# *CellB Codec* 15

The Cell codec discussed in the preceding chapter is useful primarily in authoring applications. Its advantages are its fast decoding and the high quality of the images it produces, particularly on indexed-color frame buffers. The CellB codec, which derives from its Cell counterpart, is intended for use primarily in videoconferencing applications. It features a greater balance between the time spent compressing and decompressing images than the Cell codec and employs a fixed colormap. The CellB codec's strengths include

- Software compression at interactive rates
- Very fast decoding and display, especially on indexed-color frame buffers
- Low rates of CPU use
- Good quality output

The remainder of this chapter is divided into four sections. First, the chapter explains how the CellB compressor/decompressor works. Second, it explains briefly how to create a CellB CIS. Third, it discusses the CIS attributes that apply specifically to the CellB codec (as opposed to the general CIS attributes covered in the section "General CIS Attributes" on page 257.) Fourth, the chapter introduces the subject of accelerating the playback of CellB bytestreams. For further information on this subject, see Chapter 21, "Acceleration in XIL Programs."

# 15

## How the Codec Works

The CellB compressor works on $YC_bC_r$ images that conform to the guidelines set forth in CCIR Recommendation 601. The compressor performs *intraframe* compression by representing 4-by-4 blocks of pixels using cell codes. It performs *interframe* encoding using skip codes.

### Cell Codes

The images you compress using the CellB compressor must have a width and height that are multiples of four because the compressor works with 4-by-4 cells of pixels. For each frame, the compressor begins with the cell in the upper-left corner and then proceeds from left to right. The compressor processes rows of cells in this way, moving from the top of the frame to the bottom.

When the compressor encodes a cell without reference to a cell in a preceding frame, it uses a four-byte cell code to represent the content of that cell. This cell code specifies two colors and includes a 16-bit bit mask that indicates which of the two colors should be used to represent each pixel in the cell. See Figure 15-1.

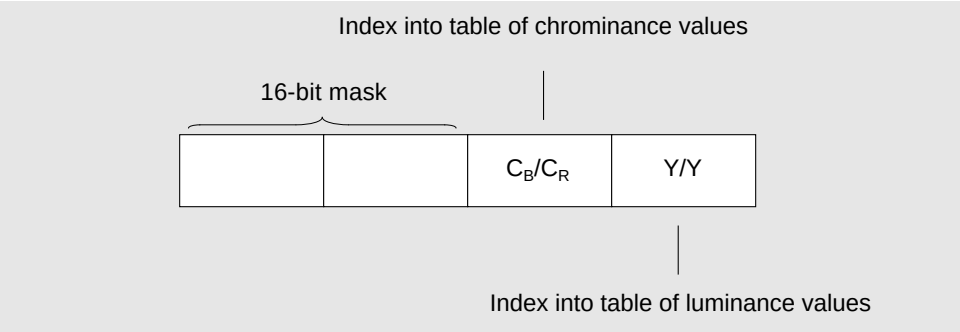


*Figure 15-1* Cell Code

The two colors are encoded as follows. The compressor calculates the average $C_b$ and $C_r$ values for the 4-by-4 block. Then, it calculates an index into a table of 256 vectors in which each vector looks like the one shown in Figure 15-2.

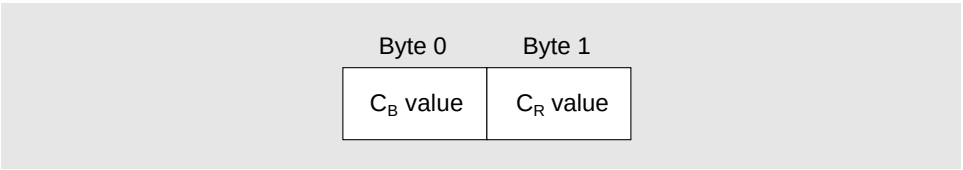


*Figure 15-2* Vectors in Chrominance Table

The values in the vector pointed to by the index are the pair of values in the table nearest to the mean $C_b$ and $C_r$ values for the cell. The compressor writes the index to this vector to the third byte of the cell code.

The compressor also analyzes the luminance values in the cell. First, it calculates the mean luminance for the cell. Second, it partitions the 16 luminance values in the cell into those values that fall below the mean and those that fall above the mean. Then, it calculates the average luminance in the two partitions.

After arriving at the average luminance values for the two partitions, the compressor calculates an index into a table of vectors of the form shown in Figure 15-3.

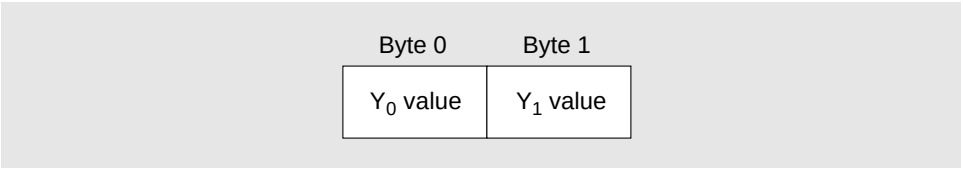


*Figure 15-3* Vectors in Luminance Table

The values in the vector pointed to by the index are the pair of values in the table closest to the average luminance values for the two partitions. The compressor then writes the index to this vector to the fourth byte of the cell code.

The first color for a cell consists of the first byte of the cell's luminance vector and the chrominance values in the cell's chrominance vector. The second color consists of the second byte of the cell's luminance vector and the same chrominance values.

The bit mask shown in Figure 15-1 is filled out in this way. Each bit in the mask is associated with a pixel in the cell. If the luminance value for a pixel is below the mean luminance value for the cell, its bit is set to 0. This pixel will be represented by the first color when the cell is decompressed. If a pixel's luminance value is above the mean, its bit is set to 1.

Because each cell code represents the values of 16 pixels using 32 bits, compression using cell codes alone leads to a compression rate of 2 bits per pixel. To better this compression rate, the CellB compressor uses skip codes to achieve interframe compression.

## *Skip Codes*

The CellB compressor encodes the first image in a sequence using cell codes exclusively. But after the first image, the compressor begins looking for cells in the current image that match—within a certain tolerance—the corresponding cells in the preceding image. Anywhere from 1 to 32 consecutive cells that match their counterparts in the previous image can be represented by a single 1-byte skip code.

The only restriction on the use of skip codes is that the cell at a particular set of coordinates can not be skipped indefinitely. There are a couple of reasons for this restriction. First, a user might join a videoconference that is already in progress. Cells represented by skip codes in the first image he receives will not be displayed correctly, and this problem must be corrected within a certain period of time. Similarly, if the CellB bytestream is being sent over an unreliable transport and a packet of data is lost, the period of the resulting error should be limited.

The policy concerning skip codes is that a particular cell must be updated at least every *n* frames, where *n* is an implementation-specific maximum. The exact value used is selected randomly for each cell each time that cell is encoded using a cell code. A random number is used to prevent the periodic bit-rate increase that might result were each cell to be updated at a fixed interval.

In a typical videoconference, about 80 percent of the cells in the average frame are represented by skip codes. This ratio leads to an average compression rate of about .8 bits per pixel.