Chapter 4 - Multimedia Compression

As we mentioned in the beginning of the last chapter, uncompressed video requires a significant amount of storage and transmission resources. For the HDTV quality video stream, we require approximately 1.5 Gigabits per second. A minute of such video would require approximately 10.4 gigabytes! Someone typing continuously at a rate of 70 words a minute would require approximately 61 years of continuous typing in order to consume the same amount of resources. In the remainder of this chapter, we will cover some of the basic image, video, and audio compression technologies that exist.

4.1 Image Compression

4.1.1 GIF

The Graphics Interchange Format (GIF) was introduced in the late 1980's by CompuServe as a way to compress digital images. GIF is pronounced "JIF". The GIF standard is a reduced color space, indexed image compression format.

The format allows for up to 256 unique colors to be represented in the image. Images that are greater than 256 colors need to be reduced to 256 colors before compression. This is accomplished through *color clustering* to pick a smaller set of representative set of colors and *dithering* to map pixels in the original image into the reduced set of colors. Dithering may result in the fine interleaving of similar colors across multiple pixels to achieve the appearance of other colors. For a given image, using fewer colors results in higher compression ratios because the probability of redundancy increases with fewer colors.

The header for GIF includes a magic cookie of either "GIF87a" or "GIF89a", image parameters, a color table, and the LZW compression parameters (e.g. minimum code size). The image parameters include the size and position of the image. The color table specifies the 256 or less color palette being used within the image. Each color is represented with 24-bits, 8-bits for each of red, green, and blue.

The pixels within the image are represented as pointers into the color table. Thus, all the pixels within the image have a range of [0-255]. To compress the data, GIF uses LZW compression on the array of color table indices.

GIF is extremely effective for images that do not have a lot of color. For example, logos for companies that have few colors, comic strips, and text are easily compressed with GIF. GIF does not do that well for images with large color dynamics or continuous tone images with color gradients. We will discuss this in section 4.3

4.1.2 PNG

After the decommissioning of the Internet from an education only network to one available for the commercial enterprise, the use of GIF and JPEG images saw a large uptick as most browsers at the time only supported the display of these two image formats. In a legal drama, Unisys (the holder of the LZW compression patent) and Compuserve (the implementer of GIF) came to an agreement to try and collect royalties of software that used GIF. After a maelstrom of activity, a number of people came together to define a new format that was not legally encumbered.

The Portable Network Graphics (PNG) format was created by a dozen or so people in 1995 and 1996. The group evaluated a number of possible approaches and ideas. At its core, however, was the LZ77 algorithm which (i) served as the basis of LZW and (ii) was not legally encumbered. In some regards PNG can be thought of as bridging the gap between an uncompressed format and GIF. It provides many of the same features as GIF but adds:

- Any number of colors – One of GIF's primary constraints is its 256 maximum color palette in order to achieve reasonable compression. PNG allows any number of colors to be used. If the user reduces the number of colors, better compression is achieved. Alternatively, with more colors, a larger file will result.
- Transparency – GIF allows for transparent backgrounds where pixels are either completely "on" or completely "off" (i.e., not displayed). PNG allows users to use an alpha channel that allows for varying levels of transparency. Each pixel can be represented as a <red, green, blue, alpha> tuple, where alpha specifies how transparent or opaque the pixel is.
- Gamma correction – this allows for colors to be adjusted depending on the particular display or printer being used to ensure constant color across devices.
- Interlacing – the interlacing algorithms was modified to provide better coarse detail to fine detail transitions

At its core, PNG consists of two major steps: Filtering and Compression. In the filtering stage, a pixel (shown as X in the figure to the right) can be encoded by itself as a pixel; with respect to A, B, or C; or through a combination. The net effect of this step is to improve the redundancy of the data. In particular, the pixel to pixel differences may be small, resulting in many values centered around 0. The second major step is using deflate (based upon LZ77) to compress the data. This is essentially the same as the LZW algorithm.

	B	C	
	A	X	

GIF and PNG under similar colors provide similar performance with PNG files being typically a bit smaller.

4.1.3 JPEG

The JPEG standard was developed by a consortium of industry and academic researchers in the late 1980's. JPEG stands for the Joint Photographics Experts Group. The main goal of the committee was to develop a general purpose compression standard for continuous-tone image applications (e.g., nature). Thus, the focus of the compression algorithm was to manage full-palette color compression. JPEG was standardized in 1991.

There were a number of goals of the standard including:

- Applicable to any continuous-tone digital source image
- Tunable compression rates which trade off image quality and resultant compressed image size
- Able to run on a range of CPUs in a reasonable amount of time.
- Support a number of modes of operation including sequential coding, progressive coding, lossless coding, and hierarchical coding. Sequential coding encodes the image from left-to-right, top-to-bottom in scan rows. Progressive coding stores the

image in a number of refines, allowing a coarse but not high-quality coding to be rendered quickly. Lossless coding allows for complete reconstruction of the image data. While not typically necessary for most image applications, some applications require the ability to store lossless data. Finally, hierarchical coding allows the image to be stored at progressively smaller resolutions. A majority of the JPEG codec implementations only support the first two modes.

Through an iterative selection and refinement process, the JPEG compression standard resulted in the selection of a DCT-based algorithm as the baseline sequential codec. For the purposes of this book, we will focus on the common sequential codec.

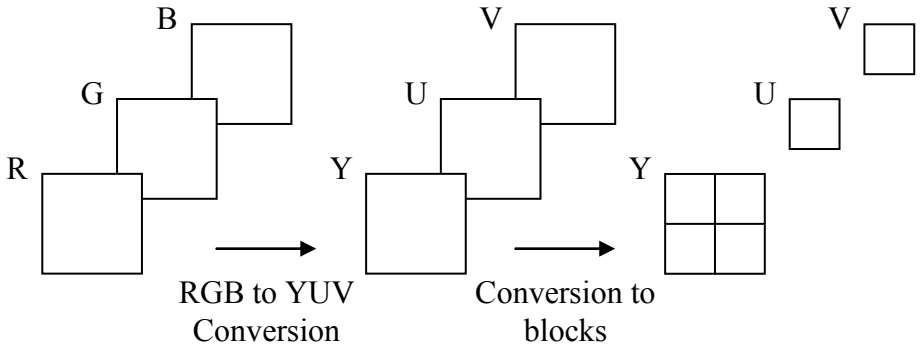
The codec consists of a number of steps, including:

- The image is split into 16x16 pixel *macroblocks* in the YUV color space
- Each macroblock is further divided into 8x8 pixel *blocks*.
- A DCT transform is performed for each block
- Quantization of coefficients is performed to reduce the amplitude of the coefficients (and to force more 0 coefficients)
- The resulting coefficients are then entropy encoded using differential coding, run-length coding, and Huffman (or Arithmetic) compression

We will describe these steps in more detail in the following subsections.

Macroblocks and Blocks

Macroblocks are the *minimum coding units* (MCUs) in the JPEG standard. For a given image, dividing the image into 16x16 pixel macroblocks results in three 16x16 arrays of image data typically in RGB format. Each of the macroblocks is then converted into the YUV color space. Once converted, the three 16x16 pixel Y, U, and V macroblocks are broken into 8x8 pixel blocks. The 16x16 Y macroblock is split into four 8x8 Y blocks. The 16x16 U macroblock is subsampled down to one 8x8 block. The 16x16 V macroblock is subsampled down to one 8x8 block. Thus, the macroblock consists of six 8x8 pixel blocks, four for Y and one each for U and V. A high-level diagram is shown below:



Note that the conversion into the six smaller 8x8 pixel blocks results in a 2:1 compression ratio (similar to that of lossless compression). The reason this works is that the human eye is much less sensitive to changes in the U and V channels than it is in the Y channel.

DCT Transform

Each 8x8 pixel block is transformed into the frequency domain using a discrete cosine transform. The main purpose of the DCT transform is that for images of reasonable size and content, the 8x8 block has relatively small high frequency components. Because of this, the DCT transform results in a majority of the frequency information in the upper left hand corner of the block. We will see why in the rest of this section.

The relation between the frequency and spatial domains can be summarized by:

$$F(u,v) = \frac{C_u}{2}\frac{C_v}{2}\sum_{0}^{7}\sum_{0}^{7} f(x,y)\cos\left[\frac{(2x+1)u\pi}{16}\right]\cos\left[\frac{(2y+1)v\pi}{16}\right]$$

$$C_u = \begin{cases} \frac{1}{\sqrt{2}} & if\ u = 0\ \&\ v = 0 \\ 1 & if\ u > 0 \end{cases} \qquad Cv = \begin{cases} \frac{1}{\sqrt{2}} & if\ v = 0\ \&\ v = 0 \\ 1 & if\ v > 0 \end{cases}$$

While the specifics of DCT transforms are beyond the scope of this textbook, it is necessary to understand the basic results of the transform. The rows of the DCT transformed block represent the amount of horizontal-ness of the block's pixels. The more horizontal lines in the block, the more the coefficients are going to be towards the bottom of the block. Similarly, the columns of the DCT represent the amount of vertical patterns that appear in the image. As a few examples:

155	155	155	155	155	155	155	155
155	155	155	155	155	155	155	155
155	155	155	155	155	155	155	155
155	155	155	155	155	155	155	155
155	155	155	155	155	155	155	155
155	155	155	155	155	155	155	155
155	155	155	155	155	155	155	155
155	155	155	155	155	155	155	155

1240	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

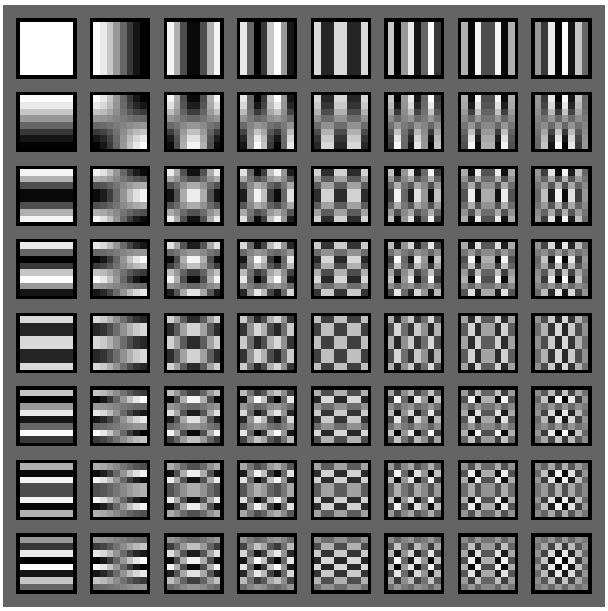
(a)

155	155	155	155	0	0	0	0
155	155	155	155	0	0	0	0
155	155	155	155	0	0	0	0
155	155	155	155	0	0	0	0
155	155	155	155	0	0	0	0
155	155	155	155	0	0	0	0
155	155	155	155	0	0	0	0
155	155	155	155	0	0	0	0

620	562	0	-197	0	132	0	-112
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

(b)

Each of the frequency domain coefficients represents a scaled pattern in the spatial domain. The upper left coefficient is called the DC coefficient and represents the average brightness of the entire block. The patterns can be thought of as basis functions (patterns) whose sums will represent the original spatial block.



This is why in (b), for example, all the DCT coefficients are in the upper row because there are now horizontal lines within the block. For most normal applications, the DCT operation results in a majority of the larger DCT coefficients being clustered in the upper left hand corner of the block.

Quantization

The next major step in JPEG compression is the quantization of the DCT coefficients. Quantization converts floating point numbers from the DCT operations into integers as well as creates coarser granularity of coefficients. The JPEG standard specifies default quantization tables that can be used. Alternatively, one can specify an application specific quantization but it needs to be included in the compressed file. An example quantization matrix is shown below.

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

For the quantization matrix above and the second DCT example, we have the following resultant matrix after quantization. The first two coefficients are obtained by dividing 620 by 16 and 562 by 11, respectively.

620	562	0	-197	0	132	0	-112
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Pre-quantization

38	51	0	-13	0	3	0	-2
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Post-quantization

Quantization generally results in many smaller coefficients being set to zero. In addition, JPEG allows a quantization factor to be specified, which allows users to trade off the visual quality and the resultant size of the image. For example, suppose we set the quantization factor to 4. Then each of the entries in the quantization matrix is multiplied by the quantization factor *before* dividing it into the DCT coefficient. Thus, for the example above, we end up with:

620	562	0	-197	0	132	0	-112
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Pre-quantization

9	12	0	-4	0	0	0	-1
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

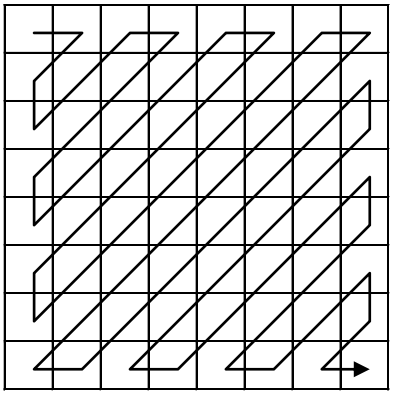
Post-quantization

There are two reasons increasing the quantization factor decreases the size of the file. First, more coefficients are typically forced to 0, leading to longer runs of 0 in entropy encoding. Second, the resultant values tend to be closer to 0. In the entropy encoding steps, the compression algorithm is biased towards using smaller number of bits for smaller coefficients. The reason increasing the quantization factor decreases picture quality is that (i) the actual values of the coefficients are quantized at a coarser granularity and (ii) the coefficients that have been reduced to 0 mean the basis function patterns no longer are represented in the compressed bit stream. In the decompression algorithm, the decompressor

uses the quantization factor (that is stored with the image) to reconstruct an approximate value of the original coefficient. Essentially, the value stored in the file is multiplied by the quantization factor and then multiplied by the corresponding quantization value from the quantization matrix to get an approximation of the original coefficient.

Entropy Encoding

The entropy encoding process finishes JPEG compression. The entropy encoding step does a number of things. First, because the DC coefficient tends to be the largest coefficient to be encoded but tends to be similar to surrounding blocks, it is differentially coded with respect to the last block of the same channel. The resultant differential is then Huffman encoded using either (i) a default table or (ii) an encoder-specified huffman table (which needs to be placed in the JPEG header). Second, for the remainder of the AC coefficients, they are zig-zag reordered, run-length encoded, and Huffman compressed. Because the coefficients tend to be clustered in the upper left corner, the coefficients are reordered so that most of the zero coefficients appear more clustered together. The zig-zag order is shown below:



To encode the coefficients, the run-length is encoded as the number of 0's in a row until the next *non-zero* coefficient. As an example, the following matrix:

9	12	0	-4	0	0	0	-1
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

The algorithm encodes the following (0,12), (4,-4), (22, -1), EOB, where EOB is a special end of block number. As shown by this example, the idea of the DCT is to convert the data

into a smaller number of non-zero coefficients. Each of the run-length components is compressed using a Huffman lookup table.

4.1.3.1 Encoding JPEG Files

Most JPEG files use the *JPEG File Interchange Format (JFIF)*. JFIF is essentially a wrapper around the JPEG format that allows JPEG bitstreams to be used across a wide number of platforms and applications. The basic format of the JFIF header are:

byte		
1-2	Start of Image (SOI) Marker	0xFFD8
3-4	APP0: App Segment	0xFFE0
5-6	Length	Length starting at this byte
7-11	Identifier	0x4A46494600 (null terminated "JFIF")
12-13	Version	Typically 0x0101 (version 1.01)
14	Units	0: none; 1: inch; 2: cm
15-16	Xdensity	Horizontal pixel density
17-18	Ydensity	Vertical pixel density
19	Xthumbnail	Thumbnail size (x direction)
20	Ythumbnail	Thumbnail size (ydirection)
21-(20+3n)	Thumbnail	Thumbnail data (R, G, B, R, G, B, …) *n* = Xthumbnail * Ythumbnail

This is followed by the detailed JPEG file data. There are three main components, each headed by a unique marker:

Quantization Marker	0xFFDB
Quantization table used	Defines the quantization table
Huffman Table Marker	0xFFC4
Huffman table used	Defines the Huffman table used
Start of Frame Marker	0xFFC0
Rest of image data	Actual DCT, quantized, zig-zag reordered, and Huffman compressed data

The quantization marker and data as well as the Huffman marker and data are optional.

4.1.3.2 Managing JPEG Image Quality

The quality and size of the compressed file in JPEG is highly dependent upon the actual content of the image. Images with many edges and corners result in many high frequency components, which need to ultimately be encoded to achieve good image fidelity. Outdoor images trees, mountains, and sky tend to result in high compression due to the overall smoothness of naturally occurring data. As previously mentioned, the main mechanism to support the quality and size of the compressed JPEG file is the quantization value. The larger the quantization factor, the smaller the resultant file and the more artifacts that are introduced into the decompressed image. Such artifacts tend to occur around sharp edges (high frequencies) in the image.

Generally speaking, JPEG results in compression ratios of:

- 0.25 – 0.5 bits per pixel – good quality
- 0.5 – 0.75 bits per pixel – good to very good quality
- 0.75 – 1.5 bits per pixel – excellent quality
- 1.5 – 2.0 bits per pixel – usually indistinguishable from the original

This means that JPEG can result in nearly perfect image quality while still achieving as 12:1 compression ratio (using 2 bits per pixel). Thus, it is much more compressed than using a lossless compression algorithm.

4.2 Video Compression

4.2.1 Video Compression Overview

There are a number of video compression algorithms that have been proposed and standardized for video compression. While there are a number of formats that are propriety such as Real Network's video compression and Windows Media, we will primarily focus on standardized video compression formats. There are two main organizations that have been involved with standardizing video compression algorithms. The *International Telecommunication Union* (ITU) is responsible for the H.*** formats, while the *International Standards Organization* (ISO) and the *International Electro-Technical Commission* (IEC) have been responsible for the various MPEG standards that have been developed. The ITU is primarily focused on interactive communications and telephony-based applications. The ISO and IEC missions are more generic in nature. The following table describes the applications and standards that have been designed:

Application	Standard	Target bit rate
Low-bitrate video and video phone	H.263	< 22 kbps
	MPEG-4	< 64 kbps
Video conferencing and video phone	H.261	p x 64kbps p={1..30}
Storage of CIF quality video	MPEG-1	1.5 Mbps
All digital TV and HDTV	MPEG-2	Up to 20 Mbps

While an exhaustive description of all video compression algorithms that exist today would be extremely long, we will describe several of the standards in more detail.

Generally, there are two types of video streams that can be generated. The first is a *constant bit-rate video* stream. In constant bit-rate (CBR) video, the video rate over several second intervals is more or less constant. This makes planning for resources such as network bandwidth potentially easier to manage. For example, the H.261 video standard generates data to fit within a constant rate of multiples of 64 kbps. To accomplish CBR video, the encoder must adaptively alter the quantization value in order to ensure a constant bit rate. The key drawback of such an approach, however, is that the quality of the video stream varies over time. To achieve CBR video, two approaches are used. *One pass* encoders alter the quantization levels dynamically and in real-time, as necessary for video conferencing applications. While requiring less resources, the quality tends to vary more compared to their two pass counterparts. *Two pass* encoders typically encode the video stream with a constant quantization level on the first pass. Then during the second pass, the stream is further massaged to fit within the CBR bandwidth. These types of encoders are typically employed for storing data to a disc (e.g., MPEG-1 stored on CD-ROM or MPEG-2 to DVD).

The second type of video streams are called *variable bit-rate (VBR)* video streams. As the name implies, the video generated has variable requirements over time. With VBR video, which is sometimes called constant quality video, the video is encoded with a fixed quantization level. While it is the easiest to implement, it is harder to support from a systems perspective because the peak bandwidth rate for the video can be extremely large. Most implementations of VBR encoders today allow a maximum burst size and the use some quality degradation (i.e., quantization increase) to keep the maximum bit rate underneath some maximum value.

4.2.2 Motion-JPEG Compression

To compress video, one of the obvious was to do so is to repeatedly apply an image compression algorithm to the individual frames of the video. A common, but not standardized format, is the Motion-JPEG (M-JPEG) compression algorithm. There are a few minor differences between the JPEG format and the common Motion-JPEG formats. The most important is that each frame of the M-JPEG stream starts with the SOF marker that we described in the JPEG compression section. Thus, all the header information, quantization tables, and Huffman tables do not need to be repeated for every image in the M-JPEG stream.

The chief advantage to M-JPEG is the relatively simple algorithm needing to be implemented to create M-JPEG streams. Using an existing software JPEG implementation or JPEG compression chip, one could easily extend it to support M-JPEG compression. Furthermore, because the algorithm is applied to only one frame of data at a time it is generally considered to be space and computation efficient when compared to other video compression algorithms such as MPEG. The primary disadvantage, however, is that it does not take advantage of the large amount of redundancy that exists between two adjacent frames of video. This is the one area that the standardized video compression algorithms aim to achieve larger compression ratios. Before describing the video compression standards, however, we first provide a basic overview of *block-based motion compensation* techniques.

4.2.3 Block-based Motion Compensation Techniques

Block-based motion compensation techniques attempt to find, for a given 16x16 macroblock of data, a close visual match to a *reference* frame. The main idea of such compensation is to find an area within a reference frame *that has already been encoded*. If a close enough match is found, then the current 16x16 macroblock needing to be encoded can just make a reference to the already encoded macroblock. There are two main components of block-based motion compensation (i) defining what a match is and (ii) how to find the matching block.

4.2.3.1 Matching Criteria

There are a number of *matching criteria* that are used in searching for close visual matches in a reference frame including: Mean Square Error, Mean Absolute Difference, and Maximum Matching Pel Count. The mean square error technique uses the following criteria to evaluate a potential block in a reference frame offset by d_1 and d_2 (the x and y motion vectors, respectively).

$$MSE(d_1, d_2) = \frac{1}{N_1 N_2} \sum_{(n_1, n_2) \in \beta} [s_{current}(n_1, n_2) - s_{reference}(n_1 + d_1, n_2 + d_2)]^2$$

MSE does a pixel by pixel squared differentiation between the current macroblock and the candidate block. For a 16x16 pixel macroblock N_1 and N_2 are 16. n_1 and n_2 range from 0 to 15 and all combinations of n_1 and n_2 are computed. This gives the mean square error for a particular match. For all combinations of d_1 and d_2 that are searched, the one that has the minimum MSE is chosen.

Similarly, the mean absolute difference matching criteria is defined as:

$$MAD(d_1, d_2) = \frac{1}{N_1 N_2} \sum_{(n_1, n_2) \in \beta} \left| s_{current}(n_1, n_2) - s_{reference}(n_1 + d_1, n_2 + d_2) \right|$$

The difference between MAD and MSE is that MSE gives a higher weight bias against pixels that might be far off.

Finally, an even simpler algorithm is to simply count the number of pixels that fall below some threshold of closeness to the match being evaluated. This can be defined as:

$$Match(n_1, n_2, d_1, d_2) = \begin{cases} 1 & if \ \left| s_{current}(n_1, n_2) - s_{reference}(n_1 + d_1, n_2 + d_2) \right| < T \\ 0 & otherwise \end{cases}$$

$$MPC(d_1, d_2) = \sum_{(n_1, n_2) \in \beta} Match(n_1, n_2, d_1, d_2)$$

The MPC algorithm obviously favors near exact matches (i.e. ones whose pixels are less than T away from the candidate pixel) over anything else.

4.2.3.2 Motion Estimation Searching

Finding the closest match in a reference is much easier said than done. Suppose we have a 720x480 pixel DVD video stream. For a given 16x16 pixel block to match on pixel boundaries and a reference frame, there are (720-16)*(480-16) or 326,656 possible motion

vector combinations for the match. Given that each combination requires the evaluation of at least 256 comparisons due to the 16x16 pixels, this results in 83.6 million pixel difference operations that need to computed for just one macroblock! Thus, to do motion compensation for the entire frame which consists of (720/16)*(480/16) or 1350 macroblocks, this results in 113 billion differences that need to be calculated *per frame*!

There are several techniques that can be used to help limit the amount of searching: limiting the search range and using subsampled searches.

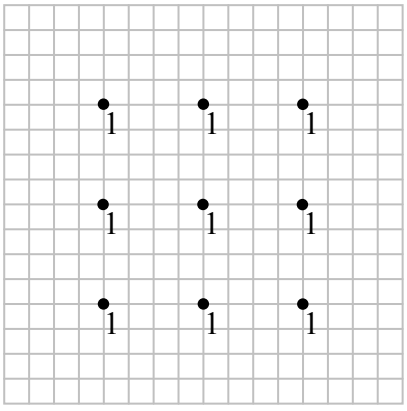
Limiting search range

One obvious way to minimize the amount of work is to limit the area within the reference frame that you look for a match. The idea of limiting the search range is that for a given object in the video, it can only move a certain amount of distance between frames. Assuming, the object provides the best visual match, the area necessary to search for such a match should not be that far. Suppose we limit the motion vectors to -16 to 16 pixels away in both the x and y direction. Assuming, the match has to be within a frame (i.e., it cannot go over the edge of the reference frame), then the 83.6 million pixel difference operations per macroblock can be reduced to 262k difference operations for non-edge macroblocks, 131k for edge macroblocks, and 65k for corner macroblocks. Thus, two orders of magnitude fewer comparisons need to be made. This still, however, requires approximately 334 million differences per frame.

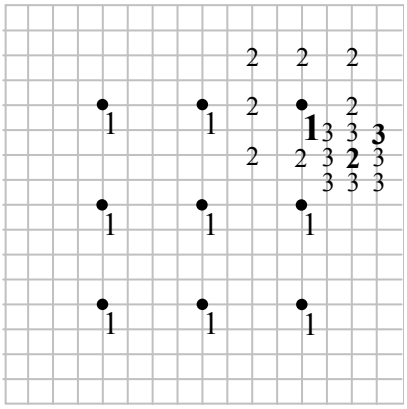
Subsampled searching

Another mechanism to reduce the amount of data searched in the motion compensation process is to perform a subsampled search. One simple subsampled search is to perform a *granular matching algorithm*. Here, instead of evaluating every pixel in the block, one could simply evaluate every *n*th pixel. If every other pixel is evaluated, then the amount of differences that need to be computed gets cut by 4. There are two popular searching algorithms that are widely used in video compression implementations (i) the Three Step search and (ii) a Cross search.

The *Three Step Search* (TSS) algorithm was introduced by Koga et. al in 1981. In the Three Step Search, an initial maximum search range is picked. For our examples, we will use 4. Given a macroblock whose upper corner is located at offset (64, 64), motion vectors of {(0,0), (-4,-4), (-4,0), (-4,4), (0,-4), (0,4), (4,-4), (4,0), (4,4)} are evaluated. This is shown below:

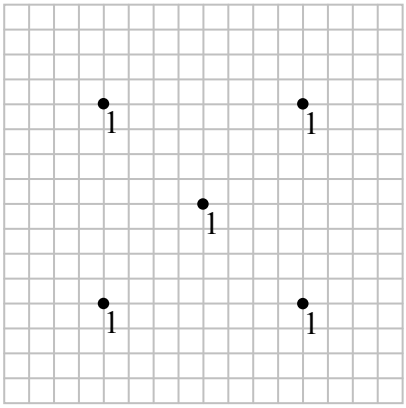


The one motion vector that yields the best match is chosen as the center of the subsequent search *but with half the search range.* Suppose, the upper right "1" has the best match of the 9, then the next set of motion vectors that are tried are {(0,0), (-2,-2), (-2,0), (-2,2), (0,-2), (0,2), (2,-2), (2,0), (2,2)} *from the upper right* "1". The process continues for three steps or until 9 adjacent pixels are evaluated. A sample process is shown below:

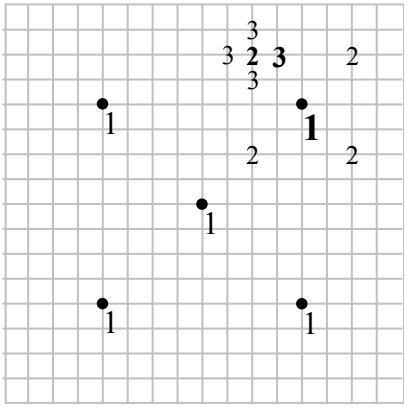


In step 2, the lower right match was the best, and in step 3, the upper right yielded the best result. The final motion vector for this search would then yield (7, -3). Obviously, just a few calculations are being used per search. The search is therefore heuristical in its performance. Using this search for one block reduces the number of difference calculations from 83.6 million to just 6400, several orders of magnitude smaller! For the entire DVD video frame, the cost is approximately 8.6 million.

The *Cross* Search algorithm is similar to the TSS algorithm in that it logarithmically searches for the best candidate match. Introduced in 1990 by Ghanbari, the algorithm starts the same as the TSS algorithm, except rather than having a 9 square pattern, it uses a "X" pattern as shown below:



The algorithm continues by finding the best match of the 5 points. Using that point, it cuts the search space in half and then centers itself around the best match found. This continues until the spacing is 1. When the spacing is 1, if the best match is in the upper right or lower left, then the algorithm uses a "+" pattern rather than an "X" pattern. If it is in the upper left or lower right, then the algorithm uses the "X" pattern. An example is shown below:



For this algorithm, the number of computation required to find a candidate motion vector is reduced from 83.6 million differences to 3328 difference operations. For the entire frame, this results in approximately 4.5 million operations.

Because the TSS algorithm and Cross algorithm subsample the reference frame, it is possible that the best match is never examined. In practice, however, these algorithms are able to find a suitable enough match that their computational savings far outweigh the slightly better match that may (or may not) exist elsewhere in the reference frame. Finally, an application may determine a certain threshold of "closeness". If any matching criteria returns something better than the threshold, the rest of the search can be skipped.

4.2.4 H.261

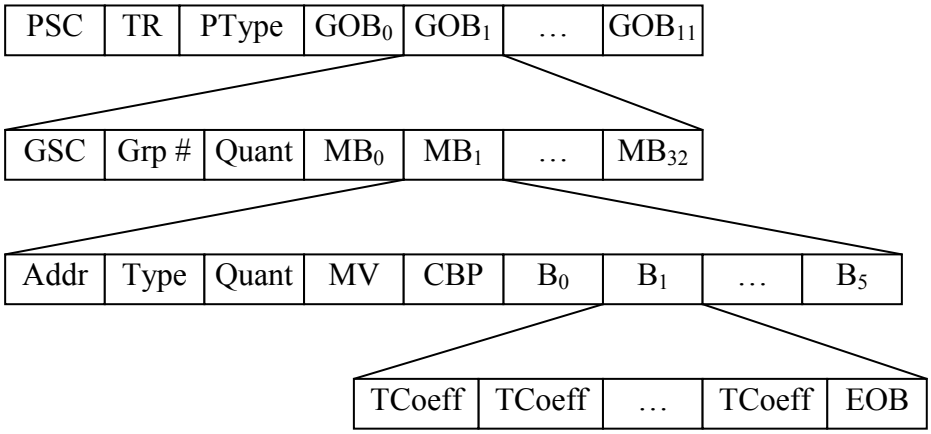
The H.261 standard was originally designed for the transmission of digital video over multiples of traditional phone circuits (64 kbps). The origins of H.261 came from the Specialist Group on Coding for Visual Telephony, a group formed in 1984. The H.261 standard was frozen in Mar. 1989. The standard calls for the transmission of p x 64kbps video where p varies from 1 to 30. For one 64 kbps channel ($p = 1$), the H.261 standard is meant for low quality videophone applications. For $p > 6$, the standard is meant to support generic video conferencing services. For video, the *common interchange format (CIF)* was agreed upon. CIF video frames are 352 x 288 pixels in size. Later on, QCIF was also adopted (176 x 144 pixels in size). H.261 is a DCT-based algorithm, whose basic compression item is the macroblock (similar to JPEG compression).

H.261 specifies two types of frames: *intra-coded* and *inter-coded*. Intra-coded (or I-frames) are independently decodeable and do not need any other additional information to decode. Inter-coded (of P-frames) are coded with respect to a previous frame, the reference frame, in order to maximize coding efficiency. The macroblocks within P-frames can be either intra-coded if an sufficient match is not found in the reference frame or can be an inter-coded frame if a suitable match is found in the reference frame.

H.261 Structure

The H.261standard is layered. At the highest layer are *pictures*, which are 352x288 pixel images. Each picture is broken into 12 *group of blocks (GOBs)* of size 176x48 for CIF-sized video. Each GOB, then contains 33 *macroblocks* of information. Each macroblock, in turn,

contains 6 8x8 pixel *blocks* (4 for the luminance channel and 1 each for both chrominance channels as in JPEG). This structure is shown below:



At the picture level:

- PSC stands for Picture Start Code, a byte-aligned marker
- TR is a temporal reference
- Ptype is the picture type (I or P frame)
- GOB_i is a group of blocks

At the GOB level:

- GSC is a GOB Start Code, a byte-aligned marker
- Grp# is a group number so that entire GOBs can be skipped (e.g. no change in the image at all)
- Quant is the quantization value for the DCT
- MB_i are the various macroblocks that make up the GOB

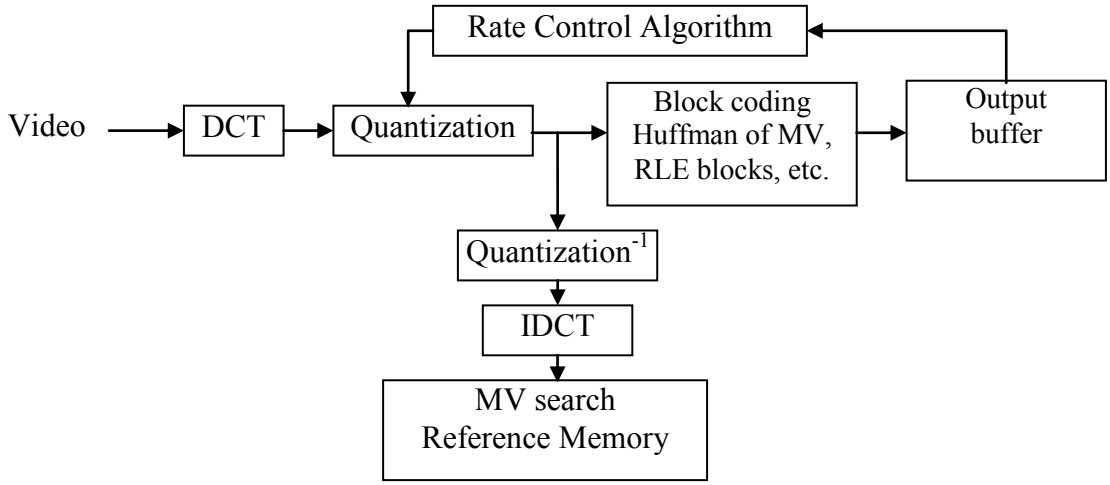
At the Macroblock level:

- Addr is the address of the macroblock within the GOB
- Type is whether the block is intra or inter coded (for P-frames)
- Quant is the quantization value if it is being changed
- MV is the motionvector for the macroblock
- CBP stands for coded block pattern. It is a 6-bit bitmap of blocks b0 to b5 and tells whether or not the block is coded or not. As an example: 100011 means only the first luminance block is encoded and the two chrominance blocks are encoded.
- B_i are the various blocks that make up the macroblock

As in JPEG, the block consists of a number of run-length encoded coefficients as in the JPEG algorithm and is terminated with an EOB marker.

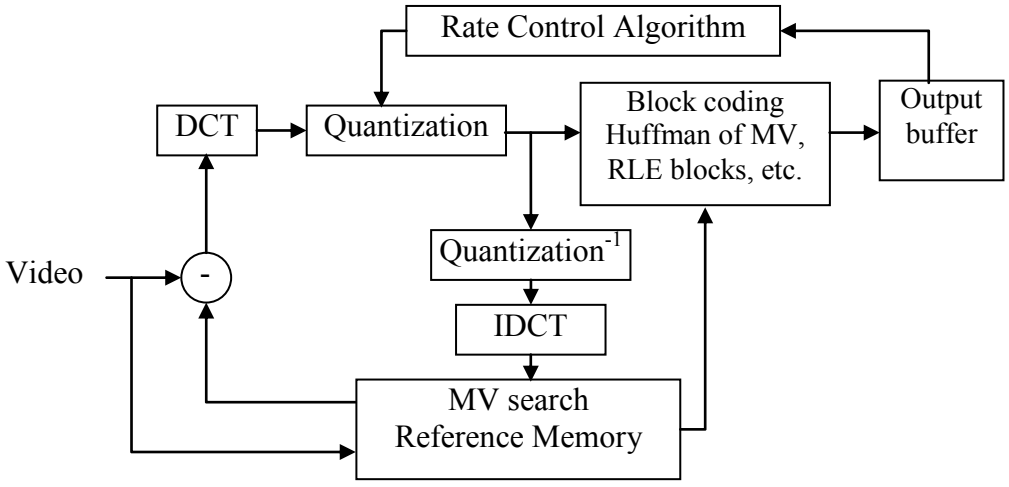
H.261 System Diagrams

At a high-level, the H.261 video standard can be constructed with a number of system components. This high-level diagram for the Intra-coded frames is shown below:



In this figure, the DCT, Quantization, Block coding, and output buffer are the basic compression steps (similar to that of JPEG). The Quantization^{-1}, IDCT and MV search boxes are for motion compensation. The Quantization^{-1} and IDCT are necessary because it reconstructs what the decoder will ultimately have, allowing the motion estimation and decompression to be more accurate. Finally, the rate control algorithm is used to adaptively manage the bit-rate.

The high-level diagram for P-frames is shown below:



The main difference is the video is fed into the MV search block, which finds the best motion vector per macroblock. It is the subtracted from the video, whose residual is then processed as normal. The motion vectors are also passed for encoding into the bitstream. Note, this is a pipelined process per macroblock.

H.261 Rate Control

Because H.261 is designed for multiples of 64 kbps, the output of the compression algorithms needs to keep the output around the channel allocated to it. H.261 does not specify an exact algorithm for managing the rate control algorithm. The difficulty with implementing H.261 rate control is two-fold. First, the data rate being generated at any time is dependent upon the quantization factor and the contents of the video itself. The latter makes it more difficult to control the amount of data being generated. Second, the rate control algorithm needs to work in real-time. Therefore, multiple passes for a video stream in order to fit it into the available resources may not be possible.

The basic technique rate control algorithms typically use is based upon monitoring an output buffer from the video coder. The output buffer is being drained by the network at a constant rate, say 128 kbps. If the amount of data being generated is greater than 128 kpbs, then the amount of video data sitting in the buffer will grow. If the amount of data being generated is less than 128 kbps, then the amount of data in the buffer will shrink as the network is draining data faster than the video codec is generating it. To match the video data rate with the network, as the amount of video data within the buffer grows, the quantization value can be increased, leading to a higher compression factor being used. As the amount of video data within the buffer shrinks, the quantization factor can be reduced to maximize the quality of the video.

4.2.5 H.263

H.263 is a newer version of the H.261 standard. The core algorithms for the standards are very similar. H.263 specifically targets low data rate video compression. H.263 was adopted in 1996 as a standard. Some of the differences include:

- More formats. H.263 specifies a number of formats including Sub-QCIF (128x96), 4CIF (704x576) and 16CIF (1408x1152). Support for SQCIF and QCIF are required by H.263. The other formats (CIF, 4CIF, and 16CIF) are optional in the standard.
- Half-pixel motion estimation. Half pixel motion estimation allows for better motion compensation at the expense of a larger search space.
- B-frames. B-frames (as in the MPEG standard) are optional in H.263

4.2.6 MPEG-1 Compression

At about the same time the JPEG format was being standardized, it was obvious that with digital imaging becoming more commonplace that digital video was not far behind. The Motion Pictures Experts Group (MPEG) was formed to develop a standard for the compression of VHS quality video and its associated audio into a 1.5 Mbps stream. The main reason for 1.5 Mbps is that it was thought to be a reasonable data rate for both storage and transmission. The standard 1x data rate for CD-ROM is approximately around this value. It is important to note that the MPEG standard defines the format of valid MPEG compressed streams, but does not specify an exact algorithm to get there. A sample implementation is provided with the standard, however.

The MPEG standard consists of three main components:

- *MPEG-Video* describes the coding of the video content for MPEG

- *MPEG-Audio* describes the audio compression format. The popular MP3 audio compression technology is part of this substandard.
- *MPEG-System* describes how to put the audio and video together. Specifically, it describes synchronization and multiplexing of multiple compressed audio and video streams.

Arriving at a Standard

One of the main goals of the MPEG group was to avoid having multiple incompatible *de facto* standards. Unfortunately, the way the Internet evolved, the distribution of video has been built around multiple proprietary standards as well as the MPEG-based standards. To arrive at the standard, the MPEG group held a competition in 1988 and 1989 to allow participant companies to come up with proposals for the MPEG-1 compression standards. This included companies such as AT&T, C-Cube, IBM, Mitsubishi, NEC, and Sony. After the competition, the MPEG group took the best of all the proposals and merged them into a single coherent standard. The standard was finalized in 1991.

As part of the competition, the designers of proposed standards had to support the following feature requires:

- Random access – any frame should be accessible within a limited amount of time. The target for random access was ½ second. This implied that the coding of the video required the ability to start decompression somewhere in the middle of the video stream, potentially hurting the ability to take advantage of temporal redundancy.
- Fast forward / reverse searches – the user should be able to scan through the compressed stream in the forward and reverse directions. Some could argue that this is even more demanding form than random access.
- Audio / video synchronization – an explicit mechanisms needed to keep the audio and video synchronized.
- Robustness to error – media and communications are not always 100% reliable. The compression scheme should be able to tolerate some media or communications errors without catastrophic failure (i.e. inability to continue decompressing the data).
- Coding / decoding delay – the proposed standard needed to support coding and decoding delays set by the application. The target for this delay was set to one second.
- Editability – the format should support the ability to modify and construct editable units of video coded with reference to themselves. Editability is easier to achieve with formats that support random access.
- Cost tradeoffs – the proposals would be evaluated to see if the decompression algorithm could be implemented in a small number of chips (for 1990). This would allow simple machines to decompress such video with relatively modest equipment costs.

A Difficult Challenge

The designers of the MPEG standard had a difficult challenge. On the one hand, the quality requirements demanded a high compression ratio that was not achieveable with intra-frame coding alone. On the other hand, random access is best achieved with intraframe

coding because every frame depends only upon itself. This meant that the proposers of the MPEG standard had to delicately balance inter-frame coding for compression efficiency and intra-frame coding in order to provide random access.

4.2.6.1 MPEG Overview

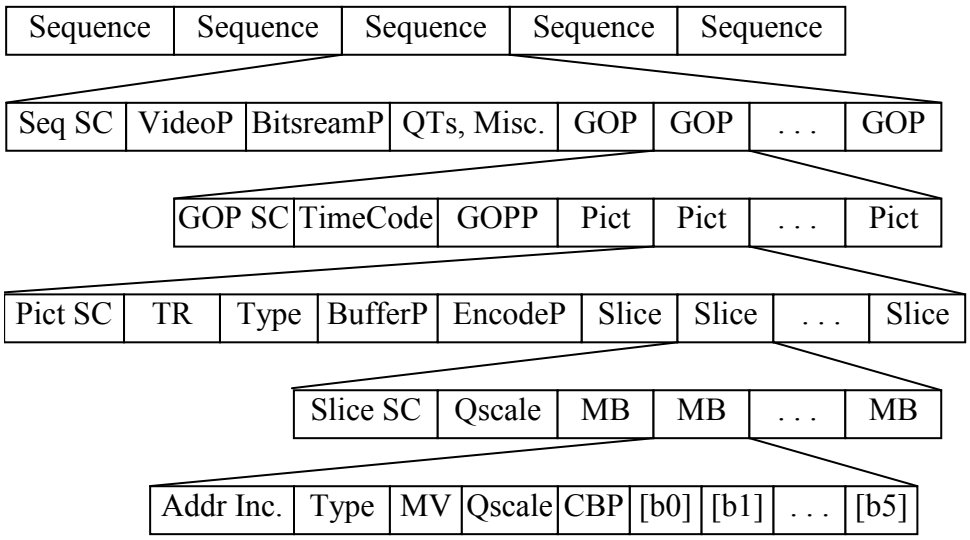
The underlying DCT transform, quantization, zig-zag reordering, run length encoding, and Huffman compression techniques used in MPEG are similar to those used in JPEG image compression. The main difference is what happens to a block of pixels *before* the DCT transform and subsequent steps. To maximize the compression ratio, MPEG-1 compression provides for block-based motion compensation to reduce temporal redundancy between frames of video. Block-based motion compensation attempts to find the best match for a macroblock to *reference* frames.

MPEG-1 supports three types of compressed video frames that use *reference* frames in different ways. Intra-coded frames or I-frames are coded without reference to other frames. That is, they are independently coded can be decoded without any additional information outside the frame. Predictive-coded frames or P-frames are coded with respect to a *previously captured reference frame*. For each macroblock within the P-frame, a suitable match from a previously encoded I or P frame is searched for. As a result, the P-frame requires one additional frame to be present in order to be decoded. Bi-directional-coded frames (B-frames) are coded with respect to both a *previously captured reference frame and a future reference frame*. For each macroblock within the B-frame, a suitable match is found through combinations of forward and reverse references to either I or P frames. Obviously, the future reference frame needs to have already been captured to do so. There are a number of advantages of B-frames. First, they deal better with uncovered areas. For example, on a panning shot, some of the video may exists in a future reference frame but not the past. Second it has better statistical properties in coding by minimizing the introduction of noise across two reference frames. Finally, each macroblock has many more reference coding options that using only predictive coding. Because I-frames are completely coded and do not take of any temporal redundancy, they are typically the larges. P-frames are the next smallest video frames and B-frames are typically the smallest.

In the remainder of this subsection, we will describe a number of components in MPEG video compression.

4.2.6.2 MPEG-1 Structure

MPEG-1 video streams are constructed of a number of layers. At the highest layer, an MPEG-1 video stream is constructed of a number of *sequences*. Each sequence consists of a number of *groups of pictures* (GOPs). GOPs consist of anywhere from 6 to 20 *frames*. While the MPEG-1 standard does not specify an exact number of frames per GOP, most first generation coders used a fixed number of frames per GOP. The GOP is the lowest-level randomly addressable entity within MPEG-video. Each frame can be one of the three types of frames: I, P, or B. Each frame consists of a number of *slices*. Slices are the primary mechanism for which MPEG systems recover from transmission or bit-errors. Each slice consists of a number of *macroblocks* which encode the actual information contained within the stream. We will describe the macroblock coding in more detail later. The overall structure is summarized below:



At the Sequence level:

- Seq SC – is a byte-aligned sequence start marker (0x000001B3)
- Video and bitstream parameters – include video information such as the horizontal and vertical size of the video, the aspect ratio, picture rate, bit rate and other parameters
- QTs and misc – include the quantization tables if they are specified by the encoder. It also includes user data.
- GOP – are the Groups of Pictures that make up the sequence

At the GOP level:

- GOP SC – is a byte-aligned GOP start marker (0x000001B8)
- Time code – is the time synchronization data for the first picture in the GOP.
- GOPP – are several GOP parameters such as whether or not the GOP is closed.
- Pict – are the individual frames that make up the GOP.

At the Picture level:

- Pict SC – is a byte-aligned sequence start marker (0x00000100)
- TR – is a temporal reference that allows the decoder to reorder frames properly for output.
- Type – is the frame type (e.g. I, P, B, or D frame)
- BufferP and EncodeP – are parameters for decoder buffer control and other encoding controls
- Slice – are the slices that make up the picture. Each picture must have a minimum of at least one slice.

At the Slice level:

- Slice SC – is a byte-aligned slice start marker (0x00000101 → 0x00001AF)
- Qscale – used to modify the quantization scale for the slice

- MB – are the macroblocks that make up the slice

At the MB level:

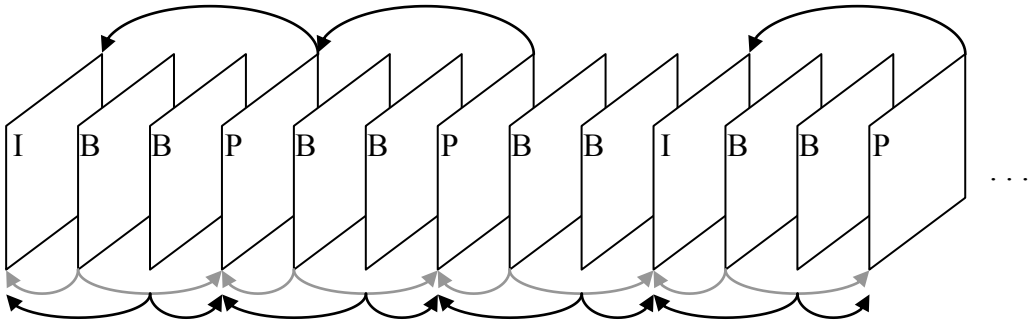
- Addr Inc – is an increment for the macroblock number. This allows macroblocks to be completely skipped if they are with some threshold as the previous image.
- Type – is the macroblock type. For example, it can be a forward predicted, intra-coded, or skipped macroblock for a P-frame. For B-frames, this can be forward predicted, backward predicted, interpolated, or skipped, among other options[3].
- MV – is the motion vector for the block
- Qscale – allows the quantization factor to be modified
- CBP – is the coded block pattern. It is the same as in H.261.

Frame Type Usage

The MPEG format does not specify the type of frame that should be used at any time. Rather, it just specifies the proper format of the stream. The actual usage is up to the individual coder. Most first generation MPEG-1 video coders used a fixed pattern of I, P, and B frames. For example:

I B B P B B P B B I B B P B B P B B I . . .

is a popular sequence of frames to use. The frame dependencies for this pattern are shown below:



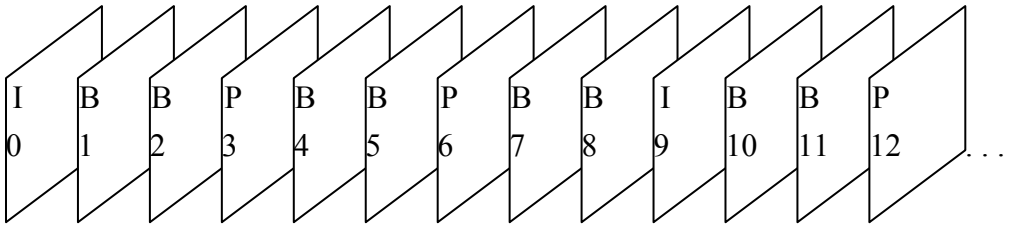
Typically every I-frame starts a new GOP. The reason for this is that the GOP is the minimum randomly addressable unit in MPEG. Using a GOP header for every I-frame means that it maximizes the ability to scan through the MPEG file.

Compression vs. Display Order

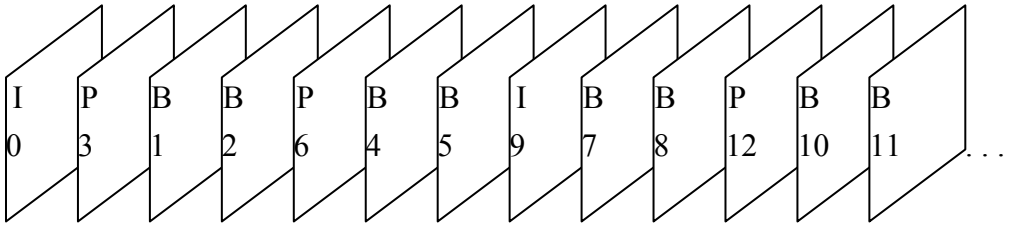
In order to make the decompressor hardware as simple as possible, the MPEG standard specifies that if a frame depends upon another frame that it should appear *before it* in the decompressed frame. If the data is ordered in this way, a decompressor can be made that only requires enough memory to store two complete uncompressed frames. An example is shown below:

[3] This is just a subset. Readers are referred to the reference for the macroblock details.

Capture Order



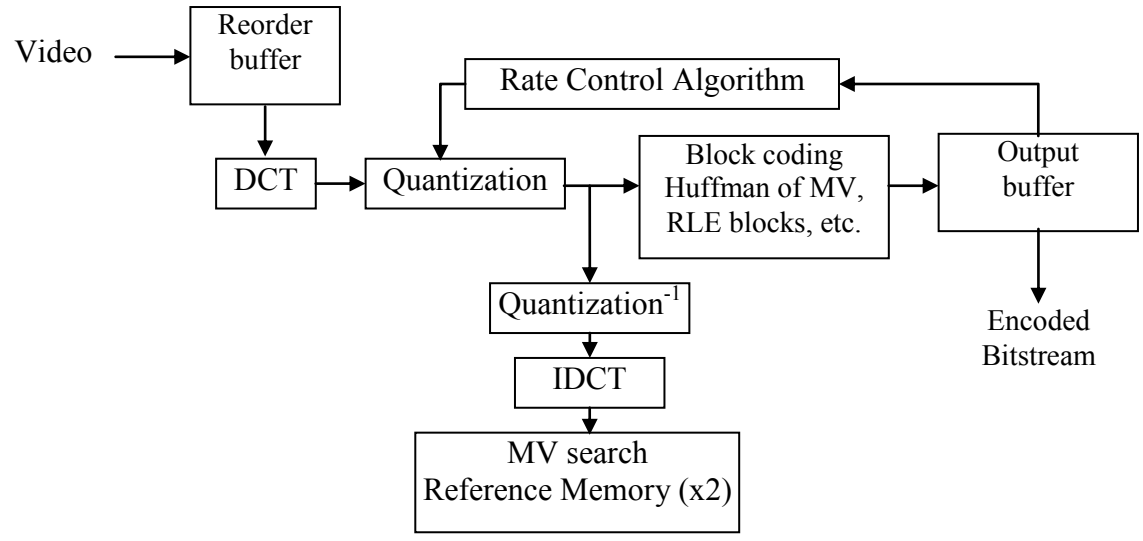
Compressed Order



As shown in the above example, the decompression algorithm needs to first decompress and store the frames labeled 0 and 3. The decompressor will have two storage areas: the *past reference frame* and the *future reference frame.* These are filled with frames 0 and 3, respectively. After decompressing frame 3, the algorithm can start to display frame 0, while decompressing frame 1. Then, frame 1 is displayed while frame 2 is being decompressed. Once another I or P frame is encountered, the future reference frame is moved to the past reference frame and the new I or P frame is placed into the future frame. In this case, 3 is moved to the *past reference frame* and 6 is placed into the *future reference frame*. This continues until the end of the movie. Note that the entire decompression can be accomplished with memory to hold only 2 reference frames.

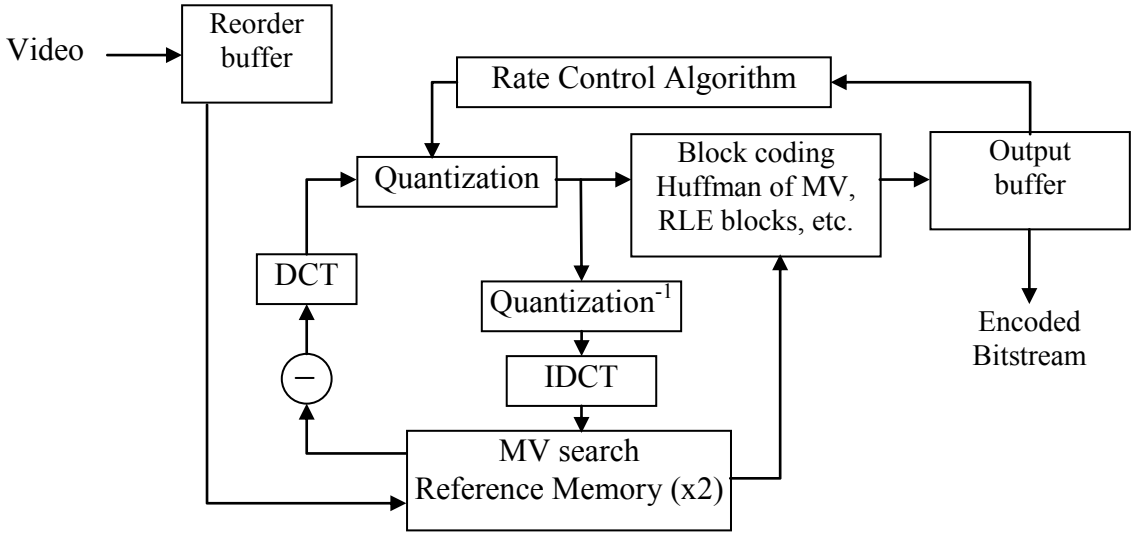
4.2.6.3 MPEG System Diagrams

The MPEG systems diagram for intra-coded I-frames is very similar to the H.261 and is shown below:



There are two main differences between the diagram for MPEG-1 and H.261. First, a reorder buffer needs to buffer data in order to properly re-order the frames for compression order. This obviously adds some delay to the compression process but is necessary. The second difference is that the reference memory must hold two uncompressed frames of data for the past and future reference frames.

The system diagram for inter-coded frames is also similar to the H.261 inter-coded frames. It is shown below:



As with H.261, motion compensation is performed for the inter-coded video frames. The "MV search and Reference Memory" box contain logic to switch between the two reference frames that are being used for B-frames.

4.2.7 MPEG-2

The MPEG-2 video compression standard was originally intended to be the standard for digital TV, HDTV, and DVD (digital video disc). MPEG-2 was standardized in 1994. MPEG-3 was meant to support even higher bit-rate video compression. Due to the superiority of the MPEG-2 format, MPEG-3 was folded into the MPEG-2 standard.

The MPEG-2 standard supports a number of resolutions. Those specified in the standard include:

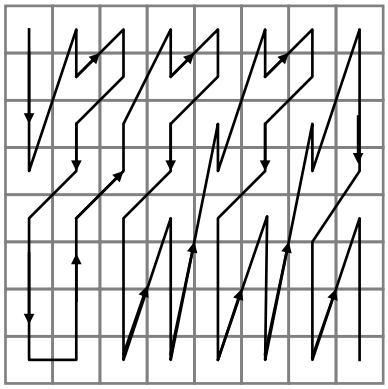
- Low - 352x288 pixel video encoded at 4 Mbps
- Main – 720x576 pixel video encoded at 15 Mbps. This standard was designed for studio TV.
- High 1440 – 1440x1152 pixel video encoded at 60 Mbps. This standard was designed for consumer HDTV.
- High – 1920x1152 pixel video encoded at 80 Mbps. This standard was designed for film production.

Clearly, MPEG-2 was not designed for the lower bit-rate applications.

While MPEG-1 and MPEG-2 are very similar in their frame structure and compression algorithms, there are a number of notable differences between the MPEG-2 and MPEG-1 standard. These include:

- Support for both interlaced and progressive inputs
- Much higher definition
- Alternate subsampling for the chrominance channels
- Improved quantization and entropy encoding

Of particular note is the alternate zig-zag scan for entropy encoding. Because interlaced fields of video tend to be much more squat (due to half of the lines missing), MPEG-2 supports an optional alternate zig-zag reordering that is more amenable to interlaced signals. It is shown below:



4.2.8 MPEG-4 / H.264

As the ITU and ISO continued development of video standards, it was becoming clear that the telephony network that ITU represented and computing that ISO represented were quickly converging. Today, we now have telephony networks that support data (e.g., DSL modems) and computer networks like cable modems delivering telephony (e.g., Voice over IP). In addition, without a huge leap in the way video compression can be accomplished, the MPEG-4 and H.264 teams looked at refinements, rather than the radical change in compression methodology that the MPEG, H.26*, and JPEG standards represented.

The MPEG-4 and H.264 sought to increase the quality of the encoded bit stream while reducing the overall bit rate. In order to do so meant reducing the residual component after motion compensation. In order to reduce residuals, however, requires it to search more possible motion vectors to use as matches for macroblocks. There are many, many changes that have been made in order to make video encoding better. The major ones include: intra prediction, more motion block compensation sizes, quarter pel motion vectors, multiple reference frames, and DCT sizes. In order to improve the visual quality on the decompressed stream an additional deblocking filter is provided.

In intra-prediction, rather than searching a different reference frame during motion compensation, a prediction is sought after in the current frame. It can only be applied to blocks in the frame that have already been encoded. Thus, searching occurs in blocks above and to the left of the current block.

To provide better refinement in the motion compensation, particularly where two objects overlap and are moving in different direction, many different motion compensation block sizes are specified. This allows much smaller residuals in such regions. Similarly, providing finer granularity in motion-estimation can provide better prediction. Specifically, quarter pixel motion vectors are allowed. Thus, pixels are interpolated, weighted by ¾ from on pixel and ¼ from another for the entire block in order to provide the prediction.

The techniques provided thus far are fairly decoder neutral. That is, most of the work is added in the motion compensation step on the encoder, with a little additional work being added at the decoder (e.g., quarter-pel math for motion vectors).

Finally, to allow for even more possible matches in reducing residuals, multiple reference frames can be specified. While this adds yet more searching possibilities, the major impact is on the decoder. In particular, with multiple reference frames, the decoder needs to save (or have) enough memory to buffer all the reference frames that will be needed in the decoding process.

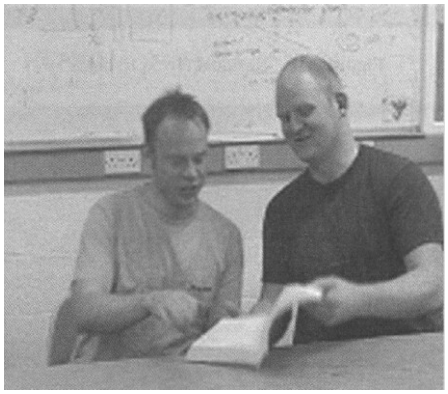
After motion compensation, the two last improvements are allowing for the DCT encoding to be performed on 4x4 blocks as well as the standard 8x8 block. This allows the DCT to be matched with the smaller motion compensation block sizes.

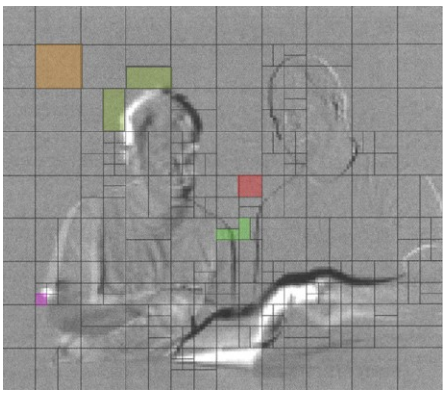
All of these improvements are shown in the table below:

	MPEG-2 / H.263	MPEG-4/AVC
Intra prediction	None	4x4, 16x16
Motion block size	16x16, 16x8, 8x16	16x16, 16x8, 8x16 8x8, 8x4, 4x8, 4x4
Motion vectors	Full pel / Half pel	Full pel / Half pel / Qtr. pel
P Frames	Single	Single / Multiple
B-Frames	1 each way	1 each way, multiple, weighted
DCT	8x8	8x8, 4x4
In loop filters	None	De-blocking

Comparison between MPEG-2 and MPEG-4

To demonstrate the difference between MPEG-2 and MPEG-4, an example image is shown below on the left. The right figure shows the block encoding under MPEG-4:





Source: H.264 and MPEG-4 Video Compression, Lain E.G. Richardson, Wiley

4.3 Audio Compression

To some degree, the compression of audio data because it fewer degrees of compression when compared with their image and video counterparts. With audio signals occurring over time (and not space), the avenues to pursue in reducing redundancy is fairly limited. The conversion of analog sound to digital is inherently lossy. Using a small number of bits per sample is to some degree compression. For the purposes of our discussion, we will assume the "best" signal needing to be represented is sampled at 44,100 Hz using 16-bits per sample with 2 channels (left / right). Thus, our target uncompressed signal has a bit rate of approximately 1.4 Mbps.

4.3.1 Basic Audio Compression Techniques

There are a number of basic audio "compression" technologies that reduce the amount of data necessary for transmission of audio signals. These include applying entropy coding, subsampling the audio signal to a smaller bit rate, using differential compression, or removing silent periods.

Entropy Encoding

One could employ entropy encoding techniques for the compression of audio samples. As with image and video data, the typical audio signal exhibits enough variation over time that applying entropy encoding is not that effective. Furthermore, exact compression of audio data is not necessarily required for audio.

Using Lower Resolution, Lower Frequency Sampling

One of the simplest ways to "compress" an audio stream is to not sample with as high a bit-depth or as high a frequency. Sampling with a smaller bit-depth introduces progressively more quantization error in the captured signal. Sampling with a lower frequency limits the signals that can be captured.

As an example the ITU G.711 μ -law format specifies that an audio signal is sampled at 8000 Hz with 8-bit samples and only one channel. The samples are obtained from a subsampling of 16-bit samples. The samples are subsampled in a logarithmic (perceptually

uniform) way. That is, the 256 samples represent linear increases in loudness. The relation between samples and μ -law samples can be expressed as:

$$y = \frac{\text{sgn}(m)}{\ln(1+u)} \ln\left(1 + u\left|\frac{m}{p}\right|\right)$$

where m is the sample, p is the peak magnitude (65536 for 16-bit samples), and u is 255. μ-law is used in North America and Japan. A-law, a slight variant of μ -law is used elsewhere.

Silence Compression

Another way to compress an audio stream is to remove silence periods. It has been shown that in a discussion between two participants that nearly 40% of the time, there is silence. The telephony companies have indeed employed such compression, much to the dismay of its callers who complain about it sounding like no one is on the phone. While still employed today, the telephony companies now substitute "noise" on the receiving side to mitigate these problems. Nevertheless, the amount of compression achievable is not that high with silence suppression. Obviously, such compression has very limited use in continuous audio streams such as music.

4.3.2 MPEG Audio Compression

Generic audio compression algorithms are typically more difficult to compress than their image and video counterparts. The main reason is that for images, the redundancy in signal occurs in two-dimensions, allowing the algorithms to take advantage of spatial redundancy.

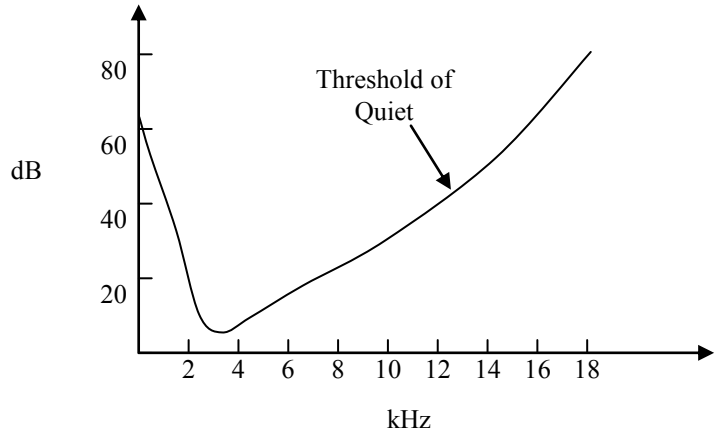
Like the JPEG and MPEG algorithms, MPEG audio compression attempts to remove signals that the human auditory system is incapable of hearing. To some degree our ideal sampling of audio signals (16-bit samples at 44,100 Hz) is already somewhat tailored to the human auditory system. In order to understand MPEG audio compression, we need to first understand how the human auditory system works and what limitations exist in our ability to hear.

4.3.2.1 Human Hearing 101

As we have previously mentioned, the human hearing range is approximately 20 Hz to 20,000 Hz. Humans have the highest acuity in the 2,000 to 4,000 Hz range. Furthermore, the normal speaking voice range is approximately 500 to 2,000 Hz. This is why most telephony systems focus on this relatively narrow band for transmission.

Threshold of Quiet

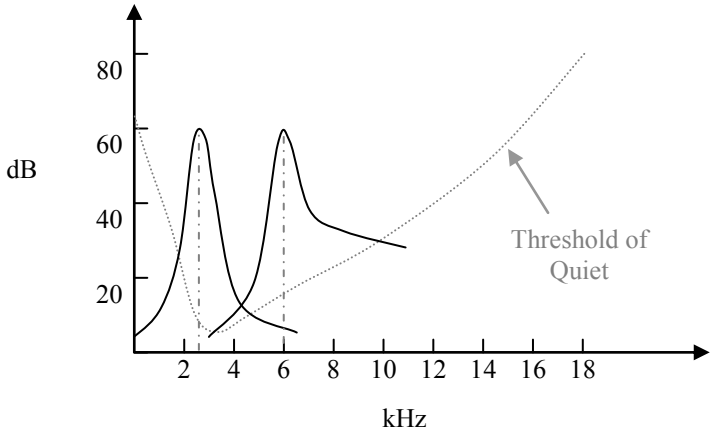
Humans can hear some frequencies better than other frequencies. The figure below shows a representative example of the loudness required in order for humans to hear a particular frequency.



The above graph shows for a particular frequency that exists in the audio signal but is below the threshold of quiet that it can be completely removed from the signal without the user hearing any difference. For example, if there is a 14 kHz signal at approximately 20 dB, this signal can be removed from the audio. Of course, an audio signal is made up of a sum of frequencies. We will describe how this is used in a little bit.

Frequency Masking

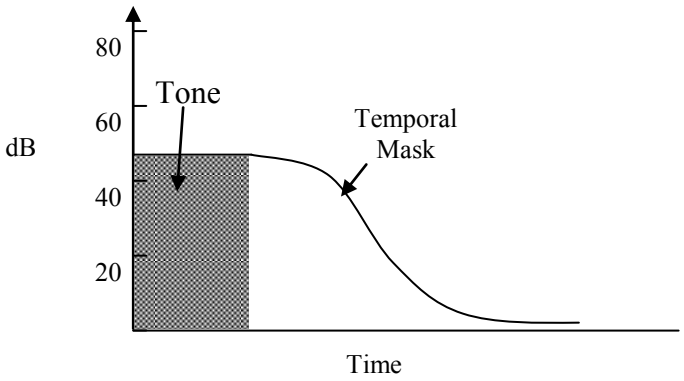
Frequency masking occurs when a loud tone *masks* surrounding frequencies. An example of this is when there are a large number of people talking in a room. We tend to hear the people close by. This is partially due to the attenuation of the far away voices. It is also due to the nearby voices masking out some of the signal. An example of frequency masking is shown below:



In this example, two loud tones exist: one at 2300 Hz and another at 6000 Hz. Now, consider a 30 dB tone at 7000 Hz. Normally, this tone would be heard because it is above the threshold of quiet. However, because of the tone at 6000 Hz, it cannot be heard. The 6000 Hz tone drowns out the 7000 Hz tone. Similarly, a 30 dB tone at 3000 Hz also cannot be heard because of the louder tone at 2300 Hz.

Temporal Masking

Temporal Masking occurs when a loud sound is heard and suddenly stops. Sounds such a drums and doorbells will temporarily continue to mask other surrounding frequencies after they stop.



As shown above, a solid tone will continue to mask the frequency that it's centered around as well as continue to provide frequency masking.

Critical Bands

It turns out that when a particular pitch is being played, the ability of the human ear to distinguish distinct pitches (or "resolution") is not that great. For frequencies in the lower hearing frequency range, the ability to distinguish pitches is less than 100 Hz. For frequencies in the higher frequency range, the ability to distinguish pitches is around 4,000 Hz. What this means is that for several tones that are nearby each other, the human ear combines them into a single pulsating pitch. In all the human audio range can be made up of 25 critical bands.

4.3.2.2 Psychoacoustic Models

Psychoacoustic models are used to capture the limitations of human hearing. Given an input signal, psychoacoustic models return the set of signals that can be heard and removes the signals that are masked out. The actual tuning of psychoacoustic models is well beyond the text of this book.

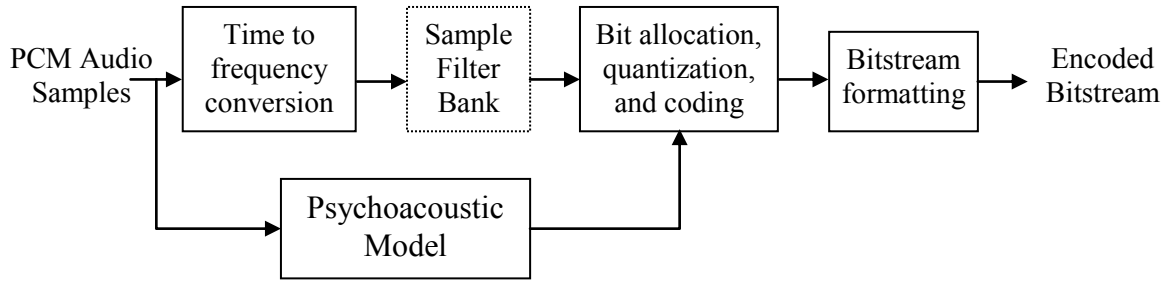
4.3.2.3 MPEG Audio Compression

Like its video compression counterpart, MPEG audio specifies the format of compliant compressed streams, but does not specify how encoders should compress data into the format. The MPEG audio compression algorithm uses a psychoacoustic model to remove signals that are not readily picked up by the human ear. The standard supplies two example psychoacoustic models, but does not require a specific model in compression. Obviously, the psychoacoustic model is not necessary for decompression as the signals that cannot be heard have already been removed. MPEG audio compression results in compression ratio ranging from 2.7 – 24:1.

The MPEG model has quite a large number of options:

- The sampling rate can be 32, 44.1, or 48 kHz.
- Four different channel modes:
    1. Monophonic – single audio channel
    2. Dual monophonic – two independent channels (e.g. languages)
    3. Stereo – two channel audio
    4. Joint stereo – compress data between two stereo channels
- Compressed bit-rates at several predefined fixed bit-rates ranging from 32 kpbs to 224 kpbs per channel. In addition, there is a "free" mode that allows for variable-bit-rate compression.
- Three layers of compression that trade off codec complexity and compressed audio quality. More on this later.

At a high layer, the MPEG audio compression standard consists of a number of components. They are connected as shown below:



The time to frequency conversion takes a number of samples and performs a time to frequency domain conversion on the signal. This transformation divides the audio signal into 32 equal-width frequency subbands. While this transformation is not optimal, it allows for the encoder to be simplified. A more optimal transformation would require the splitting of the signal into the critical bands which are not uniformly spaced (or perceived).

As previously mentioned, the MPEG audio algorithm allows for three separate layers. We will describe each in more detail below.

Layer 1

Layer I encodes the audio sample in frames of 384 audio samples. Spread across the 32 subbands, this means that it takes 12 samples from each subband. Each of the 12 samples gets a bit allocation and a scale factor. For layer 1, this can be 0 to 15 bits per subband. Each scale factor has a 6-bit representation. The encoding of samples is shown below:

Header (32 bits)	CRC (0 or 16 bits)	Bit allocation (128-256)	Scale factors (0-384 bits)	Data	Ancillary data

Layer II

Layer II uses larger groups of samples, in order to achieve better compression efficiency. In Layer II, the encoder uses 1,152 samples of audio rather than the 384 from Layer 1. Thus,

Layer II will use 36 samples from each of the 32 subbands at a time. The 36 samples are then grouped into 3 sets of 12 samples. Each 12 sample groups receives its own scale factor in order to avoid distortion. Scale factors are shared, however, when they are sufficiently close or if the distortion caused by sharing is not detectable by humans. The encoding of a frame of Layer II audio is shown below

Header (32 bits)	CRC (0 or 16 bits)	SCFSI (0-60)	Bit allocation (128-256)	Scale factors (0-384 bits)	Data	Ancillary data

The SCFSI bits specify how scale factors are shared, if any.

Layer III

Layer III is the audio compression standard used for the popular "*mp3*" format. The Layer III compression algorithm is the most complicated, yet refined, algorithm of the three layers. It filters the output of the samples with a modified DCT algorithm. The modified DCT further divides the subband outputs in frequency to provide better spectral resolution. In addition, there are a number of other refinements present in the Layer III option including: non-uniform quantization, scale-factor bands, and entropy encoding of data values (using Huffman). These refinements are beyond the scope of this textbook. At a high level, the coding for Layer III audio can be viewed as:

Header (32 bits)	CRC (0 or 16 bits)	Side information (136,256)	Data

Unfortunately, this is an oversimplification of the bit coding process in Layer III. Layer III contains a "bit reservoir" process that allows the bits to be allocated across frames, allowing maximal use of the bits. The bit reservoir allows the encoder to take advantage of bits that were not used in previous frames for achieving constant bit rate compressed audio.

4.4 Problems

1. One of the drawbacks of GIF compression is the limited color palette. For continuous gradient images, this ends up manifesting itself as banding in the image. Find two images online that, when converted to GIF, result in such banding.

2. Suppose the GIF standard were implemented with Huffman compression instead of LZW compression. What would the relative advantages and disadvantages be of such an approach?

3. Suppose the GIF standard were implemented with Huffman compression instead of LZW compression. Under what conditions would you expect the Huffman algorithm to outperform the LZW-based compression? Under what conditions would you expect the LZW-based algorithm to outperform the Huffman algorithm?

4. Suppose we have the following 8x8 block for JPEG compression:

155	155	0	155	155	0	155	155
155	155	0	155	155	0	155	155
155	155	0	155	155	0	155	155
155	155	0	155	155	0	155	155
155	155	0	155	155	0	155	155
155	155	0	155	155	0	155	155
155	155	0	155	155	0	155	155
155	155	0	155	155	0	155	155

What is the DCT coefficients using the DCT equation provided in the chapter? Hint: Think about where the non-zero coefficients are first.

5. The H.261 compression algorithm has an interesting mapping of MCUs to GOBs. In particular, it creates GOBs of size 176x48 and encodes the image into 12 such GOBs, with each GOB containing a header. Why do suppose this was done? That is, what advantages are there for structuring the MCUs in such an order?

6. Suppose the H.261 predictive coding systems diagram (on page 42) did not have the Quantization^{-1} or IDCT blocks in the system diagram. Rather, the reference frames were stored directly from the source they were derived. What would be the impact on video quality and performance of the encoder?

7. One of the goals of MPEG-1 was "editibility". Suppose a user modified 30 frames of a video and wanted to write them out into a new MPEG file. Explain the steps involved with such a process. Do you believe that MPEG-1 is editiable?

4.5 References

C. Wayne Brown, Barry J. Shepherd, *Graphics File Formats Reference and Guide*, Manning Publishing Co., ISBN 1-994777-00-7.

Communications of the ACM Special Issue on Digital Multimedia Systems, Vol. 34., No. 4, April 1991.

Readings in Multimedia Computing and Networking, Edited by Kevin Jeffay and HongJiang Zhang, pp. 42 -56.