modified to search for boundary pixels belonging to only one region at a time.

Fig. 7 shows two types of next-direction finders. Fig. 7(a) is a region image with three regions and Fig. 7(b) is its associated border image. Bixels **A** and **B** in Fig. 7(b) are the first bixels processed in their respective regions. The next-direction finder searches as *left* as possible for new bixels using a CW search sequence at **A**. A new search sequence is established after bixels in the first region are processed and before processing the second region. The next-direction finder searches as *right* as possible using a CCW search sequence at **B**.

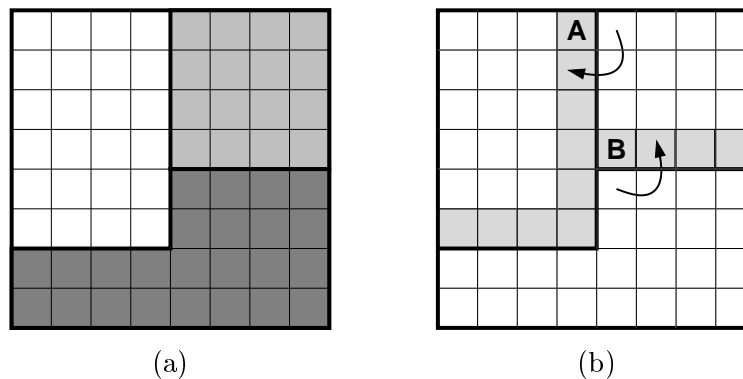


Figure 7: An example showing the two next-direction finders. (a) shows a region image with three regions and (b) is the associated border image. The shaded pixels in (b) are the bixels associated with the region image. Bixel **A** employs a clockwise search sequence for next-direction finding in its associated region while **B** uses a counter-clockwise search sequence.

⁴Searching for the next “entity” in the chain is often referred to as *border following*.

The next-direction finder's search sequence is based on the location of the previously discovered bixels. Since starting bixels inherently have no previous direction, the initial search sequence is established based on the region labels of pixels neighboring the starting bixel. Fig. 8 shows how the search sequence changes based on the orientation of the previous bixel in the chain. Fig. 8(a) shows a portion of a region image with letters indicating different regions and shaded squares representing bixels. Fig. 8(b) shows the search sequencing based on a previous direction of South. Fig. 8(c) shows the search sequencing based on a previous direction of Southeast. The numbers near the small arrow indicate the order used to examine local pixels in the search bixels. This example uses a clockwise search sequence in its next-direction finder.

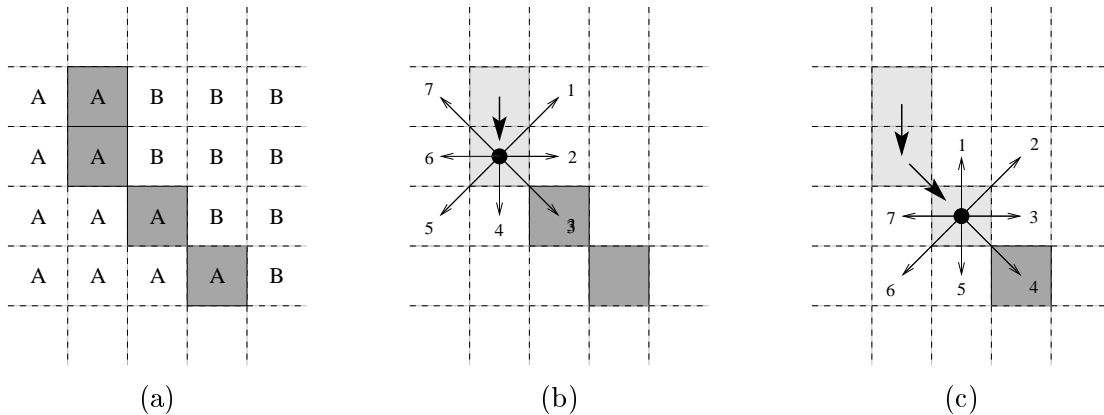


Figure 8: An example showing the next-direction finding using a CW search sequence. (a) shows a portion of a region image with labels **A** and **B** differentiating the regions and shaded squared representing the bixels. (b) and (c) show the search sequencing based on a previous directions of South and Southeast, respectively. The shading of bixels in (b) and (c) lighten as bixels are processed. The numbers near arrow points represent the order that directions are searched when to locating the next bixel in the bob.

4.3 Bixel Types

Each bixel in the border image is classified as either a *standard* or *non-standard* bixel. Bixel classification is determined during region boundary mapping. Standard bixels form links in the bixel chain by providing directional information to locate the next bixel in the chain. Non-standard bixels contain extra information in addition to directional information. A standard bixel contains only two bixels in its local neighborhood. One bixel is the previously discovered bixel and the other is the next link in the bixel chain. Non-standard bixels are used to represent more complex characteristics present in the border image.

4.3.1 Non-standard Bixels

Non-standard bixels are discovered in both the region boundary mapping and region boundary encoding phase. The non-standard bixels found during region boundary mapping are re-

quired to represent the boundaries of regions. Non-standard bixels appearing during region boundary encoding are *assigned* based on certain characteristics of the bixel chain. These assignments are made to increase coding efficiency as described in Section 5. The non-standard bixels associated with region boundary mapping are described below.

Finish Bixels: bixels indicating no undiscovered bixels are connected to the current bixel. There are two types of finish bixels: *temp-finish* bixels and *final-finish* bixels. Temp-finish bixels indicate a dead-end in bixel processing. In this case, region boundary mapping *restarts* at some other non-contiguous bixel in the bob. Final-finish bixels indicate the end of processing for bixels in the current region. Examples of temp-finish and final-finish bixels are shown in Fig. 9. In this figure, boundary mapping progresses from bixel \mathbf{x} to \mathbf{y} and onto \mathbf{A} . Bixel \mathbf{A} is a temp-finish bixel because there are no other contiguous and non-processed bixels belonging to that region after \mathbf{A} is mapped. Once \mathbf{A} is processed, boundary mapping continues at \mathbf{B} . Bixel \mathbf{C} is a final-finish bixel because all other bixels in the current region have been mapped. Finish bixels contain no directional information.

Restart Bixels: bixels where the region boundary mapping *restarts* after a temp-finish bixel is encountered. There is necessarily one restart bixel for every temp-finish bixel. Fig. 9 shows an example of a restart bixel. Bixel \mathbf{B} is a restart bixel and is where processing for the region continues once the temp-finish bixel \mathbf{A} is processed. The restart bixel \mathbf{B} is discovered by and associated with the bixel \mathbf{x} . The next link in the chain after \mathbf{x} is \mathbf{y} and the chain continues on to \mathbf{A} . The restart-temp-finish bixel pair enables the chain code to encode boundaries without resorting to backtracking. Extra information associated with restart bixels is the direction from the finding bixel to the restart bixel.

Newstart Bixels: indicate a bixel is found that is not a member of the current region being mapped. Newstart bixels indicate the discovery of a new region. Processing of this new region begins after region boundary mapping for the current region is complete. Fig. 10 shows an example of a newstart bixel. In this figure, bixel \mathbf{x} discovers bixel \mathbf{A} in its neighborhood. Bixel \mathbf{A} becomes the newstart bixel since its region label is different from the region currently being processed. Region boundary mapping continues with \mathbf{y} being the next bixel discovered after \mathbf{x} . Extra information associated with newstart bixels is the direction from the finding bixel to the newstart bixel.

Seed Bixels: bixels used to indicate pixels required to start the decoders region growing process in areas that otherwise would be grown improperly. The decoder uses seed bixels to continue growing regions that are *cut-off* by the bixels of that region. Fig. 11 shows an example of two instances requiring seed bixels to ensure

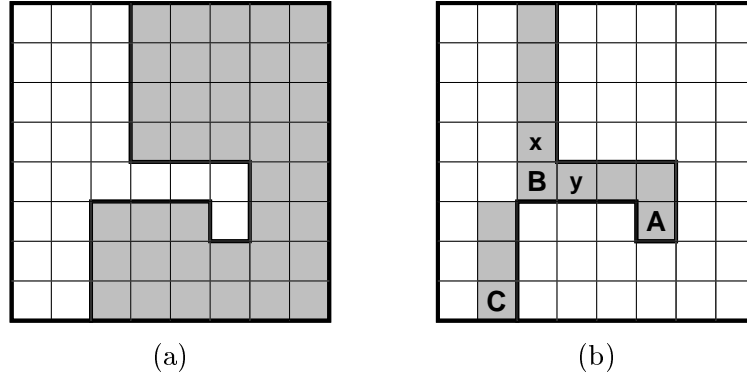


Figure 9: An example showing *finish* bixels and a *restart* bixel. (a) shows a region image containing two regions, (b) shows the associated border image with examples of a temp-finish bixel **A**, a restart bixel **B**, and a final-finish bixel **C**. Region boundary mapping proceeds from bixel **x** to **y**, and onto temp-finish bixel **A**. Region boundary mapping restarts at bixel **B** and eventually terminates at final-finish bixel **C**.

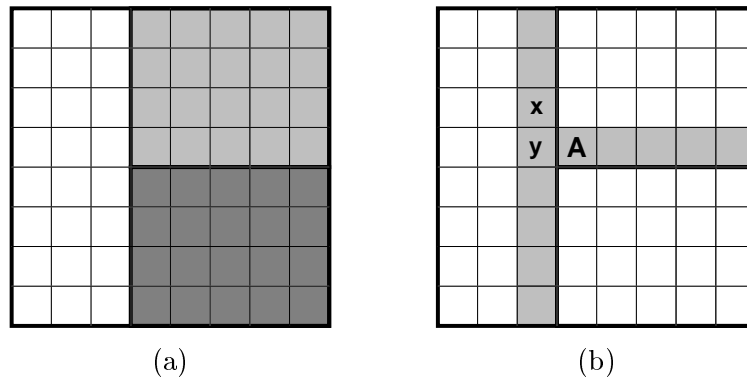


Figure 10: An example showing a *newstart* bixel. (a) shows a region image containing three regions and (b) shows the associated border image with a newstart bixel **A**. The newstart bixel **A** is associated with the finding bixel **x**. Bixel mapping continues with the mapping of bixel **y**.

proper region image reconstruction. In this figure, bixel \mathbf{x} is the first *pixel* discovered in the region. This pixel is marked by the seed bixel \mathbf{A} and region growing starts for the pixels of that region beginning at \mathbf{x} . In the second image region, the growing process ends at the pixel labeled \mathbf{y} . Seed bixel \mathbf{B} marks pixel \mathbf{z} as a seed to restart the growing process in the remaining portion of the region. Extra information associated with seed bixels is the direction from the seed bixel to the pixel where the seeding occurs.

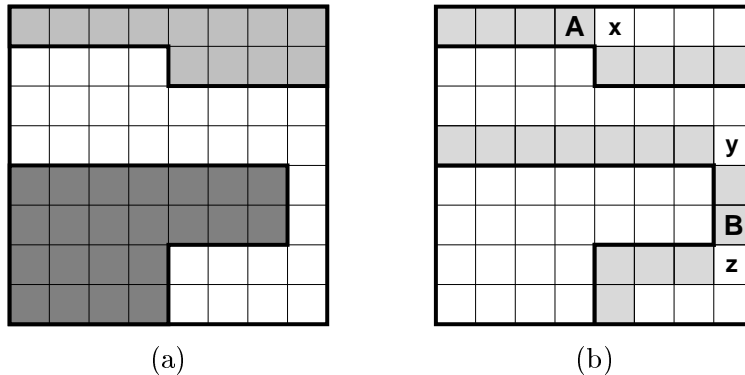


Figure 11: An example showing two situations where *seed* bixels are required. (a) is a region image containing three regions and (b) shows the associated border image with seed bixels labeled \mathbf{A} and \mathbf{B} . Seed bixels \mathbf{A} and \mathbf{B} are used to indicate that pixels \mathbf{x} and \mathbf{y} continue the region growing process in the decoder for the associated regions.

Hole Bixels: A *hole* bixel indicates the current bixel is used as an anchor for a hole region. Hole bixels have no extra information associated with them. Hole processing is described in Section 4.5.

4.4 The Region Boundary Mapping Algorithm

As each bixel is mapped, a bixel object is created and added to a list of previously mapped bixels. The bixel object contains information used by the region boundary encoding phase to encode the bob as described in Section 5. Bixel objects contain queues to store occurrences of non-standard bixels associated with bixels. Each bob in the border image is represented by a list of bixels.

A flowchart for the region boundary mapping algorithm is shown in Fig. 12. The algorithm starts in the upper-left corner of the border image and searches for bixels using raster scan order. The first bixel found becomes the starting bixel for that bob. For each new region encountered, a search sequence is established for next-direction finding. The search sequence is established after starting, restarts, and newstart bixels. Every bixel encountered is examined to determine if extra properties need to be assigned to it. The bixel is first examined to determine if it is a hole or seed bixel, respectively. Next, the local neighborhood is examined to determine

if newstart bixels are present; newstarts bixels are added to the newstart queue. Each bixel in the local neighborhood that is neither the previously processed bixel or the next bixel in the chain is a potential restart bixel; restart bixels are added to the restart stack. Processing continues until no more bixels are found contiguous to the current bixel. The current bixel is then either a temp-finish or final-finish bixel. When a temp-finish bixel is encountered, potential restart bixels are popped off the restart stack in a search for valid restart bixels; valid restart bixels are bixels on the restart stack that are not previously processed. If a temp-finish bixel is encountered, a valid restart must exist. When a final-finish bixel is encountered, the status of the newstart queue is examined. Bob processing continues if the newstart queue is non-empty. Bob processing begins from the newstart bixel removed from the queue when the newstart queue is non-empty. An empty newstart queue indicates the processing of the current bob is complete. In this case, the raster scan order search for bixels in the border image is resumed. If there are other non-processed bobs in the image, a new bixel list is created and processing starts on the new bob.

4.5 Hole Region Mapping

The number of bits required to encode hole-bobs is reduced by using relative starting locations [5]. The probability of saving bits is increased when non-hole bixels are present in the image. Since hole regions are by definition contiguous to only one other region, non-hole bobs are not connected to other hole-bobs. The general solution to encoding a hole would be to encode the starting point of each hole as is done with non-hole bobs. This would require $\log_2[rows * cols]$ bits to encode each hole-bob starting bixel. The number of bits is reduced by encoding hole-bobs after all non-hole-bobs are encoded which allows for the use of starting locations relative to other non-hole bixels.

As initially described in section 4.3.1, hole-bob starting locations are anchored by hole bixels in non-hole bobs. Hole-bobs are anchored after all non-hole-bobs are generated to increase the efficiency of locating hole starting bixels. Fig. 13 shows an example of how hole-bobs are anchored and relative addressing is applied. Fig. 13(a) is a region image with one hole and three non-hole regions. The bob with the three non-hole regions is shown in Fig. 13(b). The hole-bob associated with the hole of Fig. 13(a) is shown in Fig. 13(c). The hole-bob can be anchored by one of the bixels **A**, **B** or **C** from the non-hole-bob, as shown in Fig. 13(d). The dotted lines of Fig. 13(d) represent the linear distances measured in pixels associated with **A**, **B**, and **C** relative to hole-bob starting bixel **r**. The spans of pixels from **A** to **x**, **B** to **y**, and **C** to **z** are the possible locations of the hole starting bixels. The non-hole-bob bixel associated with the shortest of the three spans, is assigned as the hole bixel.

The number of bits required to store the distance from the non-hole bixel to the start location of the hole bixel is calculated as shown in Eqn. 1. The distance in pixels to the hole starting pixel is encoded using the number of bits specified by Eqn. 1. Fig. 13 is used to demonstrate how the encoder applies Eqn. 1 this calculation. The encoder knows bixel **r** in Fig. 13(b) is the start bixel for the hole. The direction to the start bixel of the hole is also known. The pixel labeled **x** is the maximum distance away the hole starting bixel *could* be located in the southern

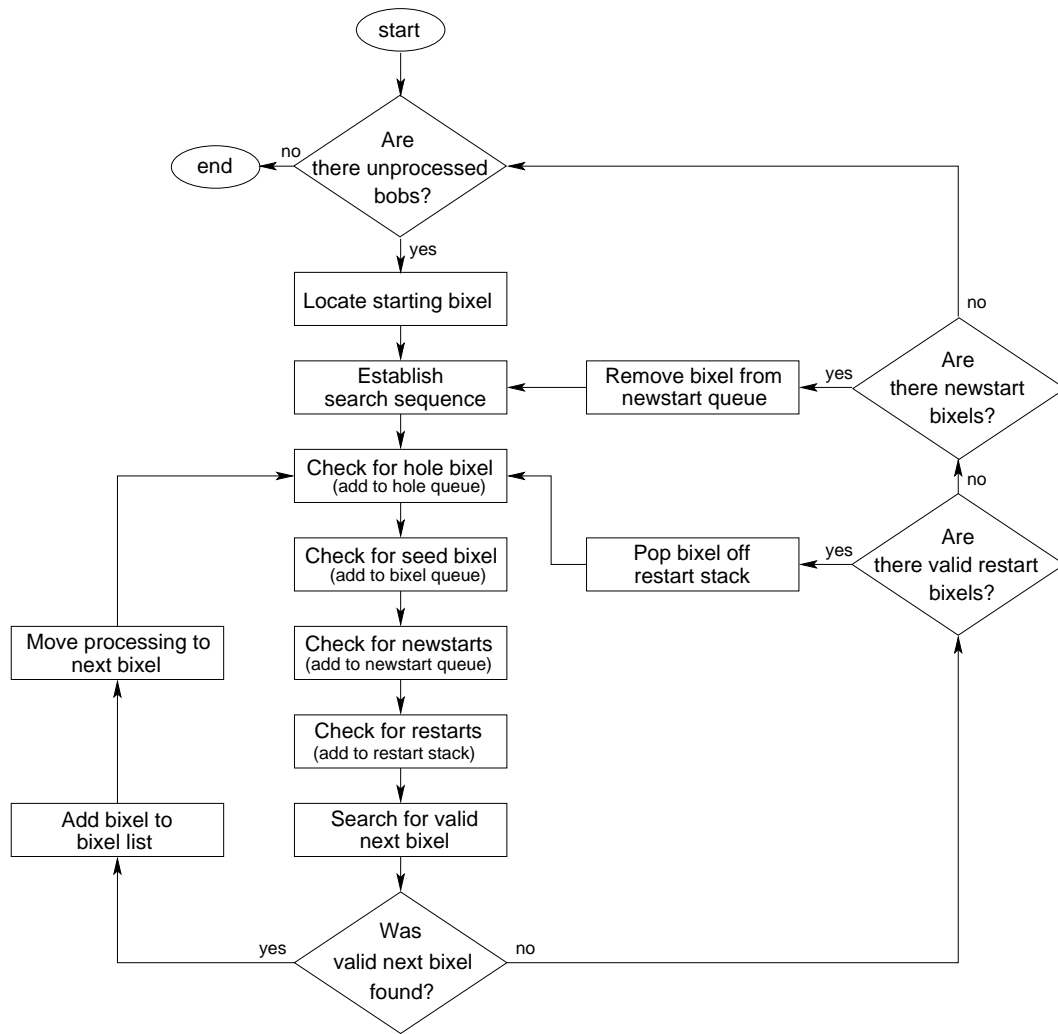


Figure 12: Flowchart for the region boundary mapping algorithm.

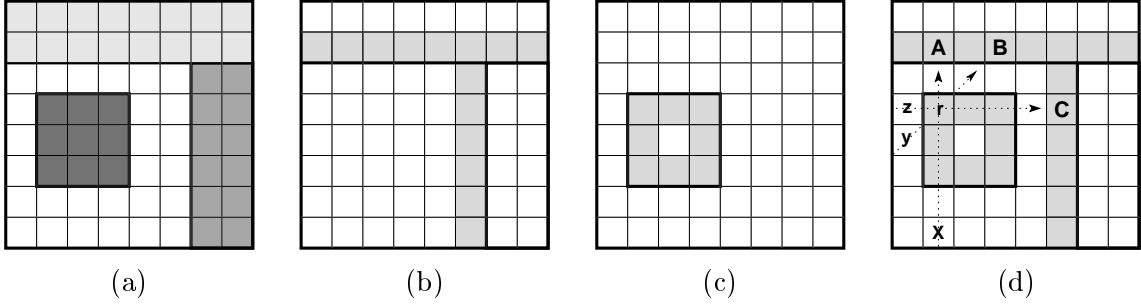


Figure 13: An example showing how hole-bobs are anchored and relative addressing is applied. (a) shows a region image containing four regions including one hole region. (b) and (c) show the bobs associated with the non-hole and hole regions, respectively. (d) shows the associated border image with the potential hole bixels marked **A**, **B**, and **C**. Pixels labeled **x**, **y**, and **z** are the most distant possible locations of hole start bixels from potential hole bixels **A**, **B**, and **C**, respectively. The hole bixel is chosen based on the minimum span in pixels from **A** to **x**, **B** to **y**, and **C** to **z**.

direction. The value of D_h in Eqn. 1 is represented by the pixel distances from **A** to **x**, **B** to **y**, and **C** to **z** for the individual calculations. The distance from bixel **A** to pixel **x** is greater than the distance from bixel **C** to pixel **z**. The distance from **B** to **y** is the least of these distances. Although the hole-bobs can be anchored by bixels **A**, **B**, or **C**, the optimal to choice in this example is **C** due to the shorter span of pixels in which the hole starting pixel could be located.

$$B_h = \log_2[D_h] \quad (1)$$

where

$$\begin{aligned} D_h &= \text{Distance (in pixels) to nearest non-hole bixel or image edge} \\ B_h &= \text{Number of bits used to store } D_h \end{aligned}$$

The example in Fig. 13 represents the general case for anchoring hole-bobs. In this example, the linear span of pixels are delineated by the edges of the image. These distances can also be delimited by the bixels of other regions. The encoder searches for the shortest distance from a non-hole bixels to another non-hole bixel or the edge of the image. The search for the shortest distance can use any of the four linear directions⁵ expanding outward from a hole-bob starting bixel.

The encoder stores all non-hole bob information prior to storing hole bob information. Hole bob information is stored in the order the hole bixels are encountered during region boundary mapping. Hole-bob representation is identical to the previously described representation of non-hole bobs.

⁵The four directions result from linear pairings of the eight directions in the 8-connected pixel model.

5 Region Boundary Encoding

Bixels are ready for encoding after every bixel in the border image is mapped. Every mapped bixel is designated as either a standard or non-standard bixel. At this point, an opportunity exists to change the information associated with the bixels. Bixels previously designated as standard may become non-standard by assigning them extra properties. Bixels previously classified as non-standard can have even more properties associated with them.

Region boundary mapping creates lists of bixels that represent regions in the partition. During region encoding, it is possible to examine these lists to extract *runs* of bixels. This enables the chain coder to run-length encode (RLE) bixels by assigning extra properties to bixels. The next section describe the conditions where RLE is applied.

5.1 Encoding Phase Non-Standard Bixels

Below are the two non-standard bixels assigned during region boundary encoding. Bixels are not limited to the number of non-standard conditions associated with them.

Finish-Run Bixels: indicate the previous direction encoded continues until the bixel chain either collides with a previously encoded bixel or until an edge of the image is reached. This allows the encoder to avoid explicitly coding the remaining bixels in that portion of the chain. Since bixels in the finish-run are implicitly encoded as part of the run, the bixels in the finish-run must not contain non-standard bixels discovered during region boundary mapping. Fig. 14 shows an example of two types of finish-runs. In this figure, the encoder notes that the direction from \mathbf{x} to \mathbf{A} is the same as the remaining bixels up to final-finish bixel \mathbf{y} . Bixel \mathbf{A} is marked as a finish-run bixel. A similar case exists for finish-run bixel \mathbf{B} . The direction from \mathbf{r} to \mathbf{B} is the same as every next direction until the final-finish bixel \mathbf{s} . Finish-run bixels use the direction of the previous link as the direction of the run and have no extra information associated with them.

Direction-Run Bixels: indicate the previous direction encoded continues for a designated number of bixels. An example of direction run bixels are shown in Fig. 15 shows an example of direction run bixels. In this figure, the direction from \mathbf{x} to \mathbf{A} is repeated until \mathbf{y} is encountered. The encoder marks bixel \mathbf{A} as a direction-run bixel and encodes the length of the run to \mathbf{y} . Normal processing proceeds from \mathbf{y} . Direction runs allow the encoder to avoid explicit coding of the bixels between \mathbf{A} and \mathbf{y} . Since bixels in the finish-run are implicitly encoded as part of the run, the bixels in the finish-run must not contain non-standard bixels discovered during region boundary mapping. Direction-run bixels use the direction of the previous link for the run direction and require no other information for encoding. The run-lengths are preset in the model and are encoded as either *short* or *long* runs.

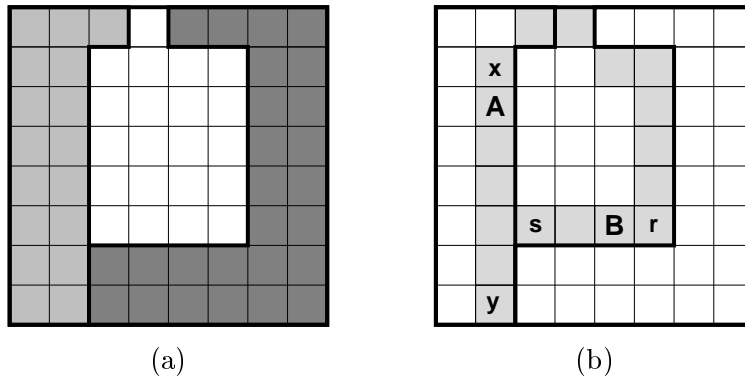


Figure 14: An example showing the two types *finish-run* bixels. (a) shows a region image containing three regions and (b) shows the associated border image with the two types of finish-run bixels labeled **A** and **B**. The direction from x to **A** is the same direction for all bixels up to the final-finish bixel y . Similarly, the direction from r to **B** is the same direction for all bixels up to the final-finish bixel s . Bixels between **A** and y and between **B** and s do not require explicit coding.

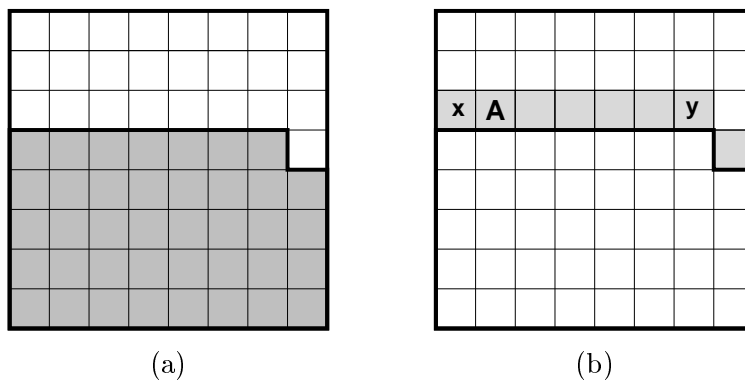


Figure 15: An example of a *direction-run* bixel. (a) shows a region image containing two regions and (b) shows the associated border image with a direction-run bixel labeled **A**. The direction from x to **A** is the same as every next-direction to the y . Bixels between **A** and y do not require explicit coding.

5.2 Image Partition Coding Example

The three-step process of image partition coding are demonstrated in Fig. 16. The left-most image shows an image partition containing two regions. The middle image shows the border image associated with the original partition in the left-most image with bixels indicated with shading. The bixel locations contain numbers to show the order of processing and arrows to indicate the next direction traversed in the chain. Three types of non-standard bixels appear in the middle image and are labeled accordingly. The right-most portion of Fig. 16 shows the bixel list generated from the border image. The numbers in the right-most image correspond to the numbers listed in the border image. The shaded bixels in the middle image likewise correspond the shaded squares in the right-most image. The arrows from the middle image are represented by direction indicators appearing in the shaded squares in the right-most image. Non-standard bixels in the right-most image are represented with the extra information in the boxes connected to the right-side of the right-most image.

The image partition coding example of Fig. 16 indicates how the backtracking problem is avoided. Bixel mapping temporarily stops at the temp-finish bixel in middle image of Fig. 16. Bixel mapping continues at the restart bixel thus avoiding backtracking through bixels label 4 and 5.

5.3 Dynamic Template Switching (DTS)

Although chain codes historically only use one link direction template, characteristics of the chain to be encoded are tracked more closely if unused template vectors are not considered. The chain coder uses one of four link vector templates to encode each region boundary. The template that best matches the characteristics of the region boundary to be encoded is chosen. Directions for the next bixel in the chain are monitored during region boundary mapping allowing the encoder to choose the template that best matches the next-direction characteristics of region boundaries. Template choice is provided as side information by the encoder prior to encoding each region boundary. Fig. 17(a) shows the standard chain coding template while Fig. 17(b)-(e) show the four templates available for region boundary encoding. The template of Fig. 17(a) is used in special situations as described in Section 5.4.

5.4 Template Realignment and Extended Vector Templates

Fig. 17 demonstrates a unique feature of the region boundary encoder. Differential chain coding is a common method of removing redundancy from the directions to the next link in the chain. The chain coder uses a differential approach combined with templates containing extended South direction vectors. The chain coder becomes differential by applying a novel technique we refer to as *template realignment* (TR).

During processing of the bixel lists, the current template is realigned so that the South direction is oriented to the direction of the previously encoded link. Using the template of Fig. 17(a), and not allowing for backtracking, would render this approach equivalent to standard differential coding. The new approach, however, differs from differential coding because the

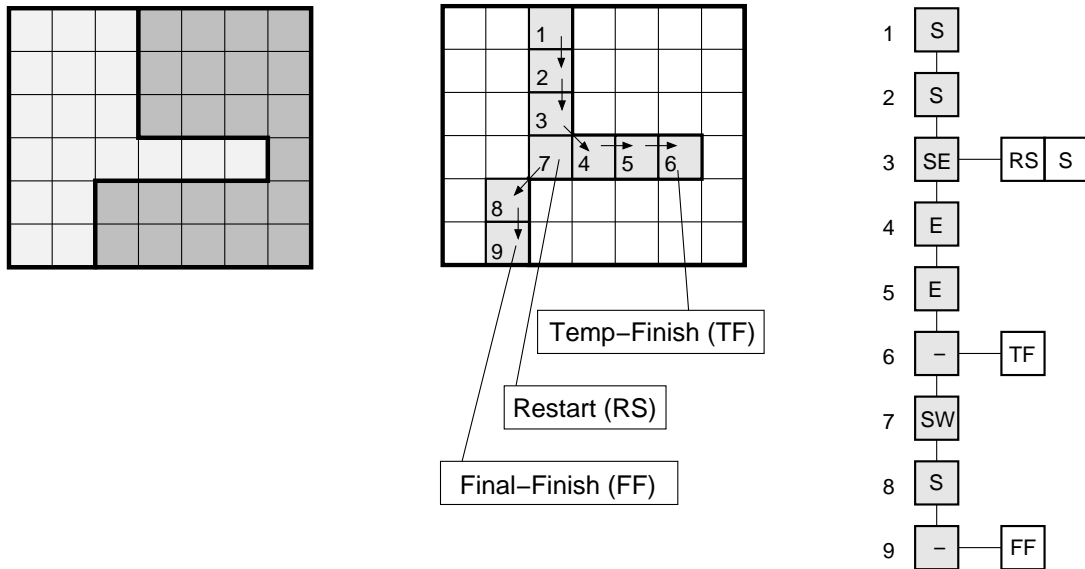


Figure 16: A detailed example of image partition encoding. The left-most image shows the regions in a partitioned image. The middle image shows the resulting bixels (shaded squares) after the region boundary extraction algorithm is applied. The arrows represent the next directions traversed in the chain and the numbers indicate the order in which bixels are processed. Three types of non-standard bixels are labeled in the middle image. The right-most diagram shows the resulting bixel list. The numbers and directions in the right-most image correspond to the numbers and arrows in middle image, respectively. Non-standard bixels are indicated with the extra information attached to the right of the bixel.

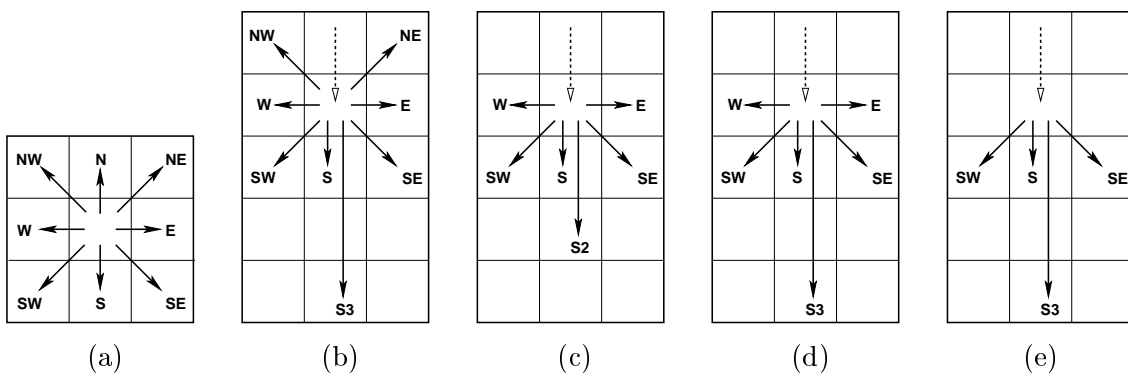


Figure 17: An example showing a set of link vector templates. (a) is the standard chain coding template used to describe directions of the next links in the chain. The other templates serve the same process yet are made functionally differential by *template realignment*. The dotted vertical line in the middle top square of each template for (b)-(e) represents the previous direction in the chain.

extended South direction vectors can be used. The extended South direction vector found in the extended vector templates (EVTs) can then be used to take advantage of the skew found in the probability distribution of the next link directions⁶. Template realignment combined with extended vector templates allow the encoding of multiple bixels in *any* direction using a single link, while adding only one additional vector to the link vector template. The realigned template direction, d_r , is calculated using Eqn. 2. In Eqn. 2, d_{i-1} and d_i are the true previous and next directions, respectively. d_S is the numerical value associated with the South direction and N_d is the number of directions in the pixel connectivity model.

$$d_r = (d_{i-1} - d_i + d_S + N_d) \bmod N_d \quad (2)$$

Fig. 18 demonstrates how templates are realigned after a single link is encoded. In Fig. 18(a), the template is aligned with to the previous direction of South. The next-link direction in the chain is Southeast. Fig. 18(b) shows the new template alignment after the Southeast direction is encoded. Fig. 18 shows how the extended South direction vectors are applied in any direction.

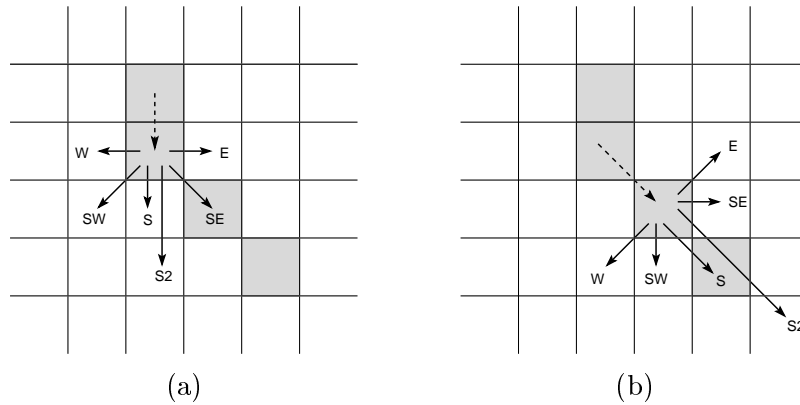


Figure 18: An example of template realignment after encoding a single link. (a) shows how the template is aligned after the previous link direction of South is encoded. (b) shows how the template is realigned after the direction of Southeast is encoded.

Fig. 19 demonstrates how a standard chain coder and a chain coder using template realignment encode a chain. In Fig. 19(a), a sequence of bixels is shown with arrows representing template direction vectors to the next link in the chain. Fig. 19(b) and Fig. 19(c) show the directions the standard template of Fig. 17(a) and the extended vector template of Fig. 17(c) use to encoded the bixels of Fig. 19(a). After each link is encoded, the South direction is realigned with the previous directions traversed. When the extended vector template is used, chain links labeled \mathbf{x} do not require coding.

The bixel sequence in Fig. 19(a) could be encoded using a standard chain code, a standard differential chain code, or the template realigned codes described above. The entropy associated

⁶Skew in the probability distribution exists because next-link directions are correlated to previous link directions.

with the bixels in Fig. 19(a) is 2.34, 1.79, and 2.27 with respect to these three techniques. Despite the higher entropy associated with the new technique when compared to the the standard differential technique, the new technique encodes the sequence with less total bits. The total bits required by the standard differential coder is 32.3 bits compared to 31.8 bits for the new technique.

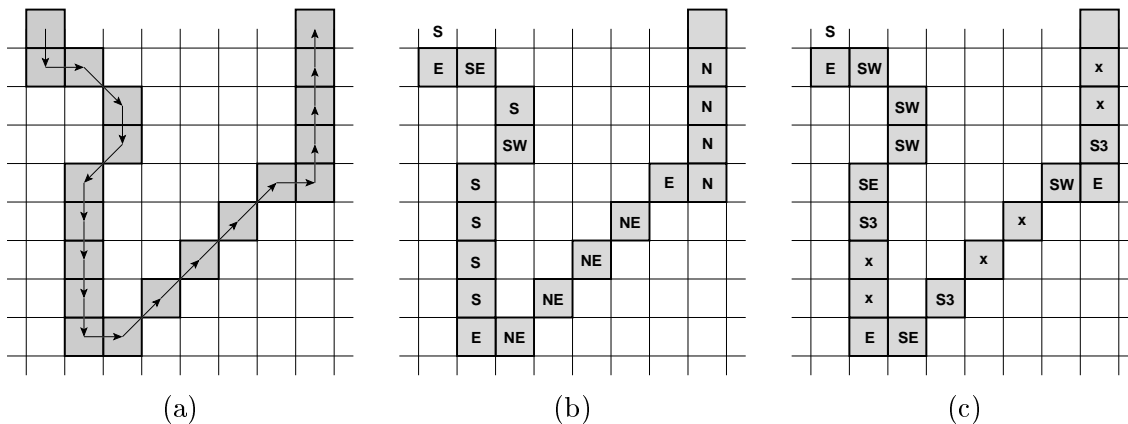


Figure 19: An example showing a chain encoding sequence using an extended vector template and template realignment. (a) is a sequence of bixels with vectors indicating the direction of the next bixel in the chain. (b) shows the standard chain code direction assignments associated with the vectors in (a) and (c) shows the direction assignments using template realignment with the extended template of Fig. 17(b). Bixels labeled **x** are not explicitly encoded in the chain of (c).

5.5 Region Image Reconstruction

Reconstruction of the region image is a two-step process. The first step decodes the region boundaries and reconstructs the original border image. The second step generates the original region image from the decoded border image. Region image reconstruction is the reversal of the encoding process minus the region boundary mapping algorithm described in Section 4. Decoding is therefore faster than encoding which renders the codec asymmetrical. The output of the decoding process is the original region image.

6 Results and Comments

The image partition encodes region boundaries using an adaptive binary arithmetic coder. Each symbol is encoded via a sequence of binary decisions structured as a binary tree. Counts of previous decisions are stored for each tree node and are used as decision probabilities.

Table 1 shows the breakdown of image partition encoding entities for the lena, tripod, and peppers test images. The fifth column in Table 1 is the total number of non-standard bixels. The terms N_{ns} and N_{rs} are the number of newstart and restart bixels, respectively. The terms

N_h and N_{sd} list the number of hole and seed bixels, respectively. The terms N_{fn} , N_{fr} , and N_{dr} are the number of finish, finish-run, and direction-run bixels, respectively.

Table 2 shows the number and type of templates used for the lena, tripod, and peppers test images. Templates B-E correspond to the templates shown in Fig. 17(b)-(e) respectively. The template of Fig. 17(a) is used for the first bixel in a bob and for directions required by non-standard bixels only, and therefore is not listed in this table.

image	N_{regs}	N_{bobs}	N_{bix}	N_{NS}	N_h	N_{ns}	N_{rs}	N_{sd}	N_{fn}	N_{fr}	N_{dr}
lena	57	4	4551	175	1	51	38	11	54	0	20
tripod	41	1	3884	80	2	37	10	7	20	3	1
peppers	74	3	4625	175	4	66	35	15	47	2	6

Table 1: Bixel content and description for three test images.

image	regions	bobs	bixels	templt B	templt C	templt D	templt E
lena	57	4	4551	27	4	11	14
tripod	41	1	3884	21	3	4	12
peppers	74	3	4625	30	10	6	27

Table 2: Template selection breakdown for the three test images.

Table 3 shows the results of applying the image partition encoder to the test image set. The second column lists *normalized image size* (NIS) for each image. The NIS is value calculated by dividing the number of pixels in the image by 65,536, the number of pixels in a 256x256 image. The terms N_r , N_{bx} , N_b , and N_h list the number of regions, bixels, bobs, and holes for each image. The column labeled “base” represents the baseline coder which is differential, but does not apply extended vector templates (EVT), dynamic template switching (DTS), or run-length encoding (RLE). The next four columns show results from varied applications of these three techniques. The first row in Table 3 list the techniques applied for each encoding experiment. Bits-per-symbol (bps) is used to measure results in these five experiments. The final column of Table 3 lists the bits-per-pixel (bpp) value required to encode the region boundaries. This result is based upon the experiment that applied only DTS (column 11) in the image partition encoder.

The experiment that added only EVT to the encoder (Table 3, column 8) causes decreased compression performance. Smoother boundaries would render this technique more effective. A similar general degradation of results are seen when RLE is added to the DTS and EVT experiment (Table 3, column 10). Once again, the efficacy of RLE techniques is increased in images containing high quantities of straight edges. The most effective of the three techniques is DTS since overall results improved when only DTS was applied. This result provides the new “baseline” image partition encoder for the test image set. The column that lists average results for only DTS represents a 8.9% improvement in compression over the average baseline result.

						base	EVT	DTS EVT	RLE DTS EVT	DTS	
image	NIS	N_r	N_{bx}	N_b	N_h	bps	bps	bps	bps	bps	bpp
lena	1.00	57	4551	4	1	2.22	2.29	2.12	2.12	1.99	0.138
tripod	1.00	41	3384	1	2	2.26	2.25	2.16	2.16	2.07	0.107
peppers	1.00	74	4625	3	4	2.28	2.37	2.17	2.15	2.03	0.143
air	1.00	53	3040	9	3	2.37	2.44	2.23	2.25	2.12	0.098
boat	1.00	68	4409	6	6	2.45	2.51	2.32	2.32	2.19	0.147
fern	2.93	68	6891	8	4	2.33	2.40	2.28	2.29	2.21	0.081
big_sur1	3.46	70	8364	2	6	2.32	2.39	2.27	2.27	2.17	0.080
ucsc3flr	3.46	125	14085	5	6	1.75	1.84	1.70	1.71	1.54	0.096
ucsc_bd1	3.46	42	5367	5	5	1.70	1.78	1.67	1.70	1.57	0.038
ucsc_sc1	3.46	128	13078	11	9	2.05	2.15	2.04	2.04	1.89	0.109
ucsc_tr1	3.46	45	5326	5	1	1.99	2.09	1.93	1.95	1.77	0.042
sf_sky	3.62	142	14325	9	6	2.06	2.15	2.03	2.03	1.86	0.113
big_sur2	3.90	82	9468	5	23	2.13	2.17	2.14	2.14	2.03	0.075
big_sur3	3.90	99	9465	3	22	2.42	2.50	2.41	2.41	2.32	0.086
balloon	6.33	92	6863	4	3	1.78	1.86	1.64	1.69	1.47	0.035
girl	6.33	265	25137	19	11	2.13	2.12	2.02	2.03	1.89	0.115
AVG	3.08	90	8648	6.2	7.0	2.14	2.21	2.07	2.08	1.95	0.094

Table 3: Information and results for boundary encoding of the test images.

6.1 Pixel-based Boundary Encoding

The image partition coder is designed for the general case which enables it to encode all possible region characteristics associated with partitioned images. Unfortunately, accounting for all possible conditions is not always optimal. In particular, there are several region characteristics that degrade coder performance.

The encoding of non-standard bixels degrades coder performance. If the non-standard bixel count is minimized, coder performance improves. Placing restrictions on region characteristics is one method of reducing the non-standard pixel count. Constraining regions to be at least two pixels wide eliminates the need to assign seed bixels. Regions wider than two pixels combined with boundary smoothing, as discussed below, would remove the need to assign newstart bixels. These constraints similarly would reduce the need to assigned finish bixels. Collectively, adding constraints to regions improves results, but likewise compromises the generality of the encoder.

An interesting boundary characteristic that the encoder handles is shown in Fig. 20. Fig. 20(a) shows a region image with five regions. All of the lower regions are one pixel wide. Fig. 20(b) shows the corresponding border image. Every pixel in the lower portion of the border image is marked as a bixel. The encoder is able to handle this boundary case, but does so at the cost of efficiency. This example demonstrates an extreme case but a similar "bunching" of bixels does appear in border images at a smaller scale. This effect is removed when pixel-wide regions are not present in the region image.

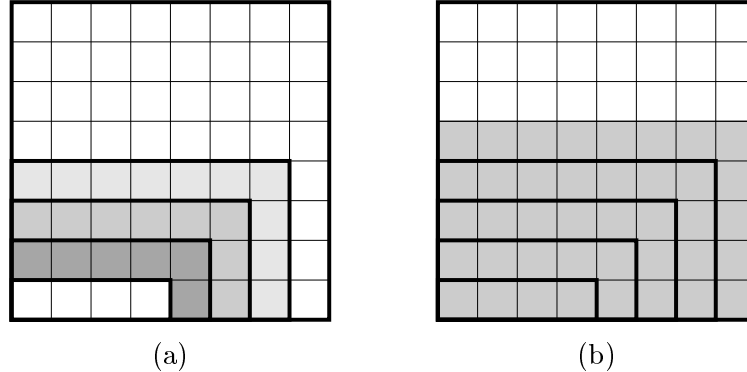


Figure 20: An example showing an region image and associated border image. This shows how pixel-wide regions can cause a "bunching" effect in the border image. (a) is the region image and (b) shows that most of the pixels in the border image are classified as bixels (shown as pixels with shading) in order to represent the region image.

6.2 Two-pass Image Partition Encoding

The image partition coder described in this chapter uses a two-pass approach for two reasons. First, the coder is to be able to choose the best template to encode each of the region boundaries. A one-pass approach would be required to use the most general template that included

all possible directions. Second, a two-pass approach enables the encoder to apply run-length encoding. It is possible to implement the encoder with run-length encoding by buffering chain links before coding. This approach, however, would increase the complexity of the encoder and likewise assume qualities of a two-pass approach.

6.3 Other Pixel-based Encoding Schemes

The results of the chain encoder compare favorably with results stated in other edge-based chain coding approaches. For example, Ester and Algazi [5] reported average coding rates of 1.874 bps using a QM-code with a zero-order model depth. It should be noted, however, that these results are for link directions only, and do not include other side information such as chain start and chain termination data. The stated results for experiments using higher order modeling suggest further performance remains to be extracted in our image partition encoder.

Conducting a fair comparison of the results reported by chain encoding schemes is challenging. First, pixel-based chain codes generally encode region using fewer symbols than edge-based schemes as demonstrated in Fig. 1. This causes the *bps* metric to be less meaningful when the number of symbols required to encode the image is not considered. Second, chain encoding performance is highly dependent upon the chains being encoded. A fair comparison can only be made when the same chain is encoded by all methods being compared.

7 Conclusion

We presented a technique to generate pixel-based region boundaries and a baseline image partition encoder utilizing a pixel-based, differential chain coder. The results showed that the bit encoding rates of the boundaries are relatively low compared to the number of pixels in the image over a test image set. Our results compared well to other pixel-based encoders (in bits-per-symbol) despite the fact our encoder contains extra information embedded in the chain code. Despite the addition of several features to the boundary encoder, the chain coder retained the simplicity of chain codes described in early works on the subject. The technique of dynamic template switching was shown to improve the bit-encoding rate of the generated boundaries.

A Image Partition Coding Language Specification

This appendix contains the Backus-Knorr Language specification for the language used by the image partition coder. The grammar listed below uses bold face for terminals and the normal font for nonterminals. The *start* symbol is listed first.

```
Border_Image ::= bobs

bobs ::=
  bob
  | bobs, bob

bob ::=
  bixel_list

bixel_list ::=
  bixels
  | bixel_list, bixels

bixels ::=
  standard_bixel
  | nonstandard_bixel

non_standard_bixel ::=
  mapping_phase_nonstandard_bixel
  | encoding_phase_nonstandard_bixel

mapping_phase_non_standard_bixel ::=
  restart_bixel
  | newstart_bixel
  | seed_bixel
  | hole_bixel
  | temp-finish_bixel
  | final-finish_bixel

encoding_phase_non_standard_bixel ::=
  direction_run
  | finish_run_bixel
```

References

- [1] P. Ausbeck. *Piecewise Smooth Modeling of Digital Images*. Ph.D. dissertation, Univ. California Santa Cruz, June 1996.
- [2] F. Bossen and T. Ebrahimi. A Simple and Efficient Binary Shape Coding Technique Based On Bitmap Representation. In *International Conference on Acoustics, Speech, and Signal Processing*, volume 4, pages 3129–3132. IEEE, 1997.
- [3] N. Brady, F. Bossen, and N. Murphy. Context-based Arithmetic Encoding of 2D Shape Sequences. In *International Conference on Image Processing*, volume 1, pages 29–32. IEEE, April 1997.
- [4] B.B. Chaudhuri and S. Chandrashekhar. Neighboring Direction Runlength Coding: An Efficient Contour Coding Scheme. *IEEE Transactions on Systems, Man, and Cybernetics*, 20(4):916–921, July 1990.
- [5] R.R. Estes and V.R. Algazi. Efficient Error Free Chain Coding of Binary Documents. In *Proc. Data Compression Conference*, pages 122–131. IEEE Computer Society, 1995.
- [6] H. Freeman. On the encoding of Arbitrary Geometric Configurations. *IRE Transactions on Electronic Computers*, EC-10:260–268, June 1961.
- [7] T. Kaneko and Masashi Okudaira. Encoding of Arbitrary Curves Based on the Chain Code Representation. *IEEE Transactions on Communications*, 33(7):697–706, July 1985.
- [8] A.K. Katsaggelos, L.P. Kondi, F.W. Meier, J. Ostermann, and G.M. Schuster. MPEG-4 and Rate-Distortion-Based Shape-Coding Techniques. *Proceedings of the IEEE*, 86(6):1126–1154, June 1998.
- [9] Z. Kulpa. Area and Perimeter Measurements of Blobs in Discrete Binary Pictures. *Computer Graphics and Image Processing*, 6(4):434–451, December 1977.
- [10] M. Kunt, M. Benard, and R. Leonardi. Recent Results in High Compression Image Coding. *IEEE Transactions on Circuits and Systems*, CAS-34(11):1306–1336, November 1987.
- [11] G. Langdon and Jorma Rissanen. Compression of Black-White Image with Arithmetic Coding. *IEEE Transactions on Communications*, 29(6):858–867, June 1981.
- [12] C. Lu and J. Dunhan. Highly Efficient Coding Schemes for Contour Lines Based on Chain Code Representations. *IEEE Transactions on Communications*, 39(10):1511–1514, October 1991.
- [13] S. Makrogiannis, G. Economou, and S. Fotopoulos. Region Oriented Compression of Color Images Using Fuzzy Inference and Fast Merging. *Pattern Recognition*, 35(9):1807–1820, September 2002.

- [14] B. Mealy. Region Boundary Generation and Compression. In *Thirty-Fifth Asilomar Conference on Signals, Systems, and Computers*, pages 205–209, 2001.
- [15] T.H. Morrin. Chain-Link Compression of Arbitrary Black-White Images. *Computer Graphics and Image Processing*, 5:172–189, 1979.
- [16] P. Nunes, F. Pereira, and F. Marques. Multi-Grid Chain Coding of Binary Shapes. In *International Conference on Image Processing*, volume 3, pages 114–117. IEEE, April 1996.
- [17] A.J. Pinho. Encoding of Image Partitions Using a Standard Technique for Lossless Image Compression. In *International Conference on Acoustics, Speech, and Signal Processing*, volume 6, pages 3141–3144. IEEE, 1999.
- [18] A.J. Pinho. Adaptive Context-Based Arithmetic Coding of Arbitrary Contour Maps. *IEEE Signal Processing Letters*, 8(1):4–6, January 2001.
- [19] S.R. Tate. Lossless Compression of Region Edge Maps. Technical Report CS-1992-9, Duke Univ., Department of Computer Science, Durham, NC, 1992.
- [20] Martin J. Turner. *The Contour Tree Image Encoding Technique and File Format*. PhD thesis, Univ. of Cambridge, New Museums Site, Pembroke St., Cambridge CB23QG, England, July 1994.
- [21] P. Zingaretti, M. Gasparroni, and L. Vecci. Fast Coding of Region Boundaries. *IEEE Transactions Pattern Analysis and Machine Intelligence*, 20(4):407–415, April 1998.