CHAPTER 17

Lossless Bilevel Image Compression

MICHAEL W. HOFFMAN

17.1 BILEVEL IMAGE COMPRESSION

This chapter describes two international standards for lossless bilevel image compression that are commonly referred to as "JBIG" and "JBIG2." JBIG2 also allows lossy compression of bilevel images as a trade for greater compression in areas of documents in which a lossless reproduction may not be warranted, such as a halftone background or fill pattern. The JBIG standard is described in the next section. In Section 17.3 the JBIG2 standard is described. Finally, the performance results of current lossless bilevel compression standards are compared using published reports.

17.2 JBIG

ITU-T Recommendation T.82 [3] specifies the eponymous JBIG standard that was drafted by the Joint Bilevel Image Experts Group (JBIG). This standard provides an alternative to Recommendations T.4 and T.6 in providing lossless compressed representations of bilevel images for more efficient transmission and storage of these images.

17.2.1 Overview of JBIG Encoding/Decoding

While it can be used effectively on grayscale or color images, JBIG was primarily designed for the compression of bilevel images, i.e., those images that have two colors: background and foreground. The primary applications for JBIG compression include facsimile transmission, document storage, and document distribution. These documents may contain text, line drawings, and halftone images. This variety of applications requires different modes of decomposing, transmitting, and reconstructing the bilevel images. In a facsimile transmission, for example, there may be little

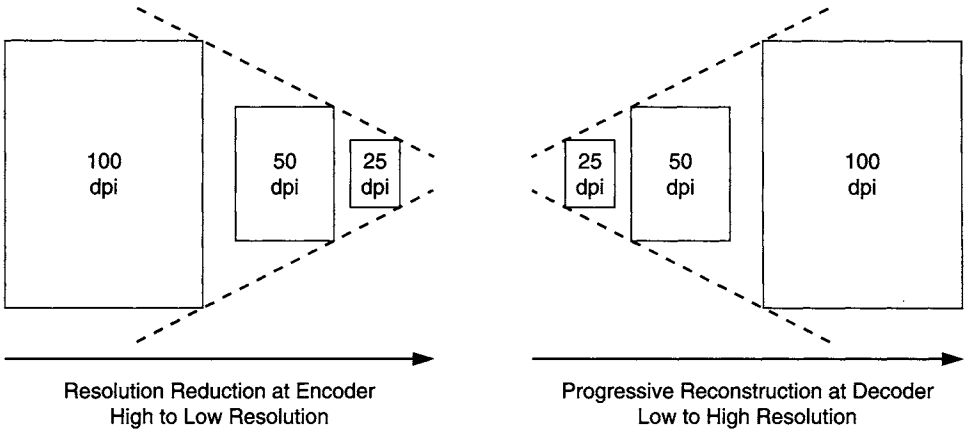


FIGURE 17.1
Resolution reduction and progressive encoding and reconstruction.

need for a progressive reconstruction of the image. In this case, a sequential transmission (left to right, top to bottom) of the original image pixel data may be preferred. If documents are being archived for later retrieval and distribution over networks, however, a progressive reconstruction may significantly reduce the network resources or operator effort required to search for a specific document or type of document. Other considerations, such as the memory required to build up progressive approximations before printing a full-sized image, also impact the decisions on how to store, access, or transmit the image data.

Figure 17.1 illustrates the resolution reduction and progressive reconstruction used in JBIG progressive transmission. At the encoder, the order of processing is from the original and highest resolution image to the lowest resolution image. The decoder reconstructs from the lowest resolution image first, successively refining the image (and increasing the resolution) until the original image is recovered. In Fig. 17.1 the lower resolution images are shown as smaller in size than their higher resolution counterparts to indicate that there are fewer data bits in a low-resolution image. Of course, if these different resolution images were rendered onto a typical facsimile page, the size of these rendered images would be the same. The number of resolution reductions is variable—a selection that results in a final resolution of 10 to 25 dots per inch (dpi) allows for useful thumbnail representations. Original image source resolutions may be 200 to 400 dpi for facsimile applications or 600 dpi or higher for documents destined for laser printing. In a practical sense, the standard imposes very few restrictions on these basic parameters.

JBIG allows for a variety of operational modes. Figure 17.2 shows the relationship among three distinct modes of operation. Single-progression sequential mode means that the pixels are scanned in raster order (left to right, top to bottom) and encoded for the entire image. Progressive mode and progressive-compatible sequential mode both use identical progressive reconstructions of the image. However, the ordering of the data is different in the two modes. The progressive mode reconstruction occurs for the entire image at one resolution. Reconstructions for subsequent resolutions are also made over the entire image.

An image can be broken into horizontal stripes and these stripes can be (loosely) treated as separate images. Progressive-compatible sequential mode performs the progressive refinement over a stripe of the image until that stripe is reconstructed at the highest resolution and then continues with the next stripe of the image. These image stripes are depicted in Fig. 17.3. The standard allows the number of lines in a stripe at the lowest resolution to be a free parameter. Note that the stripe is required to run the full width of the image. If an odd number of lines are

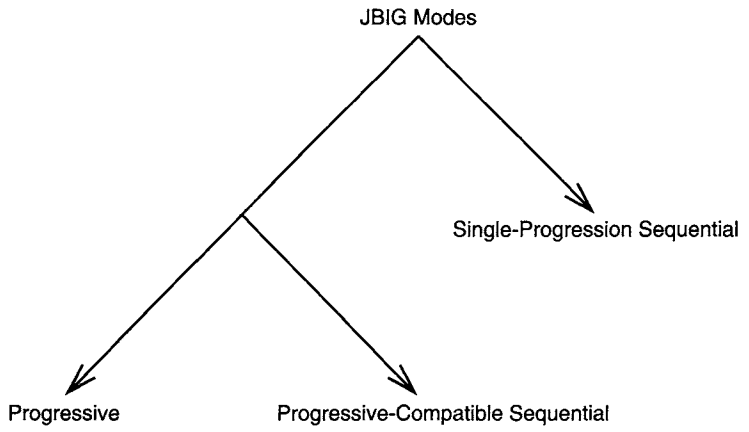
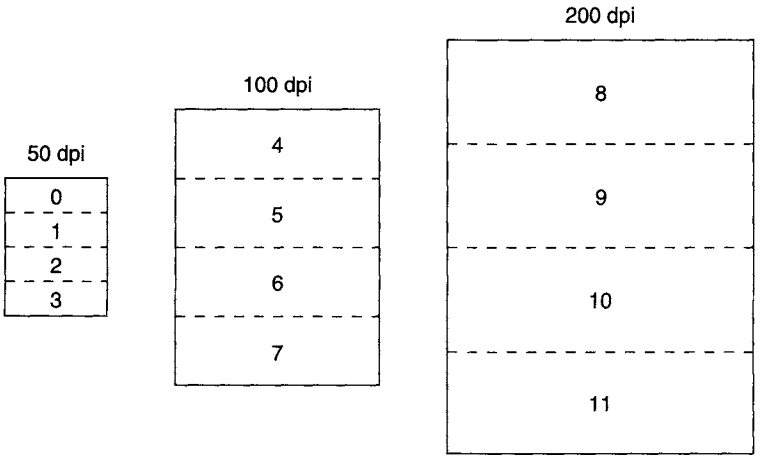


FIGURE 17.2
JBIG modes of operation.



Stripes and Data Ordering

HITOLO	SEQ	Example Order
0	0	0,1,2,3,4,5,6,7,8,9,10,11
0	1	0,4,8,1,5,9,2,6,10,3,7,11
1	0	8,9,10,11,4,5,6,7,0,1,2,3
1	1	8,4,0,9,5,1,10,6,2,11,7,3

FIGURE 17.3
Assignment of stripes within various resolutions of a given image and the possible data orderings for a single bit-plane.

included at a stripe in a higher resolution, the final line in the stripe needs to be replicated before resolution reduction.

Figure 17.3 also indicates various choices for ordering the progressive stripe data. The binary parameter SEQ allows either progressive full image data ordering (SEQ = 0) or progressive-compatible sequential ordering (SEQ = 1). The binary parameter HITOLO determines whether the data is ordered in highest to lowest resolution, the order in which it is processed at the encoder (HITOLO = 1), or in lowest to highest resolution, the order in which it is processed at the decoder (HITOLO = 0). Note that either the decoder or encoder is burdened with storing the bulk of the representation—this choice, however, can be made depending on the needs of a particular application. As mentioned previously, note that the compressed data transmitted is the same for either the progressive or progressive-compatible sequential modes; however, the order in which it is transmitted is different. Finally, if more than a single bit-plane is being considered, there are other additional parameters for determining the ordering of the stripes within each resolution and bit-plane.

17.2.2 JBIG Encoding

This section describes the encoding details of JBIG. JBIG encoding includes the following components: resolution reduction, prediction, model templates, and encoding. Resolution reduction results in a lower resolution image as well as data that needs to be encoded so that reconstruction of the current resolution is possible given the lower resolution image. Prediction attempts to construct the higher resolution pixels from the lower resolution pixels and neighboring higher resolution pixels that are available to both the encoder and the decoder. Two basic types of prediction are used: typical prediction and deterministic prediction. Neighboring pixels are used to provide a template or context for more efficient entropy coding. Both fixed templates and adaptive templates are used. Finally, a context-based adaptive arithmetic encoder is used to encode those pixels that cannot be predicted and require encoding. Some differences exist between encoding operations at a differential layer and at the bottom layer and these will be described as required.

17.2.2.1 Resolution Reduction

D defines the number of doublings of resolution in an encoding of an image. At the highest resolution there are X_D pixels per row and Y_D rows in the image. R_D is the sampling resolution of the original image (typically stated in dpi). If sequential coding is desired, D is set to 0 and the original image is treated as the bottom layer (i.e., the lowest resolution layer). Figure 17.4

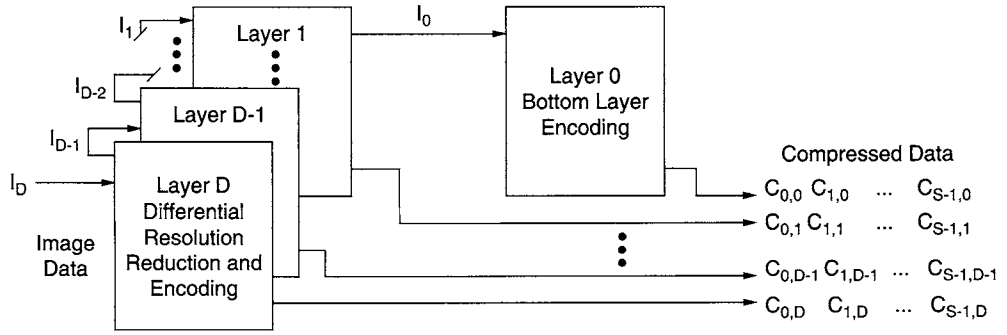


FIGURE 17.4
JBIG encoder resolution reduction and encoding.

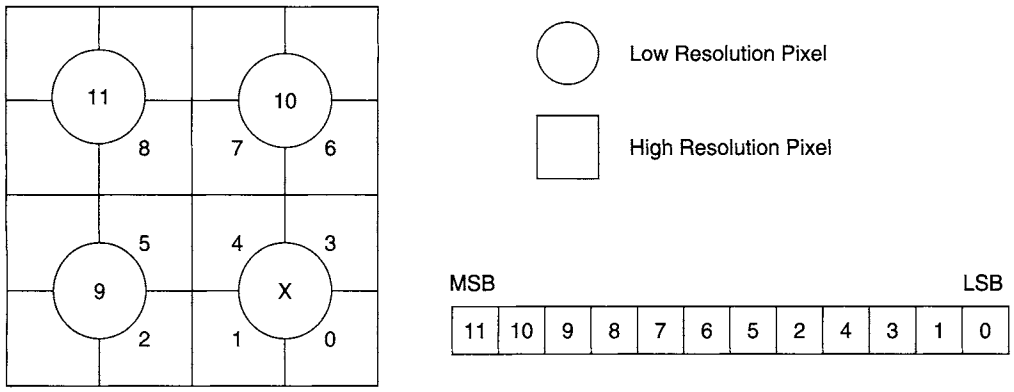


FIGURE 17.5
JBIG encoder resolution reduction neighborhood.

illustrates the resolution reduction and data generation for a JBIG encoder. The image at resolution $D - k$ is referred to as I_{D-k} ($k = 0, \ldots, D$). The coded image data for stripe s ($s = 0, \ldots, S - 1$) at resolution $D - k$ is denoted $C_{s,D-k}$. The functional blocks for each differential layer resolution reduction and encoding are identical. The bottom layer encoding contains significant differences from the differential layer encoding. Each differential layer takes as input the image data from the next higher resolution differential layer. In addition, each differential layer performs a resolution reduction and outputs coded data that allows for reconstruction of the input (higher resolution) image given the output (lower resolution) image.

The resolution reduction algorithm consists of a table-based look-up given a neighborhood of surrounding pixels taken from both the high- and low-resolution images. Figure 17.5 illustrates the relationship among the neighboring pixels in both the high- (squares) and low- (circles) resolution images. The pixel to be determined in the reduced resolution image is denoted X, while the pixels numbered 11, 10, ..., 0 form the 12-bit context for pixel X. This context is used as the index into a resolution reduction table with 4096 one-bit entries. The bit from the table for a given context is used as the value for X. While it is possible to define a different resolution reduction table than that included in the JBIG recommendation, it is important to note that this change will require matching tables for deterministic prediction. The goal in designing the table set forth in the standard was to preserve edges and lines via exceptions to a general resolution reduction rule. While not necessarily enhancing compression, these exceptions do enhance the visual information content of the low-resolution images.

In many cases, the pixels needed for a given context may not exist in the original image or the stripe being processed. In those cases the standard provides the following general rules for edges:

- the top, left, and right edges are assumed to be surrounded by the backgound color (0), and
- the bottom line is assumed to be the replication of the last line in the image (or stripe).

So if either or both of the vertical and horizontal dimensions of the image at level d are odd, additional pixels at the right and bottom are added according to these general rules. In addition, these rules are followed as needed for generating context pixels for edge pixels in the image. When pixels are required across stripe boundaries pixels from stripes above are used from the actual image since these will be available to the decoder, but pixels from below the current stripe are generated by replication from the current stripe even if these pixels exist in the original image.

17.2.2.2 Differential Layer Prediction

The prediction used in the differential and bottom layers is sufficiently different to warrant separate discussions. In differential layer encoding, after resolution reduction there are three possibilities for encoding a high-resolution pixel:

- it can be ignored (not coded) because typical prediction uniquely determines its value,
- it can be ignored (not coded) because deterministic prediction uniquely determines its value, or
- it is encoded by an arithmetic coder based on a context of neighboring pixels (from high- and low-resolution images).

In this section the first two possibilities are discussed.

17.2.2.2.1 Differential Typical Prediction Figure 17.6 illustrates the neighborhood of the eight low-resolution pixels closest to low-resolution pixel X. The four phases of the high-resolution pixels associated with X are also indicated in the figure. A pixed is called "not typical" if it and all eight of its neighboring pixels are the same color, but at least one of the four high-resolution pixels associated with it is not. If a line of low-resolution pixels contains any "not typical" pixels, then the *line* is referred to as not typical. For each pair of high-resolution lines, a fictitious pixel called LNTP (Line Not Typical) is inserted to the left of the first pixel in the first of the two-line set of high-resolution pixels. LNTP is set to zero unless a low-resolution line is found to be "nontypical." In that case this pixel is set to 1. A particular context that is rarely used in coding normal bilevel images is reused for arithmetic encoding of the LNTP pixel.

When a low-resolution pixel is found to be "typical" in a line deemed typical, then the four phases of the high-resolution pixels associated with that pixel are not encoded. The introduction to the JBIG recommendation states that on text or line-art images, typical differential prediction allows coding to be avoided for approximately 95% of the pixels. This allows a considerable reduction in execution time since no other processing is required when typical prediction is used. The overhead for using typical prediction includes sending the extra pixel LNTP for every other

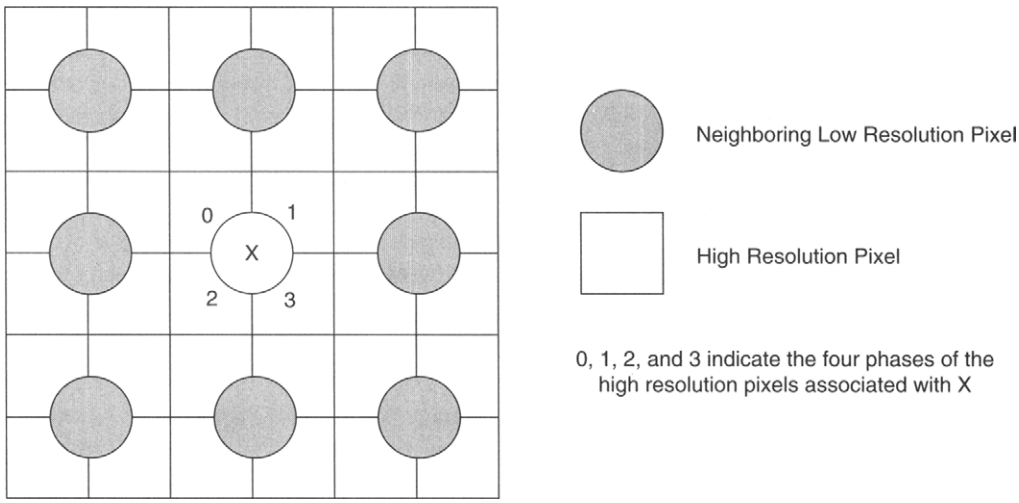
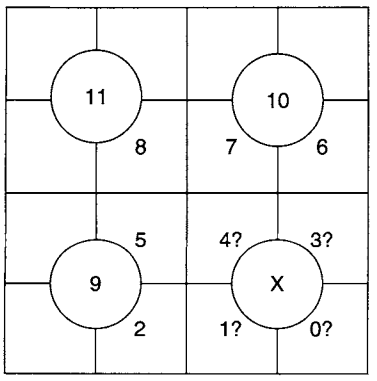


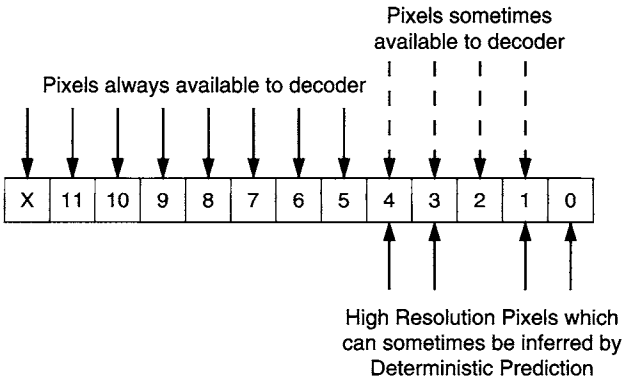
FIGURE 17.6
The neighborhood of the eight low-resolution pixels closest to X and the four phases of the high-resolution pixels associated with X.

row of the high-resolution image. Note that at the decoder the lowest levels are reconstructed first. This allows determination of whether a given low-resolution pixel can be typical at the decoder. When the line in a low-resolution image is at a stripe boundary, the general rules for edge pixels are used to replicate the last row in the stripe even though the actual pixel may be available at the decoder. Progressive-compatible sequential encoding, for example, would not have this pixel available, while progressive encoding would—but the edge pixel rules are used in both cases. Finally, note that typical prediction is optional for differential layers.

17.2.2.2.2 Deterministic Prediction (differential layer only). Since the table look-up for resolution reduction sometimes results in a high-resolution pixel being determined by the information already available to the decoder, there is no need to encode this pixel. In JBIG this is referred to as deterministic prediction. Deterministic prediction is available only for differential layer encoding. Figure 17.7 illustrates the relationship between the table-based resolution reduction algorithm and deterministic prediction. Given a resolution reduction table (such as the default table), there are patterns of bits X, 11, 10, 9, ... available to the decoder that can be used to predict the values of the four phases of the high-resolution pixels associated with X. These phases are bit positions 4, 3, 1, and 0 from the resolution reduction context. For each of these pixels there is a chance that the particular sequence of bits in the context already available to the decoder removes any ambiguity about the pixel and obviates the need for coding that pixel. The number of deterministic hits for each of the four phases (using the default resolution reduction tables) is indicated in a table provided in the figure. The standard recommendation indicates that using deterministic prediction allows additional compression of about 7% on the bilevel images in the test set.



Resolution Reduction Context



Phase	Pixel	Number of Contexts	Number of Deterministic Hits with Default Resolution Reduction Tables
0	4	256	20
1	3	512	108
2	1	2048	526
3	0	4096	1044

FIGURE 17.7
Resolution reduction context and its relationship to deterministic prediction.

Deterministic prediction for the differential layers is optional in JBIG. In addition, should a different resolution reduction table be used for a particular application, then either deterministic prediction should not be used or the deterministic prediction table must be matched to the particular resolution reduction algorithm used. This deterministic prediction table needs to be included in the JBIG header if deterministic prediction is to be used.

17.2.2.3 Bottom Layer Typical Prediction

For the bottom layer image data only typical prediction is available. (Note that if the sequential encoding mode is used, then this bottom layer image is simply the original image.) As with the differential layer prediction, bottom layer typical prediction is an option. The ith row (or line) in the bottom layer image data is said to be not typical if it differs in any pixel from the line above it. If it is not typical, then $LNTP_i$ is set to 1. A fictitious pixel called *SLNTP* ("silent pea") is generated as

$$SLNTP_i = !(LNTP_i \oplus LNTP_{i-1}), \tag{17.1}$$

where $\oplus$ indicates exclusive-or and ! indicates logical negation. For the top line, $LNTP_{-1}$ is set to 1. The virtual pixel location for sameness indicator $SLNTP_i$ is to the left of the first pixel in line i. Note that for the bottom layer, the virtual pixel occurs in every line when typical prediction is used.

17.2.2.4 Model Templates

When typical or differential prediction is used, the predictors generate a 0 or 1 (for the value of the pixel being predicted) or a 2 to indicate that this particular form of prediction cannot be applied. Note that this output does not need to be transmitted since the prediction is designed to be performed in an identical fashion at both the encoder and decoder. When prediction cannot be used, then the pixel in question needs to be encoded. Context-based adaptive arithmetic coding is used to encode these pixels. In this section the templates used to generate the contexts are described.

17.2.2.4.1 Differential Layer Templates Figure 17.8 depicts the templates used for differential layer encoding of the four phases of the high-resolution pixels. In the figure, pixels labeled X or A are the pixels used to form the context for coding the pixel starting with P. In addition to the 10 bits representing these pixels, 2 bits are required to identify the coder used for each of the four phases of high-resolution pixels. This results in 4096 possible contexts (one of which is also used to code *LNTP*).

The X pixel locations are fixed in the positions indicated in Fig. 17.8. The A pixel denotes the adaptive part of the template or, simply, the adaptive template. This pixel can be moved to a wide range of locations to allow better coding for repetitive regions such as may occur in half-toned computer generated images. The range of motion is $\pm M_x$ in the horizontal direction and from 0 to M_y in the vertical direction. M_x can be set up to a maximum of 127 while M_y can be set up to a maximum of 255. These two parameters limit the range of values that can be used for a given bilevel image. The actual values can be varied within a stripe of data by using an escape sequence followed by the line in which the change occurs and the value of the horizontal and vertical offsets for the adaptive template. Up to four moves per stripe are allowed, with the order of the move escape sequences corresponding to the increasing number of lines in which the moves become effective. Setting M_x and M_y to zero for an image indicates that the default position will be the only location for the adaptive template.

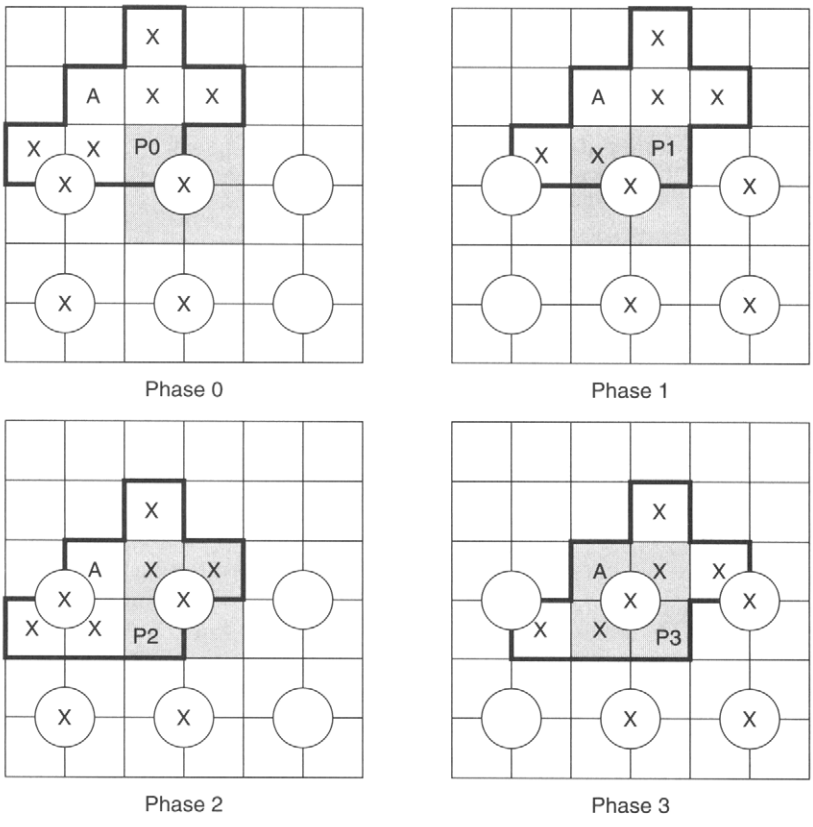


FIGURE 17.8
Differential layer templates for each of the four phases of the high-resolution pixels.

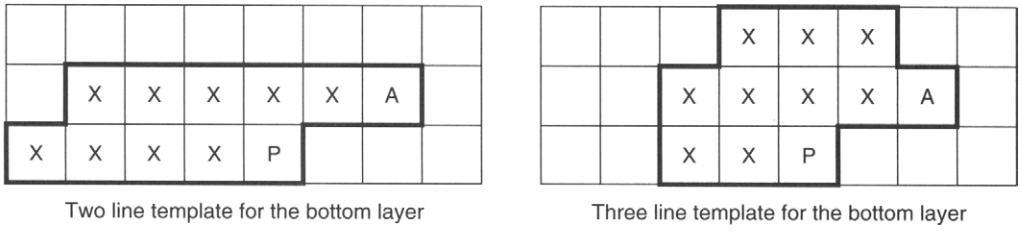


FIGURE 17.9
The two- and three-line bottom layer templates.

17.2.2.4.2 Bottom Layer Templates Figure 17.9 illustrates the two- and three-line templates used for the bottom layer. As with the previous figure for the differential layer templates, the *X* pixels are set in a fixed location within the template and the *A* pixel denotes the adaptive template. The restrictions on the location of the adaptive template are identical to those for the differential layer adaptive template described previously. As is the case with the differential layer adaptive template, a location specification of (0, 0) indicates the default location indicated in the figure. Note that only 10 bits specify the context for the bottom layer templates. This means a total of 1024 contexts for the arithmetic encoder for either the two- or three-line contexts.

Note that the order in which the template bits are arranged to form the context is unimportant, since it is only the state of the arithmetic coder that needs to be tracked—and this is done in an identical fashion for the encoder and the decoder regardless of how the template bits are ordered to form the context index. As long as the formation of the context is consistent throughout the encoding (decoding) process, it is irrelevant that it is done in a manner different than that used in the decoding (encoding) process. This is true for both the bottom layer and differential layer encoding.

17.2.2.5 Adaptive Arithmetic Coding

For each pixel being encoded the values generated by typical prediction (TP) and deterministic prediction (DP) are used to decide whether arithmetic coding is needed. If either prediction mode is not being used, the corresponding value for this predictor is always set to 2. When either prediction provides the value for the current pixel, then this value (0 or 1) indicates that no arithmetic encoding is needed. If arithmetic encoding is required (both TP and DP equal 2), then the context and the value of the current pixel are used by the adaptive arithmetic encoder. The top level flow diagram for JBIG adaptive arithmetic encoding is shown in Fig. 17.10.

The actual encoding depicted in the figure is a variant of the Q-coder adaptive binary arithmetic coder. The coder tracks a most probable symbol (MPS) and a least probable symbol (LPS) and applies a multiply-free set of update equations (see [1], for example). For each context there is a 1-bit value of the MPS and a 7-bit value indicating 1 of 113 possible states for the coder. As pixels are encoded, the arithmetic coding interval is updated and, if required, a renormalization (i.e., a set of doublings) of the interval occurs. This renormalization process generates the bits that form the coded sequence. After each renormalization process a probability estimation process is undertaken to reset the state of the coder for a given context. It is possible that the MPS switches during encoding—certain states of the encoder are associated with this switch and the table look-up for the state indicates when a switch occurs. At the end of a stripe of data, a flush routine removes unnecessary all 0 bytes from the datastream. These will be generated within the decoder as required to allow for complete decoding of the coded bitstream.

17.2.3 Data Structure and Formatting

The fundamental unit of JBIG image data is the Bilevel Image Entity (BIE). The BIE is composed of a Bilevel Image Header (BIH) and Bilevel Image Data (BID). It is possible, but not required, to use multiple BIEs for a single image. Figure 17.11 illustrates the relationship of the BIH and BID within the BIE. In addition, the figure breaks out the components of the BIH with labels and sizes of key elements. The BIH contains the initial resolution layer, D_L, within the BIE and the final resolution layer, D, within the BIE. The value P specifies the number of bit-planes in the image (P is 1 for bilevel images). X_D and Y_D are the size of the image at its *highest* resolution. These 4-byte entries are encoded with most significant byte first and allow for image sizes of $2^{32} - 1$ (or over 4×10^9) pixels per dimension. L_0 specifies the number of lines in a stripe at the lowest resolution (this is also a 4-byte entry coded most significant byte first). M_x and M_y are 1-byte entries specifying the maximum allowed range of motion for the adaptive template in the BIE. The *Order* byte contains the 1-bit values *HITOLO* and *SEQ* that specify the order in which the stripes and bilevel data are presented within the BIE. *ILEAVE* and *SMID* perform similar functions when the image consists of multiple bit-planes—these are used with *HITOLO* and *SEQ* to provide up to 12 different orderings of data segments. In the *Options* byte, the *LRLTWO* bit determines whether the two-line template (one) or three-line template (zero) is used in encoding the lowest resolution data. The *VLENGTH* bit allows a redefinition of the image length to occur. *TPDON* indicates that typical prediction in the differential layer is used when it is set to 1 while

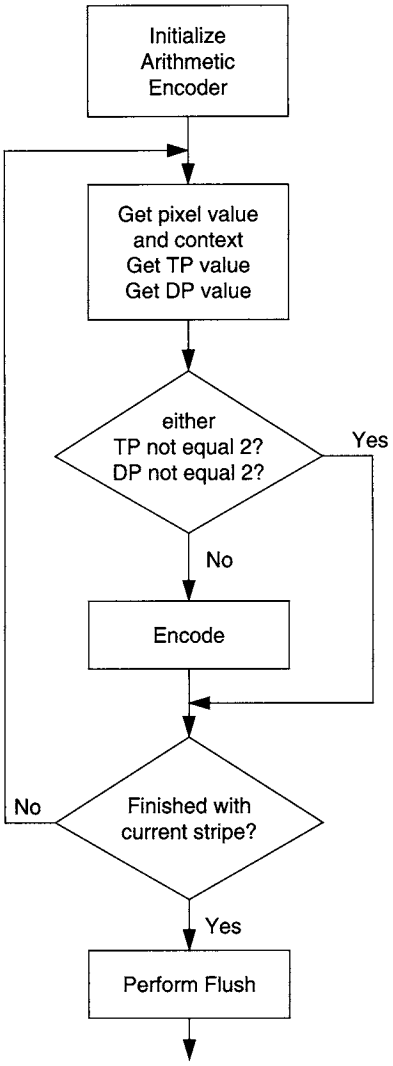


FIGURE 17.10
Flow chart for the adaptive arithmetic encoding performed in JBIG.

TPBON performs a similar function for bottom layer typical prediction. *DPON* indicates that the deterministic prediction is used (recall that this affects only differential layer image data). The *DPPRIV* bit indicates that a private resolution reduction algorithm has been used and it specifies that a private deterministic prediction table needs to be used when it is set to 1. The *DPLAST* bit indicates whether to use the previously loaded private deterministic prediction table (one) or to load a new DP table (zero). Finally, if a private deterministic prediction table needs to be loaded the 1728 bytes of data in this table follow the *Options* byte.

The BID is composed of floating marker segments followed by Stripe Data Entities (SDE). An escape byte, ESC, is used to indicate marker codes. These marker codes can be used to terminate Protected Stripe Coded Data (PSCD), to abort a BID prematurely, or to indicate a floating marker segment. The floating marker segments include an adaptive template move segment, a redefinition of image length segment, a comment segment, and a reserved marker byte. The adaptive

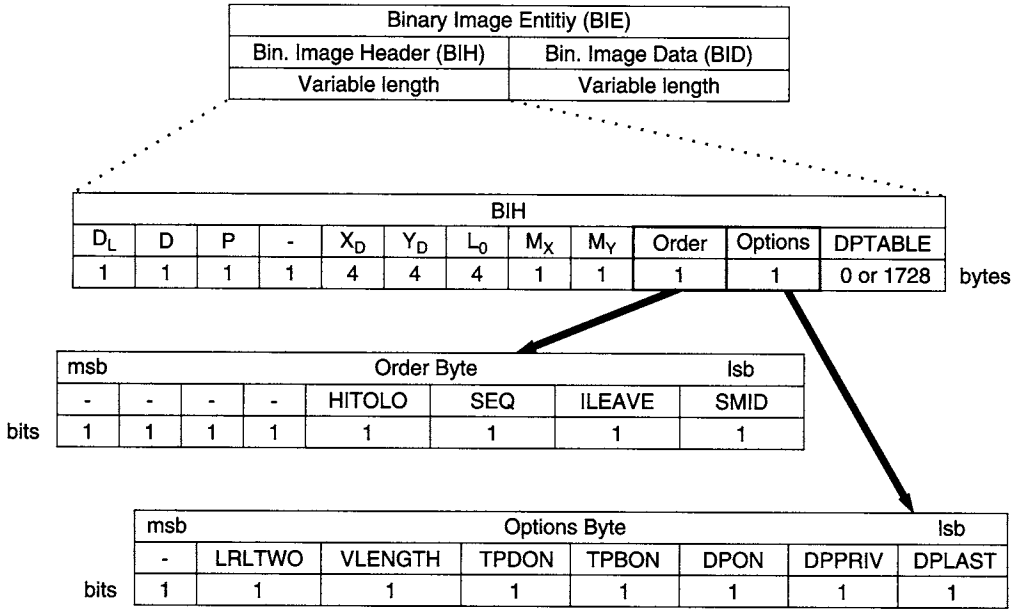


FIGURE 17.11
JBIG data structure and header information.

template move segment specifies the line at which the change occurs and the positions (horizontal and vertical) to which the adaptive template is moved. The new length segment specifies the new length of the image. To avoid improper operation when the arithmetic coder binary data produces a byte-aligned ESC symbol, the Stripe Coded Data (SCD) is searched and the occurrences of byte-aligned ESC symbols are replaced with the ESC symbol followed by a STUFF byte. This produces the PSCD that is included in the SDE within the BID. There can be an arbitrary number of SDEs in a BID. Floating marker segments occur between SDEs within the BID.

17.2.4 JBIG Decoding

JBIG decoding is a fairly straightforward reversal of the steps used in the encoder. The decoder starts with the coded data from the lowest level image data and reconstructs this bottom layer image. If sequential encoding was used, the decoder is finished. If progressive coding was used, the decoder then undoes the resolution reduction by reconstructing the high-level image from the low-level image and the coded image data for that resolution layer.

The adaptive arithmetic decoding mirrors the adaptive process used by the encoder. If typical prediction and deterministic prediction are used, then the JBIG decoder is required to detect situations in which the encoder would have avoided encoding a given pixel and use its own local prediction in these cases. The determination of whether particular lines are typical is made by the encoder and this information is transmitted as the first pixel in each line (bottom layer) or in every other line (differential layer). As with the encoder, the operation of the differential layers at the decoder is identical. One difference is that if the bilevel image is to be rendered, the final differential decoding can be sent directly to the display device rather than being stored. Note that the progressive-compatible sequential mode can be used to avoid the memory issues involved in reconstructing large images by defining stripes to be a manageable number of lines.

To determine the number of lines, lines per stripe, or pixels per line in a decoded differential layer, the decoder uses the actual number of rows and columns in the highest resolution image, the number of lines per stripe at the lowest layer, and the algorithm for resolution reduction to reconstruct the correct number of lines. These data are included in the BIE header information. For example, if the number of pixels per line at resolution d is 21, then it is possible for the number of pixels per line at resolution $d + 1$ to be 41 or 42. While one may be tempted to say "the correct answer is 42," it is possible that the number of pixels per line is 41, as would occur if the original image contained 2561 pixels per line.

According to the JBIG recommendation, the best compression is typically obtained using the sequential mode of operation. Based on implementation and use considerations, the progressive mode or progressive-compatible sequential mode may be more appropriate for a given application. If a particular image is to be compressed for use on a variety of platforms, progressive operation allows different users to meet their fidelity requirements and produce a version of the document without undue computation or delay. Recall that the only difference between progressive and progressive-compatible sequential coding is the order in which the same coded data is organized and transmitted.

17.3 JBIG2

ITU-T Recommendation T.88 [4] specifies the standard commonly referred to as JBIG2. This recommendation was drafted by the Joint Bilevel Image Experts Group committee, the same group that drafted the JBIG recommendation. This recently (2000) approved standard provides an alternative to Recommendation T.82 (JBIG) [3] and Recommendations T.4 and T.6 in providing lossless compressed representations of bilevel images for more efficient transmission and storage of these images. In addition, JBIG2 provides for lossy compression of bilevel images. The amount and effect of this lossy compression are controlled during encoding and are not specified as part of the standard. Generally, lossy compression of halftone regions of bilevel images can provide dramatic reductions in the compressed image size in exchange for small reductions in quality.

17.3.1 Overview of JBIG2

JBIG2 allows bilevel documents to be divided into three parts: text regions, halftone regions, and generic regions. Text regions consist primarily of symbols (letters, numbers, etc.) that are relatively small, such as the standard font sizes used for document preparation. These symbols are aligned in either a left–right or top–bottom row format. JBIG2 allows the formation of arbitrary symbol dictionaries. These dictionaries form the basis of the compressed representations of the symbols. The fact that particular symbols are mapped to a finite set of dictionary symbols indicates a certain lossy quality to the representations for symbol occurrences. Hence, a refinement step for generating the final symbol from the initial estimate provided by the dictionary entry is included as an option.

Halftone regions consist of regularly occurring halftone patterns that allow bilevel images to be used for representing various shades and fill patterns, as well as grayscale images. In a halftone region a particular pattern dictionary is used to represent small sections of the bilevel image. The indices of the patterns in the dictionary that are matched to the original image are treated as a pseudo-grayscale image at a reduced spatial resolution from the bilevel image. This pseudo-grayscale image is encoded in bit-planes and transmitted to the decoder along with angular orientation and grid size information to provide for a lossy representation of a halftone portion of a bilevel image.

Generic regions consist of line drawings, large symbols, or other bilevel components that have not been identified or encoded as halftone or text regions. In addition, the algorithms used for compressing generic regions are the basis of the representations for the symbol dictionary components, the pattern dictionary components, and the bit-planes of the pseudo-grayscale images used in the halftone regions. Generic region coding also allows refinements to be applied to other regions. This permits *quality progressive* representations to be encoded.

Note that, in contrast to JBIG which treated each page or page stripe as a single image, JBIG2 is focused on compressing documents containing different types of content and multiple pages with similar components. To this end, JBIG2 is a segment-based encoding scheme, compared to JBIG, which is an image-based encoding scheme. Segments of a document can contain one of the basic region types mentioned previously, dictionaries for text or patterns, general page layout information, or decoding control information. The ordering of the segments is quite flexible and allows both quality progressive and *content progressive* representations of a document. In a content progressive representation, perhaps text segments would be displayed first, followed by line art and, finally, halftone background segments. These types of representations are particularly useful for network distribution of documents. Note that the rectangular regions defined as text, halftone, or generic can overlap and there can be virtually any number of these per page. Intermediate results are stored in auxiliary buffers that are combined to form the final page buffer. Figure 17.12 illustrates a bitmap, one of the basic building blocks used within JBIG2 regions. The overall structure of the JBIG2 file can be sequential, in which each segment header is immediately followed by the data header and the data for that segment, or random access, in which the segment headers are concatenated at the beginning and the header data and data for each segment follow in the same order as the segment headers.

Whereas the basic compression technology in JBIG was context-based arithmetic coding, JBIG2 allows either context-based arithmetic coding (MQ coding similar to JBIG) or run-length encoding (Modified Modified Relative Element Address Designate (READ) (MMR) coding similar to that used in the Group 4 facsimile standard, ITU-T.6). The primary advantage of MQ coding is superior compression. The primary advantage of MMR coding is very fast decoding.

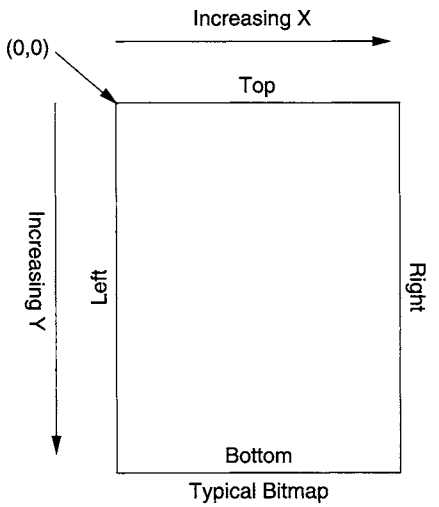


FIGURE 17.12
JBIG2 bitmap: four edges with (0,0) referring to the top left corner. Bitmaps can be given one of four orientations within a JBIG2 region.

Depending on the application, either of these alternatives may be more appealing. MMR coding is described in Chapter 20.

17.3.2 JBIG2 Decoding Procedures

The JBIG2 standard specifies only the decoding procedures for a JBIG2 datastream. There are no normative recommendations for how a JBIG2 encoder is supposed to operate. This allows room for substantial variation in the performance of JBIG2 compatible compression products. In addition, the focus on decoding leaves room for future improvements in encoding strategies, as long as they can be cast into the rather broad and flexible description provided by the JBIG2 recommendation.

In this section the decoding procedures for the primary JBIG2 region types and dictionaries are presented with the goal of providing insight into how and why the recommendation allows substantial compression of bilevel images. The first part of this section describes the decoding required for generic regions. These regions fill or modify a buffer (i.e., a rectangular grid of bits) directly. The next part describes generic refinement region decoding. While the generic regions can be coded with either run-length MMR Huffman coding or context-based arithmetic coding, the refinements use only context-based arithmetic coding and use context bits from both the original and the refined bitmaps. The third part of this section describes the symbol dictionary decoding. Concatenated sets of similar height characters (or symbols) are encoded either directly or with an aggregate refinement based on previously defined symbols. The next part describes the decoding procedures for text regions. These regions make use of existing dictionary symbol entries and can also refine these entries in a manner consistent with the generic refinement region decoding procedures. The fifth part of this section describes the pattern dictionary decoding procedures in which the dictionary entries are concatenated and encoded as a generic region. The final part describes the halftone region decoding procedures, including the orientation of the halftone pattern grid and the pseudo-grayscale image generation and coding.

17.3.2.1 Generic Region Decoding

Generic region decoding generates a bitmap of the specified size (GBW by GBH pixels) in a straightforward manner from a received bitstream. Various control information is first decoded, including an MMR encoding flag, the size of the resulting bitmap, a template selection, and a typical prediction flag. The generic region decoding is reminiscent of both the bottom layer decoding in JBIG if adaptive arithmetic coding is used and the Group 4 facsimile standard if MMR Huffman run-length encoding is used. If the MMR flag is set to 1 then the decoding is identical to the MMR decoder described in ITU-T.6, save a few minor differences. The differences include assigning "black" pixels a value "1" and "white" pixels a value "0," not using an EOFB symbol in cases in which the number of bytes contained in the bitmap will be known in advance and consuming an integer number of bytes in decoding the MMR data (even if bits need to be skipped at the end of a bitmap).

If the MMR flag is set to zero, then the decoding is quite similar to that used in decoding the bottom layer of a JBIG bilevel image. The size of the bitmap (two 4-byte unsigned integers) is decoded first, followed by 2 bits used for selecting the template to be used to generate the contexts for the arithmetic decoding. The four templates available for use in generic region decoding are illustrated in Fig. 17.13. There are one 16-bit, one 13-bit, and two 10-bit templates. All of the templates contain at least one adaptive template bit. The 16-bit template has four adaptive template bits. These adaptive template bits can be positioned from -128 to $+127$ bits in the horizontal (X) direction and from -128 to 0 bits in the vertical (Y) direction. Note that the two 10-bit templates are identical to those used in the bottom layer of JBIG. As with the JBIG bottom layer, bits can be

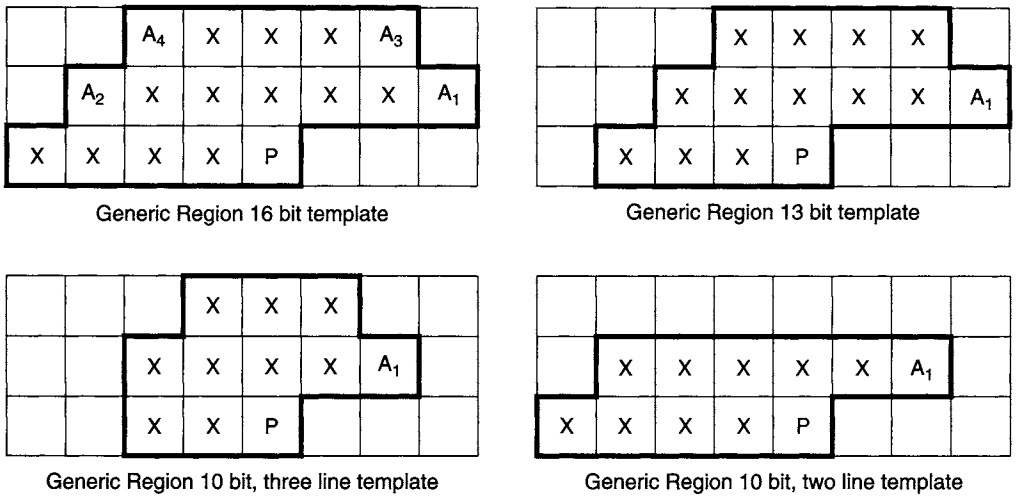


FIGURE 17.13
Generic region templates for context-based adaptive arithmetic coding. A_i indicates adaptive template pixels, X indicates fixed template pixels, and P indicates the pixel to be coded. Adaptive templates are shown in their default locations.

arithmetically encoded using the context provided by the templates. If template bits do not exist within the bitmap, these bits are assumed to be zero.

Typical prediction is also available in generic region decoding. If typical prediction is selected, then a pseudo-pixel is differentially encoded (via exclusive OR) as the first pixel prior to each row in the bitmap. If a given line is identical to the line immediately preceding it, then the pseudo-pixels are different from one line to the next. If the line is not identical to the preceding line, then the pseudo-pixels in the two rows are the same. In this way (as in the bottom layer decoding in JBIG) repeated lines can be represented by this single pseudo-pixel. As in JBIG, certain unlikely contexts are used to encode the pseudo-pixels.

17.3.2.2 Generic Refinement Region Decoding

The generic refinement region decoding procedure refines a reference bitmap already available to the decoder. Generic refinement uses context-based arithmetic coding to produce the refined bitmap given a reference bitmap. The set of inputs needed includes the width and height of the bitmap, a choice of templates, a reference bitmap with offsets that indicate how the two bitmaps are aligned, a typical prediction flag, and the location of the adaptive templates (if these are used). As was the case with generic region decoding, the edge convention employed assumes that any bits not located on a bitmap are zero.

Two templates are used in generic refinement decoding. Figure 17.14 illustrates both the 13- and 10-pixel templates. In Fig. 17.14, there are two grids associated with each of the two templates. One grid is from the reference bitmap, and one is from the refined bitmap. The pixels labeled X or RA_i are used in the template. These pixels are combined (in an arbitrary but consistent order) to form the context used in the arithmetic coding. Note that the pixels taken from the refined bitmap include only those that would be available to the decoder. All pixels in the reference bitmap are assumed available. The shaded pixel P indicates the location of the pixel being decoded in the refined bitmap. The shaded pixel X in the reference bitmap indicates the corresponding location of the P pixel in the reference bitmap. In the 13-bit template, the pixels RA_1 and RA_2 are the refinement adaptive templates. The restrictions on location for RA_1 are identical to those imposed on the generic region adaptive templates, while RA_2 can range from -128 to 127 in both

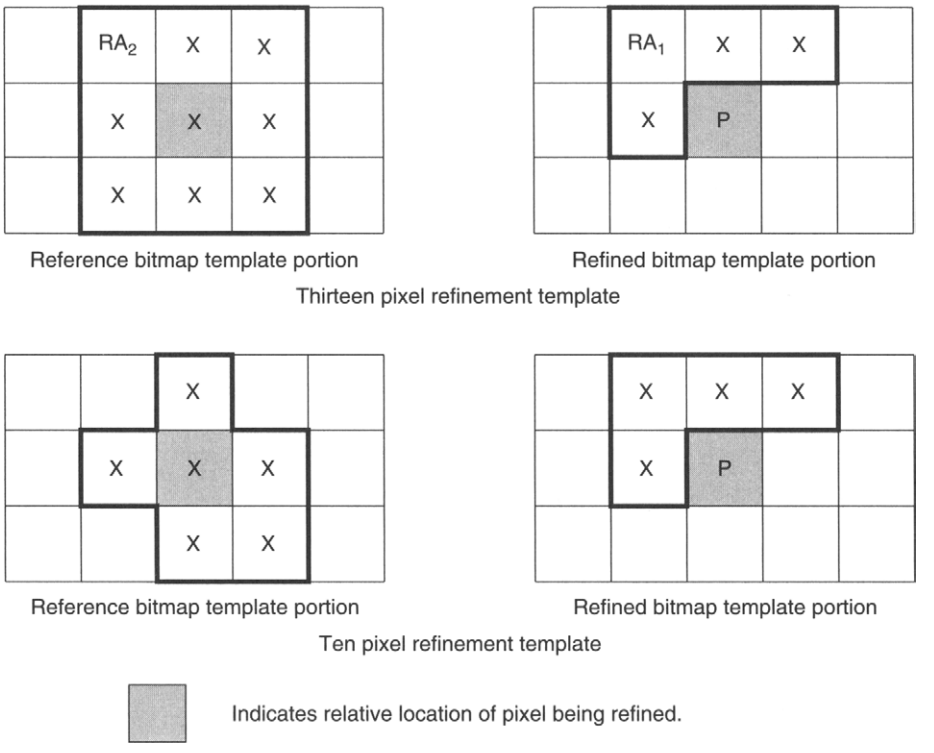


FIGURE 17.14
Generic refinement region templates for context-based adaptive arithmetic coding. RA_i indicates refinement adaptive template pixels, X indicates fixed template pixels, and P indicates the pixel coded in the refined bitmap. Adaptive templates are shown in their default locations.

the horizontal and vertical directions since the reference bitmap is completely available during decoding. There is no adaptive portion of the 10-bit template. The use of pixels from both the reference and refined bitmaps is similar to the use of pixels from both high and low resolutions in the differential layer encoding used in JBIG. Note, however, that the resolution of the two bitmaps is the same in generic refinement.

Typical prediction is allowed in generic refinement region decoding. If typical prediction is used, then a pseudo-pixel is differentially encoded at the beginning of each line to indicate whether that line is typical. If a line is deemed typical (LTP = 1), typical prediction allows certain pixels not to be coded. Figure 17.15 illustrates the generic refinement typical prediction template. If all of the pixels in the reference bitmap portion of the typical prediction template have the same value and LTP = 1, then a given pixel P in the refinement bitmap is set to that same value. If the pixels in the reference bitmap portion of the typical prediction template are not all the same, then the given pixel P in the refinement bitmap needs to be decoded.

17.3.2.3 Symbol Dictionary Decoding

The symbol dictionaries that are required to encode text segments of a document are transmitted as part of the JBIG2 datastream. These dictionary entries can then be exported, i.e., used to build up text regions or other dictionary segments, or they can merely be used as intermediate entries for building up subsequent dictionary entries, i.e., aggregate or refined dictionary entries. The data in a symbol dictionary is organized by height classes, as is illustrated in Fig. 17.16. Within each height class the data for the symbols can be encoded as one collective bitmap or individually

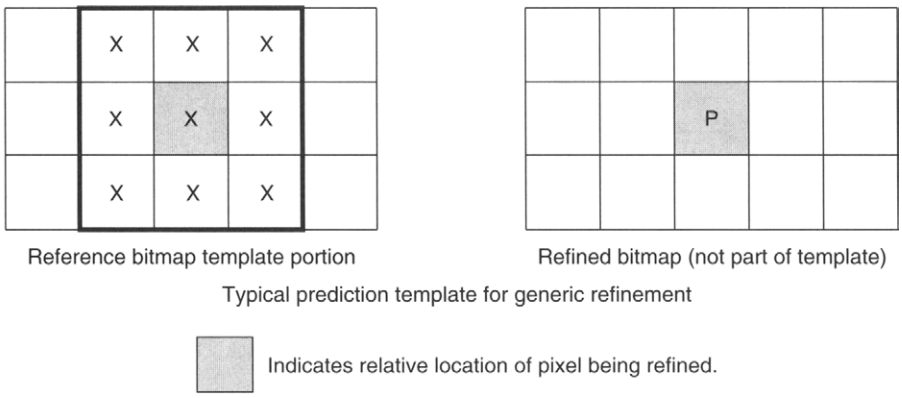


FIGURE 17.15
Template used for typical prediction in generic refinement. *X* indicates fixed template pixels and *P* indicates the pixel coded in the refined bitmap.

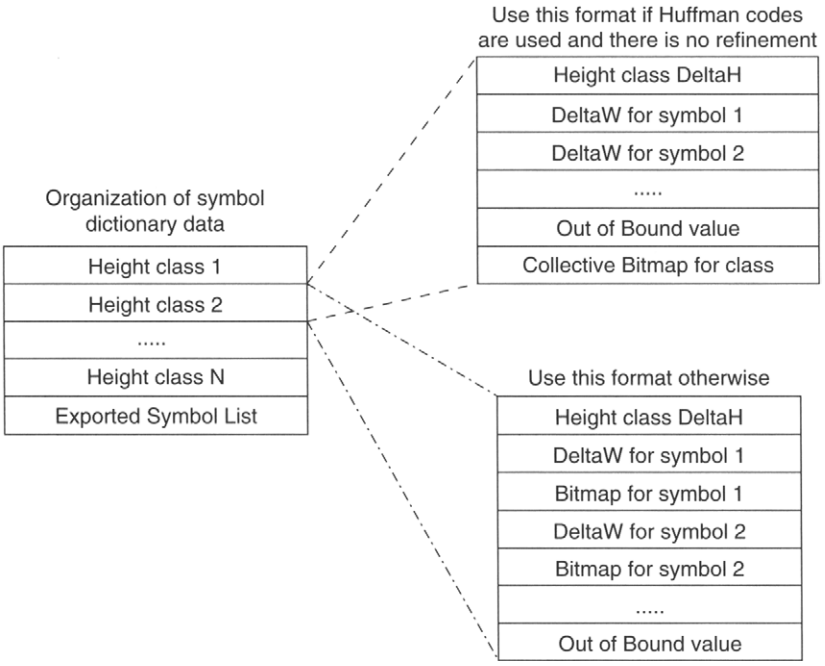


FIGURE 17.16
Symbol dictionary organization with height class data decoded in one of two possible ways.

as illustrated in Fig. 17.16. If Huffman coding is selected and there is to be no refinement or use of aggregate symbols, then one collective bitmap is encoded using the MMR selection of generic region encoding for the entire bitmap. The symbol width data is subsequently used to parse the bitmap into specific symbols.

If Huffman coding is not selected or if refinement or the use of aggregate symbols is used within a height class, then one of two forms of decoding is used. With no refinement or aggregation,

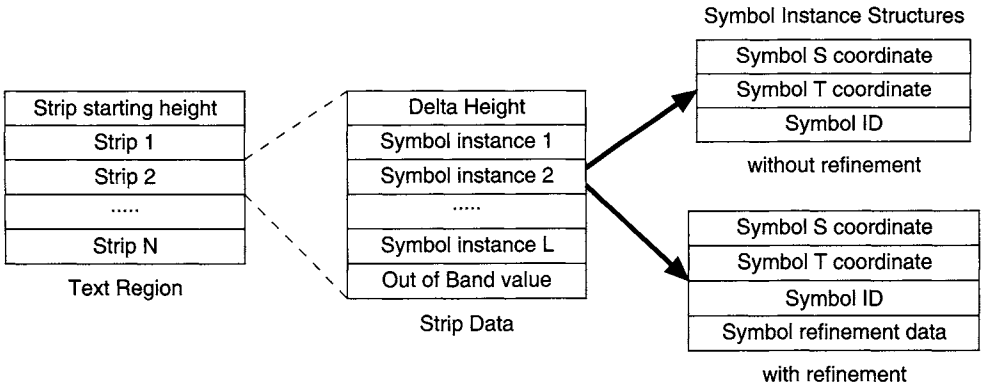


FIGURE 17.17
Text region data organization.

then generic region decoding is used. Otherwise, if aggregation or refinement is used, then one of two cases is decoded. If more than a single instance of previously defined symbols is used to define an aggregation, then text region decoding is used (this will be discussed in the next section). If only a single refinement is used, then a previously decoded symbol is refined using generic refinement decoding. In either case, a single bitmap is generated and placed in the dictionary. All height classes are decoded in this manner (note that *DeltaH* values can be negative). The final information decoded for the symbol dictionary is the exported symbol list. This list includes all the symbols that are available to other segments for decoding of text regions or other symbol dictionary entries.

17.3.2.4 Text Region Decoding

Text region decoding uses the indices in the symbol dictionaries to represent the bitmaps of text characters (or symbols). To do this, the S and T coordinates are decoded, along with the index of the dictionary entry for a given symbol and, if used, an optional refinement bitmap to slightly modify the dictionary entry and more closely or exactly represent the original symbol's bitmap. The S and T coordinates can be thought of as corresponding to the horizontal and vertical locations in a buffer, respectively (unless the symbols are transposed or a particular choice of reference corner determines otherwise—however, these parameters affect only the final orientation of the buffer). The data organization for text regions is indicated in Fig. 17.17. After the starting location of the first point is specified, each strip of data is decoded as a set of symbol instances until an out of band value indicates the end of that strip. Each symbol instance consists of an S and a T location update and a symbol index value. If refinement is not used, then this bitmap corresponding to this index in the dictionary is placed as specified. If refinement is used, then a refinement bitmap is decoded using generic refinement region decoding as specified previously.

17.3.2.5 Pattern Dictionary Decoding

Halftone regions are decoded using a pattern dictionary. To understand how this pattern dictionary is made available to the decoder consider the set of patterns illustrated in Fig. 17.18. Each of the entries in a pattern dictionary is positioned horizontally one after the next as shown to form a collective bitmap. This collective bitmap is decoded using generic region decoding. The number of patterns in the bitmap is given by $(GrayMax + 1)$, requiring $\lceil \log_2(GrayMax + 1) \rceil$ bits to represent each pattern, where $\lceil x \rceil$ indicates the smallest integer greater than or equal to x.

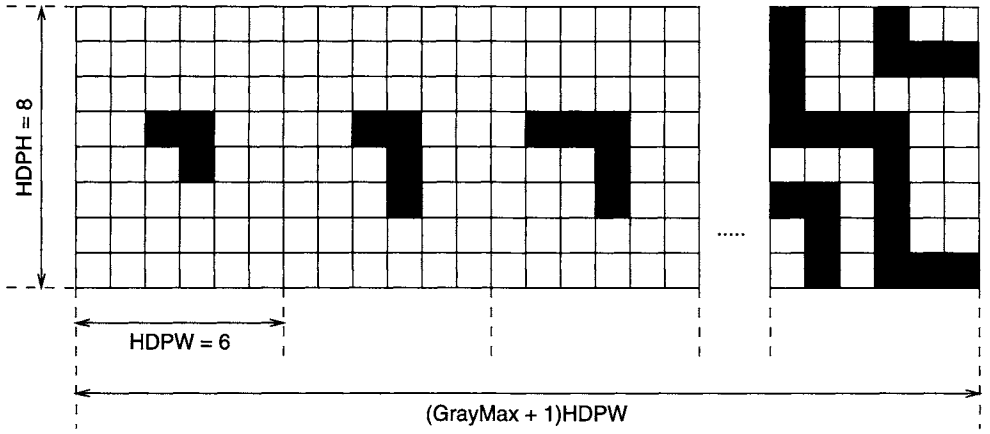


FIGURE 17.18
Collective bitmap for pattern dictionary entries.

GrayMax is a 4-byte unsigned integer, as are the width and height of each pattern in the dictionary (*HDPW* and *HDPH* from Fig. 17.18, respectively).

17.3.2.6 Halftone Region Decoding

The pattern dictionary is used to decode halftone regions. Figure 17.19 illustrates the relationship between a page buffer, a halftone region, and the halftone grid. The grid is specified by parameters X, Y, dX, and dY in the figure, along with the width and height of the pseudo-grayscale image that lies on the halftone grid. The X and Y parameters are 32-bit quantities, while the dX and dY parameters are 16-bit quantities. The least significant 8 bits for all of these parameters are interpreted as a fractional component. A bitmap can be used to skip certain entries in the halftone grid which are not needed. In addition, depending on the relative angle of the halftone grid and the halftone region, there may be several grid points which are not required since they fall completely out of the halftone region or even the page buffer.

The $\lceil \log_2(GrayMax + 1) \rceil$ bitplanes of the pseudo-grayscale image are extracted (least significant bit-plane first) from a single bitmap. The dimensions of this single bitmap are the number of vertical halftone grid points by the product of the number of horizontal halftone grid points and the number of bit-planes. Generic region decoding is used to obtain this bitmap, which is organized into a pseudo-grayscale image via gray coding. Within the halftone region, these $\lceil \log_2(GrayMax + 1) \rceil$ bit values are then used as the indices into the pattern dictionary. The selected pattern from the dictionary is placed at its grid location (these are indicated in the figure by the dashed boxes). Where these patterns overlap, a set of combination operations is possible. These include logical operations such as OR, AND, and XOR, as well as a REPLACE operation in which grid locations decoded at a later time simply replace bit values decoded previously.

17.3.3 Decoding Control and Data Structures

JBIG2 datastreams are broken into segments. The primary segment types include:

- generic region segment types (immediate, intermediate, and immediate lossless)
- generic refinement segment types (immediate, intermediate, and immediate lossless)
- text region segment types (immediate, intermediate, and immediate lossless)

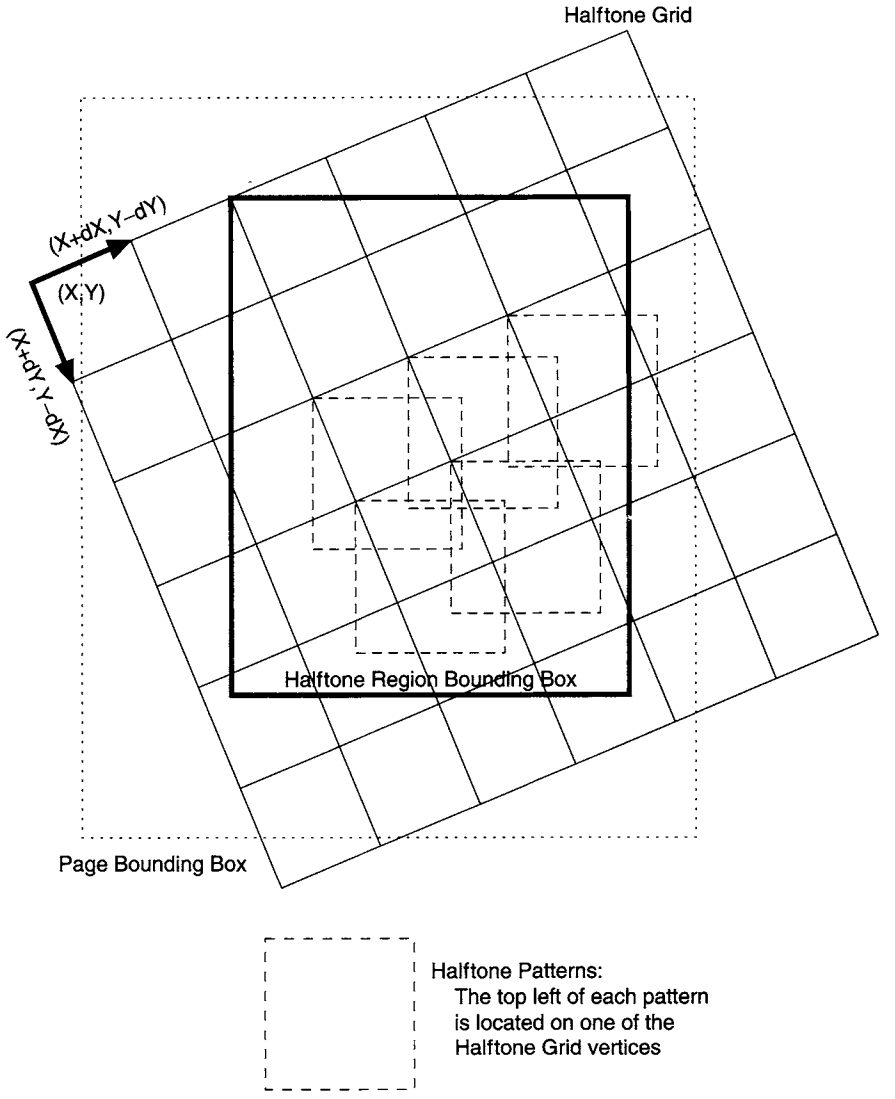


FIGURE 17.19
Halftone region grid for reconstruction of halftone region and representation of indices as a pseudo-grayscale image.

- halftone region segment types (immediate, intermediate, and immediate lossless)
- symbol dictionary segments
- pattern dictionary segments
- page information segments
- end of page, end of stripe, end of file, profiles, tables, and extension segments.

The final two groups of segment types control the decoding process. Most of these control segment names clearly indicate their function. As in the JBIG standard, it is possible to reduce required buffer sizes by striping page data, although this may come at the expense of reduced compression.

Within each segment there is a segment header, a segment data header, and segment data. This basic structure is indicated in Fig. 17.20. The segment header contains an ID (segment number),

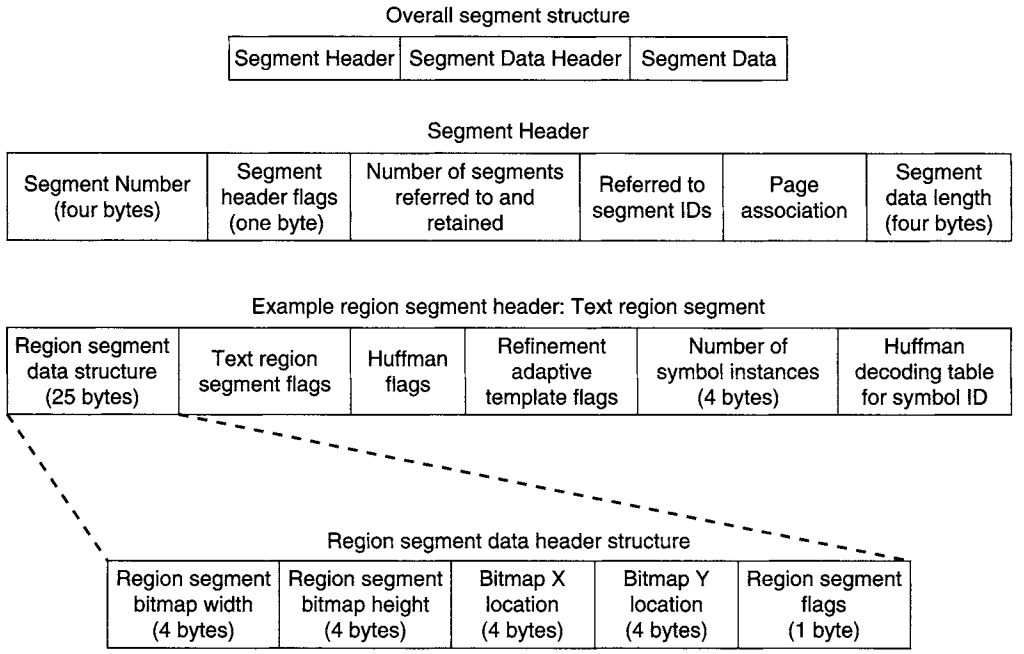


FIGURE 17.20
JBIG2 segment and segment header structure.

1 byte of flags that include an indication of segment type, and a count of other segments which are referred to in that segment (as well as an indication of whether these other segments will need to be retained for later use—this helps the decoder free unnecessary data from memory). Other segment header information includes the segment numbers of the other segments being referred to, a page number with which the current segment is associated, and the segment data length.

An example text region segment data header is also illustrated in the figure. Region segment information, such as the width and height of the segment's bitmap and its location (X, Y) in the buffer, is included in the region segment data header. Segment flags, Huffman coding flags, refinement adaptive template flags, and the number of symbol instances to be decoded by this particular text region segment are also included. Finally, if Huffman codes are to be used for decoding the symbol instance IDs, then this table is decoded from information included in the text region data header.

17.4 SUMMARY

In this section some lossless bilevel image compression results originally presented in [2] are given to allow comparison of the relative performance of JBIG and JBIG2, as well as the T.4 and T.6 facsimile standards. Table 17.1 contains lossless bilevel image compression results for four coders and three sets of images. The four coders are G3, G4, JBIG, and JBIG2, all used for lossless compression. The three image sets include nine 200-dpi images, nine 300-dpi images, and four mask images that were generated from segmenting a page of an electronics catalog [2].

While JBIG2 is slightly better in terms of compression than JBIG, both JBIG and JBIG2 provide significant improvement relative to the G3 and G4 facsimile standards. It should be noted that if

Table 17.1 Total Number of Bits in Compressed Lossless Representation of Bilevel Image Test Sets (from [2])

	Bilevel Images		
Coder	9-200 dpi	9-300 dpi	4 mask
G3 (T.4)	647,348	1,344,030	125,814
G4 (T.6)	400,382	749,674	67,116
JBIG (T.82)	254,223	588,418	54,443
JBIG2 (T.88)	236,053	513,495	51,453

lossy compression of some parts of the bilevel images is allowed, then the JBIG2 representations may be substantially further reduced in size.

17.5 REFERENCES

1. Sayood, K., 2000. *Introduction to Data Compression, 2nd ed.*, Morgan-Kaufmann, San Mateo, CA.
2. Malvar, H. S., 2001. Fast adaptive encoder for bi-level images. *Proceedings of the Data Compression Conference*, March 27–29, 2001, pp. 253–262.
3. ITU-T Recommendation T.82, 1993. Information Technology—Progressive Bi-Level Image Compression, JBIG Recommendation.
4. ITU-T Recommendation T.88, 2000. Information Technology—Lossy/Lossless Coding Bi-Level Images, JBIG2 Recommendation.