*Efficient Algorithms for*
*MPEG Video Compression*

# *Efficient Algorithms for MPEG Video Compression*

December 11, 2001

**Dzung Tien Hoang and Jeffrey Scott Vitter**

A Wiley-Interscience Publication
**JOHN WILEY & SONS, INC.**

New York / Chichester / Weinheim / Brisbane / Singapore / Toronto

# *Contents*

# *List of Figures*

# *List of Tables*

# *Preface*

In today's information-driven society, video and other forms of information are being increasingly generated, manipulated, and transmitted in digital form. This trend is manifested in the increased level of automation in businesses, the ubiquity of personal computers, the explosive growth of the Internet and the World Wide Web, and the growing library of multimedia software that incorporates digital audio, images, and video. Within the past decade, we have seen secondary storage on desktop personal computers mushroom from a mere 20 megabytes to tens of gigabytes and beyond. Modem technology has pushed the transmission bandwidth through plain telephone lines from 300 bits/second to 56,000 bits/second, and with asynchronous digital subscriber line we can attain transmission speeds of megabits per second on existing phone lines. Even with technological improvements in transmission bandwidth and storage capacity, the information explosion is quickly making current technologies seem inadequate. For example, application software that once fit onto a few floppy disks now demands multi-megabytes of disk space.

Crucial to the management of digital information are data compression techniques that help make more efficient use of the limited transmission and storage resources available. For storage applications, data compression increases the effective storage space, allowing more data to be stored on a given storage device. For transmission applications, data compression increases the effective bandwidth, allowing a higher volume of data to be transmitted over a given transmission medium. Data compression can be viewed as a logical transformation of the data and is independent of the underlying transmission

or storage technology. Data compression will not be made obsolete by advances in these technologies, as there will be an ever-present need for even more storage and even greater bandwidth.

A basic idea in data compression is that most information sources of practical interest are not random, but possess some structure. Recognizing and exploiting structure is a major theme in data compression. The amount of compression that is achievable depends upon the amount of redundancy or structure present in the data that can be recognized and exploited. For example, by noting that certain letters or words in English texts appear more frequently than others, we can represent them using fewer bits than the less frequently occurring letters or words. This principle is used in Morse Code, where letters are represented using a varying number of dots and dashes. The recognition and exploitation of statistical properties of a data source form the basis for much of lossless data compression and entropy coding.

In lossy coding, there is a direct relationship between the length of an encoding (or coding rate) and the amount of loss (or distortion) incurred. Redundancy exists when an information source exhibits properties that allow it to be coded with fewer bits with little or no perceived distortion. For example, in coding speech, distortion in high-frequency bands is not as perceptible to the ear as is distortion in lower-frequency bands. As a result, the high-frequency bands can be coded with less precision using fewer bits. In Chapter 1 we explore the nature of redundancy for lossy coding, especially as it relates to video coding.

Video belongs to a class of information called *continuous media*. Continuous media is characterized by the essentially continuous manner in which the information is presented.[1] This is in contrast to *discrete media*, in which there is no essential temporal component. Text, images, and graphics are examples of discrete media, while movies, sound, and computer animation are examples of continuous media. Even though a slide show is a time-based presentation of images, it is not a continuous medium since each image is viewed as an individual item. On the other hand, a video clip, while also consisting of a sequence of images, is a continuous medium since each image is perceived in the context of past and future images.

With continuous media, therefore, the temporal dimension becomes important. For example, a video sequence compressed with a constant image quality for every frame is often more desirable than one in which the image quality varies noticeably over time. However, because the compressibility of individual frames varies over time, maintaining a constant image quality results in a variation in coding rate over time. The process of controlling the coding rate to meet the requirements of a transmissions channel or storage device, while maintaining a desired level of quality, is called *bit rate control*.

[1]The information may be discrete in representation, but it should be presented to give an *illusion* of continuity.

In this monograph, we focus on the rate control of compressed video. Specifically, we present a new framework for allocating bits to the compression of pictures in an MPEG video sequence.

Existing optimal rate control techniques typically regulate the coding rate to minimize a sum-distortion measure. Whereas these techniques can leverage the wealth of tools from least-mean-square optimization theory, they do not guarantee constant-quality video, an objective often mentioned in the literature. In Chapter 2, we develop a framework that casts rate control as a resource allocation problem with continuous variables, nonlinear constraints, and a novel lexicographic optimality criterion that is motivated for uniform video quality. With the lexicographic criterion, we propose a new concept of coding efficiency to better reflect the constancy in quality that is generally desired from a video coder.

In Chapters 3 and 4, rigorous analysis within the lexicographic framework reveals a set of necessary and sufficient conditions for optimality for coding at both constant and variable bit rates. With these conditions, we are able to construct polynomial-time algorithms for optimal bit rate control. Experimental implementations of these algorithms confirm the theoretical analysis and produce encodings that are more uniform in quality than those achieved with existing rate control methods. Details of the implementations and results are presented in Chapter 5. With further analysis of the optimality conditions, we describe a more efficient algorithm in Chapter 6 that can recover from model errors and operates in $O(N \log N)$ time and linear space for a video sequence of length $N$.

In Chapter 7, we modify the optimal VBR algorithm to operate in real-time. Simulations show that the real-time VBR algorithm performs well compared with the optimal algorithm. A review of the literature suggests that our real-time VBR algorithm is well suited for use in conjunction with a channel rate control algorithm to jointly control source and channel rates in an ATM setting.

As evidence of the generality and flexibility of the framework, we show how to extend the framework in Chapter 8 to allocate bits among multiple variable bit rate bitstreams that are to be transmitted over a common constant bit rate channel and to encompass the case of discrete variables.

The starting point for the research described in this monograph was Dzung Hoang's Ph.D. dissertation at Brown University, under the supervision of Jeffrey Vitter. Some of the initial work has appeared in various journals [29, 31] and conferences and in a U.S. patent [30].

# *Acknowledgments*

We wish to thank most dearly our families for their love and support, which helped carry us through this rate control project, even when the desired rate seemed beyond our control! The work embodied in this book also owes much to the interactions with and encouragement from colleagues, friends, and family, most especially Swarup Acharya, Thomas Alexander, Boumediene Belkhouche, Mark Benard, Tia Chou, Shirish Gadre, César Gonzales, Johnette Hassell, Chi-Yuan Hsu, P. Krishnan, Elliot Linzer, Daniel Lopresti, Dimitrios Michailidis, Jim Munro, T. M. Murali, Apostol Natsev, Duc Ngo, Antonio Ortega, Taner Özcelik, Khai Phan, John Savage, Choh-Man Teng, Darren Vengroff, Eric Viscito, Tu Vu, Min Wang, and Jian Zhou. To all those wonderful people we owe a deep sense of gratitude.

We gratefully acknowledge the support provided along the way by the National Science Foundation through a graduate fellowship and grants CCR–9522047 and CCR–9877133, by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grant F49620–94–1–0217, and by the Army Research Office under grants DAAH04–93–G–0076 and DAAD19–01–1–0725.

*To my parents, my wife Kim, and our newborn Beatrice Huyền-Trân.*

*San José, California* D. T. Hoang
*November 2001*

*To my wife Sharon and our kids Jillian, Scott, and Audrey.*

*New Orleans, Louisiana* J. S. Vitter
*November 2001*

# *Acronyms*

| | |
|---|---|
| ATM | Asynchronous Transfer Mode |
| BMMC | Block-Matching Motion Compensation |
| CBR | Constant Bit Rate |
| CIF | Common Intermediate Format |
| DCT | Discrete Cosine Transform |
| DVD | Digital Video Disk |
| GOB | Group of Blocks |
| GOP | Group of Pictures |
| HRD | Hypothetical Reference Decoder |
| ITU | International Telecommuncation Union |
| ITU-R | ITU Radiocommunication Assembly |
| ITU-T | ITU Telecommunication Standardization Sector |
| JPEG | Joint Photographic Experts Group |
| MB | Macroblock |
| MPEG | Motion Pictures Expert Group |
| NTSC | National Television Systems Committee |
| OBMC | Overlapped Block Motion Compensation |

| | |
|---|---|
| PAL | Phase Alternating Line |
| QCIF | Quarter-CIF |
| RBG | Red Green Blue (color system) |
| SECAM | Système Électronique Couleur Avec Mémoire |
| SIF | Source Input Format |
| VBR | Variable Bit Rate |
| VBV | Video Buffering Verifier |
| VLC | Variable Length Code |
| VQ | Vector Quantization |

# 1

# *Preliminaries*

In this chapter we survey aspects of video compression that will be useful for understanding the later chapters of this book. After a brief discussion about the digital representation of video, we motivate video compression with two illustrative examples that underscore the need for lossy compression. We then describe basic lossy compression techniques for reducing spatial and temporal redundancy in video, focusing on techniques commonly used in international standards for video coding. Next, we present an overview of some of the more popular standards for video coding, with special focus on the bit rate control algorithms used in the development and evaluation of the standards. We conclude with an introduction to rate-distortion theory and the operational rate-distortion framework that forms the basis of much work on optimal bit rate control of compressed video.

This chapter is by no means intended to be comprehensive; for more in-depth discussions of the fundamentals and applications of video coding, the reader is referred to [2, 26, 55, 57, 67, 73].

## 1.1 DIGITAL VIDEO REPRESENTATION

For compression to be meaningful, a standard representation should be defined for the data to be compressed. In this section, we give an overview of some of the more popular standard representations for digital video that are in use today.

### 1.1.1 Color Representation

Although visible light consists of a continuum of wavelengths, it has been known for several centuries that a small set of primary colors, when mixed in the right proportions, can simulate a wide range of perceived colors.[1] Red, green, and blue (RGB) light sources form one set of primary colors; this is an *additive* color system since the presence of all the primary colors at their maximum intensities results in the perception of the color white. In painting and printing, cyan, magenta, and yellow (CMY) pigments or inks form another set of primary colors; this is a *subtractive* color system since the absence of all primary colors yields the color of the canvas, which is usually a shade of white.

The phenomenon of color perception reflects the way that the human eye detects and processes light and makes it possible to represent a visual image as a small set of intensity signals (e.g., red, green, and blue). For example, color televisions and computer monitors render a color image by exciting red, green, and blue phosphors with electron beams of varying intensity.

There are numerous color representation systems in use today. Here we briefly describe several color systems most relevant to digital video.

In the United States, the National Television Systems Committee (NTSC) has defined an $RGB$ color system based upon three types of phosphor that emit light in the red, green, and blue regions of the spectrum. Normalizing the RGB components to the range [0,1], the color white is represented as $R = G = B = 1$.

The NTSC RGB color components, however, are not used for the actual television signal transmission. Instead, the color system called $YIQ$ is used for transmission. The YIQ system was engineered to maintain compatibility with the preexisting black-and-white television system. The $Y$ component captures the luminance (brightness) information that can be compatibly decoded by a black-and-white television. Supplementing the $Y$ component are two chrominance (color) components, $I$ and $Q$. The $I$ component captures flesh tones and near flesh tones, and the $Q$ component captures other colors [13]. In order to fit the color YIQ signals into the same channel bandwidth as a black-and-white signal, the color components $I$ and $Q$ are transmitted at a reduced bandwidth compared with the luminance component. This approach takes advantage of the human visual system's reduced sensitivity to color changes. To simplify the color decoding of YIQ to RGB, the YIQ components are defined to be linearly related to the RGB components as described by the following linear system of equations [37]:

$$Y = 0.299R + 0.587G + 0.114B;$$

[1] In the 17th century, Isaac Newton discovered that a small number of colors from the spectrum produced by a prism can be mixed to produce other colors, including those not in the spectrum.

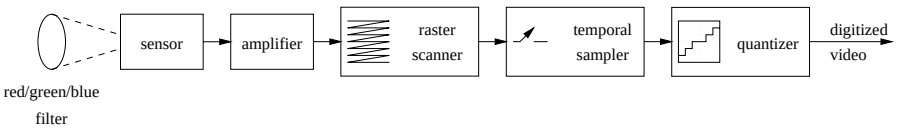



*Fig. 1.1* **Block Diagram of a Video Digitizer.** This figure shows the typical processing steps involved in the digitization of video. After signal acquisition and amplification, the key processing steps are spatial sampling, temporal sampling, and quantization.

$$\begin{aligned} I &= 0.596R - 0.274G - 0.322B; \\ Q &= 0.211R - 0.523G + 0.896B. \end{aligned}$$

Outside the United States, the PAL and SECAM television systems are widely used. The PAL system uses the *YUV* color system, and the SECAM system uses the *YDbDr* color system. In both YUV and YDrDb systems, the $Y$ component is identical to that of YIQ. As with YIQ, the YUV and YDrDb components are linearly related to the RGB components. The conversion equations [26] for YUV are

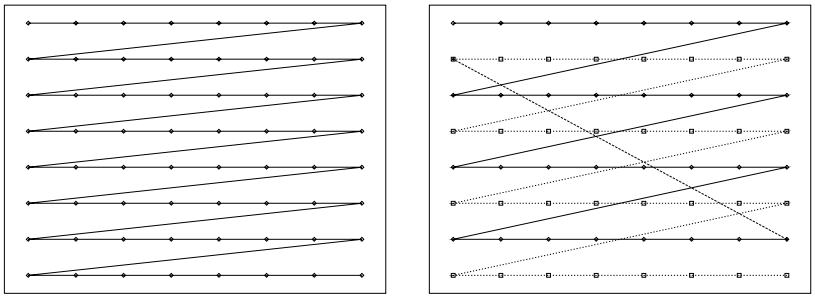
$$\begin{aligned} Y &= 0.299R + 0.587G + 0.114B; \\ U &= -0.147R - 0.289G + 0.436B \\ &= 0.492(B - Y); \\ V &= 0.615R - 0.515G - 0.100B \\ &= 0.877(R - Y). \end{aligned}$$

The conversion equations [26] for YDrDb are

$$\begin{aligned} Y &= 0.299R + 0.587G + 0.114B; \\ Dr &= -0.450R - 0.883G + 1.333B \\ &= 3.059U; \\ Db &= -1.333R + 1.116G - 0.217B \\ &= -2.169V. \end{aligned}$$

### 1.1.2 Digitization

In order to be processed by computers, analog video that is captured by a light sensor must first be digitized. Digitization of video consists of three steps: 1) spatial sampling, 2) temporal sampling, and 3) quantization. A block diagram of the digitization process is depicted in Figure 1.1 for one color component. The steps need not be performed in the order indicated and some steps may be combined into one operation.



(a) Progressive Scan (b) Interlaced Scan

*Fig. 1.2* **Scanning Techniques for Spatial Sampling.** In a progressive scan, consecutive lines in a frame are sampled in order. In an interlaced scan, the lines are divided into odd and even sets; the even lines are sampled after the odd lines.

### 1.1.3 Spatial Sampling

Spatial sampling consists of taking measurements of the underlying analog signal at a finite set of sampling points in a finite viewing area (or *frame*). To simplify the process, the sampling points are restricted to lie on a lattice, usually a rectangular grid. The two-dimensional set of sampling points are transformed into a one-dimensional set through a process called *raster scanning*. The two main ways to perform raster scanning are shown in Figure 1.2: *progressive* and *interlaced*. In a progressive (or non-interlaced) scan, the sampling points are scanned from left to right and top to bottom. In an interlaced scan, the points are divided into odd and even scan lines. The odd lines are scanned first from left to right and top to bottom. Then the even lines are scanned. The odd (respectively, even) scan lines make up a field. In an interlaced scan, two fields make up a frame. It is important to note that the odd and even fields are sampled and displayed at different time instances. Therefore the time interval between fields in an interlaced scan is half of that between frames. Interlaced scanning is commonly used for television signals and progressive scanning is typically used for film and computer displays.

### 1.1.4 Temporal Sampling

The human visual system is relatively slow in responding to temporal changes. By showing at least 16 frames of video per second, an illusion of motion is created. This observation is the basis for motion picture technology, which typically performs temporal sampling at a rate of 24 frames/sec. For television, sampling rates of 25 and 30 frames/sec are commonly used. With

interlaced scan, the sampling rate is sometimes expressed as the number of fields per second, which is twice the number of frames per second.

### 1.1.5 Quantization

After spatial and temporal sampling, the video signal consists of a sequence of continuous intensity values. The continuous intensity values are incompatible with digital processing, and one more step is needed before this information can be processed digitally. The continuous intensity values are converted to a discrete set of values in a process called *quantization*.

Quantization can be viewed as a mapping from a continuous domain to a discrete range.[2] A particular quantization mapping is called a *quantizer*. An example is shown in Figure 1.3. In the figure, there are eleven discrete quantization levels, also called *bins*. Each bin has an associated size, which is the extent of the continuous values that map to that bin. In the example, each bin, except for the bins for $-5$, 0, and 5, has the same size, which is sometimes referred to as the *quantizer step size*. This type of quantizer is called a *uniform* quantizer. A binary encoding can be assigned to each of the bins. Typically the initial quantization of a continuous source is done using a number of quantization levels that is a power of 2, so that a fixed number of bits can be used to represent the quantized value.[3] This process of representing a continuous value by a finite number of levels using a binary code is often referred to as *pulse code modulation* (PCM).

In conclusion, after spatial sampling, temporal sampling, and quantization, we have $N \times M$ picture elements, commonly called *pixels* or *pels*, represented using a fixed number of bits.

### 1.1.6 Standard Video Data Formats

To promote the interchange of digital video data, several formats for representing video data have been standardized. We now review some of the more popular standard representations.

*CCIR-601 Standard.* Because they are designed for analog television, the YIQ, YUV, and YDrDb color systems are inherently analog. The CCIR-601 digital video standard[4] defines a standard digital representation of video in terms of digital *YCrCb* color components [3]. CCIR-601 defines both 8-bit and 10-bit

[2] This definition is intended also to encompass mappings from a discrete domain to a discrete range.

[3] Further quantization of digitized data may use a number of quantization levels that is not a power of 2 and employ variable-length entropy coding.

[4] The CCIR has changed its name to the International Telecommunication Union Radiocommunication Assembly (ITU-R), and the latest revision of the CCIR-601 standard is formally known as Recommendation ITU-R BT.601-5. We use the term CCIR-601 since it is still in common usage.

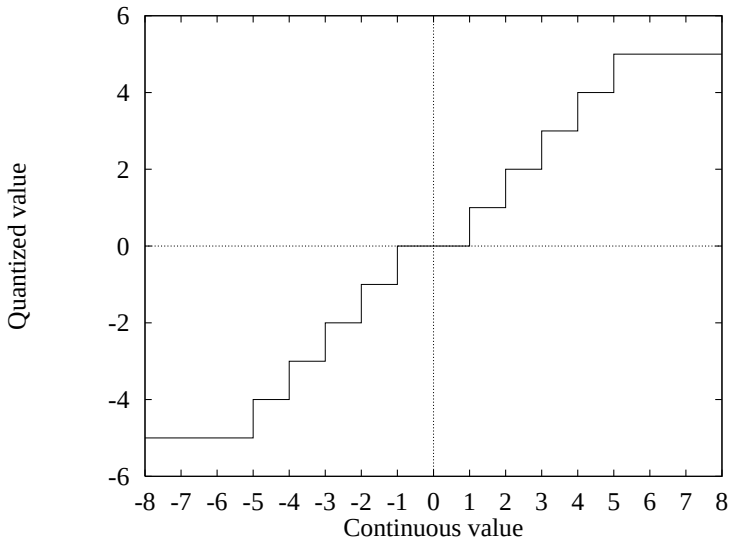



*Fig. 1.3* **Example of a Uniform Quantizer.** This example shows a quantizer that has 11 possible discrete output values, or *bins*.

digital encodings. In the 8-bit encoding, assuming that the RGB components have been digitized to the range $[0, 255]$, the YCrCb components are defined as follows [26]:

$$\begin{aligned} Y &= 0.257R + 0.504G + 0.098B + 16; \\ Cr &= 0.439R - 0.368G + -0.071B + 128; \\ Cb &= -0.148R - 0.291G + 0.439B + 128. \end{aligned}$$

The CCIR-601 standard defines a family of digital video formats. The most commonly used member of the family is the 4:2:2, 13.5 MHz format. In this format, the luminance component is sampled at a rate of 13.5 MHz with 720 active samples per line. The chrominance components, $Cr$ and $Cb$, each are sampled at 6.75 MHz with 360 active samples per line. For NTSC, this sampling yields 486 active lines per frame at 60 fields/sec. For PAL, the sampling yields 576 active lines per frame at 50 fields/sec.

In terms of pixels per frame, the 4:2:2 CCIR-601 format specifies spatial sampling of $720 \times 486$ for NTSC and $720 \times 576$ for PAL. Temporal sampling is interlaced 60 fields/sec for NTSC and interlaced 50 fields/sec for PAL. The chrominance components are subsampled horizontally with respect to the luminance component to take advantage of the human visual system's reduced spatial sensitivity to color. This subsampling process is referred to as the 4:2:2 format and is depicted in Figure 1.4.

*Source Input Format.* The Source Input Format (SIF) specifies spatial sampling of $360 \times 240$ and progressive temporal sampling at 30 frames/sec for

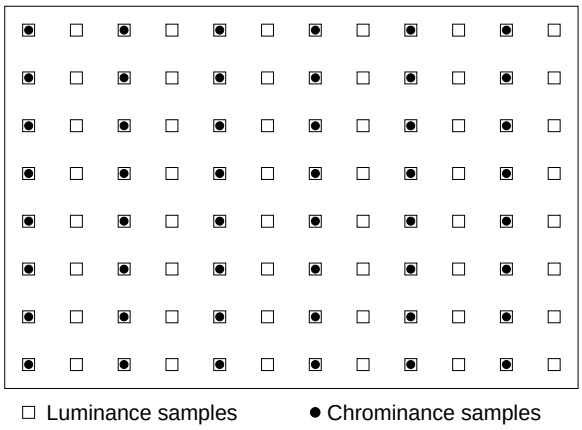



*Fig. 1.4* **4:2:2 Color Subsampling.** With 4:2:2 chroma subsampling, the two chroma components are subsampled by a factor of two horizontally. The positioning of the chrominance values relative to the luminance values is shown as specified by the CCIR-601 standard.

NTSC-originated video, and $360 \times 288$ spatial sampling at a progressive frame rate of 25 frames/sec for PAL-original video.[5] As with CCIR-601, color is represented using three components: $Y$, $Cb$, and $Cr$. Each component is quantized linearly using eight bits. The chrominance components, $Cb$ and $Cr$, are subsampled by a factor of two both horizontally and vertically, yielding a chrominance sampling of $180 \times 120$ at 30 frames/sec and $180 \times 144$ at 25 frames/sec.[6] This subsampling format is referred to as the 4:2:0 format[7] and is illustrated in Figure 1.5.

*Common Intermediate Format.* One drawback with the CCIR-601 and SIF formats is that they specify different spatial and temporal sampling parameters for NTSC and PAL systems. As its name suggests, the Common Intermediate Format (CIF) was proposed as a bridge between NTSC and PAL. As with CCIR-601, color is represented using YCrCb, quantized linearly using eight bits. The CIF format uses 4:2:0 color subsampling with an image size of $352 \times 288$. Temporal sampling is set at 30 frames/sec. For use with PAL systems, the CIF format requires conversion of the frame rate to 25 frames/sec. For NTSC systems, a spatial resampling may be necessary.

[5] For some applications, such as MPEG-1 and MPEG-2 video, it is convenient for the spatial dimensions to be a multiple of 16. For this reason, a horizontal dimension of 352 is often used.

[6] When the horizontal image dimension of 352 is used, the horizontal chrominance sampling would correspondingly be 176.

[7] The 4:2:0 format should not be confused with the 4:1:1 format in which the chrominance components are subsampled by a factor of 4 only in the horizontal direction.

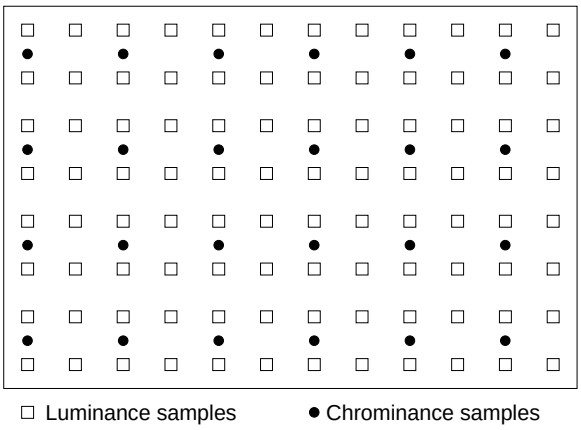



*Fig. 1.5* **4:2:0 Color Subsampling.** With 4:2:0 subsampling, the chroma components are subsampled by a factor of two both horizontally and vertically. The positioning of the chrominance values relative to the luminance values is shown as specified in the MPEG-2 standard. In the MPEG-1 standard, the chrominance samples are positioned in the center of four adjacent luminance samples.

For video conferencing and other low-bit-rate, low-resolution applications, a scaled-down version of CIF called Quarter-CIF (QCIF) is commonly used. QCIF specifies an image with half the resolution of CIF in each spatial dimension: $176 \times 144$. For many low-bit-rate applications, the frame rate is reduced from 30 frames/sec to as low as 5 frames/sec.

## 1.2 A CASE FOR VIDEO COMPRESSION

Now that we have described several standard representations for digital video, we can estimate the compression ratio required for some typical applications.

For a two-hour movie encoded in the NTSC CCIR-601 4:2:2 format, the uncompressed video representation would require about 151 gigabytes to store:

$$(720 \cdot 486 + 2 \cdot 360 \cdot 486)\frac{\text{bytes}}{\text{frame}} \cdot 30\frac{\text{frames}}{\text{sec}} \cdot 3600\frac{\text{sec}}{\text{hr}} \cdot 2 \text{ hrs} = 1.512 \cdot 10^{11} \text{ bytes.}$$

In order to store the movie on one single-sided digital video disk (DVD), which has a capacity of 4.7 gigabytes, we need to compress the video by a factor of about 32:1. To allow room for audio and other auxiliary data (such as text captioning), an even higher compression ratio is needed.

As another example, consider low-bit-rate video conferencing over a telephone modem. Assuming that the uncompressed video is encoded in QCIF

format at 10 frames/sec, the uncompressed bit rate is computed to be:

$$(176 \cdot 144 + 2 \cdot 88 \cdot 72)\frac{\text{bytes}}{\text{frame}} \cdot 8\frac{\text{bits}}{\text{byte}} \cdot 10\frac{\text{frames}}{\text{sec}} = 3.041 \cdot 10^6 \frac{\text{bits}}{\text{sec}}.$$

To transmit video in this format over a 28.8 Kbits/sec modem would require a compression ratio of about 106:1. At such a high compression ratio, depending upon the complexity of the video sequence, the quality of the compressed video may have to be sacrificed. Alternatively, the frame rate could be reduced to increase the image quality, at the expense of increased jerkiness in the motion.

The above examples show why compression is a must for some important digital video applications. For example, without compression, a single-sided DVD can hold less than four minutes of CCIR-601 digital video.

## 1.3 SPATIAL REDUNDANCY

Redundancy exists in a video sequence in two forms: spatial and temporal. The former, also called *intraframe* redundancy, refers to the redundancy that exists within a single frame of video, while the latter, also called *interframe* redundancy, refers to the redundancy that exists between consecutive frames within a video sequence.

Reducing spatial redundancy has been the focus of many image compression algorithms. Since video is just a sequence of images, image compression techniques are directly applicable to video frames. Here, we outline some popular image coding techniques applicable to lossy video coding.

### 1.3.1 Vector Quantization

In vector quantization (VQ) [21], an image is segmented into same-sized blocks of pixel values. The blocks are represented by a fixed number of vectors called *codewords*. The codewords are chosen from a finite set called a *codebook*. This process is analogous to the quantization described in Section 1.1.2, except that now quantization is performed on vectors instead of scalar values. The size of the codebook affects the coding rate (number of bits needed to encode each vector) as well as the distortion; a bigger codebook increases the coding rate and decreases the average distortion, whereas a smaller codebook has the opposite effects.

With vector quantization, encoding is more computationally intensive than decoding. Encoding requires searching the codebook for a representative codeword for each input vector, whereas decoding requires only a table lookup. Usually, the same codebook is used by the encoder and the decoder. The codebook generation process is itself computationally demanding. Some applications of VQ in video compression can be found in [19, 78].

### 1.3.2 Block Transform

In block transform coding, an image is divided into blocks, as with vector quantization. Each block is mathematically transformed into a different representation, which is then quantized and coded. The mathematical transform is chosen so as to redistribute most of the useful information into a small set of coefficients. The coefficients are then selectively quantized so that after quantization most of the "unimportant" coefficients are 0 and can be ignored, while the "important" coefficients are retained. In the decoder, a inverse quantization process is followed by an inverse transformation.

Block transform coding can be viewed as an instance of vector quantization where the codebook is determined by the transform and quantization performed. Viewed in this way, for any source, a vector quantizer can be designed that will be at least as good (in a rate-distortion sense) as a particular block transform. A motivation for using block transforms is that for certain block transforms with fast algorithms, encoding can be done faster than full-blown vector quantization. However, with block transforms, decoding has approximately the same complexity as encoding, which is more complex than decoding with vector quantization.

Mathematical transforms that have been used for block transform coding include discrete Fourier, discrete cosine, discrete sine, Karhunen-Loeve, slant, and Hadamard [37].

### 1.3.3 Discrete Cosine Transform

For images, the two-dimensional discrete cosine transform (2D-DCT) is a popular block transform that forms the basis of the lossy JPEG standard [62] developed by the Joint Photographic Experts Group. Because of its success within JPEG, the 2D-DCT has been adopted by many video coding standards, such as H.261, H.263, MPEG-1, MPEG-2, and MPEG-4. We now describe the mathematical basis of the DCT and show how it is applied to code an image.

*Forward DCT Transform.* The JPEG standard specifies a block size of $8 \times 8$ for performing the 2D-DCT. This block size is small enough for the transform to be quickly computed but big enough for significant compression. For an $8 \times 8$ block of pixel values $f(i,j)$, the 2D-DCT is defined as

$$F(u,v) = \frac{1}{4} C(u) C(v) \sum_{i=0}^{7} \sum_{j=0}^{7} f(i,j) \cos \frac{\pi u(2i+1)}{16} \cos \frac{\pi v(2j+1)}{16}, \qquad (1.1)$$

where $F(u,v)$ are the transform coefficients and

$$C(x) = \begin{cases} \frac{1}{\sqrt{2}} & \text{if } x = 0; \\ 1 & \text{otherwise.} \end{cases}$$

$$\begin{bmatrix} 8 & 16 & 19 & 22 & 26 & 27 & 29 & 34 \\ 16 & 16 & 22 & 24 & 27 & 29 & 34 & 37 \\ 19 & 22 & 26 & 27 & 29 & 34 & 34 & 38 \\ 22 & 22 & 26 & 27 & 29 & 34 & 37 & 40 \\ 22 & 26 & 27 & 29 & 32 & 35 & 40 & 48 \\ 26 & 27 & 29 & 32 & 35 & 40 & 48 & 58 \\ 26 & 27 & 29 & 34 & 38 & 46 & 56 & 69 \\ 27 & 29 & 35 & 38 & 46 & 56 & 69 & 83 \end{bmatrix}$$

*Fig. 1.6* **Default Intraframe Quantization Matrix.** This figure shows the default MPEG-2 intraframe quantization matrix to be applied to 2D-DCT coefficients. With this quantization matrix, transform coefficients are quantized more coarsely with increasing horizontal and vertical spatial frequencies.

*Inverse DCT Transform.* To be useful for coding, a block transform needs an inverse transform for purposes of decoding. The two-dimensional inverse discrete cosine transform (2D-IDCT) for an $8 \times 8$ block is defined as

$$f(i,j) = \frac{1}{4} \sum_{u=0}^{7} \sum_{v=0}^{7} F(u,v) C(u) C(v) \cos \frac{\pi u(2i+1)}{16} \cos \frac{\pi v(2j+1)}{16}. \quad (1.2)$$

*Quantization.* Since the DCT and IDCT are transform pairs, they do not result in any compression by themselves. Compression is achieved by subsequent quantization of the transform coefficients.

Quantization as applied to transform coefficients can be viewed as division followed by integer truncation. Specifically, the transform coefficients are first divided by a (prespecified) matrix of integers that is weighted by a *quantization scale* $Q$. After division, the results are truncated to integer values. In the dequantization, the quantized values are multiplied by the quantization matrix and adjusted according to the quantization scale. Typically 8 to 12 bits of precision are used.

An example of a quantization matrix is shown in Figure 1.6. The coefficients can be specified to exploit properties of the human visual system. Since the human eye is more sensitive to low spatial frequencies and less sensitive to high spatial frequencies, the transform coefficients corresponding to high spatial frequencies can be quantized more coarsely than those for low spatial frequencies. This frequency-selective quantization can be seen in Figure 1.6.

*Zig-Zag Scan.* Because of the coarse quantization of coefficients corresponding to high spatial frequencies, those coefficients are often quantized to 0. An effective way to code the resulting set of quantized coefficients is with a combination of a zig-zag scan of the coefficients as shown in Figure 1.7 and run-length encoding of consecutive zeros. Typically, the DC coefficient, $F(0,0)$, is coded separately from the other coefficients and is not included in the zig-zag scan.

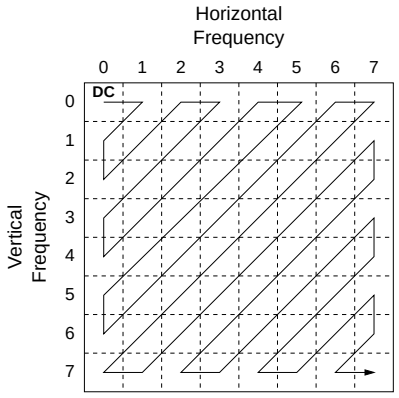



*Fig. 1.7* **Zig-Zag Scan.** This figure shows a zig-zag scanning pattern for coding quantized transform coefficients as a one-dimensional sequence. A run-length encoding of the zero values of the one-dimensional sequence is then performed.

## 1.4 TEMPORAL REDUNDANCY

Successive frames in a video sequence are typically highly correlated, especially for scenes where there is little or no motion. The spatial decorrelation techniques described in the previous section only operate within a single frame and do not exploit the redundancy that exists between frames. We now review some basic techniques for reducing temporal redundancy.

### 1.4.1 Frame Differencing

A very simple technique for exploiting temporal redundancy in a video sequence is to code the difference between one frame and the next. This technique is called *frame differencing* and is an extension of the basic differential pulse code modulation (DPCM) coding techniques (see, e.g. [57]). A block diagram of an encoder that uses frame differencing is shown in Figure 1.8.

At some initial point, a frame must be coded without frame differencing and using only spatial coding techniques. Such a frame is commonly referred to as an *intracoded* frame, or I frame for short. Because they do not take advantage of interframe redundancy, I frames consume more bits than predictive frames of comparable quality. To prevent degradation in image quality from the accumulation of prediction error and to allow for easy random access to frames in a video, frames are periodically coded as I frames.

If there is little motion between successive frames, frame differencing yields a difference image that is mostly uniform and can be coded efficiently. However, frame differencing fails when there is appreciable motion between frames or when a scene change occurs. An effective strategy when frame differenc-

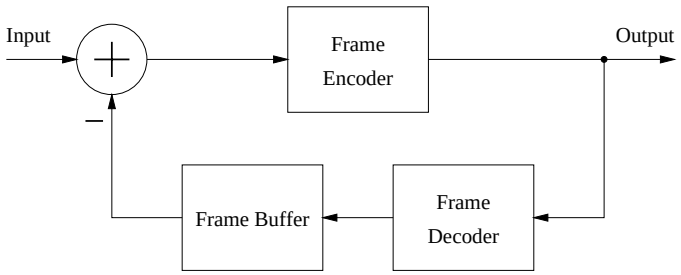



*Fig. 1.8* **A Simple Frame-Differencing Coder.** This block diagram shows a simple frame-differencing coder. The frame buffer stores the previously decoded frame which is used to compute a difference frame.

ing fails is to switch to intracoding mode. However, this technique does not improve the compression efficiency of frame differencing.

### 1.4.2 Motion Compensation

Frame differencing can be viewed as a predictive coding technique where the prediction is simply the previous decoded frame. By improving the prediction, we can potentially obtain better compression. *Motion compensation* is one such technique that uses a model of the motion of objects between frames to form a prediction. Using the model, the encoder performs *motion estimation* to determine the motion that exists between a reference frame and the current frame. The reference frame can occur temporally before the current frame (forward prediction) or after the current frame (backward prediction). An advanced technique, called bidirectional prediction, uses two reference frames, one each for forward and backward prediction, and interpolates the results. This usually gives better prediction and handles the case where an object is temporarily occluded.

The encoding process is illustrated in Figure 1.9. After motion estimation and compensation, the motion information and prediction error are transmitted to the decoder, which reconstructs the predicted frame from the motion information and the decoded reference frame. Note that the reference frame must have already been decoded for the decoder to be able to reconstruct the current frame. As with frame differencing, an I frame is needed to seed the motion compensation process.

Frames that are coded using forward prediction are called P frames, short for *predicted* frames. A P frame uses as a reference a past I frame or P frame.

Backward prediction is typically not used exclusively, but as an option for B frames, short for *bidirectionally predicted* frames. A B frame is coded from a past reference frame and a future reference frame, as shown in Figure 1.10. At first, this might seem to present a causality problem since there is a dependence upon a future frame. To avert any such problem, the frames

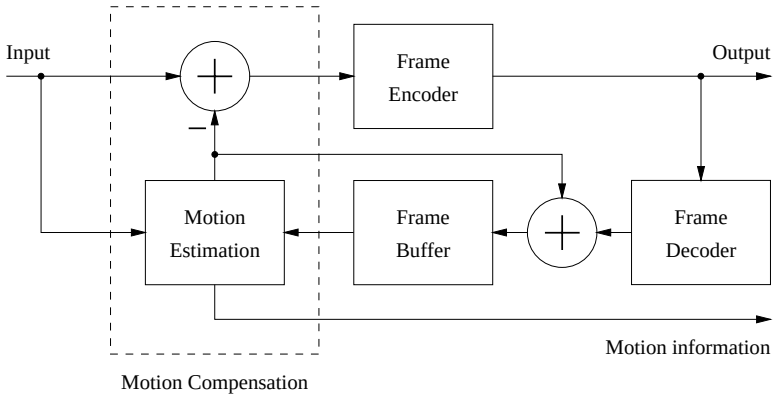



*Fig. 1.9* **A Generic Motion-Compensated Video Encoder.** This figure shows a block diagram of a generic motion-compensated video encoder. The dashed box encloses the motion compensation unit, which consists of a motion estimator and a frame differencer.

are reordered so that all reference frames that are required by a B frame or P frame come before that frame in the reordered sequence. An example is shown in Figure 1.13. In practice, this reordering introduces some encoding and decoding delays and requires two frame buffers to hold the reference frames. For non-real-time applications, such as stored video, the additional delay is not a serious issue. For real-time applications, such as video conferencing, the distance between successive reference frames are kept small to reduce the delay. B frames may be omitted altogether to further reduce the delay.

For interframe coding, perceptual weighting as per Figure 1.6 is not usually applied since the block to be coded is the block of *prediction errors*, which does not share the perceptual properties of the original spatial block of pixel values. The MPEG standards specify a default interframe quantization matrix with uniform values, as shown in Figure 1.11. In practice, the modified interframe quantization matrix shown in Figure 1.12 is often used. In this matrix, the quantization step size increases gradually as the frequency index increases. This modified interframe quantization matrix works well in practice and has been adopted as the default matrix in the MPEG-4 visual coding standard [36].

In the final step, indicated in Figure 1.9, the prediction error that results from motion compensation is coded with an intraframe coder, for instance, one of the techniques mentioned in Section 1.3.

### 1.4.3 Block Matching

A motion model that is commonly used is the *block translation* model developed by Jain and Jain [38]. In this model, an image is divided into non-

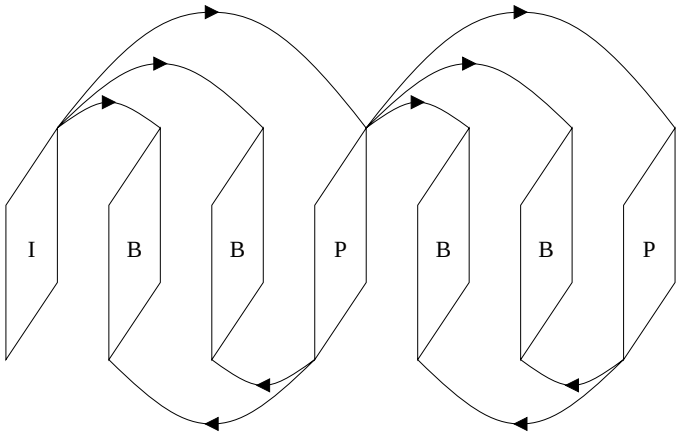



*Fig. 1.10* **Frame Types for Motion Compensation.** This figure illustrates three types of frames use in motion compensation. An I frame is coded using intraframe techniques and has no temporal dependencies. A P frame is predicted from a previous reference frame, which may be either an I frame or a P frame. B frames are predicted from a past and a future reference frame.

$$\begin{bmatrix} 16 & 16 & 16 & 16 & 16 & 16 & 16 & 16 \\ 16 & 16 & 16 & 16 & 16 & 16 & 16 & 16 \\ 16 & 16 & 16 & 16 & 16 & 16 & 16 & 16 \\ 16 & 16 & 16 & 16 & 16 & 16 & 16 & 16 \\ 16 & 16 & 16 & 16 & 16 & 16 & 16 & 16 \\ 16 & 16 & 16 & 16 & 16 & 16 & 16 & 16 \\ 16 & 16 & 16 & 16 & 16 & 16 & 16 & 16 \\ 16 & 16 & 16 & 16 & 16 & 16 & 16 & 16 \end{bmatrix}$$

*Fig. 1.11* **Default Interframe Quantization Matrix.** This figure shows the default MPEG-2 interframe quantization matrix to be applied to 2D-DCT coefficients. With this quantization matrix, transform coefficients are quantized the same independent of frequency.

$$\begin{bmatrix} 16 & 17 & 18 & 19 & 20 & 21 & 22 & 23 \\ 17 & 18 & 19 & 20 & 21 & 22 & 23 & 24 \\ 18 & 19 & 20 & 21 & 22 & 23 & 24 & 25 \\ 19 & 20 & 21 & 22 & 23 & 24 & 26 & 27 \\ 20 & 21 & 22 & 23 & 24 & 26 & 27 & 28 \\ 21 & 22 & 23 & 24 & 26 & 27 & 28 & 30 \\ 22 & 23 & 24 & 26 & 27 & 28 & 30 & 31 \\ 23 & 24 & 25 & 27 & 28 & 30 & 31 & 33 \end{bmatrix}$$

*Fig. 1.12* **Modified Interframe Quantization Matrix.** This figure shows a modified MPEG-2 interframe quantization matrix that is often used in practice and has been incorporated into the MPEG-4 standard as the default interframe matrix [36]. With this quantization matrix, transform coefficients are quantized gradually coarser as the frequency increases.

| Frame Type: | **I** | **B** | **B** | **P** | **B** | **B** | **P** | **B** | **I** |
|---|---|---|---|---|---|---|---|---|---|
| Temporal Index: | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

(a) Original Sequence (Temporal Order)

| Frame Type: | **I** | **P** | **B** | **B** | **P** | **B** | **B** | **I** | **B** |
|---|---|---|---|---|---|---|---|---|---|
| Temporal Index: | 1 | 4 | 2 | 3 | 7 | 5 | 6 | 9 | 8 |

(b) Reordered Sequence (Encoding Order)

*Fig. 1.13* **Reordering with B Frames.** This figure shows the reordering of I and P frames to allow for causal decoding of B frames. A reference frame (I or P frame) is moved immediately before the first B frame that uses it for backward prediction.

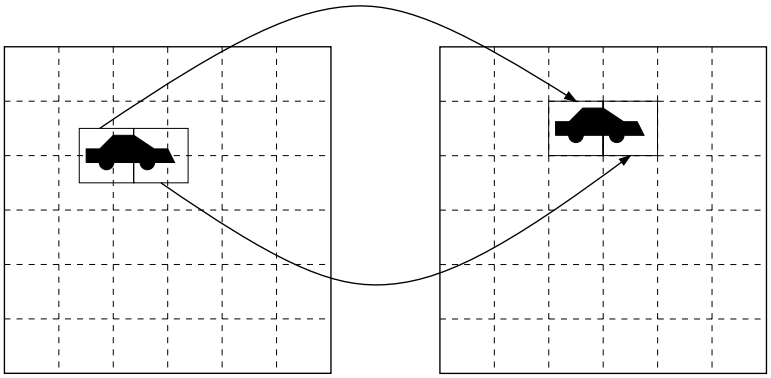



*Fig. 1.14* **Illustration of the Block Translation Model.** In this figure, two blocks in the Current Frame are shown to be copied from a similarly sized region in the Reference Frame. This action forms the basis of the block translation motion model.

overlapping rectangular blocks. Each block in the predicted image is formed by a translation of a similarly shaped source region from the reference frame. The source region needs not coincide with the block boundaries. This model does not consider any rotation or scaling of the blocks, simplifying the motion estimation procedure at the expense of decreased accuracy. A motion vector may be specified in integer or fractional pixel (pel) increments. Fractional-pel motion compensation involves interpolation of the pixel values in the source block. The block translation model is illustrated in Figure 1.14. For each block, the encoder transmits a motion vector that specifies the displacement in the translation model.

Motion estimation algorithms using the block translation model are commonly called block matching algorithms since the procedure involves matching (regularly-positioned) blocks in the current frame with (arbitrarily-positioned) blocks in the reference frame. Because of its simplicity, block matching is commonly used in current video coding standards.

## 1.5 H.261 STANDARD

In 1990, the International Telegraph and Telephone Consultative Committee (CCITT)[8] approved an international standard for video coding at bit rates of $p \times 64$ Kbits/sec, where $p$ is an integer between 1 and 30, inclusive [23, 49].

[8]The CCITT has since changed its name to the International Telecommunication Union Telecommunication Standardization Sector (ITU-T).

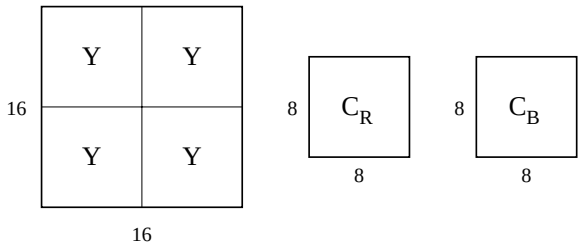



*Fig. 1.15* **Structure of a Macroblock.** This figure shows the structure of a macroblock when the 4:2:0 chroma format is used. The macroblock consists of four adjacent $8 \times 8$ blocks of $Y$ samples, a co-sited $8 \times 8$ block of $Cr$ samples, and a co-sited $8 \times 8$ block of $Cb$ samples. This macroblock structure is used by H.261, H.263, MPEG-1, MPEG-2, and MPEG-4 for 4:2:0 sources.

Officially known as CCITT Recommendation H.261, the standard is intended for low-bit-rate applications such as videophone and video conferencing. We now provide a summary of some key aspects of the standard.

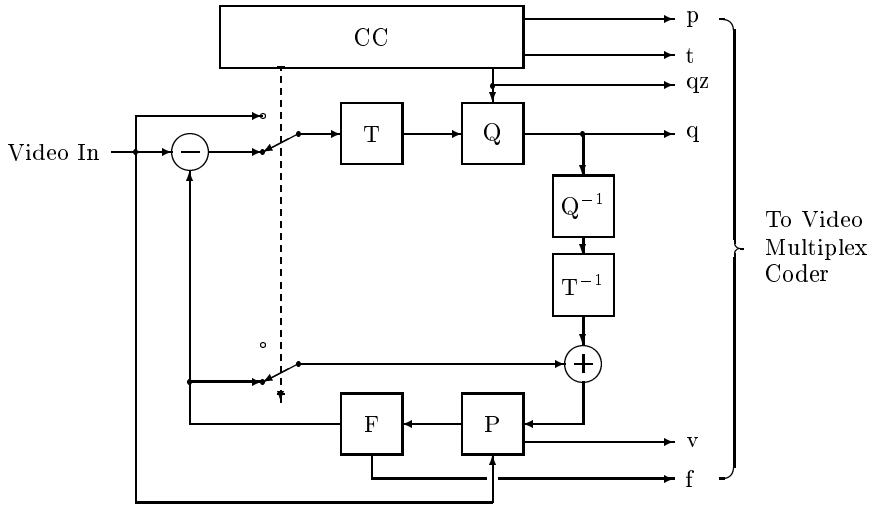
### 1.5.1 Features

The H.261 standard uses a combination of block matching motion compensation (BMMC) and 2D-DCT coding, as described in Sections 1.3.2 and 1.4.3. Since H.261 is intended for real-time video conferencing applications, there is a requirement for low encoding delay, which precludes the use of bidirectional predictive motion compensation. Therefore only intraframe coding and forward predictive coding are used, with a predicted block depending only upon the previous frame. The real-time requirement also restricts the complexity of higher-level algorithms, such as motion estimation and rate control.

The Common Intermediate Format (CIF) and Quarter-CIF (QCIF), described in Section 1.1.6, are specified for video frames. A video frame is divided into *Groups of Blocks* (GOBs) made up of a number of macroblocks (MBs). As depicted in Figure 1.15, each macroblock is composed of four $8 \times 8$ luminance blocks and two $8 \times 8$ chrominance blocks, one each for the $Cb$ and $Cr$ color components. Integer-pel motion compensation is performed at the macroblock level; that is, there is one motion vector per macroblock.

### 1.5.2 Encoder Block Diagram

A block diagram of a basic H.261 coder is shown in Figure 1.16. At a high level, the basic encoding process works as follows: The encoder first decides whether to code a macroblock $M$ using intraframe or interframe coding. For intraframe coding, the techniques outlined in Section 1.3.3 are used. If interframe coding is selected, the encoder performs motion estimation to choose a motion vector $\vec{v}$ (how this is done is left unspecified in the standard). If the




T: Transform
Q: Quantizer
P: Picture Memory with motion-compensated variable delay
F: Loop Filter
CC: Coding Control

p: Flag for INTRA/INTER
t: Flag for transmitted or not
qz: Quantizer indication
q: Quantizing index for transform coefficients
v: Motion vector
f: Switching on/off of the loop filter

*Fig. 1.16* **Block Diagram of an H.261 Video Coder.** This figures shows a block diagram of a typical H.261 video coder [23].

previous macroblock is intracoded, $\vec{v}$ is transmitted using a static Huffman code, otherwise the difference between $\vec{v}$ and the motion vector for the previous macroblock is sent using a static Huffman code. For each $8 \times 8$ block $B$ contained in $M$, a lossy version of the block of prediction errors obtained by using $\vec{v}$ to predict $B$ is then transmitted. This is done by applying the 2D-DCT to the block of prediction errors, quantizing and scanning the transform coefficients, and encoding the results using a run-length/Huffman coder, as prescribed in Section 1.3.3.

The encoder has the option of changing certain aspects of the above process. First, the encoder may simply not transmit the current macroblock; the decoder is then assumed to use the corresponding macroblock in the previous frame in its place. If motion compensation is used, there is an option to apply a linear low-pass filter to the previous decoded frame before using it for prediction.

### 1.5.3 Hypothetical Reference Decoder

In order to place a practical limit on the size of decoder buffers, the H.261 standard defines a Hypothetical Reference Decoder (HRD). Compliant encoders must generate bitstreams that meet the requirements of the HRD.

One requirement of the HRD is that the number of bits used to code any single picture shall not exceed a maximum that depends upon the picture format. The limit is $64 \cdot K$ bits for QCIF and $256 \cdot K$ bits for CIF, where $K = 1024$.

Another requirement is that the HRD buffer shall not overflow. The HRD buffer size is $B + 256 \cdot K$ bits, where $B = 4R_{\max}/29.97$ and $R_{\max}$ is the maximum video bit rate. The HRD buffer is initially empty. The HRD buffer is examined at display intervals that occur every $1/29.97$ sec. If there is at least one complete coded picture in the buffer, the decoder instantaneously removes the bits for the earliest coded picture in the buffer. Otherwise, the decoder waits until the next display interval to examine the buffer. After removal of the coded picture bits, the buffer fullness must be less than $B$. This requirement prevents the encoder from overflowing the decoder buffer.

### 1.5.4 Heuristics for Coding Control

The H.261 standard does not specify how to make coding decisions. However, to aid in the evaluation of different coding techniques, the CCITT provides an encoder simulation model called Reference Model 8 (RM8) [4]. In RM8, motion estimation is performed to minimize the mean absolute difference (MAD) of the prediction errors. A fast three-step search, instead of an exhaustive full-search, is used for motion estimation. RM8 specifies several heuristics used to make the coding decisions. We describe three such heuristics.

The variance $V_{\mathrm{P}}$ of the prediction errors for the luminance blocks in $M$ after motion compensation using $\vec{v}$ is compared against the variance $V_{\mathrm{Y}}$ of the original luminance blocks in $M$ to determine whether to perform intraframe or interframe coding. The intraframe/interframe decision diagram, as specified in RM8, is plotted in Figure 1.17.

If interframe motion compensation mode is selected, the decision of whether to use motion compensation with a zero motion vector or with the estimated motion vector is made by comparing the MAD with zero motion against that with the estimated motion vector. If the zero motion vector is chosen, this is indicated by a special coding mode and no motion vector is coded. The motion vector decision diagram, as recommended in [4], is shown in Figure 1.18.

The loop filter is enabled if a nonzero motion vector is used. The decision of whether to transmit the block transform coefficients is made individually for each block in a macroblock by considering the values of the quantized transform coefficients. If all the coefficients are zero for a block, they are not transmitted for that block.

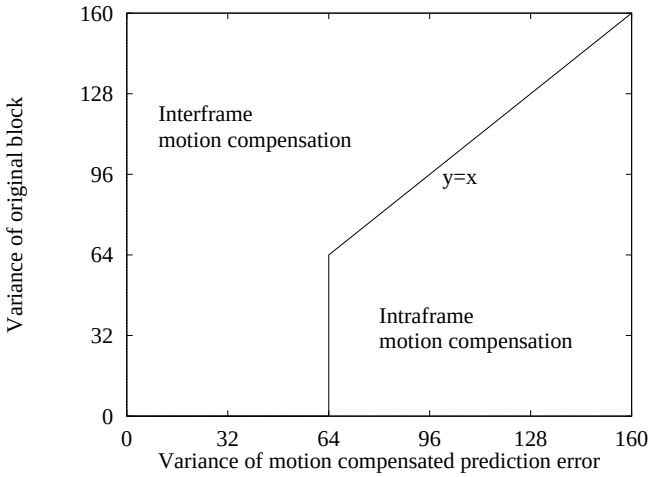



*Fig. 1.17* **Intraframe/Interframe Coding Decision Diagram.** This is the intraframe/interframe coding decision diagram of Reference Model 8 [4]. The variance of the original source block is compared with the variance of the motion-compensated prediction error to determine whether to use intraframe or interframe coding for the source block.

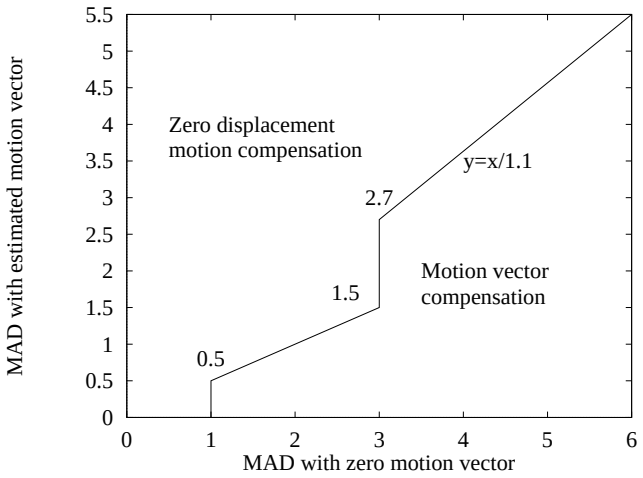



*Fig. 1.18* **Motion Vector Decision Diagram.** This figure shows the motion vector decision diagram of Reference Model 8 [4]. The mean absolute difference (MAD) is used as the measure of motion compensated prediction error. A motion-estimation search is performed to determine a candidate motion vector with the least MAD. The MAD with the candidate motion vector is compared with the MAD with the zero motion vector to determine whether the candidate motion vector or the zero motion vector is used for motion compensation.

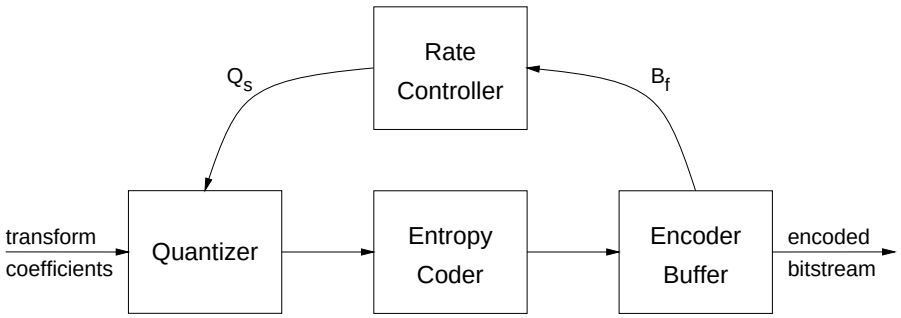



*Fig. 1.19* **Block Diagram of Encoder Rate Control.** This figure shows the functional blocks in a typical video encoder that are involved in bit rate control. The Rate Controller monitors the Encoder Buffer and adjusts the settings of the Quantizer in order to prevent the Encoder Buffer from overflowing or underflowing.

### 1.5.5 Rate Control

Video coders often have to operate within fixed bandwidth limitations. Since the H.261 standard uses variable-length entropy coding of quantized transform coefficients and side information, resulting in a variable bit rate, some form of *bit rate control* is required for operation on bandwidth-limited channels. For example, if the coder's output exceeds the channel capacity, then frames could be dropped or the quality decreased in order to meet the bandwidth constraints. On the other hand, if the coder's output is well below the channel's capacity, the quality and/or frame-rate can be increased to better utilize the channel.

A simple technique for rate control that is specified in RM8 uses a buffered encoding model as shown in Figure 1.19. In this model, the output of the encoder is connected to a buffer whose purpose is to even out the fluctuations in bit rate. By monitoring the fullness of the buffer, the rate controller can adjust the quantization scale $Q_{\mathrm{s}}$, which affects the encoder's bit rate, to prevent the buffer from underflowing or overflowing. In the model, the buffer is defined for the purpose of regulating the output bit rate and may or may not correspond to an actual encoder buffer.

RM8 gives some parameters and prescriptions for the rate control process. The size of the buffer is specified to be $p \times 6.4$ Kbits, which translates to a maximum buffering delay of 100 ms. For purposes of rate control, the first frame is coded using a fixed quantization scale that is computed from the target bit rate. After the first frame is coded, the buffer is reset to be half full. The quantization scale $Q_{\mathrm{s}}$ is determined from the buffer fullness $B_{\mathrm{f}}$ using the formula:

$$Q_{\mathrm{s}} = \min\left\{\lfloor 32B_{\mathrm{f}} \rfloor + 1,\ 31\right\},$$

where $Q_s$ has an integral range of $[1, 31]$, and $B_f$ is normalized to have a real-valued range of $[0, 1]$. The quantization scale is adjusted once for each GOB (11 macroblocks in RM8).

## 1.6 MPEG-1 AND MPEG-2 STANDARDS

In 1988, the International Standards Organization (ISO) formed the Moving Pictures Expert Group (MPEG), with the formal designation ISO-IEC/JTC1 SC29/WG11, to develop standards for the digital encoding of moving pictures (video) and associated audio. In 1991, the MPEG committee completed its first international standard, MPEG-1 [35, 41], formally ISO/IEC 11172.

As a generic video coding specification, MPEG-1 supports multiple image formats, including, CIF, SIF, and QCIF. Image sizes up to $4,095 \times 4,095$ are supported. However, only progressive scan and 4:2:0 color subsampling are supported. While MPEG-1 proved successful for the computer entertainment industry, its lack of support for interlaced scan prevented its use in digital television.

In 1990, the MPEG committee started work on MPEG-2 [24], formally ISO/IEC 13818.[9] MPEG-2 is an extension of MPEG-1 that remedies several major shortcomings of MPEG-1 by adding support for interlaced video, more color subsampling formats, and other advanced coding features. To leverage existing MPEG-1 titles and to promote its adoption, MPEG-2 retains backward compatibility with MPEG-1.

As an international standard, MPEG-2 has been very successful. For example, it has been adopted for use by the Advanced Television Systems Committee (ATSC) as the video compression engine for digital television in the United States. MPEG-2 is currently used in several digital satellite broadcast systems as well as in the consumer digital video disk (DVD).

### 1.6.1 Features

As with the H.261 standard, the MPEG standards specify a syntax for the coded bitstream and a mechanism for decoding the bitstream. Not covered by the standard are details about the encoding process, thus allowing for flexibility and innovation in encoder design and implementation.

Like H.261, the MPEG standards employ a hybrid video coding scheme that combines BMMC with 2D-DCT coding. Unlike H.261, the MPEG standards allow for bidirectional prediction in addition to intraframe coding and forward prediction, all of which are described in Section 1.4. Additionally, the MPEG standards support motion compensation at half-pel accuracy to allow for better image quality at the expense of additional computation. By

[9] Recommendation H.262 is the ITU-T designation of the MPEG-2 standard.

supporting advanced coding techniques, the MPEG standards allow an encoding system to trade off between computation and image quality. This flexibility can be a great advantage for non-real-time encoding systems that can be afforded time to code a video sequence well, especially for applications such as movies and commercials, where the quality of the coded video is of utmost importance since the video will be played many times. However, by using a subset of the coding features, the MPEG standards can also be used for real-time applications such as video conferencing, news, and other live broadcasts.

### 1.6.2 Encoder Block Diagram

A basic encoder block diagram is shown in Figure 1.20, with the embedded decoder highlighted. The structure of the encoder is very similar to that in Figure 1.16. The main differences, as outlined above, are hidden in the Coding Control, Motion Estimation, and Motion Compensation blocks.

### 1.6.3 Layers

In the syntax of the MPEG standards, a video sequence is partitioned into a hierarchy of *layers*. The presence of a layer in the bitstream is indicated by a *start code* indicating the layer type followed by a *header* that specifies parameters for that layer. At the top of the hierarchy is the *sequence layer*. The sequence header specifies information for the entire video sequence, such as the frame size, frame rate, bit rate, and quantization matrix. Below the sequence layer is the *group of pictures layer* (GOP layer). A GOP is structured as a set of contiguous frames that contains at least one I frame. A GOP can start with either a B frame or an I frame and end with either an I frame or a P frame. The GOP structure is designed to support random access, indexing, and editing. An example of a GOP unit can be found in Figure 1.10(a). If a GOP begins with a frame that does not depend upon a preceding frame, it can be decoded and displayed independently of the previous GOP and is called a *closed* GOP. GOPs that are not closed are referred to as *open*.

Below the GOP layer is the *picture layer*. The picture header contains information about each picture[10] coded, such as picture type (I, P, or B) and temporal reference. A picture is divided into *slices*, each of which consists of a segment of consecutive *macroblocks*. Dividing a picture into slices limits the effects of transmission errors and allows the decoder to recover from these errors.

A macroblock consists of a number of $8 \times 8$ *blocks* of intensity and chrominance values. The number of blocks in a macroblock depends upon the color

[10]With progressive scan, a *picture* in MPEG's terminology is equivalent to what we have been calling a *frame*. With interlaced scan, a picture may refer to a single field.

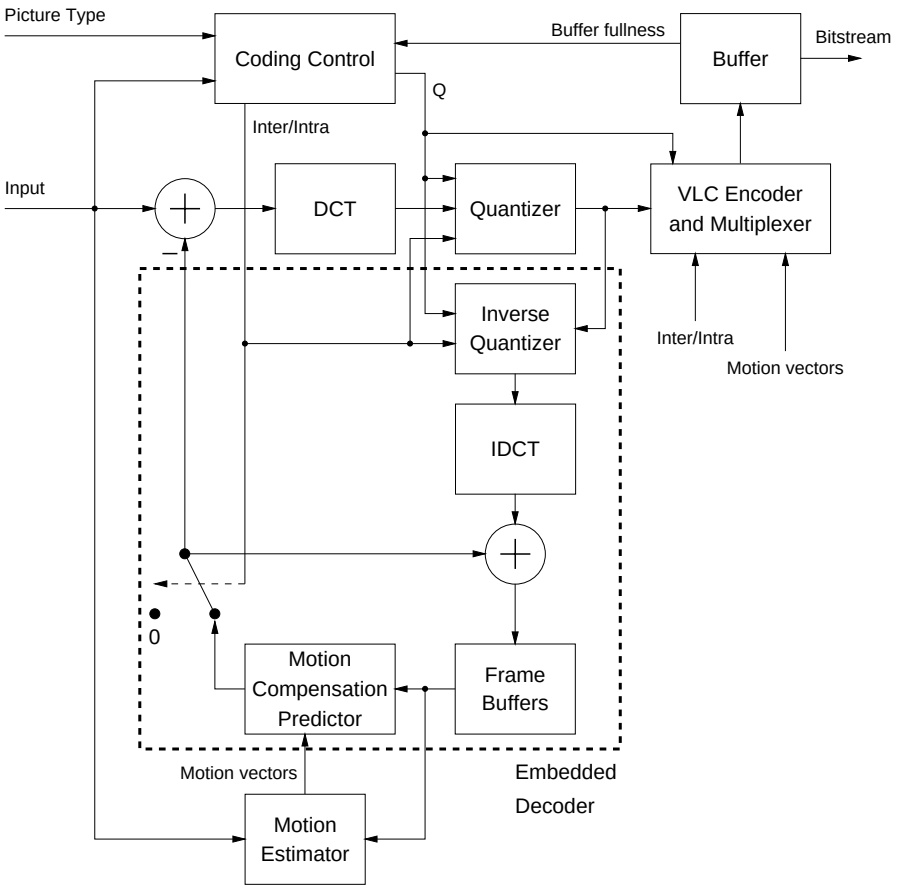



*Fig. 1.20* **Block Diagram of an MPEG Encoder.** This figure shows a block diagram of a typical MPEG encoder. The dashed box contains the blocks that perform the same functions as an MPEG decoder. The *embedded decoder* is needed to generate the same reference frames that a real decoder would.

subsampling scheme used (see Figures 1.4 and 1.5). For 4:2:0 subsampling, the structure of a macroblock is shown in Figure 1.15. For 4:2:2 subsampling, a macroblock contains four $Y$ blocks, two $Cb$ blocks, and two $Cr$ blocks. MPEG-2 supports an additional color subsampling mode, 4:4:4, in which the $Cb$ and $Cr$ color components have the same spatial resolution as the luminance component $Y$. Thus for 4:4:4 color subsampling, a macroblock consists of a total of twelve blocks.

As with JPEG and H.261, the $8 \times 8$ block is the basic unit for DCT coding and quantization.

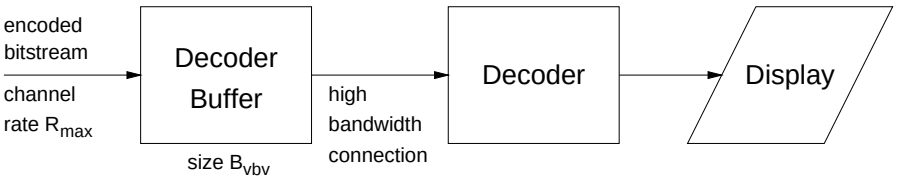



*Fig. 1.21* **MPEG Video Buffering Verifier.** This figure shows a block diagram of the MPEG Video Buffering Verifier, which is an idealized decoder model used to derive buffering constraints.

### 1.6.4 Video Buffering Verifier

In addition to defining a bitstream syntax and decoding process, the MPEG video standards define an idealized decoder model called the *Video Buffering Verifier* (VBV). The purpose of the VBV is to put quantifiable limits on the variability in the coding rate such that an encoded bitstream can be decoded with reasonable buffering requirements. As diagrammed in Figure 1.21, the VBV consists of a decoder buffer, a decoder, and a display unit. Encoded bits enter the decoder buffer at piecewise-constant rates up to $R_{\max}$. The decoder buffer stores the incoming bits for processing by the decoder. At regular display intervals, the decoder *instantaneously* removes, decodes, and displays the earliest picture in the buffer. It should be emphasized that the VBV is only an idealized decoder model and not a prescription of how to build a decoder or how an actual decoder would function. The VBV model, however, is useful in establishing rate constraints on encoded video such that the encoding would be decodable with the specified buffering requirements.

In a special *low-delay* mode, the VBV operates much like the HRD of H.261. In this mode, the VBV examines the buffer every display interval. If all the bits for the earliest picture are present in the buffer, those bits are removed for decoding. Otherwise, the decoder waits until the next display interval. The low-delay mode is intended for real-time video communications, where encoder and buffering delays are to be minimized. However, the vast majority of MPEG applications do not use the low-delay mode and we do not consider this mode further.

When not in low-delay mode, the VBV has three prescribed modes of operation: a *constant bit rate* (CBR) mode and two *variable bit rate* (VBR) modes. MPEG-1 supports only the CBR mode while MPEG-2 supports all three modes. In CBR mode, bits enter the decoder buffer at a constant rate $R_{\max}$ as specified in the sequence header. Initially, the buffer is empty and fills for a prespecified amount of time before bits for the first picture are removed and decoded. Afterwards, the buffer continues to fill at the channel rate $R_{\max}$ while the decoder removes bits for coded pictures at regular display intervals. The CBR mode models operation of a decoder connected to a constant bit rate channel with a fixed channel delay, as shown in Figure 1.22.

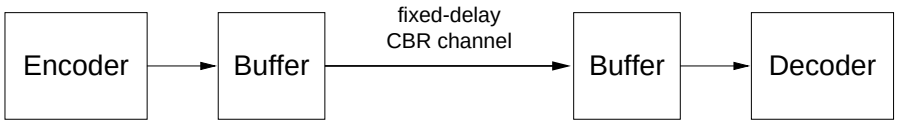



*Fig. 1.22* **Fixed-Delay CBR Video Transmission System.** This figure shows an encoder connected to a decoder through a fixed-delay CBR channel. A buffer is needed after the Encoder to smooth the bit rate for transmission over the CBR channel. Another buffer is needed before the Decoder to collects bits for decoding.

The amount of time that the start code for a given picture is to reside in the VBV buffer before that picture is decoded is specified in the picture header with a parameter called **vbv_delay**.[11] In CBR mode, **vbv_delay** is related to the VBV buffer fullness $B_{\mathrm{f}}$ in the following manner:

$$\mathbf{vbv\_delay} = \frac{90000 \cdot B_{\mathrm{f}}}{R_{\max}}.$$

In the first VBR mode, the compressed bits for picture $n$ enter the buffer at a piecewise-constant rate $R(n)$ that may vary from picture to picture, up to the maximum rate $R_{\max}$ specified in the sequence header. The relationship between $R(n)$ and **vbv_delay** is as follows:

$$R(n) = \frac{s_n}{\tau(n) - \tau(n+1) + t(n+1) - t(n)},$$

where

$$\begin{aligned} R(n) &= \text{the rate at which bits for picture } n \text{ enter the VBV buffer,} \\ s_n &= \text{the number of bits for picture } n, \\ \tau(n) &= \text{the decoding delay coded in } \mathbf{vbv\_delay} \text{ for picture } n, \text{ and} \\ t(n) &= \text{the time when picture } n \text{ is removed from the VBV buffer.} \end{aligned}$$

In the second VBR mode, **vbv_delay** is set to 65535 for all pictures. The VBV buffer is initially filled to capacity at the peak rate $R_{\max}$ before the first picture is removed. Thereafter, in each display interval, bits enter the buffer at the peak rate until the buffer is full, at which point bits stop entering the buffer until the next picture has been decoded. When the buffer is full, bits are not discarded, however. The channel is assumed to be able to hold the bits until needed by the VBV. For stored-video applications, this requirement can be met using the double-buffering scheme shown in Figure 1.23, for example. With a maximum disk latency of $T_L$, a buffer size of $T_L R_{\max}$ is sufficient to guarantee timely delivery of bits to the VBV.

[11] The parameter **vbv_delay** is coded as an integer in the range $[0, 65534]$ and is expressed in units of 1/90000 sec.

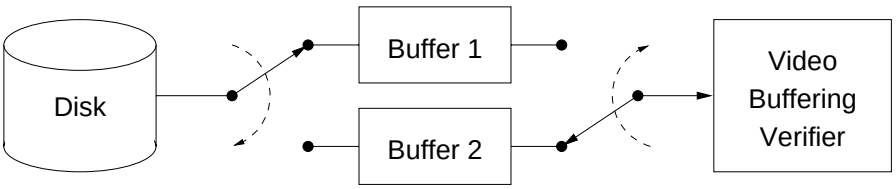



*Fig. 1.23* **Stored-Video System with Double Buffering.** This figure shows the block diagram of a video decoder that plays compressed video stored on a disk storage medium. The data transfer from the disk is double-buffered to hide the latency of the disk medium.

Since the decoder buffer stops receiving bits when it is full, a potentially variable number of bits can enter the buffer in each display period. The second VBR mode can be thought of as modeling the operation of a decoder connected to a channel or device (e.g., a disk drive) that can transfer data at a variable rate up to a peak rate $R_{\max}$.

For proper operation in any mode, the decoder buffer should not exceed its capacity $B_{\text{vbv}}$ as specified in the sequence header.[12] Also, the buffer should contain at least the number of bits needed to decode the next picture at the time it is to be decoded. As will be shown in Chapter 2, these requirements impose constraints on the number of bits that the encoder can produce for each picture.

A compliant encoder must produce a bitstream that results in proper operation of the VBV. The VBV is not intended to be a prescription for an actual decoder implementation. However, a compliant decoder implementation should be able to decode successfully (within resource limits) any bitstream that meets the VBV buffering constraints.

### 1.6.5 Rate Control

As with H.261, the MPEG standards do not specify how to perform rate control. To allow for testing and experimentation using a common set of encoder routines, MPEG created a series of test models. Here, we describe the rate control strategy outlined in the MPEG-2 Test Model 5 (TM5) [34]. In TM5, rate control is broken down into three steps:

1. **Target bit allocation**. In this step, the complexity of the current picture is estimated based upon the encoding of previous pictures to allocate a number of bits to code the picture.

[12] This requirement is always met in the second VBR mode.

2. **Rate control**. A reference quantization scale is determined using a virtual buffer in a feedback loop to regulate the coding rate so as to achieve the target bit allocation.

3. **Adaptive quantization**. The reference quantization scale is modulated according to the spatial activity in each macroblock to determine the actual quantization scale with which to code the macroblock.

*Target Bit Allocation.* The number of bits to be allocated to a picture depends upon its type: I, P, or B. For each picture type, there is a complexity model that attempts to relate the number of bits that would result from coding a picture of a given type to the quantization scale used. The complexity models are of the form

$$S_{\mathrm{i}} = \frac{X_{\mathrm{i}}}{Q_{\mathrm{i}}}, \qquad S_{\mathrm{p}} = \frac{X_{\mathrm{p}}}{Q_{\mathrm{p}}}, \qquad S_{\mathrm{b}} = \frac{X_{\mathrm{b}}}{Q_{\mathrm{b}}},$$

where $S$, $X$, and $Q$ denote number of bits, complexity, and quantization scale, respectively; and the subscript indicate the picture type.

Initially the complexity values are set to:

$$X_{\mathrm{i}} = \frac{160 \cdot \textbf{bit\_rate}}{115}, \qquad X_{\mathrm{p}} = \frac{60 \cdot \textbf{bit\_rate}}{115}, \qquad X_{\mathrm{b}} = \frac{42 \cdot \textbf{bit\_rate}}{115},$$

where **bit_rate** is measured in bits/sec.

After a picture of a given type is coded, its associated complexity model is updated based upon the average quantization scale used and number of bits produced:

$$X_{\mathrm{i}} = S_{\mathrm{i}} Q_{\mathrm{i}}, \qquad X_{\mathrm{p}} = S_{\mathrm{p}} Q_{\mathrm{p}}, \qquad X_{\mathrm{b}} = S_{\mathrm{b}} Q_{\mathrm{b}}.$$

Bit allocation is performed with the goal that the average bit rate is achieved at the GOP layer. A corresponding number of bits is assigned to code each GOP. Bits are allocated to each picture in a GOP based upon the complexity models, the number of bits available to code the remaining pictures in the GOP, and the number of remaining I, P, and B pictures in the GOP. Let $N$ be the total number of pictures in a GOP. Let $N_{\mathrm{i}}$, $N_{\mathrm{p}}$, and $N_{\mathrm{b}}$ be the number of remaining I, P, and B pictures, respectively. The bit target for each type of picture is calculated according to

$$T_{\mathrm{i}} = \max\left\{\frac{R}{1 + \frac{N_{\mathrm{p}} X_{\mathrm{p}}}{K_{\mathrm{p}} X_{\mathrm{i}}} + \frac{N_{\mathrm{b}} X_{\mathrm{b}}}{K_{\mathrm{b}} X_{\mathrm{i}}}}, \frac{\textbf{bit\_rate}}{8 \cdot \textbf{picture\_rate}}\right\};$$

$$T_{\mathrm{p}} = \max\left\{\frac{R}{N_{\mathrm{p}} + \frac{N_{\mathrm{b}} K_{\mathrm{p}} X_{\mathrm{b}}}{K_{\mathrm{b}} X_{\mathrm{p}}}}, \frac{\textbf{bit\_rate}}{8 \cdot \textbf{picture\_rate}}\right\};$$

$$T_{\mathrm{b}} = \max\left\{\frac{R}{N_{\mathrm{b}} + \frac{N_{\mathrm{p}} K_{\mathrm{b}} X_{\mathrm{p}}}{K_{\mathrm{p}} X_{\mathrm{b}}}}, \frac{\textbf{bit\_rate}}{8 \cdot \textbf{picture\_rate}}\right\},$$

where $K_{\mathrm{p}}$ and $K_{\mathrm{b}}$ are constants that depend upon the quantization matrices,[13] and $R$ is the number of bits available to code the remaining pictures in the GOP.

After all the pictures in a GOP have been coded, any difference between the target and actual bit allocation is carried over to the next GOP.

*Rate Control.* Given a target bit allocation for a picture, a virtual encoding buffer is used to determine a reference quantization scale in a way similar to the procedure in Section 1.5.5 for the H.261 standard. A separate virtual buffer is maintained for each picture type. The reference quantization scale is computed for each macroblock from the buffer fullness as

$$Q_j = \frac{31 \cdot d_j}{r},$$

where $Q_j$ is the reference quantization scale for macroblock $j$, $d_j$ is the buffer fullness, and $r$ is a *reaction parameter* given by

$$r = \frac{2 \cdot \mathbf{bit\_rate}}{\mathbf{picture\_rate}}.$$

*Adaptive Quantization.* The rate control step provides a reference quantization scale to code each macroblock. The reference quantization scale is modulated with an *activity factor* that is determined from a measure of the spatial activity of the macroblock. The rationale is that a macroblock that has little spatial activity, such as a smooth region, should be quantized more finely than a macroblock with high spatial activity, such as a textured region, since quantization error is typically more noticeable in smooth regions than in textured regions. This approach attempts to equalize perceptual quality for a given quantization scale.

For macroblock $j$, an activity factor $\mathrm{act}_j$ is computed as one plus the minimum of the variances of the luminance blocks within the macroblock. A normalized activity factor is then computed based upon the average of the activity factor of the last encoded picture:

$$N_{\mathrm{act}} = \frac{2 \cdot \mathrm{act}_j + \mathbf{avg\_act}}{\mathrm{act}_j + 2 \cdot \mathbf{avg\_act}}.$$

For the first picture, $\mathbf{avg\_act} = 400$. The actual quantization scale $Q_{\mathrm{s}}$ used to code the macroblock is computed as

$$Q_{\mathrm{s}} = \min\{Q_j \cdot N_{\mathrm{act}},\, 31\}.$$

It should be noted that the TM5 rate control strategy does not take into account the VBV and therefore does not guarantee VBV compliance. Also, TM5 only performs rate control for CBR operation and not for VBR.

[13] For the matrices specified in TM5, $K_{\mathrm{p}} = 1.0$ and $K_{\mathrm{b}} = 1.4$.

*Table 1.1* **Standardized H.263 Picture Formats.** H.263 supports these standardized picture formats in addition to a custom picture format [25].

| Picture | Luminance | | Chrominance | |
|---|---|---|---|---|
| Format | Width | Height | Width | Height |
| sub-QCIF | 128 | 96 | 64 | 48 |
| QCIF | 176 | 144 | 88 | 72 |
| CIF | 352 | 288 | 176 | 144 |
| 4CIF | 704 | 576 | 352 | 288 |
| 16CIF | 1408 | 1152 | 704 | 576 |

## 1.7 H.263 STANDARD

Building on the H.261 standard, the ITU-T released an improved low-bit-rate video coding standard in 1996 called Recommendation H.263 [25]. H.263 offers improved compression efficiency and quality over H.261 and is designed to operate below 64 Kbits/sec. In 1998, the ITU-T released Recommendation H.263 Version 2, also known as H.263+, which is compatible with the original H.263 Version 1 with extra optional features. The ITU-T is currently working on a third version of H.263, unofficially known as H.263++, and a new standard for very low-bit-rate video coding called H.26L [67].

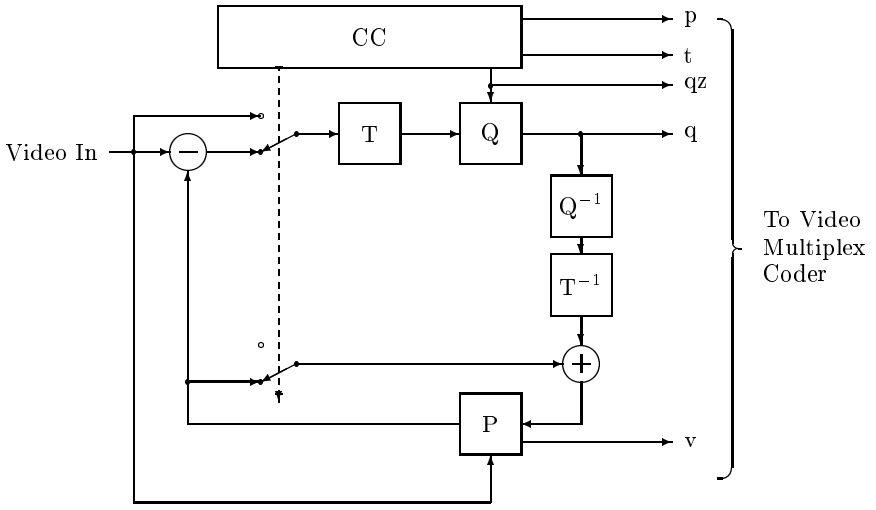
### 1.7.1 Features

As shown in Figure 1.24, H.263 maintains the same basic encoder structure as H.261, except without the loop filter. In lieu of the loop filter, H.263 employs half-pel motion vectors in order to improve motion compensation performance.

H.263 supports five standardized picture formats and a custom picture format. The standardized formats are listed in Table 1.1. A custom picture format may be specified in the picture header with a width that is a multiple of four in the range $[4, 2048]$ and a height that is a multiple of four in the range $[4, 1152]$.

In addition to the core coding tools, H.263 Version 1 provides four optional coding tools: Unrestricted Motion Vector, Syntax-based Arithmetic Coding, Advanced Prediction Mode, and PB-frames.

*Unrestricted Motion Vector.* With this tool, motion vectors are allowed to extend beyond the picture boundary. Edge pixels are used as prediction outside the picture area. This option improves compression efficiency when there is motion near picture boundaries, especially for small picture sizes. The motion vector range is also extended to benefit large pictures.




T: Transform
Q: Quantizer
P: Picture Memory with motion-compensated variable delay
F: Loop Filter
CC: Coding Control

p: Flag for INTRA/INTER
t: Flag for transmitted or not
qz: Quantizer indication
q: Quantizing index for transform coefficients
v: Motion vector

*Fig. 1.24* **Block Diagram of an H.263 Video Coder.** This figures shows a block diagram of a typical H.263 video coder [25].

*Syntax-based Arithmetic Coding.* Arithmetic coding replaces variable-length Huffman codes for improved compression with no change in image quality.

*Advanced Prediction Mode.* In this mode, an advanced motion compensation technique called Overlapped Block Motion Compensation (OBMC) is applied to the luminance component. One motion vector is associated with each $8 \times 8$ luminance block. The prediction for each pixel is the weighted average of the predictions using the motion vectors for the current block and the two nearest neighboring block in its four-connected neighborhood (left, right, above, below). For example, the prediction for pixels in the lower left $4 \times 4$ quadrant of a block is formed from the motion vectors from the current block and the two blocks to the left and below.

The increased number of motion vectors requires more bits to transmit but results in better motion compensated prediction. In addition, the overlapped blending of prediction blocks results in fewer blocking artifacts.

*PB-frames Mode.* This optional mode adds bidirectional prediction using PB-frames. A PB-frame combines two pictures into one coding unit. The second

*Table 1.2* **Minimum BPPmaxKb as Function of Picture Size.** This table lists the minimum value that the **BPPmaxKb** parameter can take as a function of the picture size in pixels, where **BPPmaxKb** is the maximum compressed size of a picture in units of 1024 bits [25].

| Picture Size in Pixels | Minimum BPPmaxKb |
|---|---|
| up to 25,344 (or QCIF) | 64 |
| 25,360 to 101,376 (or CIF) | 256 |
| 101,392 to 405,504 (or 4CIF) | 512 |
| 405,520 and above | 1024 |

picture in the unit is coded with forward prediction and the first with bidirectional prediction. The use of PB-frames allows for increased frame rate with a small increase in bits.

*Version 2 Modes.* In addition to the above coding tools, H.263 Version 2 adds twelve more negotiable options. These additional tools address coding efficiency, scalability, and error resilience. It is beyond the scope of this book to discuss these in any detail. The reader is referred to [25, 74].

### 1.7.2 Hypothetical Reference Decoder

As with the H.261 standard, H.263 defines a Hypothetical Reference Decoder (HRD). Since H.263 supports additional picture formats beyond the QCIF and CIF formats supported by H.261, the parameters of the HRD have been modified accordingly.

The number of bits used to code any single picture shall not exceed a maximum defined by the parameter **BPPmaxKb** in units of 1024 bits. A minimum value for **BPPmaxKb** is defined based upon the picture size, as shown in Table 1.2. The encoder may negotiate a larger value of **BPPmaxKb**.

The HRD buffer size in bits is defined to be $B + 1024 \cdot \mathbf{BPPmaxKb}$, where $B = 4R_{\max}/\mathrm{PCF}$, $R_{\max}$ is the maximum video bit rate, and PCF is the picture clock frequency in Hz. The HRD buffer is initially empty and is examined at picture clock intervals (1/PCF sec). If there is at least one complete coded picture in the buffer, the decoder instantaneously removes the bits for the earliest coded picture in the buffer. Otherwise, the decoder waits until the next picture clock interval. After removal of the coded picture bits, the buffer fullness must be less than $B$.

There is no buffer underflow condition because the HRD simply waits until all the bits for the next coded picture are in the buffer before removing them. This implies that some pictures may be repeated when the coding rate exceeds $R_{\max}$ for several picture clock intervals.

## 1.8 LOSSY CODING AND RATE-DISTORTION

The examples in Section 1.2 show that existing video applications require high compression ratios, over an order of magnitude higher than what is typically possible with lossless compression methods. These high levels of compression can be realized only if we accept some loss in fidelity between the uncompressed and compressed representations. There is a natural tradeoff between the size of the compressed representation and the fidelity of the reproduced images. This tradeoff between coding rate and distortion is quantified in rate-distortion theory.

### 1.8.1 Classical Rate-Distortion Theory

Let $D$ be a measure of distortion according to some fidelity criterion. In classical rate-distortion theory, as pioneered by Claude Shannon [71], a rate-distortion function, $R(D)$, is defined to be the theoretical lower bound on the best coding rate achievable as a function of the desired distortion $D$ for a given information source, by *any* compressor. In general, the fidelity criterion can be any valid metric; in practice, a squared-error distortion is often used; that is, $D(x, \hat{x}) = (x - \hat{x})^2$.

For a discrete source, $R(0)$ is simply the entropy of the source and corresponds to lossless coding ($D = 0$). In cases where the distortion is bounded from above by $D_{\max}$, then $R(D_{\max}) = 0$. Furthermore, it can be shown that $R(D)$ is a nonincreasing convex function of $D$ (see, e.g., [14]).

For some specific information sources and distortion measures, closed-form expressions for the rate-distortion function have been determined. For a zero-mean Gaussian source with variance $\sigma^2$ and a squared-error distortion measure,

$$R(D) = \begin{cases} \frac{1}{2} \log_2 \frac{\sigma^2}{D}, & \text{if } 0 \leq D \leq \sigma^2; \\ 0, & \text{if } D > \sigma^2. \end{cases}$$

This is plotted for $\sigma = 1$ in Figure 1.25.

### 1.8.2 Operational Rate-Distortion

In practice, classical rate-distortion theory is not directly applicable to complex encoding and decoding systems since sources are typically not well characterized and $R(D)$ is difficult, if not impossible, to determine. Even though not directly computable, the existence of a hypothetical rate-distortion function for a given type of information source allows a comparison to be made between competing encoding systems and algorithms.

A more practical approach is taken in [8, 72]. By measuring actual coding rates and distortion achieved by the coder under study, an *operational rate-distortion* plot similar to Figure 1.26 can be constructed. It is sometimes

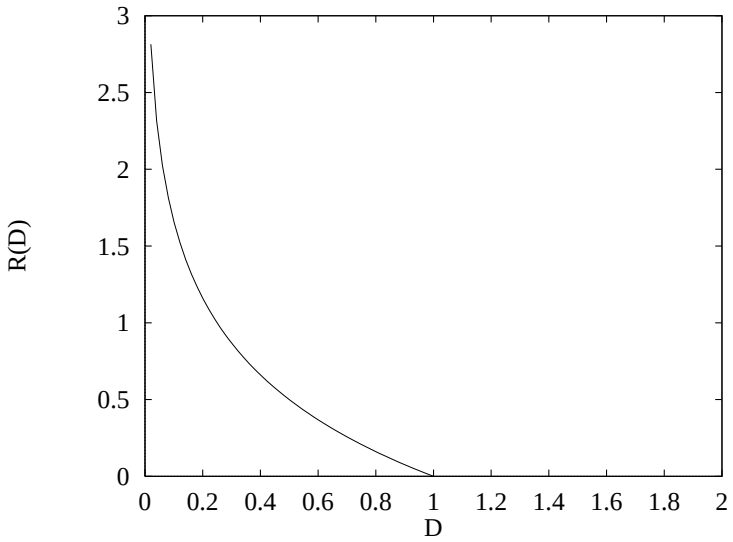



*Fig. 1.25* **Rate-Distortion Function for a Gaussian Source** This figure shows the theoretical rate-distortion function for a zero-mean Gaussian source with variance $\sigma^2 = 1$.

useful to show the convex hull of the data points to fill in the gap between points. Data points are typically generated by varying the level of quantization or other coding parameters under study.

By plotting operational rate-distortion curves for various competing coders, a comparison can be made of their effectiveness. A basic idea is that the more effective and capable a coder is, the closer is its operational rate-distortion curve to the hypothetical rate-distortion function. In Figure 1.27, Coder 1 performs better than Coder 2 for coding rates greater than about 600 bits per *coding unit*, where a coding unit is a generic term for a block of data. At coding rates less than 600 bits/unit, Coder 2 performs better.

The mean square error (MSE) distortion measure is commonly used in the literature since it is a convenient measure that lends itself to mathematical analysis. For images and video, however, MSE is not an ideal measure since it is not a good model of human visual perception. For example, in many cases, two encodings with the same MSE can have remarkably different perceptual quality. However, in keeping with convention, we assume the use of MSE as the distortion measure for the remainder of this chapter.

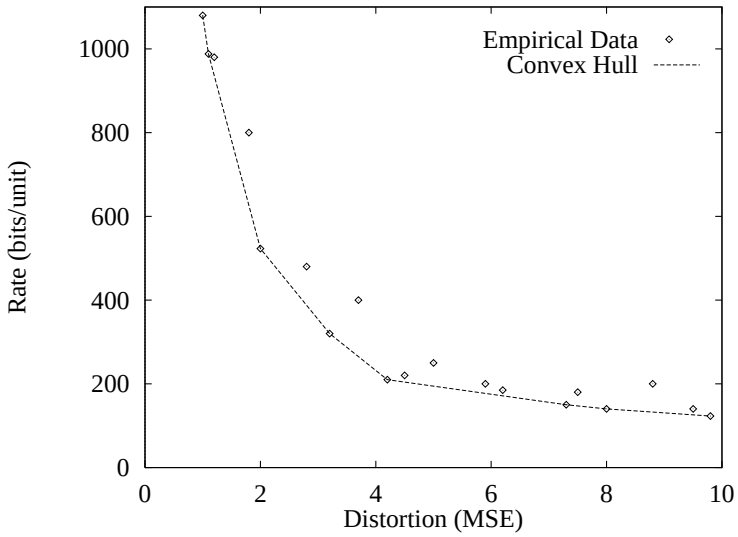



*Fig. 1.26* **Operational Rate-Distortion Plot.** This figure illustrates the practice of estimating the rate-distortion characteristics of a coder by plotting measured coding rate and distortion values. The lower convex hull of the plotted measurements forms an approximation to the rate-distortion function.

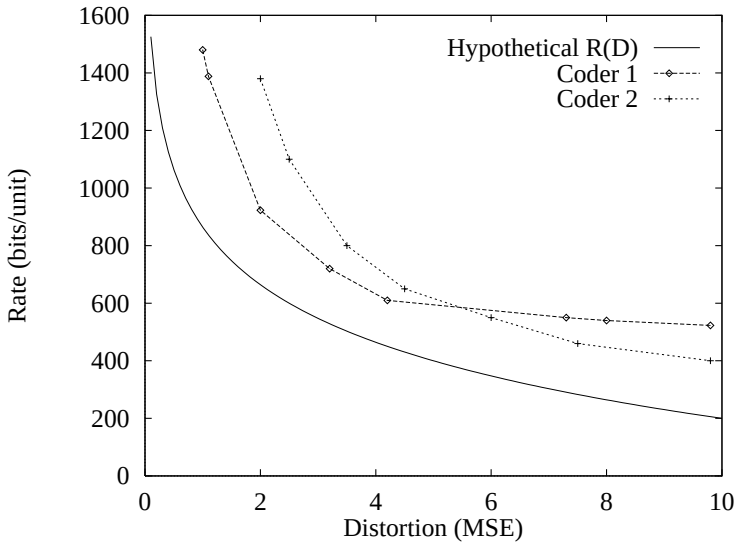



*Fig. 1.27* **Comparison of Coders Using Operational Rate-Distortion.** This figure shows how two coders can be evaluated by comparing their operational rate-distortion curves. From this plot, it can be seen that Coder 1 performs better than Coder 2 for coding rates greater than about 600 bits/unit.

### 1.8.3 Budget-Constrained Bit Allocation

A problem that well illustrates the operational rate-distortion framework is the budget-constrained bit allocation problem, which we state below. Without loss of generality, quantization is the coding parameter to be adjusted.

**Problem 1.1** *Given a set of quantizers* $\{q_1, q_2, \ldots, q_M\}$*, a sequence of* $N$ *blocks, and a target bit budget* $B$*, determine an assignment of quantizers* $\boldsymbol{Q} = \langle Q_1, Q_2, \ldots, Q_N \rangle$ *to each block that minimizes a distortion measure* $D(\boldsymbol{Q})$ *and uses* $R(\boldsymbol{Q}) \leq B$ *bits.*

### 1.8.4 Viterbi Algorithm

Problem 1.1 can be solved using a dynamic programming algorithm commonly referred to as the Viterbi algorithm (VA) [18, 77]. Assuming that quantization always produces an integral number of bits, the Viterbi algorithm works by first constructing a trellis of nodes and then finding a shortest path through the trellis. Each node represents a state and each edge a transition between states. For the bit allocation problem, we identify each state with a tuple $(b, t, d, p)$, where $t$ is a time index, $b$ is the total number of bits used in an allocation for the sequence of blocks $\langle x_1, x_2, \ldots, x_t \rangle$, $d$ is the minimum sum distortion for any allocation to those blocks using exactly $b$ bits, and $p$ is a pointer back to a previous state. There is a single start state labeled $(0, 0, 0, 0)$.

Starting with the start state, we construct the trellis by adding an edge for each choice of quantizer and creating a corresponding set of new states. The new states record the number of bits and minimum distortion for all choices of quantizer for coding the first block. There may be more than one edge entering a new state if more than one quantizer results in the same number of bits. However, only the minimum distortion is recorded as $d$, and $p$ is made to point to a source state that results in the minimum distortion. In case more than one incoming edge produces the minimum distortion, the pointer can point to any of the edges with the minimum distortion. This process is repeated so that new states for time index $k+1$ are constructed by adding edges corresponding to the quantization of block $x_{k+1}$ to the states with time index $k$. In the trellis construction, we prune out those states whose bit consumption exceeds the bit budget $B$. After the states with time index $N$ have been constructed, we pick a state with time index $N$ that has the minimum distortion. The sequence of bit allocations can then be constructed by following the pointers $p$ back from the end state to the start state.

A simple example with $M = 2$ and $N = 3$ is shown in Figure 1.28 to illustrate the Viterbi algorithm. In the example, the shaded node marks a state that exceeds the bit budget $B$ and can be pruned. An optimal path is shown with thick edges. As in this example, there may be more than one path with the minimum distortion.

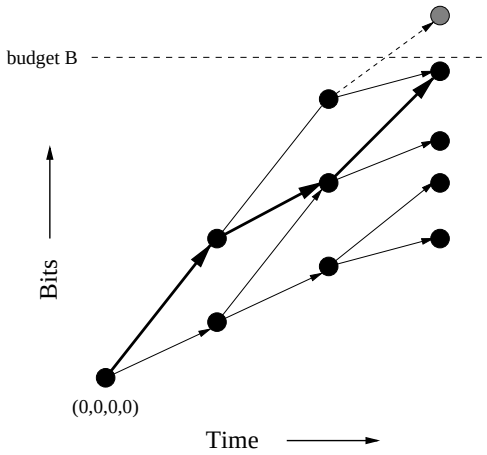



*Fig. 1.28* **Trellis Construction by the Viterbi Algorithm.** This figure shows an example of a trellis constructed with the Viterbi Algorithm. The shaded node marks a state that exceeds the bit budget $B$ and can be pruned. An optimal path is shown with thick edges. Note that there may be more than one optimal path.

### 1.8.5 Lagrange Optimization

Although the Viterbi algorithm finds an optimal solution to Problem 1.1, it is computationally expensive. There could potentially be an exponential number of states generated, on the order of $M^N$.

Shoham and Gersho [72] give an efficient bit allocation algorithm based upon the Lagrange multiplier method [17]. In this method, Problem 1.1, a constrained optimization problem, is transformed to the following unconstrained optimization problem:

**Problem 1.2** *Given a set of $M$ quantizers $\{q_1, q_2, \ldots, q_M\}$, a sequence of $N$ blocks, and a parameter $\lambda$, determine an assignment of quantizers $\boldsymbol{Q} = \langle Q_1, Q_2, \ldots, Q_N \rangle$ to each block that minimizes the cost function $C_\lambda(\boldsymbol{Q}) = D(\boldsymbol{Q}) + \lambda R(\boldsymbol{Q})$.*

Here the parameter $\lambda$ is called the Lagrange multiplier. Let $\boldsymbol{Q}^*(\lambda)$ denote an optimal assignment of quantizers for Problem 1.2 given $\lambda$, and let $R\big(\boldsymbol{Q}^*(\lambda)\big)$ denote the resulting total number of bits allocated. (There may be more than one optimal solution for a given $\lambda$, with each solution having a different coding rate $R$.) It can be shown that a solution to Problem 1.2 is also a solution to Problem 1.1 with $B = R\big(\boldsymbol{Q}^*(\lambda)\big)$, where $B$ is the target bit budget of Problem 1.1. This is proved in [17], and we reproduce the theorem and proof as presented in [72].

**Theorem 1.1** *For any $\lambda \geq 0$, a solution $\boldsymbol{Q}^*(\lambda)$ to Problem 1.2 is also a solution to Problem 1.1 with the constraint $R(\boldsymbol{Q}) \leq R\big(\boldsymbol{Q}^*(\lambda)\big)$.*

*Proof*: For the solution $\boldsymbol{Q}^*(\lambda)$, we have

$$D\big(\boldsymbol{Q}^*(\lambda)\big) + \lambda R\big(\boldsymbol{Q}^*(\lambda)\big) \le D(\boldsymbol{Q}) + \lambda R(\boldsymbol{Q})$$

for all quantizer allocations $\boldsymbol{Q}$. Equivalently, we have

$$D\big(\boldsymbol{Q}^*(\lambda)\big) - D(\boldsymbol{Q}) \le \lambda R(\boldsymbol{Q}) - R\big(\boldsymbol{Q}^*(\lambda)\big)$$

for all $\boldsymbol{Q}$. In particular, this result applies for all $\boldsymbol{Q}$ belonging to the set

$$S^* = \big\{\boldsymbol{Q} \;\big|\; R(\boldsymbol{Q}) \le R\big(\boldsymbol{Q}^*(\lambda)\big)\big\}.$$

Since $\lambda \ge 0$ and $R(\boldsymbol{Q}) - R\big(\boldsymbol{Q}^*(\lambda)\big) \le 0$ for $\boldsymbol{Q} \in S^*$, we have

$$D\big(\boldsymbol{Q}^*(\lambda)\big) - D(\boldsymbol{Q}) \le 0, \qquad \text{for } \boldsymbol{Q} \in S^*.$$

Therefore $\boldsymbol{Q}^*(\lambda)$ is a solution to the constrained problem. ∎

The Lagrange multiplier $\lambda$ can be viewed as determining a tradeoff between coding rate and distortion. A low value for $\lambda$ favors minimizing distortion over coding rate, and a high value favors minimizing coding rate over distortion. In the limit, when $\lambda = 0$, we are minimizing distortion; as $\lambda \to \infty$, we minimize coding rate.

Lagrange optimization can be interpreted graphically as shown in Figure 1.29. The minimization of the Lagrange cost function $C_\lambda$ can be viewed as finding the last point or points intersected in the rate-distortion plane as a line with slope $-\lambda$ is swept from right to left. In the example shown, there are two such points. From this graphical view, we can easily see that the only points that can be found by Lagrange optimization are those that lie on the convex hull of the set of all points. This is a fundamental limitation of the Lagrange method.

In light of the above limitation, Theorem 1.1 does not guarantee that a solution for the constrained Problem 1.1 can always be found by solving the unconstrained Problem 1.2. It only applies for cases where there is a value for $\lambda$ such that the number of bits used in a solution to Problem 1.2 is the same as the bit budget $B$ in Problem 1.1.

For a given bit budget $B$, in order to apply the Lagrange multiplier method, we need to know what value of $\lambda$ to use. In practice, an iterative search procedure can be used to determine the proper value. The search procedure takes advantage of a useful property of Lagrange optimization: The solution coding rate $R\big(\boldsymbol{Q}^*(\lambda)\big)$ is a nonincreasing function of $\lambda$. With appropriate initial upper and lower bounds for $\lambda$, a bisection search can be performed to find the proper value for $\lambda$. Details of the search procedure can be found in [72].

For an additive distortion measure, the distortion $D(\boldsymbol{Q})$ can be expressed as

$$D(\boldsymbol{Q}) = \sum_i D_i(Q_i),$$

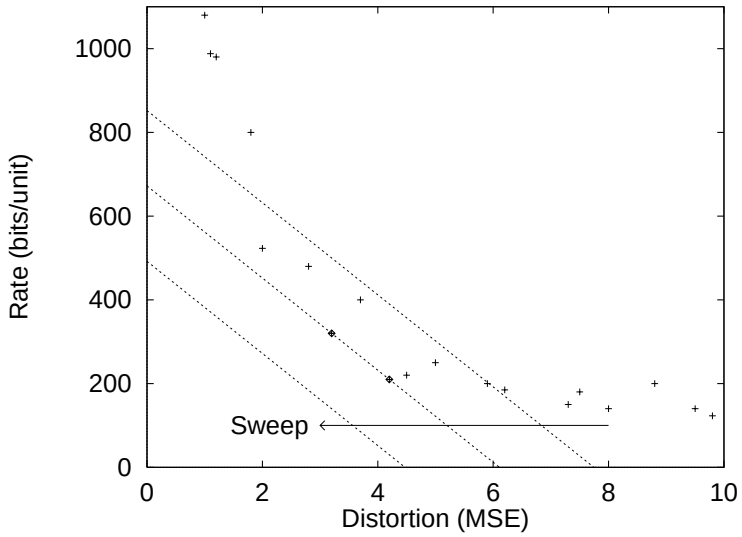



*Fig. 1.29* **Graphical Interpretation of Lagrange Optimization.** Lagrange optimization can be viewed as finding the last point(s) intersected by a right-to-left sweep of a line with slope $-\lambda$. In this example, the two data points circled are found in the minimization.

where $D_i(Q_i)$ is the distortion for block $i$ when using the quantizer specified by $Q_i$. If we assume that the coding of each block is independent of the quantization choices of other blocks, the coding rate $R(\boldsymbol{Q})$ can be expressed as

$$R(\boldsymbol{Q}) = \sum_i R_i(Q_i),$$

where $R_i(Q_i)$ is the coding rate for block $i$ when using the quantizer specified by $Q_i$. The minimization of $C_\lambda$ in Problem 1.2 can then be expressed as

$$\begin{aligned}
\min_{\boldsymbol{Q}} \{C_\lambda(\boldsymbol{Q})\} &= \min_{\boldsymbol{Q}} \{R(\boldsymbol{Q}) + \lambda D(\boldsymbol{Q})\} \\
&= \min_{\boldsymbol{Q}} \Big\{ \sum_{i=1}^{N} R_i(Q_i) + \lambda \sum_{i=1}^{N} D_i(Q_i) \Big\} \\
&= \sum_{i=1}^{N} \Big( \min_{Q_i} \{R_i(Q_i) + \lambda D_i(Q_i)\} \Big).
\end{aligned}$$

In short, we can minimize $C_\lambda(\boldsymbol{Q})$ by minimizing $R_i(Q_i) + \lambda D_i(Q_i)$, with respect to $Q_i$, separately for each block. This property makes the Lagrange method efficient for applications where blocks are coded independently.

# 2

# Lexicographic Bit Allocation Framework

In any lossy coding system, there is an inherent trade-off between the coding rate of the data and the distortion of the reconstructed signal. Often the transmission (storage) medium is bandwidth (capacity) limited. The purpose of rate control is to allocate bits to coding units and to regulate the coding rate to meet the bit rate constraints imposed by the transmission or storage medium while maintaining an acceptable level of distortion.

We consider rate control in the context of the MPEG-1 and MPEG-2 video coding standards. In addition to specifying a syntax for the encoded bitstream and a mechanism for decoding it, the MPEG standards define a hypothetical decoder called the *Video Buffering Verifier* (VBV), which places quantifiable limits on the variability in bit rate of encoded video. The VBV is diagrammed in Figure 1.21 and described in Section 1.6.

As outlined in Section 1.6.5, the overall rate control process can be broken down into three steps:

1. a high level bit allocation to coding units (video pictures),

2. a low level control of the quantization scale within a coding unit to achieve the bit allocation,

3. adjustment to the quantization scale to equalize perceptual quality.

In this chapter, we develop a framework that addresses the first step: bit allocation under VBV constraints and a total bit budget. This framework consists of three components: 1) a bit-production model, 2) a novel lexicographic optimality criterion, and 3) a set of buffer constraints for constant and

variable bit rate operation. We formalize bit allocation as a resource allocation problem with continuous variables and nonlinear constraints, to which we apply a global lexicographic optimality criterion.

The global nature of the lexicographic optimization requires the use of off-line techniques wherein the complexities of all the coded pictures, as specified with bit-production models, are known prior to computing a global bit allocation. One way to view this is as a serial computation with unlimited lookahead, wherein the inputs are the bit-production models for each picture. In practice, this would entail making multiple passes over the video sequence in order to construct the models, to compute an optimal allocation, and to compress the sequence using the computed allocation. In Chapter 5, we explore some techniques for reducing the computation by limiting the amount of lookahead used. Recognizing that no bit-production model is completely accurate, we develop efficient algorithms in Chapters 5 and 6 that can recover from model errors without having to recompute an optimal allocation from scratch. In Chapter 7, we develop an on-line VBR rate control algorithm based upon the optimal algorithm of Chapter 4 which, though suboptimal, compares favorably to the optimal algorithm.

In the next two chapters, we use the lexicographic framework to analyze bit allocation under constant bit rate and variable bit rate operation. The analyses yield necessary and sufficient conditions for optimality that lead to efficient bit allocation algorithms. In Chapter 5, we describe an implementation of these algorithms within a software MPEG-2 encoder and present simulation results.

## 2.1 PERCEPTUAL AND NOMINAL QUANTIZATION

As shown in Figure 1.19, the output bit rate of a typical video coder can be regulated by adjusting a quantization scale $Q_{\mathrm{s}}$. Increasing $Q_{\mathrm{s}}$ reduces the output bit rate, but it also decreases the visual quality of the compressed pictures. Similarly, decreasing $Q_{\mathrm{s}}$ both increases the output bit rate and improves the picture quality. Therefore by varying $Q_{\mathrm{s}}$, we can trace out a rate vs. distortion curve, such as that shown in Figure 1.25.

Although $Q_{\mathrm{s}}$ can be used to control rate and distortion, coding with a constant value of $Q_{\mathrm{s}}$ generally does not result in either constant bit rate or constant perceived quality. Both of these factors depend upon the scene content as well. Studies into human visual perception suggest that *perceptual distortion* is correlated to certain spatial and temporal properties of a video sequence [1, 59]. These studies lead to various techniques, called *perceptual quantization* or *adaptive quantization*, that take into account properties of the Human Visual System (HVS) in determining the quantization scale [11, 12, 39, 44, 65, 76, 79].

Based upon this body of work, we propose a separation of the actual quantization scale $Q_{\mathrm{s}}$ into a *nominal quantization scale* $Q$ and a *perceptual quan-*

*tization function* $P(I, Q)$ such that $Q_s = P(I, Q)$, where $I$ denotes the block being quantized. The function $P$ is chosen so that if the same nominal quantization scale $Q$ is used to code two blocks, then the blocks will have the same perceptual distortion. For example, if block $i$ is significantly more complex than block $j$ in a visual sense, then in order for the two blocks to have the same perceptual distortion, the same nominal quantization scale $Q$ must be used. However, the actual quantization scale $Q_s$ for block $i$ will be significantly smaller than the $Q_s$ of block $j$.

In this way, the nominal quantization scale $Q$ corresponds directly to the perceived distortion and can serve as the object for optimization. We favor a multiplicative model where $P(I, Q) = \alpha_I Q$.[1] Where quantization noise is less noticeable, such as in highly-textured regions, we can use a larger value for $\alpha_I$ compared with regions where quantization noise is more noticeable, such as in relatively uniform areas. In this regards, $\alpha_I$ can be viewed as a perceptual weighting factor.

The problem of determining $P(I, Q)$ has been studied elsewhere [10, 76] and is an active area of research. It is not considered further in this book. In this chapter, we address the assignment of nominal quantization $Q$ to each picture to give constant or near-constant quality among pictures while satisfying bit rate constraints imposed by the channel and decoder. We propose to compute $Q$ at the picture level; that is, we compute one $Q$ for each picture to be coded. Besides decreasing the computation over computing a different $Q$ for each macroblock, this method results in constant perceptual quality within each picture. The framework can certainly be generalized to other coding units, and in principle can be applied to code other types of data, such as images and speech.

## 2.2 CONSTANT QUALITY

The goal of optimal bit allocation traditionally has been to minimize an additive distortion measure, typically mean-squared error (MSE), averaged over coding blocks [60, 66]. Whereas this approach leverages the wealth of tools from optimization theory and operations research, it does not guarantee the constancy of quality that is generally desired from a video coding system. For example, a video sequence with a constant or near-constant level of distortion is more desirable than one with lower average distortion but higher variability, because human viewers tend to find frequent changes in quality more noticeable and annoying. A long video sequence typically contains segments which, even if encoded at a fairly low bit rate, will not contain any disturbing quan-

[1] The MPEG-2 Test Model 5 [34] also uses a multiplicative formulation, whereas an additive formulation is proposed in [63]. Our bit allocation framework works with any perceptual quantization function that is monotonically increasing.

tization artifacts. Improving the quality of pictures in those segments is less important than improving the quality of pictures in segments that are more difficult to encode, where artifacts may be readily visible.

To address these issues, we propose a *lexicographic optimality* criterion that better expresses the desired constancy of quality. The idea is to minimize the maximum distortion of a block and then minimize the second highest block distortion, and so on. The intuition is that doing so would equalize distortion by limiting peaks in distortion to their minimum. As we will show later, if a constant quality allocation is feasible, then it must necessarily be lexicographically optimal.

## 2.3 BIT-PRODUCTION MODELING AND QUANTIZATION SCALE

For simplicity, we assume that each picture has a bit-production model that relates the picture's nominal quantization scale $Q$ to the number of coded bits $B$. This assumes that the coding of one picture is independent of any other. This independence holds for an MPEG encoding that uses only intra-frame (I) pictures, but not for one that uses forward predictive (P) or bidirectionally predictive (B) pictures, for example. In practice, the extent of the dependency is limited to small groups of pictures. Nonetheless, we initially assume independence to ease analysis, and we defer treatment of dependencies until a later chapter, where we consider practical implementations.

We specify $Q$ and $B$ to be nonnegative real-valued variables. In practice, the actual quantization scale $Q_{\mathrm{s}}$ and the number of bits $B$ are positive integers, satisfying $Q_{\mathrm{s}} = \lfloor \alpha_{\mathrm{I}} \cdot Q \rfloor$. However, to facilitate analysis, we assume that there is a continuous function for each picture that maps the nominal quantization scale $Q$ to $B$.

For a sequence of $N$ pictures, we define $N$ corresponding bit-production models $\{f_1, f_2, \ldots, f_N\}$, where $f_k : [0, \infty] \to [l_k, u_k]$, with $0 \leq l_k < u_k$. (We number the $N$ pictures in encoding order and not temporal display order. See Figure 1.13.) For picture $k$, the bit production model $f_k$ maps a nominal quantization scale $Q$ to a corresponding number $f_k(Q) = B$ of bits. We require the models to have the following properties:

1. $f_k(0) = u_k$,
2. $f_k(\infty) = l_k$,
3. $f_k$ is continuous and monotonically decreasing.

From these conditions, it follows that $f_k$ is invertible with $Q = g_k\big(f_k(Q)\big)$, where $g_k = f_k^{-1}$ and $g_k : [l_k, u_k] \to [0, \infty]$. Like $f_k$, the inverse $g_k$ is also continuous and monotonically decreasing. Although monotonicity does not always hold in practice, it is a generally accepted assumption.

In video coding systems, the number of bits produced for a picture also depends upon myriad coding choices besides quantization scale, such as motion

compensation and the mode used for coding each block. We assume that these choices are made independent of quantization scale and prior to performing rate control.

## 2.4 BUFFER CONSTRAINTS

The MPEG standards specify that an encoder should produce a bitstream that can be decoded by a hypothetical decoder referred to as the Video Buffering Verifier (VBV), as shown in Figure 1.21 and described in Section 1.6. Data can be transferred to the VBV either at a constant or variable bit rate.[2] In either mode of operation, the number of bits produced by each picture must be controlled so as to satisfy constraints imposed by the operation of the decoder buffer, whose size $B_{\mathrm{vbv}}$ is specified in the bitstream by the encoder. The encoder also specifies the maximum transfer rate $R_{\max}$ into the VBV buffer and the amount of time the decoder should wait before removing and decoding the first picture. In this section, we consider constraints on the number of bits produced in each picture that follow from analysis of the VBV.

### 2.4.1 Constant Bit Rate

We first examine the mode of operation in which the compressed bitstream is to be delivered at a constant bit rate $R_{\max}$.

**Definition 2.1** An *allocation sequence* $\boldsymbol{s} = \langle s_1, s_2, \ldots, s_N \rangle$ is an $N$-tuple containing bit allocations for $N$ pictures, so that $s_n$ is the number of bits allocated to picture $n$, for $1 \leq n \leq N$.

**Definition 2.2** Let $B_{\mathrm{vbv}}$ be the size of the VBV decoding buffer. For $1 \leq n \leq N$, we use $B_{\mathrm{f}}(\boldsymbol{s}, n)$ to denote the *buffer fullness* (the number of bits in the VBV buffer) resulting from allocation sequence $\boldsymbol{s}$ just *before* the $n$th picture is removed from the buffer for decoding. We let $B^*_{\mathrm{f}}(\boldsymbol{s}, n)$ denote the buffer fullness just *after* the $n$th picture is removed from the buffer for decoding. We define $R_{\max}$ to be the rate at which bits enter the VBV buffer, and we let $T_n$ be the amount of time required to display picture $n$. We define $B_{\mathrm{inc}}(n) = R_{\max} T_n$ to be the maximum incoming number of bits that can enter the buffer in the time it takes to display picture $n$.

The buffer fullness levels before and after the $n$th picture satisfy the relation

$$B_{\mathrm{f}}(\boldsymbol{s}, n) = B^*_{\mathrm{f}}(\boldsymbol{s}, n) + s_n. \tag{2.1}$$

[2]The MPEG-1 standard only defines VBV operation with a constant bit rate, whereas the MPEG-2 standard also allows for variable bit rate.

For constant bit rate (CBR) operation, the state of the VBV buffer is described by the recurrence

$$\begin{aligned} B_{\mathrm{f}}(\boldsymbol{s},1) &= B_1, \\ B_{\mathrm{f}}(\boldsymbol{s},n+1) &= B_{\mathrm{f}}(\boldsymbol{s},n) + B_{\mathrm{inc}}(n) - s_n, \end{aligned} \tag{2.2}$$

where $B_1$ is the initial buffer fullness. Unwinding the recurrence, we can express (2.2) as

$$B_{\mathrm{f}}(\boldsymbol{s},n+1) = B_1 + \sum_{j=1}^{n} B_{\mathrm{inc}}(j) - \sum_{j=1}^{n} s_j. \tag{2.3}$$

To prevent the VBV decoder buffer from overflowing we must have

$$B_{\mathrm{f}}(\boldsymbol{s},n+1) \le B_{\mathrm{vbv}}. \tag{2.4}$$

The MPEG standards allow pictures to be skipped in certain applications. We assume that all pictures are coded, in which case all of picture $n$ must arrive at the decoder by the time it is to be decoded and displayed; that is, we have the following upper bound on the number $s_n$ of bits allocate to picture $n$:

$$B_{\mathrm{f}}(\boldsymbol{s},n) \ge s_n, \tag{2.5}$$

or equivalently,

$$B_{\mathrm{f}}^{*}(\boldsymbol{s},n) \ge 0. \tag{2.6}$$

A violation of this condition is called a buffer *underflow*.

To derive a lower bound on the number of bits to code picture $n$, we can use (2.2) and (2.4) to get

$$s_n \ge B_{\mathrm{f}}(\boldsymbol{s},n) + B_{\mathrm{inc}}(n) - B_{\mathrm{vbv}}.$$

Since we cannot produce a negative number of bits, the lower bound on $s_n$ is

$$s_n \ge \max\left\{B_{\mathrm{f}}(\boldsymbol{s},n) + B_{\mathrm{inc}}(n) - B_{\mathrm{vbv}},\, 0\right\}. \tag{2.7}$$

In summary, for constant bit rate operation, in order to pass video buffer verification, an allocation sequence $\boldsymbol{s}$ must satisfy the following for all $n$:

$$\max\left\{B_{\mathrm{f}}(\boldsymbol{s},n) + B_{\mathrm{inc}}(n) - B_{\mathrm{vbv}},\, 0\right\} \le s_n \le B_{\mathrm{f}}(\boldsymbol{s},n). \tag{2.8}$$

An exemplary plot of the evolution of the buffer fullness over time for CBR operation is shown in Figure 2.1. In this example, the decoder waits $T_0$ seconds before decoding and removing the first picture. The initial buffer fullness at time $T_0$ is $B_1$. The time to display each picture is assumed to be a constant $T$ seconds; that is, $B_{\mathrm{inc}}(n)$ is a constant for all $n$. In the plot, the upper and lower bounds for the number of bits to code picture 2 are shown as $U_2$ and $L_2$, respectively.

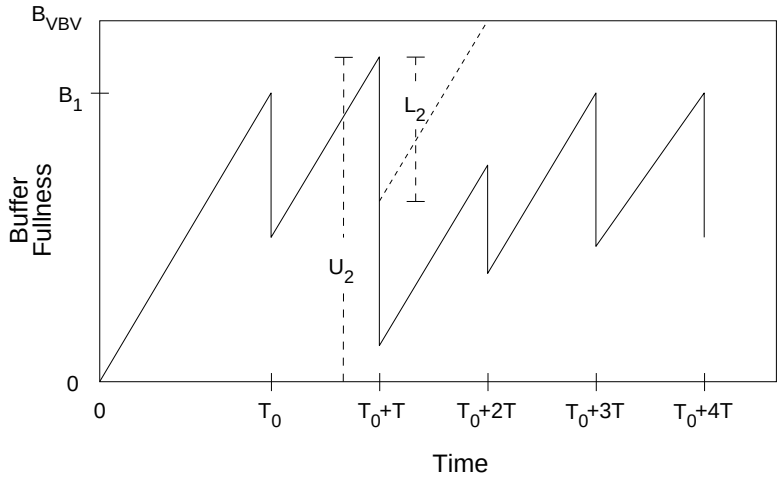



*Fig. 2.1* **Evolution of Buffer Fullness for CBR Operation.** This figure plots the evolution of the buffer fullness of the VBV for CBR operation. Bits enter the VBV decoder buffer at a constant rate until time $T_0$, when the first picture is decoded and removed. Successive pictures are removed after a time interval $T$. Upper and lower bounds on the number of bits that can be produced for the second picture are shown as $U_2$ and $L_2$, respectively. Note that $U_2 = B_{\rm f}(\boldsymbol{s}, 2)$.

### 2.4.2 Variable Bit Rate

We now examine the scenario where the compressed video bitstream is to be delivered at a variable bit rate (VBR). Specifically, we adopt the second VBR mode of the MPEG-2 VBV model (see Section 1.6.4), where bits always enter the VBV decoder buffer at the peak rate $R_{\max}$ until the buffer is full.[3] Depending upon the state of the buffer, bits enter during each display interval at a rate that is effectively variable up to the peak rate $R_{\max}$. As before, we denote the maximum number of bits entering the buffer in the time it takes to display picture $n$ as $B_{\rm inc}(n) = R_{\max} T_n$.

For VBR operation, the state of the VBV buffer is described by

$$\begin{aligned} B_{\rm f}(\boldsymbol{s}, 1) &= B_{\rm vbv}, \\ B_{\rm f}(\boldsymbol{s}, n+1) &= \min\left\{B_{\rm vbv},\, B_{\rm f}(\boldsymbol{s}, n) + B_{\rm inc}(n) - s_n\right\}. \end{aligned} \tag{2.9}$$

Unlike the CBR case, the VBV buffer is prevented from overflowing by the minimization in (2.9). When $B_{\rm f}(\boldsymbol{s}, n) + B_{\rm inc}(n) - s_n > B_{\rm vbv}$, we say

[3]We note that with the same bit allocation, the VBV buffer fullness for the second VBR mode is always equal to or higher than the fullness when operating in the first VBR mode. Intuitively, if the channel bit rate is not further constrained, a lexicographically optimal bit allocation sequence for the second VBR mode should not be worse (lexicographically) than an optimal bit allocation sequence for the first VBR mode, given the same total bit budget.

that picture $n$ results in a *virtual overflow*. When a virtual overflow occurs, the effective input rate to the VBV buffer during that display interval is less than the peak rate. Like the CBR case, underflow is possible, and to prevent it (2.5) must hold. The evolution of the buffer fullness is shown for VBR operation in Figure 2.2. The time to display each picture is assumed to be a constant $T$ seconds. As shown in the plot, the number of bits that enter the buffer during each display interval is variable, with virtual overflows occurring for pictures 2 and 4.

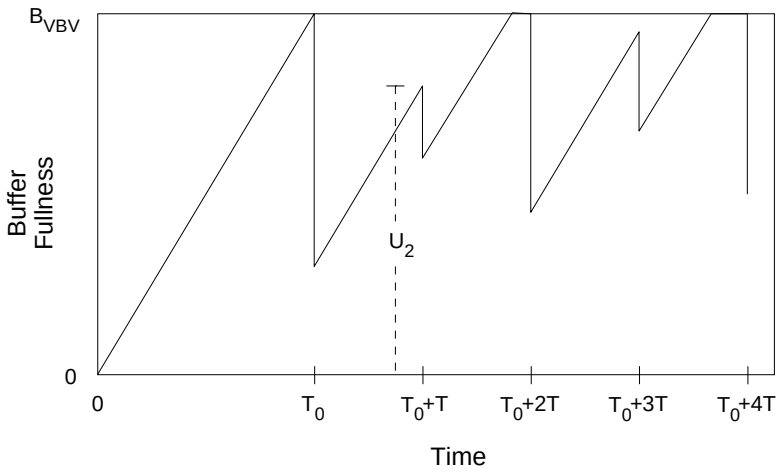



*Fig. 2.2* **Evolution of Buffer Fullness for VBR Operation.** This figure plots the evolution of the buffer fullness of the VBV for VBR operation. Bits enter the VBV buffer at a constant rate until time $T_0$, when the first picture is decoded and removed. Successive pictures are removed after a time interval $T$. By the VBR property, when the buffer becomes full, no more bits enter until the next picture is removed; hence there is no lower bound on the number of bits produced for a picture. The upper bound on the number of bits for the second picture is shown as $U_2$.

### 2.4.3 Encoder vs. Decoder Buffer

In the above discussion, we have focused solely on the decoder buffer, whereas Figure 1.19 shows the Rate Controller monitoring the fullness of the encoder buffer. By assuming a fixed channel delay the encoder buffer fullness can be shown to mirror the decoder buffer fullness, except for an initial startup period. That is, an empty decoder buffer would correspond to a full encoder buffer, and vice versa. The reader is referred to [68] for a more complete discussion of buffer constraints in video coder systems. We therefore refer exclusively to the decoder buffer in our discussions of the VBV.

## 2.5 BUFFER-CONSTRAINED BIT ALLOCATION PROBLEM

Using the bit-production model and VBV constraints defined above, we now formalize the buffer-constrained bit allocation problem.

**Definition 2.3** A *buffer-constrained bit allocation problem* $P$ is specified by a tuple

$$P = \langle N, F, B_{\text{tgt}}, B_{\text{vbv}}, B_1, B_{\text{inc}} \rangle,$$

where the terms are defined as follows:

- $N$ is the number of pictures;
- $F = \langle f_1, f_2, \ldots, f_N \rangle$ is a sequence of $N$ modeling functions (as specified in Section 2.3) that model the relationship between the nominal quantization scale and the number of coded bits for each picture;
- $B_{\text{tgt}}$ is the target number of bits to code all $N$ pictures;
- $B_{\text{vbv}}$ is the size of the VBV buffer in bits;
- $B_1$ is the number of bits initially in the VBV buffer; and
- $B_{\text{inc}}(n)$ is a function that gives the maximum number of bits that can enter the decoding buffer while the $n$th picture is being displayed (i.e., between the time when the $n$th picture is removed from the buffer and the time when the $(n+1)$st picture is removed from the buffer).

For convenience, in the sequel we shall use the shorter term "bit allocation problem" to refer to the buffer-constrained bit allocation problem.

**Definition 2.4** Given $P = \langle N, F, B_{\text{tgt}}, B_{\text{vbv}}, B_1, B_{\text{inc}} \rangle$, an allocation sequence $\boldsymbol{s}$ is a *legal allocation* if the following conditions hold for $1 \le n \le N$:

1. $\sum_{j=1}^{N} s_j = B_{\text{tgt}}$.
2. Equation (2.5) holds: $B_{\text{f}}(s, n) \ge s_n$.
3. For CBR only, (2.7) holds: $s_n \ge \max\{B_{\text{f}}(s, n) + B_{\text{inc}}(n) - B_{\text{vbv}},\, 0\}$.

In order for a CBR bit allocation problem to have a legal allocation sequence, we must have

$$B_{\text{vbv}} \ge \max_{1 \le j \le N-1} \{B_{\text{inc}}(j)\}. \tag{2.10}$$

Also, the buffer fullness at the end of the sequence must be within bounds. For an allocation sequence $\boldsymbol{s}$, by (2.1) and (2.3), the final buffer fullness is

$$B_{\text{f}}^{*}(\boldsymbol{s}, N) = B_1 + \sum_{j=1}^{N-1} B_{\text{inc}}(j) - B_{\text{tgt}}. \tag{2.11}$$

The bound on $B_{\mathrm{f}}^{*}(\boldsymbol{s}, N)$ is thus

$$0 \leq B_{\mathrm{f}}^{*}(\boldsymbol{s}, N) \leq B_{\mathrm{vbv}}. \tag{2.12}$$

From (2.11) and (2.12), we get the following CBR bounds on $B_{\mathrm{tgt}}$:

$$B_1 + \sum_{j=1}^{N-1} B_{\mathrm{inc}}(j) - B_{\mathrm{vbv}} \leq B_{\mathrm{tgt}} \leq B_1 + \sum_{j=1}^{N-1} B_{\mathrm{inc}}(j). \tag{2.13}$$

A VBR bit allocation problem does not have a lower bound for the target bit rate $B_{\mathrm{tgt}}$ since the VBV does not impose a lower bound on the number of bits produced by each picture. The upper bound for $B_{\mathrm{tgt}}$ depends upon whether $\max_{1 \leq j \leq N} \left\{ B_{\mathrm{inc}}(j) \right\} > B_{\mathrm{vbv}}$. In general, the VBR upper bound on $B_{\mathrm{tgt}}$ is

$$B_{\mathrm{tgt}} \leq B_1 + \sum_{j=1}^{N-1} \min \left\{ B_{\mathrm{inc}}(j),\, B_{\mathrm{vbv}} \right\}. \tag{2.14}$$

However, in the sequel, we assume that $\max_{1 \leq j \leq N-1} \left\{ B_{\mathrm{inc}}(j) \right\} \leq B_{\mathrm{vbv}}$. We also assume that bit allocation problems are given so that a legal allocation sequence exists.

## 2.6 LEXICOGRAPHIC OPTIMALITY

We now formally define the lexicographic optimality criterion. As mentioned in Section 2.1, we equate nominal quantization scale with perceptual distortion and define the optimality criterion based upon the nominal quantization scale $Q$ assigned to each picture.

Let $S$ be the set of all legal allocation sequences for a bit allocation problem $P$. For an allocation sequence $\boldsymbol{s} \in S$, let $\boldsymbol{Q^s} = \langle Q_1^{\boldsymbol{s}}, Q_2^{\boldsymbol{s}}, \ldots, Q_N^{\boldsymbol{s}} \rangle$ be the quantizer values to achieve the bit allocations specified by $\boldsymbol{s}$. Thus $Q_j^{\boldsymbol{s}} = g_j(s_j)$, where $g_j$ is as defined in Section 2.3. Ideally, we would like an optimal allocation sequence to use a constant nominal quantization scale. However, this may not be feasible because of buffer constraints. We could consider minimizing an $l_k$ norm of $\boldsymbol{Q^s}$. However, as discussed earlier, such an approach does not guarantee constant quality where possible, and it may result in some pictures having extreme values of $Q_j$.

Instead, our goal is to minimize the maximum $Q_j$. Additionally, given that the maximum $Q_j$ is minimized, we want the second largest $Q_j$ to be as small as possible, and so on. This condition is referred to as *lexicographic optimality* in the literature (e.g., [33]).

We define a sorted permutation DEC on $\boldsymbol{Q^s}$ such that for $\mathrm{DEC}(\boldsymbol{Q^s}) = \langle Q_{j_1}^{\boldsymbol{s}}, Q_{j_2}^{\boldsymbol{s}}, \ldots, Q_{j_N}^{\boldsymbol{s}} \rangle$ we have $Q_{j_1}^{\boldsymbol{s}} \geq Q_{j_2}^{\boldsymbol{s}} \geq \cdots \geq Q_{j_N}^{\boldsymbol{s}}$. Let $\mathrm{rank}(\boldsymbol{s}, k)$ be the $k$th largest element of $\mathrm{DEC}(\boldsymbol{Q^s})$; that is, $\mathrm{rank}(\boldsymbol{s}, k) = Q_{j_k}^{\boldsymbol{s}}$. We define a binary relation $\succ$ on allocation sequences as follows: We say that $\boldsymbol{s} \succ \boldsymbol{s'}$ if and only

if for some $1 \leq k \leq N$ we have $\text{rank}(\boldsymbol{s}, j) = \text{rank}(\boldsymbol{s}', j)$ for $1 \leq j < k$ and $\text{rank}(\boldsymbol{s}, k) > \text{rank}(\boldsymbol{s}', k)$. We define $\boldsymbol{s} \prec \boldsymbol{s}'$ if and only if $\boldsymbol{s}' \succ \boldsymbol{s}$; $\boldsymbol{s} \asymp \boldsymbol{s}'$ if and only if $\text{rank}(\boldsymbol{s}, j) = \text{rank}(\boldsymbol{s}', j)$ for all $j$. We define $\boldsymbol{s} \succeq \boldsymbol{s}'$ if and only if $\boldsymbol{s} \succ \boldsymbol{s}'$ or $\boldsymbol{s} \asymp \boldsymbol{s}'$, and similarly we define $\boldsymbol{s} \preceq \boldsymbol{s}'$ if and only if $\boldsymbol{s} \prec \boldsymbol{s}'$ or $\boldsymbol{s} \asymp \boldsymbol{s}'$.

**Definition 2.5** A legal allocation sequence $\boldsymbol{s}^*$ is *lexicographically optimal* if $\boldsymbol{s}^* \preceq \boldsymbol{s}$ for all other legal allocation sequences $\boldsymbol{s}$.

**Lemma 2.1 (Constant-$Q$)** *Given a bit allocation problem $P = \langle N, F, B_{\text{tgt}}, B_{\text{vbv}}, B_1, B_{\text{inc}} \rangle$, if there exists a legal allocation sequence $\boldsymbol{s}$ and a nominal quantization scale $Q$ such that $g_n(s_n)$ is the constant value $Q$ for all $n$, where $g_n$ is defined as in Section 2.3, then $\boldsymbol{s}$ is the only lexicographically optimal allocation sequence for $P$.*

*Proof*: First we prove that $\boldsymbol{s}$ is optimal. Since $\boldsymbol{s}$ is a legal allocation sequence, we have

$$\sum_{j=1}^{N} s_j = \sum_{j=1}^{N} f_j(Q) = B_{\text{tgt}}.$$

Suppose that $\boldsymbol{s}$ is not optimal. Let $\boldsymbol{s}'$ be an optimal allocation sequence. Then $\text{rank}(\boldsymbol{s}', k) < \text{rank}(\boldsymbol{s}, k) = Q$ for some $k$, and $\text{rank}(\boldsymbol{s}', j) \leq \text{rank}(s, j)$ for all $j$. Therefore $s'_m > f_m(Q)$ for some $m$ and $s'_j \geq f_j(Q)$ for all $j$ since $f_j$ is a decreasing function. Thus

$$\sum_{j=1}^{N} s'_j > \sum_{j=1}^{N} f_j(Q) = B_{\text{tgt}}.$$

So $\boldsymbol{s}'$ is not a legal allocation sequence, a contradiction. Therefore $\boldsymbol{s}$ is optimal.

Now we show that $\boldsymbol{s}$ is the only optimal allocation sequence. Let $\boldsymbol{s}'$ be an optimal allocation sequence. Since $\boldsymbol{s}$ and $\boldsymbol{s}'$ are both optimal, we have $\boldsymbol{s} \preceq \boldsymbol{s}'$ and $\boldsymbol{s} \succeq \boldsymbol{s}'$, thus implying $\boldsymbol{s} \asymp \boldsymbol{s}'$. Therefore $\text{rank}(\boldsymbol{s}, j) = \text{rank}(\boldsymbol{s}', j) = Q$ for all $j$, or simply $\boldsymbol{s}' = \boldsymbol{s}$. ∎

Lemma 2.1 establishes a desirable property of the lexicographic optimality criterion: If a constant-$Q$ allocation sequence is legal, it is the only lexicographically optimal allocation sequence. This meets our objective of obtaining a constant-quality allocation (via perceptual quantization) when feasible.

## 2.7 RELATED WORK

Shoham and Gersho [72] have examined the budget-constrained bit allocation problem (see Section 1.8.3) in the context of a discrete set of independent quantizers. A bit allocation algorithm based upon Lagrangian minimization is presented as a more efficient alternative to the well-known dynamic programming solution based upon the Viterbi algorithm [18, 77]. Although it

only solves the simple budget-constrained allocation problem, this work lays the foundation for much of the ensuing work on optimal bit allocation.

Uz, Shapiro, and Czigler [75] examine optimal budget-constrained bit allocation in a dependent-coding setting. They propose a parametric rate-distortion model for intraframe coding and forward predictive coding. The model has an exponential form and is motivated by theoretical rate-distortion results for stationary Gaussian sources. Lagrangian minimization is chosen as the optimization technique and a closed-form solution is obtained in terms of known statistics and the Lagrange multiplier. A search over the Lagrange multiplier then yields a solution to the budget-constrained problem. The authors acknowledge that minimizing sum-distortion does not lead to uniform distortion. They propose an alternate minimax formulation that minimizes the maximum picture distortion by equating the distortion among pictures.[4]

Budget-constrained minimax bit allocation for dependent coding is also considered by Schuster and Katsaggelos [70]. They provide a minimax solution by first showing how to find a minimum-rate solution given a maximum distortion and then using an bisection search to find the maximum distortion corresponding to the desired rate. However, the bisection search is not guaranteed to converge in a finite number of iterations.

Reibman and Haskell [68] derive bit rate constraints for buffered video coders for a general variable bit rate channel, such as that provided by an ATM network. The constraints take into account both the encoder and decoder buffers. The bit rate constraints are used in an ad hoc algorithm that jointly selects the channel and encoder rates.

Ortega, Ramchandran, and Vetterli [60] study the problem of optimal bit allocation in a buffered video coder. The authors consider video coding with CBR buffer constraints and formulate bit allocation as an integer-programming problem. They assume a finite set of quantization scales, an integral number of coded bits, and independent coding. The problem is optimally solved using a dynamic programming algorithm based upon the Viterbi algorithm (as described in Section 1.8.4). Heuristic methods based upon Lagrangian minimization and other ad hoc techniques are proposed to provide more efficient, but suboptimal, solutions.

Ramchandran, Ortega, and Vetterli [66] extend the discrete optimization framework of [60] to handle dependent coding. Except for a simple illustrative case, computing an optimal bit allocation under the dependent framework requires time and space exponential in the number of coding units. A heuristic pruning technique is proposed to reduce the number of states considered. However, the effectiveness of the heuristic depends upon the rate-distortion characteristics of the source.

[4]Our lexicographic framework produces a minimax solution since lexicographic optimality is a refinement of minimax.

The work in [60] is further extended by Hsu, Ortega, and Reibman [32] to include transmission over a VBR channel with delay constraints. Besides buffer and delay constraints, the authors also consider constraints imposed by several policing mechanisms proposed for ATM networks. Assuming a discrete set of quantizers and a discrete set of transmission rates, the quantization scale and transmission rate can be jointly optimized using the Viterbi algorithm to produce a minimum sum-distortion encoding. In the construction of the trellis used by the Viterbi algorithm, states that violate the various constraints are discarded. Unlike our framework, there is no explicit constraint on the total number of bits used.

Ding [15] also considers joint control of encoder and channel rate. Instead of considering global optimality, the author focuses on real-time control algorithms. An algorithm is proposed that separates rate control into a "short-term" process and a "long-term" process. The long term rate control sets a base quantization scale $Q_{\text{seq}}$ called the *sequence quantization parameter*.[5] In normal operation, $Q_{\text{seq}}$ is used to code each picture. Long-term rate control monitors the average fullness of a virtual encoder buffer and adjusts $Q_{\text{seq}}$ to maintain the buffer fullness between two thresholds. Short-term rate control is applied when the upper bound on encoder rate needs to be enforced. Several methods are proposed for performing short-term rate control.

Chen and Hang [5] derive a model for block transform video coders that relates bits, distortion, and quantization scale. Assuming a stationary Gaussian process, the authors derive a bit-production model containing transcendental functions. The model is applied to control the frame rate of motion-JPEG and H.261 video coders.

In the operations research literature, lexicographic optimality has been applied to such problems as resource location and allocation (e.g., [22, 40, 50, 52, 58, 64]) and is sometimes referred to as *lexicographic minimax*, since it can be viewed as a refinement of minimax theory.

## 2.8 DISCUSSION

As described above, much of the previous work on optimal rate control for video coding use the conventional rate-distortion approach of minimizing an additive distortion measure with budget and buffer constraints. However, the conventional approach does not directly achieve the constancy in quality that is generally desired. In contrast, our proposed framework, based upon a novel lexicographic optimality criterion, is designed to achieve constant quality when possible and a well-defined notion of near-constant quality otherwise.

[5] The sequence quantization parameter is similar to the notion of nominal quantization scale defined in Section 2.1.

We incorporate into the framework a set of buffer constraints based upon the Video Buffering Verifier of the popular MPEG video coding standards.

In the following chapters, we analyze rate control under both CBR and VBR constraints, and we derive a set of necessary and sufficient conditions for optimality. These conditions, intuitive and elegant, lead to efficient algorithms for computing optimal allocation sequences in polynomial time and linear space. Based upon the analytical results and the optimal algorithms, we develop a real-time VBR rate control algorithm in Chapter 7. This algorithm operates with a single encoding pass and has many practical applications.

In Chapter 8, we describe some extensions to the lexicographic framework. We show how to apply the framework to allocate bits to multiple VBR streams for transport over a CBR channel. We also show how to adapt the operational rate-distortion framework of [60] to perform lexicographic optimization with a discrete set of quantizers.

# 3

# *Optimal Bit Allocation under CBR Constraints*

In this chapter, we analyze the buffer-constrained bit allocation problem under constant bit rate VBV constraints, as described in Section 2.4.1. The analysis leads to an efficient dynamic programming algorithm for computing a lexicographically optimal solution. An even faster and more robust approach, based upon the framework we present in this chapter, will be explored in Chapter 6.

Before proceeding with a formal theoretical treatment, we first present some intuition for the results that follow. If we consider a video sequence as being composed of segments of differing coding difficulty, a segment of "easy" pictures can be coded at a higher quality (lower distortion) than an immediately following segment of "hard" pictures if we code each segment at the same average bit rate. Since we have a decoder buffer, we can vary the bit rate to some degree, depending upon the size of the buffer. If we could somehow "move" bits from the easy segment to the hard segment, we would be able to code the easy segment at a lower quality than before and the hard segment at a higher quality, thereby reducing the difference in perceptual quality between the two segments. In terms of the decoder buffer, this corresponds to filling up the buffer during the coding of the easy pictures, which are coded with less than the average bit rate. By use of the accumulated bits in the buffer, the hard pictures can be coded with effectively more than the average bit rate.

Similarly, suppose we have a hard segment followed by an easy segment. We would like to empty the buffer during the coding of the hard pictures in order to use as many bits as the buffer allows to code the hard pictures at above the average bit rate. This simultaneously leaves room in the buffer to accumulate excess bits resulting from coding the subsequent easy pictures at below the average bit rate.

This behavior of emptying and filling the buffer is intuitively desirable because it means that we are taking advantage of the full capacity of the buffer. In the following analysis, we will show that such a behavior is indeed exhibited by a lexicographically optimal bit allocation sequence.

## 3.1 ANALYSIS

First, we seek to prove necessary conditions for lexicographic optimality. To do so, we use the following lemma.

**Lemma 3.1** *Consider two allocation sequences $\boldsymbol{s}$ and $\boldsymbol{s'}$ that are identical except for the bit allocations to pictures $u$ and $v$. If* $\max\{g_u(s'_u), g_v(s'_v)\} < \max\{g_u(s_u), g_v(s_v)\}$*, where $g$ is the quantization function defined in Section 2.3, then we have $\boldsymbol{s'} \prec \boldsymbol{s}$.*

*Proof*: Suppose $\max\{g_u(s'_u), g_v(s'_v)\} < \max\{g_u(s_u), g_v(s_v)\}$. Let $j$ be the greatest index such that $\text{rank}(\boldsymbol{s}, j) = \max\{g_u(s_u), g_v(s_v)\}$. Then $\text{rank}(\boldsymbol{s}, j) > \text{rank}(\boldsymbol{s}, j+1)$. Consider $\text{rank}(\boldsymbol{s'}, j)$. Either $\text{rank}(\boldsymbol{s'}, j) = \text{rank}(\boldsymbol{s}, j+1)$ or $\text{rank}(\boldsymbol{s'}, j) = \max\{g_u(s'_u), g_v(s'_v)\}$. In either case, $\text{rank}(\boldsymbol{s}, j) > \text{rank}(\boldsymbol{s'}, j)$. Therefore, $\boldsymbol{s'} \prec \boldsymbol{s}$. ∎

The following important lemma establishes a set of necessary *switching conditions* in order for an allocation sequence to be optimal. It states that an optimal allocation sequence consists of segments of constant nominal quantization scale $Q$, with changes in $Q$ occurring only at the buffer boundaries (when the buffer is empty or full). In particular, $Q$ can decrease only when the buffer becomes empty, and $Q$ can increase only when the buffer gets full.

**Lemma 3.2** *Consider bit allocation problem $P = \langle N, F, B_{\text{tgt}}, B_{\text{vbv}}, B_1, B_{\text{inc}} \rangle$ with CBR constraints. If $\boldsymbol{s}^*$ is an optimal allocation sequence, then the following conditions hold for all $1 \le j < N$:*

1. *If $g_j(s_j^*) > g_{j+1}(s_{j+1}^*)$ (i.e., the nominal quantization decreases), then we have $B_{\text{f}}(\boldsymbol{s}^*, j) = s_j^*$ (or equivalently, $B_{\text{f}}^*(\boldsymbol{s}^*, j) = 0$) .*
2. *If $g_j(s_j^*) < g_{j+1}(s_{j+1}^*)$ (i.e., the nominal quantization increases), then we have $B_{\text{f}}(\boldsymbol{s}^*, j+1) = B_{\text{vbv}}$ (or equivalently, $B_{\text{f}}^*(\boldsymbol{s}^*, j) = B_{\text{vbv}} - B_{\text{inc}}(j)$) .*

In an optimal allocation sequence, the VBV decoder buffer is full before decoding starts on a relatively difficult scene, which is marked by an increase in $Q$ (case 2). This policy makes the entire capacity of the decoder buffer available to code the more difficult pictures. The nominal quantization scale is increased because otherwise the scene may be too costly to encode in terms of number of bits, even making use of the full capacity of the buffer. On the other hand, before decoding a relatively easy scene, which is marked by

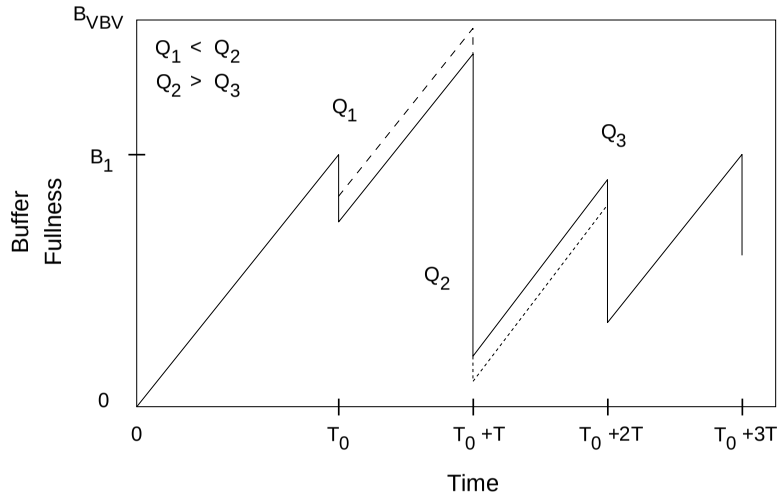



*Fig. 3.1* **Sketch of Proof of Lemma 3.2.** This figure shows the state of the VBV buffer for a hypothetical situation where $Q_1 < Q_2$ (case 2), $Q_2 > Q_3$ (case 1), and the switching conditions of Lemma 3.2 are *not* met. The dashed lines show the effects of increasing $Q_1$ and decreasing $Q_2$ without changing the total number of bits allocated. The dotted lines show the effects of decreasing $Q_2$ and increasing $Q_3$ while maintaining the total bit allocation.

a decrease in $Q$ (case 1), the buffer is emptied in order to provide the most space to accumulate bits when the easy scene uses less than the average bit rate. These observations agree with the intuitions provided earlier.

A sketch of the proof of Lemma 3.2 is shown in Figure 3.1. The proof is by contradiction. In the figure, the VBV buffer is shown for a hypothetical situation in which $Q_1 < Q_2$ and $Q_2 > Q_3$ and the switching conditions are not met. By necessity, there is a slight asymmetry in what we mean by the buffer being full and the buffer being empty: We talk about the buffer being full *before the next picture is removed from the buffer.* The buffer includes the incoming bits that arrive while the current picture is being displayed. On the other hand, we talk about the buffer being empty *after the current picture is removed from the buffer.* In this case, the buffer does not include the incoming bits that arrive during the display of the current picture.

Case 2 corresponds to the situation $Q_1 < Q_2$ (i.e., $j = 1$) in Figure 3.1. Since the buffer is not full before the next picture (picture 2) is removed, an alternate allocation can be constructed that is the same as the allocation shown, except that the VBV plot follows the dashed line for the segment between pictures 1 and 2. The dashed line corresponds to increasing $Q_1$ for the first picture and decreasing $Q_2$ for the second picture, while still maintaining the constraint $Q_1 < Q_2$ and not causing the buffer to overflow. This new allocation sequence is lexicographically better than before. Intuitively, the

new allocation shifts bits left-to-right from a relatively easy picture (with lower $Q$) to a relatively hard picture (with higher $Q$). This shifting of bits can take place until the buffer becomes full.

Case 1 corresponds to the situation $Q_2 > Q_3$ (i.e., $j = 2$) in Figure 3.1. Since the buffer is not empty after picture 2 is removed, an alternate allocation can be constructed that is the same as the allocation shown, except that the VBV plot follows the dotted line for the segment between pictures 2 and 3. The dotted line results from decreasing $Q_2$ and increasing $Q_3$ while still maintaining $Q_2 > Q_3$ and not causing the buffer to underflow. This new allocation sequence is lexicographically better than before. Intuitively, the new allocation shifts bits right-to-left from a relatively easy picture (with lower $Q$) to a relatively hard picture (with higher $Q$). This shifting of bits can take place until the buffer becomes empty.

We note that Lemma 2.1 follows directly from Lemma 3.2.

*Proof of Lemma 3.2*:
*Case 1.* We prove Case 1 by contradiction. Suppose that $g_j(s_j^*) > g_{j+1}(s_{j+1}^*)$ for some $1 \le j < N$, but that $B_{\rm f}(\boldsymbol{s}^*, j) \ne s_j^*$. Let $\Delta = B_{\rm f}(\boldsymbol{s}^*, j) - s_j^*$. Then by (2.5), $\Delta > 0$. Consider an allocation sequence $\boldsymbol{s'}$ that differs from $\boldsymbol{s}^*$ only for pictures $j$ and $j+1$; that is,

$$s'_k = s_k^*, \qquad \text{for } k \in \{1, \ldots, N\} \setminus \{j, j+1\}; \tag{3.1}$$

$$s'_k \ne s_k^*, \qquad \text{for } k \in \{j, j+1\}. \tag{3.2}$$

In order to show a contradiction, we want to find an assignment to $s'_j$ and $s'_{j+1}$ that makes $\boldsymbol{s'}$ a legal allocation sequence and "better" than $\boldsymbol{s}^*$. By "better" we mean lexicographically smaller:

$$g_j(s'_j),\ g_{j+1}(s'_{j+1}) < g_j(s_j^*), \tag{3.3}$$

or equivalently

$$s'_j > s_j^* \tag{3.4}$$

and

$$s'_{j+1} > f_{j+1}\big(g_j(s_j^*)\big). \tag{3.5}$$

To meet the target bit rate, we must have

$$s'_j + s'_{j+1} = s_j^* + s_{j+1}^*. \tag{3.6}$$

Let $\delta = s'_j - s_j^*$. Then $s_{j+1}^* - s'_{j+1} = \delta$. By (3.4), we want $\delta > 0$. We want to show that $\boldsymbol{s'}$ is a legal allocation sequence for some value of $\delta > 0$. To avoid VBV violations, (2.8) must hold for all pictures under the allocation sequence $\boldsymbol{s'}$. From (3.2) and (3.6), we have

$$B_{\rm f}(\boldsymbol{s'}, k) = B_{\rm f}(\boldsymbol{s}^*, k), \qquad \text{for } k \ne j+1. \tag{3.7}$$

Since $\boldsymbol{s}^*$ is a legal allocation sequence, there are no VBV violations for pictures 1, 2, ..., $j-1$ under $\boldsymbol{s'}$. Furthermore, if our choice for $s'_j$ does not cause a

VBV violation for picture $j$, then we are assured that there would be no VBV violations in pictures $j+1$, $j+2$, ..., $N$. So we must choose $s'_j$ subject to (2.8) and (3.4). Therefore

$$s^*_j < s'_j \le s^*_j + \Delta. \tag{3.8}$$

If $0 < \delta \le \Delta$, then $\boldsymbol{s'}$ is a legal allocation sequence. We also want (3.5) to hold. For this we need

$$\delta < s^*_{j+1} - f_{j+1}\big(g_j(s^*_j)\big). \tag{3.9}$$

Since $g_j(s^*_j) > g_{j+1}(s^*_{j+1})$, we have $f_{j+1}(g_j(s^*_j)) < s^*_{j+1}$. Therefore, $s^*_{j+1} - f_{j+1}\big(g_j(s^*_j)\big) > 0$. So for

$$0 < \delta \le \min\big\{\Delta,\, s^*_{j+1} - f_{j+1}\big(g_j(s^*_j)\big)\big\}, \tag{3.10}$$

$\boldsymbol{s'}$ is a legal allocation sequence that meets condition (3.3). By Lemma 3.1, we have $\boldsymbol{s'} \prec \boldsymbol{s^*}$, and thus $\boldsymbol{s^*}$ is not an optimal allocation sequence, a contradiction.

*Case 2.* We prove Case 2 by contradiction. Suppose that $g_j(s^*_j) < g_{j+1}(s^*_{j+1})$ for some $1 \le j < N$, but that $B_{\rm f}(\boldsymbol{s^*}, j+1) \ne B_{\rm vbv}$. Let $\Delta = B_{\rm vbv} - B_{\rm f}(\boldsymbol{s^*}, j+1)$; by (2.4), $\Delta > 0$. Consider an allocation sequence $\boldsymbol{s'}$ that differs from $\boldsymbol{s^*}$ only for pictures $j$ and $j+1$; that is,

$$s'_k = s^*_k, \qquad \text{for } k \in \{1, \ldots, N\} \setminus \{j, j+1\}. \tag{3.11}$$

We want to find an assignment to $s'_j$ and $s'_{j+1}$ that makes $\boldsymbol{s'}$ a legal allocation sequence and "better" than $\boldsymbol{s^*}$, in order to show a contradiction. By "better" we mean lexicographically smaller:

$$g_j(s'_j),\ g_{j+1}(s'_{j+1}) < g_{j+1}(s^*_{j+1}), \tag{3.12}$$

or equivalently

$$s'_{j+1} > s^*_{j+1} \tag{3.13}$$

and

$$s'_j > f_j\big(g_{j+1}(s^*_{j+1})\big). \tag{3.14}$$

To meet the target bit rate, we must have

$$s'_j + s'_{j+1} = s^*_j + s^*_{j+1}. \tag{3.15}$$

Let $\delta = s'_{j+1} - s^*_{j+1}$. Then $s^*_j - s'_j = \delta$. By (3.13), we want $\delta > 0$. We want to show that $\boldsymbol{s'}$ is a legal allocation sequence for some value of $\delta > 0$. To avoid VBV violations, (2.8) must hold for all pictures under the allocation sequence $\boldsymbol{s'}$. From (3.11) and (3.15), we have

$$B_{\rm f}(\boldsymbol{s'}, k) = B_{\rm f}(\boldsymbol{s^*}, k) \quad \text{for } k \ne j+1. \tag{3.16}$$

Since $\boldsymbol{s}^*$ is a legal allocation sequence, there are no VBV violations for pictures 1 to $j-1$ under $\boldsymbol{s'}$. Furthermore, if our choice for $s'_j$ does not cause a VBV violation, then we are assured that there would be no VBV violations in pictures $j+1$ to $N$. So we must choose $s'_j$ subject to (2.8) and (3.13):

$$\begin{aligned} B_{\rm f}(\boldsymbol{s'}, j+1) &= B_{\rm f}(\boldsymbol{s'}, j) + B_{\rm inc}(j) - s'_j \\ &= B_{\rm f}(\boldsymbol{s}^*, j) + B_{\rm inc}(j) - s'_j; \\ B_{\rm f}(\boldsymbol{s}^*, j+1) &= B_{\rm f}(\boldsymbol{s}^*, j) + B_{\rm inc}(j) - s^*_j. \end{aligned}$$

Combining the last two lines, we get

$$\begin{aligned} B_{\rm f}(\boldsymbol{s'}, j+1) &= B_{\rm f}(\boldsymbol{s}^*, j+1) + s^*_j - s'_j \\ &= B_{\rm vbv} - \Delta + \delta \\ &\le B_{\rm vbv}. \end{aligned}$$

In order for the last inequality to hold, we need $0 < \delta \le \Delta$, in which case $\boldsymbol{s'}$ is a legal allocation sequence. We also want (3.14) to hold. For this we need

$$\delta < s^*_j - f_j\big(g_{j+1}(s^*_{j+1})\big). \tag{3.17}$$

Since $g_{j+1}(s^*_{j+1}) > g_j(s^*_j)$, we have $f_j\big(g_{j+1}(s^*_{j+1})\big) < s^*_j$. Therefore, $s^*_j - f_j\big(g_{j+1}(s^*_{j+1})\big) > 0$. So for

$$0 < \delta \le \min\big\{\Delta,\, s^*_j - f_j\big(g_{j+1}(s^*_{j+1})\big)\big\}, \tag{3.18}$$

$\boldsymbol{s'}$ is a legal allocation sequence that meets condition (3.14). By Lemma 3.1, we have $\boldsymbol{s'} \prec \boldsymbol{s}^*$, and thus $\boldsymbol{s}^*$ is not an optimal allocation sequence, a contradiction. ∎

The theorem that follows is the main result of this section and shows that the switching conditions at the buffer boundaries are not only necessary but also sufficient for optimality. But first we prove a useful lemma that will be helpful in the proof of the theorem.

**Lemma 3.3** *Given two integers $u < v$ in the range $[1, N]$, let $\boldsymbol{s}$ and $\boldsymbol{s'}$ be legal bit allocation sequences such that $B_{\rm f}(\boldsymbol{s}, u) \ge B_{\rm f}(\boldsymbol{s'}, u)$ and $s_j \le s'_j$ for all $u \le j \le v$. Then $B_{\rm f}(\boldsymbol{s}, v+1) = B_{\rm f}(\boldsymbol{s'}, v+1)$ if and only if $B_{\rm f}(\boldsymbol{s}, u) = B_{\rm f}(\boldsymbol{s'}, u)$ and $s_j = s'_j$ for all $u \le j \le v$.*

*Proof*: We use (2.3) to express $B_{\rm f}(\boldsymbol{s}, v+1)$ in terms of $B_{\rm f}(\boldsymbol{s}, u)$:

$$\begin{aligned} B_{\rm f}(\boldsymbol{s}, u) &= B_1 + \sum_{j=1}^{u-1} \big(B_{\rm inc}(j) - s_j\big) \\ B_{\rm f}(\boldsymbol{s}, v+1) &= B_1 + \sum_{j=1}^{v} \big(B_{\rm inc}(j) - s_j\big) \\ &= B_{\rm f}(\boldsymbol{s}, u) + \sum_{j=u}^{v} \big(B_{\rm inc}(j) - s_j\big). \end{aligned}$$

Similarly,

$$B_{\rm f}(\boldsymbol{s'}, v+1) = B_{\rm f}(\boldsymbol{s'}, u) + \sum_{j=u}^{v} \big(B_{\rm inc}(j) - s'_j\big).$$

First we prove the "if" part. Suppose that $B_{\rm f}(\boldsymbol{s}, u) = B_{\rm f}(\boldsymbol{s'}, u)$ and $s_j = s'_j$ for all $u \le j \le v$. Then

$$\begin{aligned} B_{\rm f}(\boldsymbol{s}, v+1) &= B_{\rm f}(\boldsymbol{s}, u) + \sum_{j=u}^{v} \big(B_{\rm inc}(j) - s_j\big) \\ &= B_{\rm f}(\boldsymbol{s'}, u) + \sum_{j=u}^{v} \big(B_{\rm inc}(j) - s'_j\big) \\ &= B_{\rm f}(\boldsymbol{s'}, v+1). \end{aligned}$$

Now we prove the "only if" part. Suppose $B_{\rm f}(\boldsymbol{s}, v+1) = B_{\rm f}(\boldsymbol{s'}, v+1)$. Then

$$\begin{aligned} B_{\rm f}(\boldsymbol{s}, v+1) &= B_{\rm f}(\boldsymbol{s'}, v+1); \\ B_{\rm f}(\boldsymbol{s}, u) + \sum_{j=u}^{v} \big(B_{\rm inc}(j) - s_j\big) &= B_{\rm f}(\boldsymbol{s'}, u) + \sum_{j=u}^{v} \big(B_{\rm inc}(j) - s'_j\big); \\ B_{\rm f}(\boldsymbol{s}, u) - B_{\rm f}(\boldsymbol{s'}, u) &= \sum_{j=u}^{v} (s_j - s'_j). \end{aligned} \tag{3.19}$$

But $B_{\rm f}(\boldsymbol{s}, u) \ge B_{\rm f}(\boldsymbol{s'}, u)$ and $s_j \le s'_j$ for $u \le j \le v$. Therefore, $B_{\rm f}(\boldsymbol{s}, u) - B_{\rm f}(\boldsymbol{s'}, u) \ge 0$ and $\sum_{j=u}^{v} (s_j - s'_j) \le 0$. Combined with (3.19), this implies that $B_{\rm f}(\boldsymbol{s}, u) = B_{\rm f}(\boldsymbol{s'}, u)$ and $\sum_{j=u}^{v} s_j = \sum_{j=u}^{v} s'_j$. Since $s_j \le s'_j$ for $u \le j \le v$, we get $s_j = s'_j$ for $u \le j \le v$. ∎

Now for the main result of this section:

**Theorem 3.1** *Given $P = \langle N, F, B_{\rm tgt}, B_{\rm vbv}, B_1, B_{\rm inc} \rangle$, with CBR constraints, a legal allocation sequence $\boldsymbol{s}$ is optimal if and only if the following conditions hold for all $1 \le j \le N$. Also, the optimal allocation sequence is unique.*

1. *If $g_j(s_j) > g_{j+1}(s_{j+1})$ (i.e., the nominal quantization decreases), then we have $B_{\rm f}(\boldsymbol{s}, j) = s_j$ (or equivalently, $B^*_{\rm f}(\boldsymbol{s}, j) = 0$).*

2. *If $g_j(s_j) < g_{j+1}(s_{j+1})$ (i.e., the nominal quantization increases), then we have $B_{\rm f}(\boldsymbol{s}, j+1) = B_{\rm vbv}$ (or equivalently, $B^*_{\rm f}(\boldsymbol{s}, j) = B_{\rm vbv} - B_{\rm inc}(j)$).*

*Proof*: Lemma 3.2 established these condition as necessary for optimality. Now we need to show that these conditions are also sufficient for optimality and imply uniqueness. Let $\boldsymbol{s}^*$ be an optimal allocation sequence for $P$. Let $\boldsymbol{s}$ be a legal allocation sequence that meets both conditions of the theorem. We will show that $\boldsymbol{s} = \boldsymbol{s}^*$.

First let us restrict ourselves to a maximal segment of consecutive pictures to which $\boldsymbol{s}$ assigns its maximum quantization scale $Q_{\max}$. That is, let $Q_{\max} = \max_{1 \le j \le N} \{g_j(s_j)\}$. Let $u$ be the index of the first picture in such a segment. There are two cases to consider: $u = 1$ and $u > 1$. If $u = 1$, then $B_{\rm f}(\boldsymbol{s}, u) = B_{\rm f}(\boldsymbol{s}^*, u) = B_1$. If $u > 1$, then since $u$ is the index of the start of the segment, we have $g_{u-1}(s_{u-1}) < g_u(s_u)$, which by condition 2 implies that $B_{\rm f}(\boldsymbol{s}, u) = B_{\rm vbv}$. Since $\boldsymbol{s}^*$ is a legal allocation sequence, we have $B_{\rm f}(\boldsymbol{s}^*, u) \le B_{\rm vbv} = B_{\rm f}(\boldsymbol{s}, u)$. Therefore, in either case we have

$$B_{\rm f}(\boldsymbol{s}, u) \ge B_{\rm f}(\boldsymbol{s}^*, u). \tag{3.20}$$

Let $v$ be the index of the last picture of the constant-$Q_{\max}$ segment. Since $\boldsymbol{s}^*$ is optimal, it follows that $g_j(s_j^*) \le Q_{\max}$ for $u \le j \le v$, and

$$s_j^* \ge s_j, \qquad \text{for } u \le j \le v. \tag{3.21}$$

Combining (3.20) and (3.21) we get

$$B_{\rm f}(\boldsymbol{s}^*, j) \le B_{\rm f}(\boldsymbol{s}, j), \qquad \text{for } u \le j \le v. \tag{3.22}$$

There are two cases for $v$: $v = N$ and $v < N$. If $v = N$, then $B_{\rm f}(\boldsymbol{s}, v+1) = B_{\rm f}(\boldsymbol{s}^*, v+1) = B_1 + \sum_{j=1}^{N-1} B_{\rm inc}(j) - B_{\rm tgt}$. If $v < N$, then since $v$ is the index of the end of the segment, we have $g_v(s_v) > g_{v+1}(s_{v+1})$, which implies that

$$B_{\rm f}(\boldsymbol{s}, v) = s_v \tag{3.23}$$

by condition 1. Since $\boldsymbol{s}^*$ is a legal allocation sequence, we have $B_{\rm f}(\boldsymbol{s}^*, v) \ge s_v^*$. Combining this with (3.23) and (3.21) we get

$$B_{\rm f}(\boldsymbol{s}^*, v) \ge s_v^* \ge s_v = B_{\rm f}(\boldsymbol{s}, v). \tag{3.24}$$

By (3.22) and (3.24), we have $B_{\rm f}(\boldsymbol{s}^*, v) = B_{\rm f}(\boldsymbol{s}, v)$ and $s_v^* = s_v$. As a result, it follows that $B_{\rm f}(\boldsymbol{s}^*, v+1) = B_{\rm f}(\boldsymbol{s}, v+1)$. Therefore, in either case $v = N$ or $v < N$, we have $B_{\rm f}(\boldsymbol{s}^*, v+1) = B_{\rm f}(\boldsymbol{s}, v+1)$.

By (3.21) and Lemma 3.3, we have $B_{\rm f}(\boldsymbol{s}^*, u) = B_{\rm f}(\boldsymbol{s}, u)$ and $s_j^* = s_j$ for $u \le j \le v$. As a consequence, $B_{\rm f}(\boldsymbol{s}, j) = B_{\rm f}(\boldsymbol{s}^*, j)$ for $u \le j \le v$. This means that $\boldsymbol{s}$ and $\boldsymbol{s}^*$ agree on pictures that are assigned quantization scale $Q_{\max}$.

Now we consider the other segments of the video sequence, for which $\boldsymbol{s}$ does not assign quantization scale $Q_{\max}$. Let us partition pictures 1 through $N$ into maximal contiguous segments such that the pictures in a segment use the same nominal quantization scale $Q$. It follows that the first picture in a segment is picture 1 or it uses a different value of $Q$ than the previous picture in the video sequence. Similarly, the last picture in a segment is picture $N$ or it uses a value of $Q$ different from the next picture in the video sequence. Let $M$ be the number of such segments. We order the segments so that segment $k$ uses a value of $Q$ greater than or equal to the value of $Q$ used in the previous segments $1, 2, \ldots, k-1$. We denote the value of $Q$ used by segment $k$ as $Q_{(k)}$.

We will show that allocation sequence $\boldsymbol{s}^*$ uses the same number of bits as allocation sequence $\boldsymbol{s}$ for each picture in segment $k$, for $1 \le k \le M$. This will establish the conditions in the theorem as necessary and show that the optimal allocation sequence is unique. We will prove this claim by induction on $k$.

*Claim*: Let the $k$th video segment start at picture $u_k$ and end at picture $v_k$. That is,

- Either $u_k = 1$ or $g_{u_k-1}(s_{u_k-1}) \neq g_{u_k}(s_{u_k})$;
- Either $v_k = N$ or $g_{v_k}(s_{v_k}) \neq g_{v_k+1}(s_{v_k+1})$; and
- $g_j(s_j) = Q_{(k)}$, for all $u_k \le j \le v_k$.

Then for all $u_k \le j \le v_k$, we have $s_j^* = s_j$ and $B_{\rm f}(\boldsymbol{s}, j) = B_{\rm f}(\boldsymbol{s}^*, j)$.

We have already proven the base case ($k = 1$) of the claim. Our inductive hypothesis is that the claim is true for $k < m$. We need to show that the claim is also true for $k = m$.

Consider the $m$th segment of consecutive pictures, which $\boldsymbol{s}$ assigns nominal quantization scale $Q_{(m)}$. For notational simplicity let us denote $u = u_m$ and $v = v_m$. By the inductive hypothesis, $\boldsymbol{s}$ and $\boldsymbol{s}^*$ use the same values of $Q$ for all pictures in which $\boldsymbol{s}$ uses $Q > Q_{(m)}$. Because $\boldsymbol{s}^*$ is optimal, we have $g_j(s_j^*) \le g_j(s_j) = Q_{(m)}$ for $u \le j \le v$, and thus

$$s_j^* \ge s_j, \qquad \text{for } u \le j \le v. \tag{3.25}$$

We consider all cases for the segment boundaries. For the left boundary there are three cases: $u = 1$, $g_{u-1}(s_{u-1}) > g_u(s_u)$, or $g_{u-1}(s_{u-1}) < g_u(s_u)$. If $u = 1$, then $B_{\rm f}(\boldsymbol{s}^*, u) = B_{\rm f}(\boldsymbol{s}, u) = B_1$. If $g_{u-1}(s_{u-1}) > g_u(s_u)$, then from the inductive hypothesis, we have $B_{\rm f}(\boldsymbol{s}^*, u-1) = B_{\rm f}(\boldsymbol{s}, u-1)$ and $s_{u-1}^* = s_{u-1}$; therefore, $B_{\rm f}(\boldsymbol{s}^*, u) = B_{\rm f}(\boldsymbol{s}, u)$. If $g_{u-1}(s_{u-1}) < g_u(s_u)$, then from condition 2, we have $B_{\rm f}(\boldsymbol{s}, u) = B_{\rm vbv}$; since $\boldsymbol{s}^*$ is a legal allocation sequence, $B_{\rm f}(\boldsymbol{s}^*, u) \le B_{\rm vbv} = B_{\rm f}(\boldsymbol{s}, u)$. Therefore, for all three cases of $u$, we have

$$B_{\rm f}(\boldsymbol{s}^*, u) \le B_{\rm f}(\boldsymbol{s}, u). \tag{3.26}$$

For the right segment boundary there are three cases to consider: $v = N$, $g_v(s_v) < g_{v+1}(s_{v+1})$, or $g_v(s_v) > g_{v+1}(s_{v+1})$. If $v = N$, then $B_{\rm f}(\boldsymbol{s}^*, v+1) = B_{\rm f}(\boldsymbol{s}, v+1) = B_1 + \sum_{j=1}^{N-1} B_{\rm inc}(j) - B_{\rm tgt}$. If $g_v(s_v) < g_{v+1}(s_{v+1})$, then from the inductive hypothesis, we have $B_{\rm f}(\boldsymbol{s}^*, v+1) = B_{\rm f}(\boldsymbol{s}, v+1)$.

The last case on $v$ to consider is $g_v(s_v) > g_{v+1}(s_{v+1})$. From (3.25) and (3.26) we have

$$B_{\rm f}(\boldsymbol{s}^*, j) \le B_{\rm f}(\boldsymbol{s}, j), \qquad \text{for } u \le j \le v. \tag{3.27}$$

From condition 1 we have

$$B_{\rm f}(\boldsymbol{s}, v) = s_v. \tag{3.28}$$

Since $\boldsymbol{s}^*$ is a legal allocation sequence, we get $B_{\mathrm{f}}(\boldsymbol{s}^*, v) \geq s_v^*$. Combining this with (3.25) and (3.28), we get

$$B_{\mathrm{f}}(\boldsymbol{s}^*, v) \geq s_v^* \geq s_v = B_{\mathrm{f}}(\boldsymbol{s}, v). \tag{3.29}$$

From (3.27) and (3.29), we get $B_{\mathrm{f}}(\boldsymbol{s}^*, v) = B_{\mathrm{f}}(\boldsymbol{s}, v)$ and $s_v^* = s_v$. And thus we get $B_{\mathrm{f}}(\boldsymbol{s}^*, v+1) = B_{\mathrm{f}}(\boldsymbol{s}, v+1)$.

Therefore, for all three cases of $v$, we have

$$B_{\mathrm{f}}(\boldsymbol{s}^*, v+1) = B_{\mathrm{f}}(\boldsymbol{s}, v+1). \tag{3.30}$$

From (3.25), (3.26), (3.30), and Lemma 3.3, we have $s_j^* = s_j$ for $u \leq j \leq v$. It follows that $B_{\mathrm{f}}(\boldsymbol{s}, j) = B_{\mathrm{f}}(\boldsymbol{s}^*, j)$ for $u \leq j \leq v$, which proves the claim for $k = m$ and thus finishes the proof by induction. ∎

## 3.2 CBR ALLOCATION ALGORITHM

Theorem 3.1 is a powerful result. It says that to find the optimal allocation sequence, we need only find a legal allocation sequence that meets the stated switching conditions. In this section, we use the technique of dynamic programming (DP) to develop an algorithm to compute a lexicographically optimal CBR allocation sequence in polynomial time and linear space. We assume for purposes of this section that the bit-production model as described in Section 2.3 is perfectly accurate. (In Chapters 5 and 6, we consider the more realistic case in which the bit-production model is only approximately correct.)

### 3.2.1 DP Algorithm

The basic idea behind dynamic programming is to decompose a given problem in terms of optimal solutions to smaller problems. All we need to do is maintain invariant the conditions stated in Theorem 3.1 for each subproblem we solve. We do this by constructing optimal bit allocations for pictures 1 to $j$ that end up with the VBV buffer in one of two states: *full* or *empty*. These states are exactly the states where a change in the nominal quantization scale $Q$ may occur. For $1 \leq j < N$, let $\mathbf{Top}[j]$ be the legal allocation sequence, if such an allocation sequence exists, for pictures 1 to $j$ that ends up, immediately before the $(j+1)$st picture is removed, with the VBV buffer *full* (i.e., $B_{\mathrm{f}}(\boldsymbol{s}^*, j+1) = B_{\mathrm{vbv}}$, or equivalently $B_{\mathrm{f}}^*(\boldsymbol{s}^*, j) = B_{\mathrm{vbv}} - B_{\mathrm{inc}}(j)$). Similarly, let $\mathbf{Bot}[j]$ be the legal allocation sequence for pictures 1 to $j$ that ends up, immediately after the $j$th picture is removed, with the VBV buffer *empty* (i.e., $B_{\mathrm{f}}(\boldsymbol{s}^*, j) = s_j^*$, or equivalently, $B_{\mathrm{f}}^*(\boldsymbol{s}^*, j) = 0$). We use **Initial** to denote the empty allocation, which corresponds to the starting point with initial buffer fullness $B_1$. We denote the final complete legal allocation sequence $\boldsymbol{s}^*$ by **Final**. By (2.11), after the $N$th picture is removed, the final buffer contains $B_1 + \sum_{j=1}^{N-1} B_{\mathrm{inc}}(j) - B_{\mathrm{tgt}}$ bits.

Suppose that we have computed $\mathbf{Top}[k]$ and $\mathbf{Bot}[k]$ for $1 \leq k < j < N$. To compute $\mathbf{Top}[j]$, we search for a legal allocation sequence among $\{\mathbf{Initial}, \mathbf{Top}[1], \mathbf{Top}[2], \ldots, \mathbf{Top}[j-1], \mathbf{Bot}[1], \mathbf{Bot}[2], \ldots, \mathbf{Bot}[j-1]\}$ to which we can concatenate a constant-$Q$ segment to give a legal allocation sequence $\boldsymbol{s}$ such that the switching conditions are met and the buffer ends up full (i.e., $B_{\mathrm{f}}(\boldsymbol{s}, j+1) = B_{\mathrm{vbv}}$). Similarly, for $\mathbf{Bot}[j]$ we search for a previously computed allocation sequence in $\{\mathbf{Initial}, \mathbf{Top}[1], \mathbf{Top}[2], \ldots, \mathbf{Top}[j-1], \mathbf{Bot}[1], \mathbf{Bot}[2], \ldots, \mathbf{Bot}[j-1]\}$, so that when it is extended by a constant-$Q$ segment, it meets the switching conditions and results in the buffer being empty (i.e., $B_{\mathrm{f}}(\boldsymbol{s}, j) = s_j$, or equivalently, $B^*_{\mathrm{f}}(\boldsymbol{s}^*, j) = 0$). There is at most one legal (and therefore optimal) allocation sequence to each state.

Once we have computed $\mathbf{Top}[N-1]$ and $\mathbf{Bot}[N-1]$, we can compute the legal allocation sequence $\mathbf{Final}$ for all $N$ pictures in a process similar to the one above for computing $\mathbf{Top}[j]$ and $\mathbf{Bot}[j]$, except that the final allocation results in a final buffer state that gives the desired target number of bits $B_{\mathrm{tgt}}$. We can then trace in reverse order the path of concatenated links that lead from the empty state $\mathbf{Initial}$ to the final buffer state $\mathbf{Final}$.

The basic step in the DP algorithm is illustrated in Figure 3.2. The round nodes represent buffer states for which we have previously computed optimal allocation sequences leading to that state. Each node stores the last $Q$ used in the legal allocation sequence leading to that state and the origin of the last constant-$Q$ segment leading to that state. The square node represents the next state whose legal allocation sequence we wish to compute (e.g., $\mathbf{Top}[j]$ or $\mathbf{Bot}[j]$, for $j < N$, or $\mathbf{Final}$, for $j = N$). The dashed lines each represent a constant-$Q$ allocation that connects the respective nodes. To compute a solution for the square node, we need to search for a constant-$Q$ segment that connects the square node with a round node such that the switching conditions are met. The switching conditions are checked by comparing the $Q$ used for the segment against the last $Q$ used in the legal solution for the round node that the segment connects. The allocation implied by each constant-$Q$ segment is also checked for VBV compliance.

### 3.2.2 Correctness of DP Algorithm

When computing $\mathbf{Top}[j]$ and $\mathbf{Bot}[j]$ for $1 \leq j < N$, we have insured that the conditions of Theorem 3.1 are met. The conditions are also met in the computation of the final state $\mathbf{Final}$. Therefore, we end up with a legal allocation sequence that meets the conditions of Theorem 3.1 and is thus optimal.

### 3.2.3 Constant-$Q$ Segments

We have used the concept of a constant-$Q$ segment extensively in the above discussion. We now formalize this concept in connection with the notion of

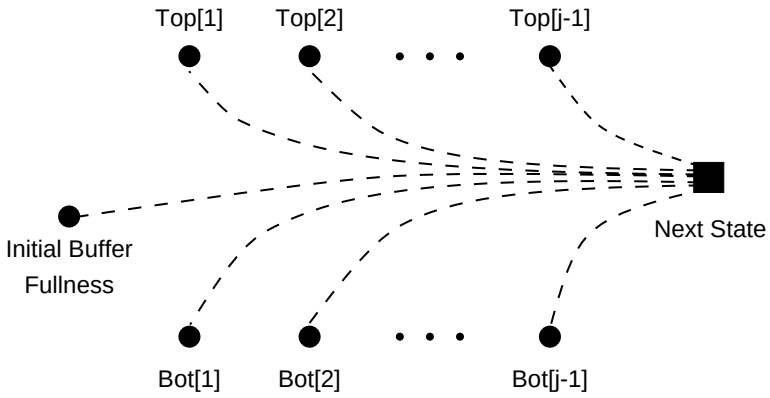



*Fig. 3.2* **Search Step in DP Algorithm.** This figure illustrates the generic search step in the DP algorithm. To extend the legal (optimal) solution to the Next State (which is either **Top**$[j]$ or **Bot**$[j]$, for $j < N$, or **Final**, for $j = N$), we search for for a constant-$Q$ connector (shown as dashed lines) that meets the switching conditions of Theorem 3.1.

bit-production models defined in Section 2.3. First, we define a family of bit-production functions $\{F_{i,j}(Q)\}$ that gives the number of bits resulting from allocating a constant nominal quantization scale $Q$ for pictures $i$ to $j$, inclusive:

$$F_{i,j}(Q) = \sum_{i \le k \le j} f_k(Q). \tag{3.31}$$

What we are really interested in, though, is the inverse of $F_{i,j}$. We denote the inverse as $G_{i,j}$ so that $G_{i,j} = F_{i,j}^{-1}$. Then $G_{i,j}(B)$ gives the constant $Q$ that results in $B$ bits being produced by pictures $i$ to $j$ collectively. Since $f_k$ is monotonically decreasing, so is $F_{i,j}$, and thus $G_{i,j}$ is monotonically increasing.

### 3.2.4 Verifying a Constant-$Q$ Allocation

The DP algorithm for CBR bit allocation given in Section 3.2.1 needs to verify whether each allocation subsequence of constant nominal quantization scale $Q$ meets the VBV buffer constraints. This can be done in time linear in the length of the allocation segment by simulating the VBV. In the DP algorithm, for each time step $1 \le j < N$, as shown in Figure 3.2, we need to verify the constant-$Q$ segments leading from the $2j-1$ round nodes $\{$**Initial**, **Top**[1], **Top**[2], ..., **Top**$[j-1]$, **Bot**[1], **Bot**[2], ..., **Bot**$[j-1]\}$ to **Top**$[j]$ and to **Bot**$[j]$, and there is a similar set of $2N-1$ verifications when $j = N$ leading to the desired final state **Final**, for a total of $2N-1+\sum_{j=1}^{N-1} 2(2j-1) = \Theta(N^2)$ verifications of constant-$Q$ allocations. If each verification requires linear time in the length of the constant-$Q$ segment, the total complexity for the DP algorithm is at least cubic time.

To rectify this situation, we can observe that the constant-$Q$ allocations to be verified all start with the buffer either full, empty, or at its initial state; and they all end with the buffer either full, empty, or at its final state. We also note that for bit allocations to a segment of pictures (say, from frames $i$ to $j$) with a fixed initial buffer state (say, $B_1$) and using $B_{\mathrm{T}}$ bits, there is a continuous range of $Q$ values that results in a legal allocation sequence. When additional pictures are considered, this range of legal $Q$ values never widens. The upper bound for $Q$ is simply the minimum $Q$ among the constant-$Q$ allocation sequences for pictures $i$ to $j$ in which the VBV buffer is exactly full immediately before picture $k+1$ is removed, for some $i \le k < j$. More formally, we have

$$G_{i,j}(B_{\mathrm{T}}) \le \min_{i \le k < j} \Big\{ G_{i,k}\Big( B_1 + \sum_{m=i}^{k} B_{\mathrm{inc}}(m) - B_{\mathrm{vbv}} \Big) \Big\}. \tag{3.32}$$

Similarly, the lower bound for $Q$ is the maximum $Q$ among the constant-$Q$ allocations for pictures $i$ to $j$ in which the VBV buffer is exactly empty immediately after picture $k$ is removed, for some $i \le k < j$. More formally, we have

$$G_{i,j}(B_{\mathrm{T}}) \ge \max_{i \le k < j} \Big\{ G_{i,k}\Big( B_1 + \sum_{m=i}^{k} B_{\mathrm{inc}}(m) \Big) \Big\}. \tag{3.33}$$

For the current frame $j$, where $1 \le j \le N$, we can use the above observations to perform the $O(j)$ VBV verifications (3.32) and (3.33) for $1 \le i < j$ in constant time per verification. For each $1 \le i \le N$ and the current $j$, we store the values of the term on the right-hand side of (3.32) and the term on the right-hand side of (3.33). Each time we increase $j$, we can update each of these values in constant time. For example, if we denote the term on the right-hand size of (3.32) by $U_{i,j}$, then by definition, $U_{i,j+1} = \min\big\{U_{i,j},\ G_{i,j}\big(B_1 + \sum_{m=i}^{j} B_{\mathrm{inc}}(m) - B_{\mathrm{vbv}}\big)\big\}$. The prefix sum $\sum_{m=i}^{j} B_{\mathrm{inc}}(m)$ can obviously be computed from $\sum_{m=i}^{j-1} B_{\mathrm{inc}}(m)$ in constant time. We show in the next section how to compute $G_{i,j}$ in constant time. The total space bound to store the terms is linear.

### 3.2.5 Time and Space Complexity

The time complexity of the DP algorithm depends upon two main factors: the time to compute a constant-$Q$ segment and the time to verify whether the constant-$Q$ segment can be appended to one of the previously computed allocation sequences. In the last section, we discussed how to verify a constant-$Q$ segment, so what remains is to show how to compute a constant-$Q$ allocation.

We assume that the bit-production model $f_k$ for picture $k$ and the inverse function $G_{i,j} = F_{i,j}^{-1}$ for pictures $i$ to $j$ can be evaluated in constant time with $O(N)$ preprocessing time and space. An example is the *hyperbolic bit-production model* $f_k(Q) = \alpha_k/Q + \beta_Q$, for positive real values $\alpha_k$ and $\beta_k$. We

have

$$\begin{aligned} F_{i,j}(Q) &= \sum_{k=i}^{j} f_k(Q) \\ &= \frac{1}{Q}\sum_{k=i}^{j}\alpha_k + \sum_{k=i}^{j}\beta_k; \\ Q &= \frac{\sum_{k=i}^{j}\alpha_k}{F_{i,j}(Q) - \sum_{k=i}^{j}\beta_k}. \end{aligned} \tag{3.34}$$

Since $F_{i,j}$ and $G_{i,j}$ are inverses, we can substitute $B$ for $F_{i,j}(Q)$ and $G_{i,j}(B)$ for $Q$ to get

$$\begin{aligned} G_{i,j}(B) &= \frac{\sum_{k=i}^{j}\alpha_k}{B - \sum_{k=i}^{j}\beta_k} \\ &= \frac{\sum_{k=1}^{j}\alpha_k - \sum_{k=1}^{i-1}\alpha_k}{B - \sum_{k=1}^{j}\beta_k + \sum_{k=1}^{i-1}\beta_k}. \end{aligned} \tag{3.35}$$

We can precompute the $N$ prefix sums $\sum_{k=1}^{j}\alpha_k$ and $\sum_{k=1}^{j}\beta_k$, $1 \le j \le N$, in linear time and space. By (3.35), we can use the prefix sums to compute $G_{i,j}$ in constant time. The same technique can be used for bit-production models of the form $f_k(Q) = \alpha_k/Q^2 + \beta_k/Q + \gamma_k$, $f_k(Q) = \alpha_k/Q^3 + \beta_k/Q^2 + \gamma_k/Q + \phi_k$, and $f_k(Q) = \alpha_k/Q^4 + \beta_k/Q^3 + \gamma_k/Q^2 + \phi_k/Q + \theta_k$. Examples of other functional forms for $f_k$ with a closed-form solution for $G_{i,j}$ can be found in [51]. Of course, we need to insure that the models are monotonically decreasing. We discuss bit-production models further in Chapter 5, including the important issue of how to handle inaccuracies in a model.

From what we learned in this section and the last, VBV verification and constant-$Q$ calculation can be done in constant time with linear-time preprocessing, and hence computing $\mathbf{Top}[j]$ and $\mathbf{Bot}[j]$ takes $O(j)$ time. Therefore, we can compute an optimal allocation sequence for the entire sequence of $N$ pictures in $\sum_{j=1}^{N} O(j) = O(N^2)$ time. If we store pointers for tracing the optimal sequence of concatenations, the algorithm requires $O(N)$ space.

## 3.3 RELATED WORK

Lin and Chan [43] give conditions similar to the switching conditions of Theorem 3.1 for optimal buffered bit allocation under a minimum sum-distortion criterion, assuming independent convex rate-distortion functions. They use the Lagrange multiplier method to find a bit allocation sequence that is optimal within a convex hull approximation. The optimal vector of Lagrange multipliers consists of constant-valued segments that increase (decrease, respectively) only when the decoder buffer is full (empty).

Salehi et al. [69] apply the theory of *majorization* [53] to reduce the variability in transmission rate for stored video. The problem is to determine a feasible transmission schedule by which a pre-compressed video bitstream can be transmitted over a communications channel to the decoder without underflowing or overflowing the decoder buffer. As applied to that problem, majorization results in minimizing the peak and variance in transmission rate. Salehi et al. [69] provide an optimal smoothing algorithm that runs in time linear in the length of the video sequence, based upon the algorithm of Lee and Preparata [42] for finding shortest paths in polygonal channels.

## 3.4 DISCUSSION

It can be easily shown that the majorization solution to optimal smoothing of [69] is, in fact, equivalent to lexicographic minimization of the transmitted rates, subject to the constraint that the total number of bits transmitted is fixed. Linear running time is possible because the buffering constraints are manifested as fixed upper and lower bounds on the cumulated number of bits transmitted. In buffer-constrained bit allocation, there is no fixed relationship between the bit allocation on the one hand and the nominal quantization scale (or equivalently, distortion) that is the object of optimization, but for any given video sequence, the relationship between the two can be precomputed in linear time and space using the approach of Section 3.2.5. As a result, the approach of [69], when modified appropriately, will yield a linear-time and linear-space approach to buffer-constrained bit allocation.

However, there is still one crucial difference between the transmission scheduling problem of [69] and buffer-constrained bit allocation: The bit-production model described in Section 2.3, which is at the heart of the relation between the buffering constraints and the quantization scale, is only *approximate* in practice. The value $f_k(Q)$ is the predicted number of bits used to encode picture $k$ with nominal quantization scale $Q$, but the actual encoding length for the $k$th picture will typically be different. Therefore, after encoding the $k$th picture, the optimal allocation sequence for the remaining pictures would have to be recomputed. Redoing the approach of [69] after each picture is encoded would take linear time for each $1 \leq k \leq N-1$, and therefore the total algorithm would become $O(N^2)$.

In Chapter 5, we consider errors in the bit-production models and show how a slight rearrangement of the DP algorithm—in which the dynamic programming is done *in reverse order* with respect to the video sequence—will allow the recomputation after each picture is encoded to be done in linear time, and thus the resulting total time is $O(N^2)$.

In Chapter 6 we show how to do better. We develop an optimal data structure for the buffer-constrained bit allocation problem for the case of inaccurate bit-production models. Recomputation after each picture takes constant time typically and $O(\log N)$ time in the worst case. The net result is

that the entire video sequence can be encoded in linear time in practice and always in linear space; in the worst case, the running time is $O(N \log N)$.

Another simplification of our analysis to date is that the bit-production model assumes that the coding of a particular picture is independent of the coding of any other picture. As noted earlier, this independence assumption does not hold for video coders that employ a differential coding scheme such as motion compensation. In this case, the coding of a reference picture affects the coding of subsequent pictures that are coded with respect to it. Therefore, the bit-production model for picture $j$ would depend causally not only upon the nominal quantization scale used for picture $j$ but also upon the nominal quantization scale used for its reference picture. The dependent coding problem has been addressed in the traditional distortion-minimization framework in [66]. How effectively the results of this chapter can be extended to a dependent-coding framework is an open problem.

# 4

# *Optimal Bit Allocation under VBR Constraints*

In this chapter, we analyze the buffer-constrained bit allocation problem under variable bit rate VBV constraints, as described in Section 2.4.2. The analysis leads to an efficient iterative algorithm for computing the unique lexicographically optimal solution.

In CBR operation, the total number of bits that a CBR stream can use is dictated by the channel bit rate and the buffer size. With VBR operation, the total number of bits has no lower bound, and its upper bound is determined by the peak bit rate and the buffer size. Consequently, VBR is useful and most advantageous over CBR when the average bit rate needs to be lower than the peak bit rate. This is especially critical in storage applications, where the storage capacity, and not the transfer rate, is the limiting factor. Another important application of VBR video coding is multiplexing multiple video bitstreams over a CBR channel [27]. In this application, statistical properties of the multiple video sequences allow more VBR bitstreams with a given peak rate $R_{\max}$ to be multiplexed onto the channel than CBR bitstreams coded at a constant rate of $R_{\max}$.

For typical VBR applications, then, the average bit rate is lower than the peak. In this case, bits enter the decoder buffer at an effective bit rate that is less than the peak during the display interval of many pictures. In interesting cases, there will be segments of pictures that will be coded with an average bit rate that is higher than the peak. This is possible because of the buffering. During the display of these pictures, the VBV buffer fills at the peak rate. Since these pictures require more bits to code than the peak rate, they are "harder" to code than the other "easier" pictures.

In order to equalize quality, the easy pictures should be coded at the same base quality. It does not pay to code any of the hard pictures at a quality higher than that of the easy pictures. The bits expended to do so could instead be better distributed to raise the quality of the easy pictures. Among the hard pictures, there are different levels of coding difficulty. Using the same intuitions from the CBR case, we can draw similar conclusions about the buffer emptying and filling behavior among the hard pictures.

In the following analysis, we show that a lexicographically optimal VBR bit allocation sequence possesses the properties described above. In particular, the hard segments of pictures in a VBR bit allocation sequence behave as in a CBR setting. In fact, the VBR algorithm invokes the CBR algorithm to allocate bits to segments of hard pictures.

## 4.1 ANALYSIS

For an optimal allocation sequence, let us refer to the pictures that are coded with the best quality (i.e., with the lowest nominal quantization scale $Q_{\min}$) as "easy" pictures. Lemma 4.1 below characterizes the easy pictures (except for picture $N$) as precisely those pictures whose encoding causes a virtual buffer overflow, in which the buffer must be filled at less than maximum rate. Lemma 4.2 characterizes the $N$th picture as easy if the final buffer state is nonempty.

**Lemma 4.1** *Consider optimal allocation sequence $\boldsymbol{s}^*$ for bit allocation problem $P = \langle N, F, B_{\text{tgt}}, B_{\text{vbv}}, B_1, B_{\text{inc}} \rangle$ with VBR constraints. If $B_{\text{f}}(\boldsymbol{s}^*, j) + B_{\text{inc}}(j) - s_j^* > B_{\text{vbv}}$ for some $1 \le j \le N$, then $g_j(s_j^*) = \min_{1 \le k \le N} \{g_k(s_k^*)\}$, where $g$ is the quantization function defined in Section 2.3.*

*Proof*: Let $Q_{\min} = \min_{1 \le k \le N} \{g_k(s_k^*)\}$. Let $j$ be an index such that $B_{\text{f}}(\boldsymbol{s}^*, j) + B_{\text{inc}}(j) - s_j^* > B_{\text{vbv}}$. Let $\Delta = B_{\text{f}}(\boldsymbol{s}^*, j) + B_{\text{inc}}(j) - s_j^* - B_{\text{vbv}}$. Thus, $\Delta > 0$.

Suppose that $g_j(s_j^*) > Q_{\min}$. Let $u$ be an index such that $g_u(s_u^*) = Q_{\min}$. Consider an allocation sequence $\boldsymbol{s}$ that differs from $\boldsymbol{s}^*$ only for pictures $j$ and $u$. We want to assign values to $s_j$ and $s_u$ that make $\boldsymbol{s}$ a legal allocation sequence with $g_j(s_j)$, $g_u(s_u) < g_j(s_j^*)$, which implies $\boldsymbol{s} \prec \boldsymbol{s}^*$, a contradiction.

The idea is that we want to shift a (positive) number of bits, say $\delta$, from picture $u$ to picture $j$ but still have a legal allocation sequence. Let $s_j = s_j^* + \delta$ and $s_u = s_u^* - \delta$ with $\delta > 0$. Then $g_j(s_j) < g_j(s_j^*)$ and $\sum_{k=1}^{N} s_k = \sum_{k=1}^{N} s_k^* = B_{\text{tgt}}$. We first need to show that $\boldsymbol{s}$ does not result in a VBV buffer underflow; that is, we want to show that $B_{\text{f}}(\boldsymbol{s}, k) \ge s_k$. There are two cases to consider: $u < j$ and $u > j$.

*Case 1:* $u < j$. Since $s_k = s_k^*$ for $k < u$, we have $B_{\text{f}}(\boldsymbol{s}, k) = B_{\text{f}}(\boldsymbol{s}^*, k)$ for $k \le u$. And since $s_u < s_u^*$ and $s_k = s_k^*$ for $u < k < j$, we have $B_{\text{f}}(\boldsymbol{s}, k) \ge B_{\text{f}}(\boldsymbol{s}^*, k)$ for $u < k \le j$. Therefore, pictures 1 to $j-1$ cannot cause any VBV

buffer underflows. If we choose $0 < \delta < \Delta$, then $B_{\rm f}(\boldsymbol{s}, j+1) = B_{\rm vbv}$ and picture $j$ cannot cause a VBV buffer underflow. Since $s_k = s_k^*$ for $k > j$ and $B_{\rm f}(\boldsymbol{s}, j+1) = B_{\rm f}(\boldsymbol{s}^*, j+1)$, pictures $j+1$ to $N$ cannot cause any VBV buffer underflows.

*Case 2:* $u > j$. Since $s_k = s_k^*$ for $k < j$, we have $B_{\rm f}(\boldsymbol{s}, k) = B_{\rm f}(\boldsymbol{s}^*, k)$ for $k \leq j$. If we choose $0 < \delta < \Delta$, then $B_{\rm f}(\boldsymbol{s}, j+1) = B_{\rm vbv}$ and picture $j$ cannot cause a VBV buffer underflow. Since $s_k = s_k^*$ for $j < k < u$, and $B_{\rm f}(\boldsymbol{s}, j+1) = B_{\rm f}(\boldsymbol{s}^*, j+1)$ by our choice of $\delta$, pictures $j+1$ to $u-1$ cannot cause any VBV buffer underflows. Since $s_u < s_u^*$, we have $B_{\rm f}(\boldsymbol{s}, k) \geq B_{\rm f}(\boldsymbol{s}^*, k)$ for $k \geq u$. Therefore, pictures $u$ to $N$ cannot cause any VBV buffer underflows.

Therefore, $\boldsymbol{s}$ is a legal allocation sequence with $g_j(s_j) < g_j(s_j^*)$. We need to guarantee that $g_u(s_u) < g_j(s_j^*)$ so that $\boldsymbol{s} \prec \boldsymbol{s}^*$. Let $\gamma = g_j(s_j^*) - g_u(s_u^*)$. Since $g_j(s_j^*) > g_u(s_u^*)$, we have $\gamma > 0$. Let $\alpha = s_u^* - f_u(g_u(s_u^*) + \gamma/2)$. Since $f_u$ is decreasing and $\gamma > 0$, we have $\alpha > 0$ and

$$\begin{aligned} s_u^* - \alpha &= f_u\left(g_u(s_u^*) + \frac{\gamma}{2}\right); \\ g_u(s_u^* - \alpha) &= g_u(s_u^*) + \frac{\gamma}{2} \\ &< g_u(s_u^*) + \gamma \\ &= g_j(s_j^*). \end{aligned}$$

Consider the assignment $\delta = \min\{\alpha, \Delta/2\}$. There are two cases: $\alpha \leq \Delta/2$ and $\alpha > \Delta/2$. If $\alpha \leq \Delta/2$, we have $\delta = \alpha$, from which follows $g_u(s_u) = g_u(s_u^* - \delta) = g_u(s_u^* - \alpha) < g_j(s_j^*)$; since $0 < \delta < \Delta$, the allocation sequence $\boldsymbol{s}$ is legal. If $\alpha > \Delta/2$, we have $\delta = \Delta/2$. Since $g_u$ is decreasing and $\alpha > \Delta/2$, we have $g_u(s_u^* - \Delta/2) < g_u(s_u^* - \alpha)$ and thus $g_u(s_u) = g_u(s_u^* - \delta) = g_u(s_u^* - \Delta/2) < g_j(s_j^*)$. Since $0 < \delta < \Delta$, the allocation sequence $\boldsymbol{s}$ is legal. Since $\boldsymbol{s}$ is a legal allocation sequence that differs from $\boldsymbol{s}^*$ only for pictures $u$ and $j$ with $g_u(s_u) < g_j(s_j^*)$ and $g_j(s_j) < g_j(s_j^*)$, from Lemma 3.1 we have $\boldsymbol{s} \prec \boldsymbol{s}^*$, and thus $\boldsymbol{s}^*$ is not optimal, a contradiction. Therefore, $g_j(s_j^*) = \min_k \left\{g_k(s_k^*)\right\}$. ∎

**Lemma 4.2** *Consider optimal allocation sequence $\boldsymbol{s}^*$ for bit allocation problem $P = \langle N, F, B_{\rm tgt}, B_{\rm vbv}, B_1, B_{\rm inc}\rangle$ with VBR constraints. If $B_{\rm f}(\boldsymbol{s}^*, N) > s_N^*$, then $g_N(s_N^*) = \min_{1 \leq k \leq N} \left\{g_k(s_k^*)\right\}$.*

*Proof*: Let $Q_{\min} = \min_{1 \leq k \leq N} \left\{g_k(s_k^*)\right\}$. Let $j$ be an index such that $B_{\rm f}(\boldsymbol{s}^*, j) + B_{\rm inc}(j) - s_j^* > B_{\rm vbv}$. Let $\Delta = B_{\rm f}(\boldsymbol{s}^*, N) - s_N^*$. Since $B_{\rm f}(\boldsymbol{s}^*, N) > s_N^*$, we have $\Delta > 0$. Suppose that $g_N(s_N^*) > Q_{\min}$. Let $u$ be an index such that $g_u(s_u^*) = Q_{\min}$. Now consider an allocation sequence $\boldsymbol{s}$ that differs from $\boldsymbol{s}^*$ only for pictures $u$ and $N$. We want to assign values to $s_N$ and $s_u$ that make $\boldsymbol{s}$ a legal allocation sequence with $g_N(s_N), g_u(s_u) < g_N(s_N^*)$, from which will follow $\boldsymbol{s} \prec \boldsymbol{s}^*$, a contradiction. Let $s_N = s_N^* + \delta$ and $s_u = s_u^* - \delta$ with $\delta > 0$. Then $g_N(s_N) < g_N(s_N^*)$ and $\sum_{k=1}^N s_k = \sum_{k=1}^N s_k^* = B_{\rm tgt}$. We now need to show that $\boldsymbol{s}$ does not result in a VBV buffer underflow; that is, we want to show that $B_{\rm f}(\boldsymbol{s}, k) \geq s_k$.

Since $s_k = s_k^*$ for $k < u$, we have $B_{\rm f}(\boldsymbol{s},k) = B_{\rm f}(\boldsymbol{s}^*,k)$ for $k \le u$. Since $s_u < s_u^*$ and $s_k = s_k^*$ for $u < k < N$, we have $B_{\rm f}(\boldsymbol{s},k) \ge B_{\rm f}(\boldsymbol{s}^*,k)$ for $u < k \le N$. Therefore, pictures 1 to $N-1$ cannot cause any VBV buffer underflows. For picture $N$, we have $B_{\rm f}(\boldsymbol{s},N) \ge B_{\rm f}(\boldsymbol{s}^*,N) = \Delta + s_N^* = \Delta + s_N - \delta$. Therefore, if we choose $\delta < \Delta$, then $B_{\rm f}(\boldsymbol{s},N) > s_N$ and picture $N$ cannot cause a VBV buffer underflow.

We have thus shown that $\boldsymbol{s}$ is a legal allocation sequence with $g_N(s_N) < g_N(s_N^*)$. We now need to guarantee that $g_u(s_u) < g_N(s_N^*)$. Let $\gamma = g_N(s_N^*) - g_u(s_u^*)$. Since $g_N(s_N^*) > g_u(s_u^*)$, we have $\gamma > 0$. Let $\alpha = s_u^* - f_u(g_u(s_u^*)+\gamma/2)$. Since $f_u$ is decreasing and $\gamma > 0$, we have $\alpha > 0$ and

$$\begin{aligned} s_u^* - \alpha &= f_u\left(g_u(s_u^*) + \frac{\gamma}{2}\right); \\ g_u(s_u^* - \alpha) &= g_u(s_u^*) + \frac{\gamma}{2} \\ &< g_u(s_u^*) + \gamma \\ &= g_N(s_N^*). \end{aligned}$$

Consider the assignment $\delta = \min\{\alpha, \Delta/2\}$. There are two cases: $\alpha \le \Delta/2$ and $\alpha > \Delta/2$. If $\alpha \le \Delta/2$, we have $\delta = \alpha$, from which follows $g_u(s_u) = g_u(s_u^* - \delta) = g_u(s_u^* - \alpha) < g_N(s_N^*)$; since $0 < \delta < \Delta$, the allocation sequence $\boldsymbol{s}$ is legal. If $\alpha > \Delta/2$, we have $\delta = \Delta/2$. Since $g_u$ is decreasing and $\alpha > \Delta/2$, we have $g_u(s_u^* - \Delta/2) < g_u(s_u^* - \alpha)$ and thus $g_u(s_u) = g_u(s_u^* - \delta) = g_u(s_u^* - \Delta/2) < g_N(s_N^*)$. Since $0 < \delta < \Delta$, the allocation sequence $\boldsymbol{s}$ is legal.

Since $\boldsymbol{s}$ is a legal allocation sequence that differs from $\boldsymbol{s}^*$ only for pictures $u$ and $N$ with $g_u(s_u) < g_N(s_N^*)$ and $g_N(s_N) < g_N(s_N^*)$, then from Lemma 3.1, we have $\boldsymbol{s} \prec \boldsymbol{s}^*$, and thus $\boldsymbol{s}^*$ is not optimal, a contradiction. Therefore, $g_N(s_N^*) = \min_k \{g_k(s_k^*)\}$. ∎

The next lemma gives a set of *switching conditions* for changes in $Q$ that are similar to the results of Lemma 3.2. Condition 1 dealing with an empty buffer state remains the same.

**Lemma 4.3** *Consider bit allocation problem $P = \langle N, F, B_{\rm tgt}, B_{\rm vbv}, B_1, B_{\rm inc}\rangle$ with VBR constraints. If $\boldsymbol{s}^*$ is an optimal allocation sequence, then the following conditions hold for all $1 \le j < N$:*

1. *If $g_j(s_j^*) > g_{j+1}(s_{j+1}^*)$ (i.e., the nominal quantization decreases), then we have $B_{\rm f}(\boldsymbol{s}^*,j) = s_j^*$ (or equivalently, $B_{\rm f}^*(\boldsymbol{s}^*,j) = 0$).*

2. *If $g_j(s_j^*) < g_{j+1}(s_{j+1}^*)$ (i.e., the nominal quantization increases), then we have $B_{\rm f}(\boldsymbol{s}^*,j+1) = B_{\rm vbv}$ and $B_{\rm f}(\boldsymbol{s}^*,j+1) + B_{\rm inc}(j+1) - s_{j+1}^* \le B_{\rm vbv}$.*

*Proof*:
*Case 1.* The proof is identical to the proof of Case 1 of Lemma 3.2, except that condition (2.5) now holds instead of (2.8).

*Case 2.* Let us consider the case in which $g_j(s_j^*) < g_{j+1}(s_{j+1}^*)$ for some $1 \le j < N$. If $B_{\rm f}(\boldsymbol{s}^*, j+1) + B_{\rm inc}(j+1) - s_{j+1}^* > B_{\rm vbv}$, then by Lemma 4.1, we have $g_{j+1}(s_{j+1}^*) \le g_j(s_j^*)$, a contradiction. Therefore, we have

$$B_{\rm f}(\boldsymbol{s}^*, j+1) + B_{\rm inc}(j+1) - s_{j+1}^* \le B_{\rm vbv}. \tag{4.1}$$

Suppose that $B_{\rm f}(\boldsymbol{s}^*, j+1) < B_{\rm vbv}$. Let $\Delta = B_{\rm vbv} - B_{\rm f}(\boldsymbol{s}^*, j+1) > 0$. Consider an allocation sequence $\boldsymbol{s}$ that differs from $\boldsymbol{s}^*$ only for pictures $j$ and $j+1$. We want to assign values to $s_j$ and $s_{j+1}$ that make $\boldsymbol{s}$ a legal allocation sequence with $g_j(s_j)$, $g_{j+1}(s_{j+1}) < g_{j+1}(s_{j+1}^*)$, from which it will follow that $\boldsymbol{s} \prec \boldsymbol{s}^*$, a contradiction.

Let $\gamma = g_{j+1}(s_{j+1}^*) - g_j(s_j^*)$ and $\alpha = s_j^* - f_j(g_j(s_j^*) + \gamma/2)$. Since $g_j(s_j^*) < g_{j+1}(s_{j+1}^*)$, we have $\gamma > 0$. Since $f_j$ is decreasing and $\gamma > 0$, we have $\alpha > 0$ and

$$\begin{aligned} s_j^* - \alpha &= f_j\left(g_j(s_j^*) + \frac{\gamma}{2}\right); \\ g_j(s_j^* - \alpha) &= g_j(s_j^*) + \frac{\gamma}{2} \\ &< g_j(s_j^*) + \gamma \\ &= g_{j+1}(s_{j+1}^*). \end{aligned}$$

Consider the assignments $s_j = s_j^* - \delta$ and $s_{j+1} = s_{j+1}^* + \delta$, where $\delta = \min\{\alpha, \Delta/2\}$. By these assignments, we have $g_{j+1}(s_{j+1}) < g_{j+1}(s_{j+1}^*)$. We now show that $g_j(s_j) < g_{j+1}(s_{j+1}^*)$. There are two cases: $\alpha \le \Delta/2$ and $\alpha > \Delta/2$. If $\alpha \le \Delta/2$, we have $\delta = \alpha$ and thus $g_j(s_j) = g_j(s_j^* - \delta) = g_j(s_j^* - \alpha) < g_{j+1}(s_{j+1}^*)$. If $\alpha > \Delta/2$, we have $\delta = \Delta/2$. Since $g_j$ is decreasing and $\alpha > \Delta/2$, we have $g_j(s_j^* - \Delta/2) < g_j(s_j^* - \alpha)$ and thus $g_j(s_j) = g_j(s_j^* - \delta) = g_j(s_j^* - \Delta/2) < g_{j+1}(s_{j+1}^*)$. In either case, we have $g_j(s_j) < g_{j+1}(s_{j+1}^*)$.

We need to show that allocation sequence $\boldsymbol{s}$ as defined above is legal. Since $s_k = s_k^*$ for $k < j$, we have $B_{\rm f}(\boldsymbol{s}, k) = B_{\rm f}(\boldsymbol{s}^*, k)$ for $k \le j$. Therefore, there are no VBV buffer violations in pictures 1 to $j-1$. Since $s_j < s_j^*$, we have $B_{\rm f}(\boldsymbol{s}, j+1) > B_{\rm f}(\boldsymbol{s}^*, j+1)$, and thus picture $j$ cannot cause a VBV buffer underflow.

Now we need to show that pictures $j+1$ to $N$ cannot cause a VBV buffer underflow. Since we assumed that $B_{\rm f}(\boldsymbol{s}^*, j+1) < B_{\rm vbv}$, we have $B_{\rm f}(\boldsymbol{s}^*, j+1) = B_{\rm f}(\boldsymbol{s}^*, j) + B_{\rm inc}(j) - s_j^*$ and

$$\begin{aligned} B_{\rm f}(\boldsymbol{s}, j) + B_{\rm inc}(j) - s_j &= B_{\rm f}(\boldsymbol{s}^*, j) + B_{\rm inc}(j) - (s_j^* - \delta) \\ &= B_{\rm f}(\boldsymbol{s}^*, j+1) + \delta \\ &< B_{\rm f}(\boldsymbol{s}^*, j+1) + \Delta \\ &= B_{\rm vbv}. \end{aligned}$$

Therefore, it follows that $B_{\rm f}(\boldsymbol{s}, j+1) = B_{\rm f}(\boldsymbol{s}, j) + B_{\rm inc}(j) - s_j$. By (4.1) we know that $B_{\rm f}(\boldsymbol{s}^*, j+1) + B_{\rm inc}(j+1) - s_{j+1}^* \le B_{\rm vbv}$, and hence $B_{\rm f}(\boldsymbol{s}^*, j+2) =$

$B_{\rm f}(\boldsymbol{s}^*, j+1) + B_{\rm inc}(j+1) - s^*_{j+1}$. Now,

$$\begin{aligned} & B_{\rm f}(\boldsymbol{s}, j+1) + B_{\rm inc}(j+1) - s_{j+1} \\ &\quad = B_{\rm f}(\boldsymbol{s}, j) + B_{\rm inc}(j) - s_j + B_{\rm inc}(j+1) - s_{j+1} \\ &\quad = B_{\rm f}(\boldsymbol{s}^*, j+1) + \delta + B_{\rm inc}(j+1) - s^*_{j+1} - \delta \\ &\quad = B_{\rm f}(\boldsymbol{s}^*, j+1) + B_{\rm inc}(j+1) - s^*_{j+1} \\ &\quad = B_{\rm f}(\boldsymbol{s}^*, j+2) \\ &\quad \le B_{\rm vbv}. \end{aligned}$$

Therefore, $B_{\rm f}(\boldsymbol{s}, j+2) = B_{\rm f}(\boldsymbol{s}^*, j+2)$. Since $s_k = s^*_k$ for $k > j+1$, we have $B_{\rm f}(\boldsymbol{s}, k) = B_{\rm f}(\boldsymbol{s}^*, k)$ for $k > j+1$. Hence, pictures $j+1$ to $N$ cannot cause a VBV buffer underflow, and $\boldsymbol{s}$ is a legal allocation sequence.

Since $\boldsymbol{s}$ is a legal allocation sequence that differs from $\boldsymbol{s}^*$ only for pictures $j$ and $j+1$ and we have $g_j(s_j) < g_{j+1}(s^*_{j+1})$ and $g_{j+1}(s_{j+1}) < g_{j+1}(s^*_{j+1})$, then from Lemma 3.1 we have $\boldsymbol{s} \prec \boldsymbol{s}^*$, which implies that $\boldsymbol{s}^*$ is not optimal, a contradiction. ∎

The following theorem is the main result of this section. It shows that the minimum-$Q$ conditions and switching conditions in the previous lemmas are sufficient for optimality.

**Theorem 4.1** *Given $P = \langle N, F, B_{\rm tgt}, B_{\rm vbv}, B_1, B_{\rm inc}\rangle$ with VBR constraints, a legal allocation sequence $\boldsymbol{s}$ is optimal if and only if the following conditions hold for all $1 \le j \le N$. Also, the optimal allocation sequence is unique.*

1. *If $B_{\rm f}(\boldsymbol{s}, j) + B_{\rm inc}(j) - s_j > B_{\rm vbv}$, then we have $g_j(s_j) = \min_k \{g_k(s_k)\}$.*
2. *If $B_{\rm f}(\boldsymbol{s}^*, N) > s^*_N$ then we have $g_N(s^*_N) = \min_{1\le k\le N} \{g_k(s^*_k)\}$.*
3. *If $g_j(s_j) > g_{j+1}(s_{j+1})$ (i.e., the nominal quantization decreases), then we have $B_{\rm f}(\boldsymbol{s}, j) = s_j$ (or equivalently, $B^*_{\rm f}(\boldsymbol{s}^*, j) = 0$).*
4. *If $g_j(s_j) < g_{j+1}(s_{j+1})$ (i.e., the nominal quantization increases), then we have $B_{\rm f}(\boldsymbol{s}, j+1) = B_{\rm vbv}$ and $B_{\rm f}(\boldsymbol{s}, j+1) + B_{\rm inc}(j+1) - s_{j+1} \le B_{\rm vbv}$.*

*Proof*: Lemmas 4.1, 4.2, and 4.3 establish these as necessary conditions. Now we need to show that these conditions are also sufficient for optimality and that they imply uniqueness.

The proof for sufficiency and uniqueness is similar to that of Theorem 3.1 except for segments with the minimum $Q$. Here we consider only segments that use the minimum $Q$.

Let $\boldsymbol{s}^*$ be an optimal allocation sequence and $Q_{\rm min} = \min_j \{g_j(s_j)\}$. By condition 2, if $g_N(s_N) > Q_{\rm min}$ then it must be that $B_{\rm f}(\boldsymbol{s}, N) = s_N$, or equivalently, $B_{\rm f}(\boldsymbol{s}, N+1) = B_{\rm inc}(N)$. Therefore, $B_{\rm f}(\boldsymbol{s}, N)$ is known if picture $N$ does not use the minimum $Q$, and we can use arguments of Theorem 3.1. Following the steps of Theorem 3.1, we can show that $s^*_j = s_j$ for all $j$ satisfying $g_j(s_j) > Q_{\rm min}$.

Let $J_{\min} = \{ j \mid g_j(s_j) = Q_{\min} \}$. Since $\boldsymbol{s}^*$ is optimal, we have $g_j(s_j^*) \leq g_j(s_j)$ for $j \in J_{\min}$. Therefore,

$$s_j^* \geq s_j, \qquad \text{for } j \in J_{\min}. \tag{4.2}$$

Since the total number of bits allocated is the same for $\boldsymbol{s}$ and $\boldsymbol{s}^*$, the number of bits they allocate to pictures in $J$ must also be the same. That is,

$$\sum_{j \in J_{\min}} s_j^* = \sum_{j \in J_{\min}} s_j. \tag{4.3}$$

But (4.2) and (4.3) both hold if and only if $s_j^* = s_j$ for $j \in J_{\min}$. Therefore, we have $s = s^*$. ∎

Although Theorem 4.1 is an important result, it does not show us how to compute the minimum nominal quantization scale $Q$ with which to code the "easy" pictures. The following lemmas and theorem show that, if we relax the bit budget constraint, we can find the minimum $Q$ and therefore the optimal allocation sequence to meet the bit budget by an iterative process, which we describe in detail in the next section. The iterative process is guaranteed to converge to the optimal allocation sequence in a finite number of steps.

**Lemma 4.4** *Consider two VBR problems* $P = \langle N, F, B_{\text{tgt}}, B_{\text{vbv}}, B_1, B_{\text{inc}} \rangle$ *and* $P' = \langle N, F, B'_{\text{tgt}}, B_{\text{vbv}}, B_1, B_{\text{inc}} \rangle$ *with* $B_{\text{tgt}} < B'_{\text{tgt}}$, *and let their optimal allocation sequences be* $\boldsymbol{s}$ *and* $\boldsymbol{s}'$, *respectively. It follows that* $\boldsymbol{s} \succ \boldsymbol{s}'$.

*Proof*: Let $J_{\text{over}} = \{ j \mid B_{\text{f}}(\boldsymbol{s}, j) + B_{\text{inc}}(j) - s_j > B_{\text{vbv}} \}$. Then $J_{\text{over}}$ contains exactly the pictures that result in virtual overflows, as defined in Section 2.4.2. If we start with allocation sequence $\boldsymbol{s}$, it is clear that we can use more bits for the pictures in $J_{\text{over}}$ without changing the buffer fullness $B_{\text{f}}(\boldsymbol{s}, n)$. Let $B_{\text{over}} = \sum_{j \in J_{\text{over}}} \big( B_{\text{f}}(\boldsymbol{s}, j) + B_{\text{inc}}(j) - s_j - B_{\text{vbv}} \big)$. Then $B_{\text{over}}$ is the maximum number of bits we can add to the pictures in $J_{\text{over}}$ without changing the buffer fullness. Let $\Delta = B'_{\text{tgt}} - B_{\text{tgt}}$. There are two cases to consider: $\Delta \leq B_{\text{over}}$ and $\Delta > B_{\text{over}}$.

*Case 1:* $\Delta \leq B_{\text{over}}$. Consider an allocation sequence $\boldsymbol{s}$ for problem $P'$ constructed as follows: Let $s_j = s_j$ for $j \notin J_{\text{over}}$. We distribute $\Delta$ bits to the pictures in $J_{\text{over}}$ without changing the buffer fullness. Then we have $s_j \geq s_j$, which implies that $g_j(s_j) \leq g_j(s_j)$. Since $\Delta > 0$, we also have $s_j > s_j$ for some $j \in J_{\text{over}}$. Since $B_{\text{f}}(\boldsymbol{s}, j) = B_{\text{f}}(\boldsymbol{s}, j)$ for all $j$, allocation sequence $\boldsymbol{s}$ does not cause any buffer underflows. We used $B_{\text{tgt}} + \Delta = B'_{\text{tgt}}$ bits in $\boldsymbol{s}$, and thus $\boldsymbol{s}$ is a legal allocation sequence for $P'$.

*Case 2:* $\Delta > B_{\text{over}}$. Consider an allocation sequence $\boldsymbol{s}''$ for problem $P'$ constructed as follows: Let $s''_j = s_j$ for $j \notin J_{\text{over}} \cup \{N\}$. We then distribute $B_{\text{over}}$ bits to pictures in $J_{over}$ by the assignments $s''_j = s_j + (B_{\text{f}}(\boldsymbol{s}, j) + B_{\text{inc}}(j) - s_j - B_{\text{vbv}})$, for $j \in J_{over}$. Finally, we distribute the remaining $\Delta - B_{over}$ bits to picture $N$ with $s''_N = s_N + \Delta - B_{over}$.

We have shown how to create a legal allocation sequence $\boldsymbol{s}''$ for $P'$ starting with $\boldsymbol{s}$. When we add more bits to $\boldsymbol{s}$ to form $\boldsymbol{s}''$, we strictly decrease $Q$ for the pictures that we add bits to and never increase $Q$ anywhere. Therefore, we have $\boldsymbol{s} \succ \boldsymbol{s}''$. Since $\boldsymbol{s}'$ is the optimal allocation sequence for $P'$, we have $\boldsymbol{s}'' \succeq \boldsymbol{s}'$. Therefore, $\boldsymbol{s} \succ \boldsymbol{s}'$. ∎

**Lemma 4.5** *Consider two VBR problems* $P = \langle N, F, B_{\rm tgt}, B_{\rm vbv}, B_1, B_{\rm inc}\rangle$ *and* $P' = \langle N, F, B'_{\rm tgt}, B_{\rm vbv}, B_1, B_{\rm inc}\rangle$ *with* $B_{\rm tgt} < B'_{\rm tgt}$*, and let their optimal allocation sequences be* $\boldsymbol{s}$ *and* $\boldsymbol{s}'$*, respectively. It follows that* $s_j = s'_j$ *for all* $1 \le j \le N$ *for which* $g_j(s_j) > \min_{1\le k\le N}\{g_k(s_k)\}$.

*Proof*: We provide an inductive proof similar to that used to prove Theorem 3.1. First we assume that $\boldsymbol{s}$ is not a constant-$Q$ allocation sequence, for if it were, the lemma would hold vacuously. Let $Q_{(k)}$ be the $k$th largest value of $Q$ assigned by allocation sequence $\boldsymbol{s}$. Let $Q_{\rm min}$ be the minimum value of $Q$ assigned by $\boldsymbol{s}$.

*Claim*: Consider a segment of consecutive pictures in the video sequence such that

- Either $u = 1$ or $g_{u-1}(s_{u-1}) \neq g_u(s_u)$;
- Either $v = N$ or $g_v(s_v) \neq g_{v+1}(s_{v+1})$; and
- $g_j(s_j) = Q^{(k)} > Q_{\rm min}$, for all $u \le j \le v$.

Then for all $u \le j \le v$, we have $s_j = s'_j$ and $B_{\rm f}(\boldsymbol{s}, j) = B_{\rm f}(\boldsymbol{s}', j)$.

We first prove the base case ($k = 1$) of the claim. Consider a maximal segment of consecutive pictures that is assigned a constant nominal quantization scale $Q_{(1)}$ by allocation sequence $\boldsymbol{s}$. Let $u$ be the index of the starting picture of such a segment. We consider two cases: $u = 1$ and $u > 1$. If $u = 1$, then $B_{\rm f}(\boldsymbol{s}, u) = B_{\rm f}(\boldsymbol{s}', u) = B_1$. If $u > 1$, then since $u$ is the index of the start of the segment, we have $g_{u-1}(s_{u-1}) < g_u(s_u)$, which implies that $B_{\rm f}(\boldsymbol{s}, u) = B_{\rm vbv}$ by Lemma 4.3; since $\boldsymbol{s}'$ is a legal allocation sequence, we have $B_{\rm f}(\boldsymbol{s}', u) \le B_{\rm vbv}$. In either case we have

$$B_{\rm f}(\boldsymbol{s}, u) \ge B_{\rm f}(\boldsymbol{s}', u). \tag{4.4}$$

Let $v$ be the index of the end of the segment. We consider two cases: $v = N$ and $v < N$. If $v = N$, then by the contrapositive of Lemma 4.2, $B_{\rm f}(\boldsymbol{s}, v) = s_v$. (Here we use the condition that $Q_{(1)} > Q_{\rm min}$.) If $v < N$, then since $v$ is the index of the end of the segment, we have $g_v(s_v) > g_{v+1}(s_{v+1})$, which implies that $B_{\rm f}(\boldsymbol{s}, v) = s_v$ by Lemma 4.3. In either case we have

$$B_{\rm f}(\boldsymbol{s}, v) = s_v. \tag{4.5}$$

From Lemma 4.4, we have $\boldsymbol{s} \succ \boldsymbol{s}'$. Therefore, we have $g_j(s'_j) \le Q_{(1)}$ for all $j$ and thus

$$s_j \le s'_j, \qquad \text{for } u \le j \le v. \tag{4.6}$$

From (4.4) and (4.6) we have

$$B_{\mathrm{f}}(\boldsymbol{s}, j) \geq B_{\mathrm{f}}(\boldsymbol{s'}, j), \qquad \text{for } u \leq j \leq v. \tag{4.7}$$

Since $\boldsymbol{s'}$ is a legal allocation sequence, we have

$$B_{\mathrm{f}}(\boldsymbol{s'}, v) \geq s'_v \geq s_v = B_{\mathrm{f}}(\boldsymbol{s}, v). \tag{4.8}$$

Combining (4.7) and (4.8), we have $B_{\mathrm{f}}(\boldsymbol{s}, v) = B_{\mathrm{f}}(\boldsymbol{s'}, v)$ and $s_v = s'_v$. Therefore, $B_{\mathrm{f}}(\boldsymbol{s}, v+1) = B_{\mathrm{f}}(\boldsymbol{s'}, v+1)$. Since $Q_{(1)} > Q_{\min}$, by the contrapositive of Lemma 4.2, we see that the buffer fullness for pictures $u$ to $v$ is updated as with CBR operation. Therefore, we can use the results of Lemma 3.3, which implies that $B_{\mathrm{f}}(\boldsymbol{s}_j) = B_{\mathrm{f}}(\boldsymbol{s'}, j)$ and $s_j = s'_j$ for $u \leq j \leq v$, and thus the claim is true for $k = 1$.

Our inductive hypothesis is that the claim is true for $k < m$. We need to show that it is also true for $k = m$, assuming that $Q_{(m)} > Q_{\min}$. Consider a segment of consecutive pictures that are assigned nominal quantization scale $Q^{(m)}$. Let $u$ be the index of the start of the segment and $v$ the index of the end of the segment. We consider all cases for the segment boundaries.

For the left segment boundary we consider three cases: $u = 1$, $g_{u-1}(s_{u-1}) > g_u(s_u)$, and $g_{u-1}(s_{u-1}) < g_u(s_u)$. If $u = 1$, then $B_{\mathrm{f}}(\boldsymbol{s}, u) = B_{\mathrm{f}}(\boldsymbol{s'}, u) = B_1$. If $g_{u-1}(s_{u-1}) > g_u(s_u)$, then from the inductive hypothesis, we know that $B_{\mathrm{f}}(\boldsymbol{s}, u-1) = B_{\mathrm{f}}(\boldsymbol{s'}, u-1)$ and $s_{u-1} = s'_{u-1}$; therefore $B_{\mathrm{f}}(\boldsymbol{s}, u) = B_{\mathrm{f}}(\boldsymbol{s'}, u)$. If $g_{u-1}(s_{u-1}) < g_u(s_u)$, then from Lemma 4.3, we have $B_{\mathrm{f}}(\boldsymbol{s}, u) = B_{\mathrm{vbv}}$; since $\boldsymbol{s'}$ is a legal allocation sequence, we have $B_{\mathrm{f}}(\boldsymbol{s'}, u) \leq B_{\mathrm{vbv}} = B_{\mathrm{f}}(\boldsymbol{s}, u)$. For all three cases we have

$$B_{\mathrm{f}}(\boldsymbol{s'}, u) \leq B_{\mathrm{f}}(\boldsymbol{s}, u). \tag{4.9}$$

For the right segment boundary we consider three cases: $v = N$, $g_v(s_v) > g_{v+1}(s_{v+1})$, and $g_v(s_v) < g_{v+1}(s_{v+1})$. If $v = N$, then by the contrapositive of Lemma 4.2, $B_{\mathrm{f}}(\boldsymbol{s}, v) = s_v$. (We use the condition that $Q_{(m)} \neq Q_{\min}$.) If $g_v(s_v) > g_{v+1}(s_{v+1})$, then by Lemma 4.3, we have $B_{\mathrm{f}}(\boldsymbol{s}, v) = s_v$. If $g_v(s_v) < g_{v+1}(s_{v+1})$, then from the inductive hypothesis, we get $B_{\mathrm{f}}(\boldsymbol{s}, v+1) = B_{\mathrm{f}}(\boldsymbol{s'}, v+1)$. For the first two cases, we have

$$B_{\mathrm{f}}(\boldsymbol{s}, v) = s_v. \tag{4.10}$$

From Lemma 4.4, it follows that $\boldsymbol{s} \succ \boldsymbol{s'}$. Therefore, $g_j(s'_j) \leq Q_{(m)}$ for $u \leq j \leq v$ and thus

$$s_j \leq s'_j, \qquad \text{for } u \leq j \leq v. \tag{4.11}$$

From (4.9) and (4.11) we have

$$B_{\mathrm{f}}(\boldsymbol{s}, j) \geq B_{\mathrm{f}}(\boldsymbol{s'}, j), \qquad \text{for } u \leq j \leq v. \tag{4.12}$$

By (4.10), (4.11), and the fact that $\boldsymbol{s'}$ is a legal allocation sequence, we get

$$B_{\mathrm{f}}(\boldsymbol{s'}, v) \geq s'_v \geq s_v = B_{\mathrm{f}}(\boldsymbol{s}, v). \tag{4.13}$$

Combining (4.12) and (4.13), we have $B_{\rm f}(\boldsymbol{s}, v) = B_{\rm f}(\boldsymbol{s'}, v)$ and $s_v = s'_v$. Therefore, $B_{\rm f}(\boldsymbol{s}, v+1) = B_{\rm f}(\boldsymbol{s'}, v+1)$. So for all three cases for $v$, we have $B_{\rm f}(\boldsymbol{s}, v+1) = B_{\rm f}(\boldsymbol{s'}, v+1)$.

Since $Q_{(m)} > Q_{\min}$, by the contrapositive of Lemma 4.2, we see that the buffer fullness for pictures $u$ to $v$ is updated in the same way as with CBR operation. Therefore, we can use the results of Lemma 3.3, which implies that $B_{\rm f}(\boldsymbol{s}, j) = B_{\rm f}(\boldsymbol{s'}, j)$ and $s_j = s'_j$ for $u \le j \le v$. And thus the claim is true for $k = m$, which finishes the proof by induction. ∎

**Lemma 4.6** *Consider two VBR problems* $P = \langle N, F, B_{\rm tgt}, B_{\rm vbv}, B_1, B_{\rm inc}\rangle$ *and* $P' = \langle N, F, B'_{\rm tgt}, B_{\rm vbv}, B_1, B_{\rm inc}\rangle$ *with* $B_{\rm tgt} < B'_{\rm tgt}$*, and let their optimal allocation sequences be* $\boldsymbol{s}$ *and* $\boldsymbol{s'}$*, respectively. It follows that*

$$\min_{1\le j\le N}\left\{g_j(s_j)\right\} > \min_{1\le j\le N}\left\{g_j(s'_j)\right\}.$$

*Proof*: Let $Q_{\min} = \min_j\left\{g_j(s_j)\right\}$ and $Q'_{\min} = \min_j\left\{g_j(s'_j)\right\}$. From Lemma 4.4 we have $\boldsymbol{s} \succ \boldsymbol{s'}$. From Lemma 4.5, the only pictures that can be assigned a different $Q$ by $\boldsymbol{s}$ and $\boldsymbol{s'}$ are those that are assigned nominal quantization scale $Q_{\min}$ by $\boldsymbol{s}$. But $\boldsymbol{s} \succ \boldsymbol{s'}$ which implies that $s'$ must assign to some picture a nominal quantization scale lower than $Q_{\min}$. Therefore, $Q_{\min} > Q'_{\min}$. ∎

We summarize Lemmas 4.4, 4.5, and 4.6 with the following theorem.

**Theorem 4.2** *Consider two VBR problems* $P = \langle N, F, B_{\rm tgt}, B_{\rm vbv}, B_1, B_{\rm inc}\rangle$ *and* $P' = \langle N, F, B'_{\rm tgt}, B_{\rm vbv}, B_1, B_{\rm inc}\rangle$ *with* $B_{\rm tgt} < B'_{\rm tgt}$*, and let their optimal allocation sequences be* $\boldsymbol{s}$ *and* $\boldsymbol{s'}$*, respectively. It follows that*

1. $\boldsymbol{s} \succ \boldsymbol{s'}$,
2. $s_j = s'_j$ *for all* $1 \le j \le N$ *such that* $g_j(s_j) > \min_{1\le k\le N}\left\{g_k(s_k)\right\}$*, and*
3. $\min_{1\le j\le N}\left\{g_j(s_j)\right\} > \min_{1\le j\le N}\left\{g_j(s'_j)\right\}$.

## 4.2 VBR ALLOCATION ALGORITHM

Theorems 4.1 and 4.2 give us a way to find the optimal allocation sequence for a given VBR allocation problem. If we know the minimum nominal quantization scale $Q$ present in the optimal allocation sequence, then it is easy to find the optimal allocation sequence. However, in general, we do not know the minimum $Q$. Theorem 4.2 gives us an iterative way to find it.

### 4.2.1 VBR Algorithm

Here we sketch an iterative algorithm for computing a VBR allocation.

1. Mark all pictures as *easy*. Let $B_{\text{easy}} \leftarrow B_{\text{tgt}}$.
2. Allocate $B_{\text{easy}}$ bits collectively to the easy pictures using a constant nominal quantizer. Let $Q_{\min}$ be the nominal quantizer used.
3. Simulate the VBV in VBR mode to identify *hard* segments of pictures. A hard segment is one that leads to a buffer underflow when $Q_{\min}$ is used. It consists of pictures that follow the most recent virtual overflow up to and including the picture the caused the underflow. After identifying a hard segment, reduce the bit allocation to the picture that caused the underflow to just prevent underflow. Reset the buffer fullness to empty and continue the simulation, adding new pictures to the existing hard segment if the buffer continues to underflow.
4. Allocate bits to each newly identified hard segment according to the optimal CBR algorithm, with a bit budget such that the underflow is just prevented. By preventing underflow in the hard segments, we are left with extra unallocated bits.
5. Let $B_{\text{hard}}$ be the total number of bits allocated to the hard pictures. Let $B_{\text{easy}} \leftarrow B_{\text{tgt}} - B_{\text{hard}}$.
6. If a new hard segment was identified in the most recent execution of Step 3, go to Step 2.

### 4.2.2 Correctness of VBR Algorithm

We now prove that the VBR algorithm computes a lexicographically optimal allocation sequence by showing that the resulting allocation satisfies the switching conditions of Theorem 4.1.

First, we make several observations about the VBR algorithm.

1. Pictures marked "easy" are assigned the same value of $Q$,
2. "Hard" pictures are marked in segments that start either at the beginning of the video sequence or with the buffer full and that end with the buffer empty.
3. Segments of hard pictures are allocated using the CBR algorithm.

The correctness of the CBR algorithm insures that within hard segments conditions 3 and 4 of Theorem 4.1 hold. In order to show that the other conditions also hold, we first need to show that the CBR algorithm does not assign a $Q$ lower than the $Q_{\min}$ computed in Step 2.

**Lemma 4.7** *Let $\mathbf{s}$ be an allocation sequence computed by the VBR algorithm. Let $i$ and $j$ denote the indices of the beginning and end, respectively, of a hard segment as identified in Step 3. Then*

$$\min_{i \le k \le j} \{g_k(s_k)\} \ge Q_{\min}.$$

*Proof*: Let $\boldsymbol{s'}$ be an allocation sequence that is the same as $\boldsymbol{s}$ except for pictures $i$ to $j$, where $\boldsymbol{s'}$ uses $Q_{\min}$. Thus, in a VBV simulation using $\boldsymbol{s'}$ for pictures $i$ to $j$, $\boldsymbol{s'}$ does not cause a virtual overflow and underflows only at picture $j$. Let $u$ and $v$ mark the beginning and end, respectively, of a segment with the minimum $Q$ in the CBR allocation sequence for pictures $i$ to $j$. We consider two cases for $u$: $u = i$ and $u > i$. If $u = i$, then we have $B_{\rm f}(\boldsymbol{s}, u) = B_{\rm f}(\boldsymbol{s'}, u)$ since $s_k = s'_k$ for $k < i$. If $u > i$, then since $u$ marks the beginning of a segment with minimum $Q$ in the CBR allocation sequence for pictures $i$ to $j$, from Theorem 3.1, $B_{\rm f}(\boldsymbol{s}, u-1) = s_{u-1}$. This implies that $B_{\rm f}(\boldsymbol{s}, u) = B_{\rm inc}(u-1)$. Since $\boldsymbol{s'}$ does not cause an underflow for picture $u-1$, $B_{\rm f}(\boldsymbol{s'}, u-1) \geq s'_{u-1}$, which implies that $B_{\rm f}(\boldsymbol{s'}, u) \geq B_{\rm inc}(u-1)$. In either case, we have

$$B_{\rm f}(\boldsymbol{s'}, u) \geq B_{\rm f}(\boldsymbol{s}, u). \tag{4.14}$$

We consider two cases for $v$: $v = j$ and $v < j$. If $v = j$, then $B_{\rm f}(\boldsymbol{s'}, v) < s'_v$ since an underflow occurs at picture $j$. Thus $B_{\rm f}(\boldsymbol{s'}, v+1) < B_{\rm inc}(v)$. But since $\boldsymbol{s}$ is a legal allocation sequence, $B_{\rm f}(\boldsymbol{s}, v+1) \geq B_{\rm inc}(v)$. If $v < j$, then since $v$ marks the end of a segment with minimum $Q$ in the CBR allocation sequence for pictures $i$ to $j$, from Theorem 3.1, $B_{\rm f}(\boldsymbol{s}, v+1) = B_{\rm vbv}$. Since $\boldsymbol{s'}$ does not cause virtual overflow, $B_{\rm f}(\boldsymbol{s'}, v+1) \leq B_{\rm vbv}$. In either case,

$$B_{\rm f}(\boldsymbol{s'}, v+1) \leq B_{\rm f}(\boldsymbol{s}, v+1). \tag{4.15}$$

Expanding for $B_{\rm f}(\boldsymbol{s}, u)$ and $B_{\rm f}(\boldsymbol{s}, v+1)$ we have

$$B_{\rm f}(\boldsymbol{s}, u) = B_1 + \sum_{k=1}^{u-1} B_{\rm inc}(k) - \sum_{k=1}^{u-1} s_k, \tag{4.16}$$

$$B_{\rm f}(\boldsymbol{s}, v+1) = B_1 + \sum_{k=1}^{v} B_{\rm inc}(k) - \sum_{k=1}^{v} s_k. \tag{4.17}$$

Subtracting (4.17) from (4.16), canceling like terms, and rearranging, we have

$$\sum_{k=u}^{v} s_k = \sum_{k=u}^{v} B_{\rm inc}(k) + B_{\rm f}(\boldsymbol{s}, u) - B_{\rm f}(\boldsymbol{s}, v+1). \tag{4.18}$$

The same manipulations with $B_{\rm f}(\boldsymbol{s'}, u)$ and $B_{\rm f}(\boldsymbol{s'}, v+1)$ yield

$$\sum_{k=u}^{v} s'_k = \sum_{k=u}^{v} B_{\rm inc}(k) + B_{\rm f}(\boldsymbol{s'}, u) - B_{\rm f}(\boldsymbol{s'}, v+1). \tag{4.19}$$

Combining (4.14), (4.15), (4.18), and (4.19) we have

$$\sum_{k=u}^{v} s_k \leq \sum_{k=u}^{v} s'_k. \tag{4.20}$$

Pictures $u$ to $v$ use a constant $Q$ in both allocation sequences $\boldsymbol{s}$ and $\boldsymbol{s'}$, where $\boldsymbol{s}$ uses $Q = \min_{i \le k \le j} \{g_k(s_k)\}$ and $\boldsymbol{s'}$ uses $Q_{\min}$. Therefore, we have

$$F_{u,v}\Big(\min_{i \le k \le j} \{g_k(s_k)\}\Big) \le F_{u,v}(Q_{\min}). \tag{4.21}$$

Since $F_{u,v}$ is a monotonically decreasing function (see Section 3.2.3), we have

$$\min_{i \le k \le j} \{g_k(s_k)\} \ge Q_{\min}.$$

∎

From Lemma 4.7, we can conclude that after each iteration of the VBR algorithm, $Q_{\min}$ is indeed the minimum $Q$. Since hard segments do not include pictures that cause a virtual overflow and does not include the last picture if it does not cause a buffer underflow, conditions 1 and 2 of Theorem 4.1 also hold.

We are now ready to state the main result of this section.

**Theorem 4.3 (Correctness of VBR Algorithm)** *Each pass through the VBR algorithm results in an allocation sequence that is lexicographically optimal for the number of bits actually allocated.*

### 4.2.3 Time and Space Complexity

We note that the loop in the VBR algorithm terminates when no more hard segments are identified. This implies that the algorithm terminates after at most $N$ iterations, where $N$ is the number of pictures.

Assuming that $G_{i,j}$ can be evaluated in constant time, we have shown in Section 3.2.5 that the CBR algorithm operates in $O(N^2)$ time and uses $O(N)$ space. Not counting the executions of the CBR algorithm, each iteration of the VBR algorithm takes $O(N)$ time and space. Since at most $O(N)$ iterations are performed, the time complexity excluding the executions of the CBR algorithm is $O(N^2)$.

We can defer actually invoking the CBR algorithm in Step 4 of the VBR algorithm until the end. This would avoid invoking the CBR algorithm more than once for each hard picture. Let $M$ be the number of hard segments found by the VBR algorithm and $L_i$ be the size of the $i$th hard segment. The time consumed by execution of the CBR algorithm can be expressed as

$$T_{\mathrm{CBR}}(N) = \sum_{i=1}^{M} O(L_i^2) = O\Big(\sum_{i=1}^{M} L_i^2\Big). \tag{4.22}$$

Since $\sum_i L_i \le N$, we have $\sum_i L_i^2 \le \left(\sum_i L_i\right)^2 \le N^2$. Therefore, the time complexity of the VBR algorithm is $O(N^2)$. For cases where there are relatively few hard segments, computing an optimal VBR allocation sequence

will likely be faster, in practice, than computing a CBR allocation sequence. Furthermore, Theorem 4.3 guarantees that we can halt the VBR algorithm after any number of iterations and have an optimal allocation sequence. The decision to continue depends upon whether the achieved bit consumption is acceptable. With each iteration the number of bits allocated increases.

## 4.3 DISCUSSION

The above complexity analysis is performed in the context of off-line global optimization. The vast majority of CBR video coders in operation today work in real-time mode without the luxury of lookahead processing. Since VBR coders can potentially give better quality for the same bit budget, they are targeted for quality-sensitive applications (such as encoding a Hollywood movie) where expensive off-line processing is a viable option. However, the above analysis does allow for "one-pass" VBR encoding. By substituting a real-time CBR algorithm for the optimal one invoked by the VBR algorithm, we can construct a one-pass real-time VBR encoder. Though necessarily suboptimal, the resulting coder would have complexity comparable to existing CBR coders. An approach along these lines is discussed in Chapter 7.

# 5

# *Implementation of Lexicographic Bit Allocation*

In this chapter, we describe an implementation of rate control using the lexicographically optimal bit allocation algorithms presented in Chapters 3 and 4 within a publicly available software MPEG-2 encoder [56]. With this implementation, we have four aims: 1) to verify the effectiveness of lexicographic optimality, 2) to assess the practical implications of the assumptions made in the framework, namely, independent coding and continuous variables, 3) to explore various bit-production models, and 4) to develop robust techniques for recovering from errors with the approximate bit-production models.

## 5.1 PERCEPTUAL QUANTIZATION

For perceptual quantization, we use the TM5 adaptive quantization scheme (described in Section 1.6.5), where the nominal quantization scale is modulated by an activity factor that is computed from the spatial activity of the luminance blocks within a macroblock. In TM5, the actual quantization scale used for coding a particular macroblock is determined from an initially computed (global) reference quantization scale, a (local) feedback factor that is dependent of the state of a virtual encoding buffer, and the activity factor. For modeling purposes, we define the nominal quantization for a picture as the average of the product of the reference quantization scale and the buffer-feedback factor over all coded macroblocks.

## 5.2 BIT-PRODUCTION MODELING

The framework in Chapter 2 presumes the existence of an exact continuous bit-production model for each picture. In practice, the rate-distortion function of a complex encoding system, such as MPEG, cannot be determined exactly for nontrivial classes of input. Therefore, approximate models are used in practice.

As the complexity analyses in Sections 3.2.5 and 4.2.3 show, the running time for the optimal bit allocation algorithms depends upon the time to evaluate $G_{i,j}$, the function that is used to compute a constant-$Q$ allocation sequence. In practice, therefore, the chosen models should admit efficient computation of $G_{i,j}$. In this section, we examine three classes of models—hyperbolic, linear-spline, and hyperbolic-spline—for which $G_{i,j}$ can be efficiently computed.

### 5.2.1 Hyperbolic Model

In [76], the following simple *hyperbolic bit-production model* forms the basis of an adaptive bit allocation algorithm:

$$f_k(Q) = \frac{\alpha_k}{Q} + \beta_k; \tag{5.1}$$

$$F_{i,j}(Q) = \sum_{k=i}^{j} f_k(Q), \tag{5.2}$$

where we associate $\alpha_k > 0$ with the complexity of coding picture $k$ and $\beta_k > 0$ with the overhead for coding the picture. The hyperbolic model is one of the simplest models to exhibit the monotonicity and concavity characteristic of rate-distortion functions.[1] Several instances of the hyperbolic model are plotted in Figure 5.1. TM5 adopts a similar model where only the complexity term is used. With adaptive quantization techniques, $\alpha_k$ and $\beta_k$ are typically estimated from the results of encoding previous pictures. The parameters can also be determined by coding a sampling of blocks in picture $k$ and fitting the parameters to the coding statistics.

We saw in (3.35) that the hyperbolic model has a simple closed-form expression for the nominal quantization scale $G_{i,j}(b)$ that uses a total of $b$ bits for pictures $i$, $i+1$, ..., $j$:

$$G_{i,j}(b) = \frac{\sum_{k=i}^{j} \alpha_k}{b - \sum_{k=i}^{j} \beta_k}. \tag{5.3}$$

As previously discussed in Section 3.2.5, we can precompute the cumulative sums for $\alpha_k$ and $\beta_k$ in linear time and space and then use these to compute $G_{i,j}$

[1] Our framework only requires monotonicity and not concavity.

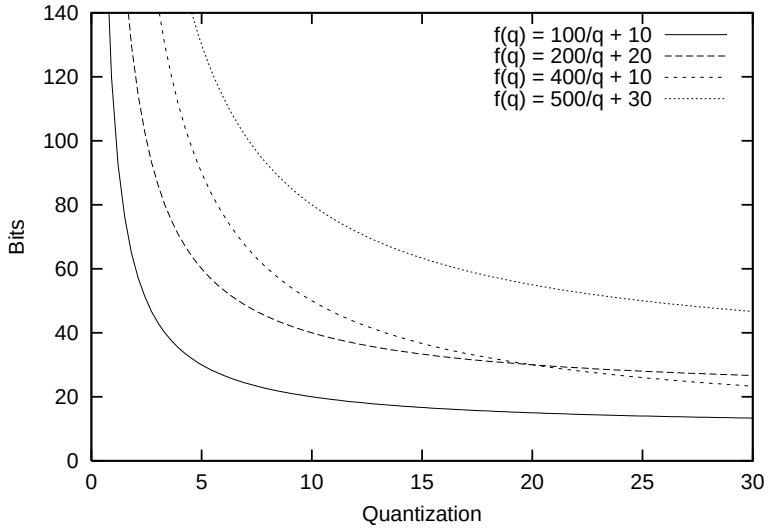



*Fig. 5.1* **Hyperbolic Bit-Production Models.** This figure shows several instances of a simple hyperbolic bit-production model.

in constant time. This results in a time complexity of $O(N^2)$ for both optimal CBR and VBR allocation.

In related work, Ding and Liu [16] propose the following more general class of bit-production models and describe its use in rate control:

$$f_k(Q) = \frac{\alpha_k}{Q^{\gamma_k}} + \beta_k. \tag{5.4}$$

The exponent parameter $\gamma_k$ is dependent on the picture type (I, P, or B) and is intended to capture the different rate-distortion characteristics for each picture type. One drawback to (5.4) is that the model is nonlinear with respect to the parameters, and we know of no closed-form solution to $G_{i,j}$ in the general setting. Although numerical techniques can be used to solve for $G_{i,j}$, this could adversely affect the computational efficiency of the bit allocation algorithms.

In preliminary experiments, we find that the hyperbolic model works well near the operating point where $\alpha_k$ and $\beta_k$ have been determined, but is not reliable at a distant operating point. This observation leads us to formulate the following encoding strategy.

1. Encode the sequence using the standard TM5 coder, keeping statistics (for each picture) on the average nominal quantization scale, the coding rate (number of bits used), and the number of bits used to code the quantized DCT coefficients.

2. Compute $\alpha_k$ and $\beta_k$ from the statistics gathered in the previous encoding pass. Allocate bits to pictures with the lexicographic bit allocation algorithm and encode the sequence using this allocation, gathering statistics as before.

3. Repeat Step 2.

The idea is that with each encoding, the accuracy of the bit models will improve as the operating $Q$ is determined and refined for each picture.

### 5.2.2 Linear-Spline Model

As noted above, the hyperbolic model works well with small changes in the nominal quantization scale $Q$. However, with a large variation in $Q$ between successive pictures, as may occur with a scene change, the model becomes less reliable. After all, the model is defined by only two parameters $\alpha_k$ and $\beta_k$. Previously, we have compensated for this limitation by performing multiple encoding passes to ensure that the parameters are determined close to the actual operating point. We now consider a different approach where more effort is expended to construct more accurate bit models that are then used to encode the video sequence in a single pass.

Lin, Ortega, and Kuo [47, 48] propose using cubic-spline interpolation models of rate and distortion in conjunction with a gradient-based rate control algorithm [45, 46]. The spline models are computed by first encoding each picture several times using a select set of $M$ quantization scales, $\{q_1, q_2, \ldots, q_M\}$ with $q_1 < q_2 < \cdots < q_M$, and measuring the coding rate (actual number of bits used). Each tuple consisting of a quantization scale and the coding rate is called a *control point*. For picture $i$, the function between two consecutive control points $(q_m, b_{k,m})$ and $(q_{m+1}, b_{k,m+1})$ has the form

$$f_k^m(Q) = a_{k,m}Q^3 + b_{k,m}Q^2 + c_{k,m}Q + d_{k,m}. \tag{5.5}$$

The real-valued parameters $a_{k,m}$, $b_{k,m}$, $c_{k,m}$, and $d_{k,m}$ are computed from four control points $(q_{m-1}, b_{k,m-1})$, $(q_m, b_{k,m})$, $(q_{m+1}, b_{k,m+1})$, and $(q_{m+2}, b_{k,m+2})$, such that $f_k^m(q_m) = b_{k,m}$ and $f_k^{m+1}(q_{m+1}) = b_{k,m+1}$ and the first-order derivatives of $f_k^m$ and $f_k^{m+1}$ are continuous at the control points. The authors suggest using the Fibonacci-like set $\{1, 2, 3, 5, 8, 13, 21, 31\}$ for the control quantization scales to exploit the exponential decay typical of rate-distortion functions.

One drawback of a cubic-spline model is that it is generally not monotonic. To ensure monotonicity, we consider a simpler *linear-spline interpolation bit-production model*, where a line segment is used to interpolate the bit-production function between control points. For picture $k$, the function between two consecutive control points $(q_m, b_{k,m})$ and $(q_{m+1}, b_{k,m+1})$ has the form

$$f_k^m(Q) = \alpha_{k,m}Q + \beta_{k,m}. \tag{5.6}$$

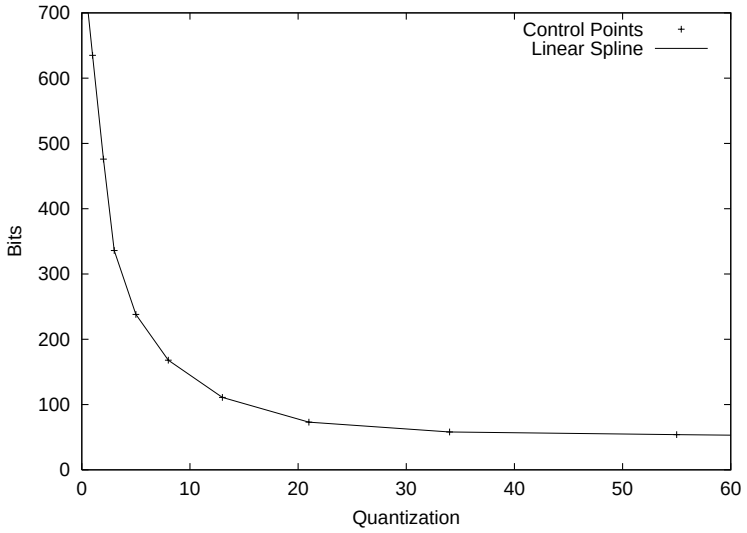



*Fig. 5.2* **Example of a Linear-Spline Interpolation Model.** This figure shows how line segments are used to interpolate between control points.

We choose the control quantization scales to be $\{1, 2, 3, 5, 8, 13, 21, 34, 55\}$ to exploit the exponential-decay property of rate-distortion functions. In case the control points themselves do not exhibit monotonicity, we enforce monotonicity by skipping those control points where the monotonicity property is violated. For nominal quantization scales less than $q_1$ or greater than $q_M$, we extrapolate using the parameters $\alpha_{k,1}$, $\beta_{k,1}$ or $\alpha_{k,M-1}$, $\beta_{k,M-1}$, respectively. An example of a linear-spline model is shown in Figure 5.2.

The linear-spline model gives a simple closed-form expression for the nominal quantization scale $Q = G_{i,j}(B)$ if we know the two control points that bracket the resulting operating point $Q$. Between the control points $q_m$ and $q_{m+1}$, we have from (5.9)

$$F_{i,j}(Q) = \sum_{k=i}^{j} f_k^m(Q) \tag{5.7}$$

$$= Q \sum_{k=i}^{j} \alpha_{k,m} + \sum_{k=i}^{j} \beta_{k,m}; \tag{5.8}$$

$$Q = \frac{F_{i,j}(Q) - \sum_{k=i}^{j} \beta_{k,m}}{\sum_{k=i}^{j} \alpha_{k,m}}. \tag{5.9}$$

Thus, the nominal quantization scale $Q$ can be computed as

$$Q = \frac{B - \sum_{k=i}^{j} \beta_{k,m}}{\sum_{k=i}^{j} \alpha_{k,m}}. \tag{5.10}$$

Because of the monotonicity property, we can determine the correct two bracketing control points using binary search. For example, suppose we start with an arbitrary value of $m$ and compute $Q$ using (5.10). If $q_m \leq Q \leq q_{m+1}$ then the correct operating point $Q$ has been found. If $Q < q_m$, the operating point must lie between two control points with lower indices. Similarly, if $Q > q_{m+1}$, the operating point must lie between two control points with higher indices. A simple binary search procedure on $m$ can be used to find the correct operating point $Q$.

Since there are a fixed number of control points, we can compute $G_{i,j}(B)$ in constant time with linear-time preprocessing. As with the hyperbolic model, we can compute optimal CBR and VBR allocation sequences in quadratic time.

The cubic-spline model of [47, 48] is used in a dependent-coding framework, where the effects of coding previous pictures are taken into account in the modeling. Our framework assumes independent coding and does not take these effects into account. However, from the switching theorems, we note that an optimal allocation sequence has segments of constant $Q$. This provides a basis for estimating the linear-spline model parameters. By encoding the video sequence multiple times with a constant $Q$ determined from the control points, we can construct a linear-spline interpolation model for each picture. We expect these models to be reasonably accurate within a segment of constant $Q$. At the boundary between segments, however, we can expect some discrepancy in the models for dependent pictures (P and B types).

### 5.2.3 Hyperbolic-Spline Model

In preliminary simulations, we found that the linear-spline model gave consistently better results than the simple hyperbolic model. This improvement is not surprising since the linear-spline model has more parameters and can better approximate a picture's rate-distortion characteristics over a wider range of quantization values. The hyperbolic model, on the other hand, better matches the rate-distortion characteristics locally. These observations suggest that we can construct a more accurate *hyperbolic-spline interpolation bit-production model* by combining the two, namely, by replacing the linear interpolation in the spline model with hyperbolic interpolation. Instead of (5.6), we use

$$f_k^m(Q) = \frac{\alpha_{k,m}}{Q} + \beta_{k,m}, \tag{5.11}$$

to interpolate for $Q$ within the range $q_m \leq Q < q_{m+1}$. A comparison between linear-spline interpolation and hyperbolic-spline interpolation using the same control points is shown in Figure 5.3. As with the linear-spline model, computing a constant $Q$ still takes constant time with linear-time preprocessing.

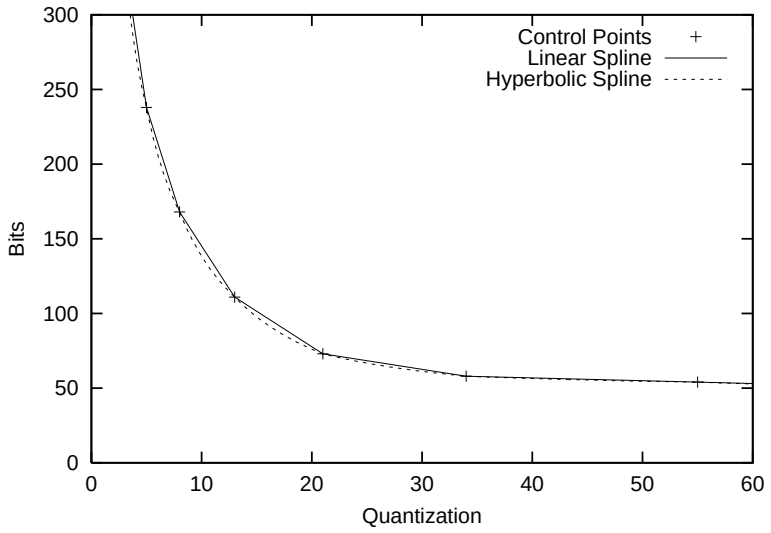



*Fig. 5.3* **Linear-Spline and Hyperbolic-Spline Models.** This figure illustrates the interpolation of points between control points using linear splines and hyperbolic splines.

## 5.3 PICTURE-LEVEL RATE CONTROL

A bit allocation specifies the number of bits and the corresponding nominal quantization to be used to code each picture. Even with accurate bit-production models, the actual number of bits produced will inevitably depart from the model. When coding a given picture, we can either use the specified nominal quantization to code the entire picture, or we can control the encoding to meet the specified bit allocation. These two approaches are often referred to as *open-loop* control and *closed-loop* control, respectively.

### 5.3.1 Closed-Loop Rate Control

A popular approach taken in TM5 is to regulate the quantization scale at the macroblock level while coding a picture so that the desired bit allocation is met. This is achieved with a *closed-loop* feedback mechanism using the fullness of a virtual encoder buffer to control the macroblock quantization. One drawback of this technique is that the coded quality within a picture may vary considerably, especially for a picture that contains regions of varying complexity. With gross errors in the bit-production models, the actual average quantization scale may differ markedly from the desired quantization scale, thereby adversely affecting the coded quality.

### 5.3.2 Open-Loop Rate Control

Another approach is to perform *open-loop* control where the assigned nominal quantization scale is used to code a picture. We can then adjust the bit allocation of the remaining uncoded pictures to compensate for the difference between desired and actual bit production. An advantage of this approach is that the quality is more constant within a picture. In addition, less processing is required to code each picture. A disadvantage is that, since the bit production is not controlled below the picture layer, the actual bit production may vary from the target and potentially cause the buffer to overflow or underflow.

After coding a picture, we can reallocate bits to the remaining pictures optimally (for the given models). Instead of recomputing an optimal allocation sequence from scratch and incurring an extra factor of $N$ in the time complexity, we can take advantage of dynamic programming to increase the time complexity by only a constant factor. We do this for a CBR allocation and for hard pictures in a VBR allocation by constructing the dynamic programming table in the CBR algorithm *in reverse*.

As presented in Section 3.2, the dynamic programming algorithm works by solving for allocation sequences for pictures 1 to $j$ for increasing values of $j$. Alternatively we can rework the dynamic programming to compute optimal allocation sequences for pictures $j$ to $N$ for decreasing values of $j$. The final buffer state, which we denote by **Final**, corresponds by (2.11) to a buffer fullness level of $B_1 + \sum_{j=1}^{N-1} B_{\mathrm{inc}}(j) - B_{\mathrm{tgt}}$ bits. For each $1 < j \le N$, we redefine $\mathbf{Top}[j]$ to represent the legal allocation sequence for pictures $j$, $j+1$, $\ldots$, $N$ that *starts* with a *full* buffer immediately before picture $j$ is removed (i.e., $B_{\mathrm{f}}(\boldsymbol{s}^*, j) = B_{\mathrm{vbv}}$, or equivalently $B^*_{\mathrm{f}}(\boldsymbol{s}^*, j-1) = B_{\mathrm{vbv}} - B_{\mathrm{inc}}(j)$) and ends up in **Final**. We redefine $\mathbf{Bot}[j]$ to represent the legal allocation sequence for pictures $j$, $j+1$, $\ldots$, $N$ that *starts* with an *empty* buffer immediately after picture $j-1$ is removed (i.e., $B_{\mathrm{f}}(\boldsymbol{s}^*, j-1) = s^*_{j-1}$, or equivalently, $B^*_{\mathrm{f}}(\boldsymbol{s}^*, j-1) = 0$ or $B_{\mathrm{f}}(\boldsymbol{s}^*, j) = B_{\mathrm{inc}}(j)$) and ends up in **Final**. We redefine **Initial** to be the optimal allocation sequence $\boldsymbol{s}^*$ for all $N$ pictures that begins with initial buffer fullness level $B_1$ and ends up in **Final**.

With this setup, we can use a dynamic programming procedure to compute $\mathbf{Top}[j]$ and $\mathbf{Bot}[j]$ from $\{\mathbf{Top}[j+1], \mathbf{Top}[j+2], \ldots, \mathbf{Top}[N], \mathbf{Bot}[j+1], \mathbf{Bot}[j+2], \ldots, \mathbf{Bot}[N], \mathbf{Final}\}$. Similarly, we can compute the full allocation sequence **Initial** from $\{\mathbf{Top}[2], \mathbf{Top}[3], \ldots, \mathbf{Top}[N], \mathbf{Bot}[2], \mathbf{Bot}[3], \ldots, \mathbf{Bot}[N], \mathbf{Final}\}$. As with the forward approach, each step takes linear time, so the total running time is $O(N^2)$.

Once this reverse dynamic programming table is precomputed, we can compute a revised allocation for picture $j$, after encoding picture $j-1$, by searching for a proper constant-$Q$ segment that starts with the corrected VBV buffer fullness before picture $j$ is removed and that connects to one of the states $\{\mathbf{Top}[j+1], \mathbf{Top}[j+2], \ldots, \mathbf{Top}[N], \mathbf{Bot}[j+1], \mathbf{Bot}[j+2], \ldots, \mathbf{Bot}[N], \mathbf{Final}\}$. With the reverse dynamic programming table available, this search consumes $O(N)$ time for the hyperbolic, linear-spline, and hyperbolic-spline

bit-production models (or for any bit-production model that can be computed in constant time). The total time to recover from bit-production errors is thus $O(N)$ per picture, or $O(N^2)$ overall during the course of the entire video sequence, the same as the time complexity for computing the initial allocation.

In Chapter 6 we will show how to do the reverse preprocessing making use of a more sophisticated technique, based upon the Lee-Preparata [42] algorithm for finding shortest paths in polygonal channels. The preprocessing takes $O(N \log N)$ time. After picture $j-1$ is encoded, the recomputation needed to encode picture $j$ will take constant time in typical cases and never more than $O(\log N)$ time, so the total encoding time for the entire video sequence will be $O(N)$, in practice, and $O(N \log N)$ in the worst case. The total space usage remains linear.

The above procedures apply to a CBR allocation and to hard pictures in a VBR allocation (which are allocated using the CBR routine). For easy pictures in a VBR allocation, we can simply recompute a new value for $Q_{\min}$. Here, we assume that errors in bit-production modeling are not severe enough to change the classification of hard and easy pictures.

### 5.3.3 Hybrid Rate Control

In early experiments, we observed that closed-loop rate control resulted in rapid fluctuations in the nominal quantization scale[2] between pictures owing to the buffer-feedback mechanism. With accurate bit-production models, however, the need to perform low-level rate control below the picture level is questionable. This suggests using open-loop control. As noted earlier, since we assume independent coding, we can expect more errors in the bit-production models at pictures where the assigned $Q$ changes. With these observations, we propose a hybrid rate control strategy, where closed-loop control is used for pictures at the boundaries of a constant-$Q$ segment and open-loop control is used for the rest. Another motivation for using closed-loop control for boundary pictures is that the VBV buffer should be either nearly empty or nearly full for these pictures, and the bit rate must be carefully controlled to avoid underflowing or overflowing the buffer.

## 5.4 BUFFER GUARD ZONES

Even with the picture-level rate control strategies outlined above, there is still the possibility of the VBV buffer overflowing or underflowing. To safeguard against this, we compute a bit allocation sequence using a slightly smaller buffer than that specified in the MPEG bitstream so that we can have *guard*

[2]By nominal quantization scale, we mean the average measured macroblock quantization scale with perceptual quantization factored out.


*Fig. 5.4* **Guard Zones.** This figure illustrates the use of guard zones to safeguard against underflow and overflow of VBV buffer. A bit allocation is computed so that the buffer fullness remains between the guard zones.

*zones* near the top and bottom of the buffer. For the experiments with CBR, have we chosen to place the guard zones at 5% and 95% of maximum buffer size. This is illustrated in Figure 5.4. For VBR mode, the upper guard zone is not needed since buffer overflow is not a concern.

## 5.5 SOFTWARE SIMULATION ENVIRONMENT

We implemented the rate control algorithms within the software encoder provided by the MPEG Software Simulation Group (MSSG) [56]. The MSSG coder follows the TM5 coding model with minor differences. We made a further modification to the MSSG coder by restricting the fullness of the virtual encoding buffers used to regulate the macroblock-level quantization scale to the range $[0, 2r]$, where $r$ is the reaction parameter defined in Section 1.6.5. In addition, we added facilities to save and restore motion vectors and coding decisions, allowing us to test different rate control algorithms while keeping the motion vectors and coding decisions fixed.

## 5.6 INITIAL SIMULATIONS

In the initial set of simulations, we evaluate the effectiveness of the hyperbolic model with closed-loop rate control and the linear-spline model with hybrid rate control using the following short video clips: `flower garden`, `football`,

`mobile`, and `table tennis`. The clips are in SIF format ($352 \times 240$). In the `flower garden` clip, the camera pans over a bed of brightly colored flowers with a house in the background and a tree in the foreground. The `football` clip shows an action sequence from a football game, with high motion within the scene. The `mobile` clip contains a scene with a lot of picture detail, moving and rotating objects, and camera zoom. The `table tennis` clip contains some action and abrupt changes in the camera angle. To simulate scene changes, the four clips are concatenated into a 418–frame video sequence in the following order: `flower garden`, `mobile`, `football`, `table tennis`.

The coding parameters for the simulations are listed in Table 5.1. For CBR mode, we specify a constant bit rate of 1.0 Mbits/sec. For VBR, we use an average bit rate of 1.0 Mbits/sec and a peak bit rate of 1.2 Mbits/sec. The VBV buffer size is set to 720,896 bits. The MPEG-2 TM5 CBR rate control algorithm is used as a reference, using the same set of coding parameters. In order to reduce factors that would affect the actual bit production, full-search motion estimation is initially performed using a fixed nominal quantization scale of 13, and the same motion vectors are then used for all the encodings. The coding decisions, however, are still determined on-line.

In the first set of simulations, we used the hyperbolic model with closed-loop rate control and performed multiple encoding passes. The results of the encodings are presented in Table 5.2 and Figures 5.5 to 5.13. The table collects some summary statistics for the various coders. Figures 5.5 to 5.13 contain plots showing the buffer fullness, smoothed instantaneous bit rate, nominal quantization, and PSNR of the various coders.

The initial pass of the Hyperbolic CBR and VBR coders used statistics gathered with the TM5 coder in order to determine the parameters of the bit-production models. Later passes of the Hyperbolic CBR (VBR) coder used statistics gathered from the previous pass. From the results in Table 5.2 and Figures 5.7 to 5.9, the Hyperbolic CBR coder does not exhibit much change between passes. However, from Table 5.2 and Figures 5.11 to 5.13, the Hyperbolic VBR coder does show a reduction in the standard deviations of both PSNR and nominal $Q$ as well as better usage of the VBV buffer with later passes.

As evident from Figure 5.5, the TM5 coder uses only a fraction of the VBV buffer and maintains the buffer relatively level. In contrast, the lexicographic coders make better use of the VBV buffer.

Comparing hyperbolic modeling with closed-loop rate control on the one hand with linear-spline modeling with hybrid rate control on the other, we see that the latter outperforms the former in all aspects. It is noteworthy that the hyperbolic model seems to underestimate the bit production, whereas the linear-spline model overestimates the bit production. The result is that the nominal quantization scales used are higher than the target for the hyperbolic model and lower for the linear-spline model.

*Table 5.1* **Parameters for Initial Simulations.** This table lists the coding parameters used for the initial simulations.

| Value | Description |
|---|---|
| 418 | number of frames |
| 15 | N (# of frames in GOP) |
| 3 | M (I/P frame distance) |
| 0 | ISO/IEC 11172-2 stream |
| 0 | 0:frame pictures, 1:field pictures |
| 352 | horizontal_size |
| 240 | vertical_size |
| 2 | aspect_ratio: 1=square pel, 2=4:3, 3=16:9, 4=2.11:1 |
| 5 | frame_rate: 1=23.976, 2=24, 3=25, 4=29.97, 5=30 frames/sec |
| 1000000 | bit_rate (bits/sec) |
| 44 | vbv_buffer_size (in multiples of 16 kbit) |
| 0 | low_delay |
| 0 | constrained_parameters_flag |
| 4 | Profile ID: Simple=5, Main=4, SNR=3, Spatial=2, High=1 |
| 8 | Level ID: Low=10, Main=8, High 1440=6, High=4 |
| 1 | progressive_sequence |
| 1 | chroma_format: 1=4:2:0, 2=4:2:2, 3=4:4:4 |
| 2 | video_format: 0=comp,1=PAL,2=NTSC,3=SECAM,4=MAC,5=unspec |
| 352 | display_horizontal_size |
| 240 | display_vertical_size |
| 0 | intra_dc_precision: 0=8 bit, 1=9 bit, 2=10 bit, 3=11 bit |
| 0 | top_field_first |
| 1 1 1 | frame_pred_frame_dct (I P B) |
| 0 0 0 | concealment_motion_vectors (I P B) |
| 1 1 1 | q_scale_type (I P B) |
| 1 0 0 | intra_vlc_format (I P B) |
| 0 0 0 | alternate_scan (I P B) |
| 0 | repeat_first_field |
| 1 | progressive_frame |
| 0 | P distance between complete intra slice refresh |
| 0 | rate control: r (reaction parameter) |
| 0 | rate control: avg_act (initial average activity) |
| 0 | rate control: Xi (initial I frame global complexity) |
| 0 | rate control: Xp (initial P frame global complexity) |
| 0 | rate control: Xb (initial B frame global complexity) |
| 0 | rate control: d0i(initial I frame virtual buffer fullness) |
| 0 | rate control: d0p(initial P frame virtual buffer fullness) |
| 0 | rate control: d0b(initial B frame virtual buffer fullness) |
| 3 3 23 23 | P: forw_hor_f_code forw_vert_f_code search_width/height |
| 1 1 7 7 | B1: forw_hor_f_code forw_vert_f_code search_width/height |
| 2 2 15 15 | B1: back_hor_f_code back_vert_f_code search_width/height |
| 2 2 15 15 | B2: forw_hor_f_code forw_vert_f_code search_width/height |
| 1 1 7 7 | B2: back_hor_f_code back_vert_f_code search_width/height |

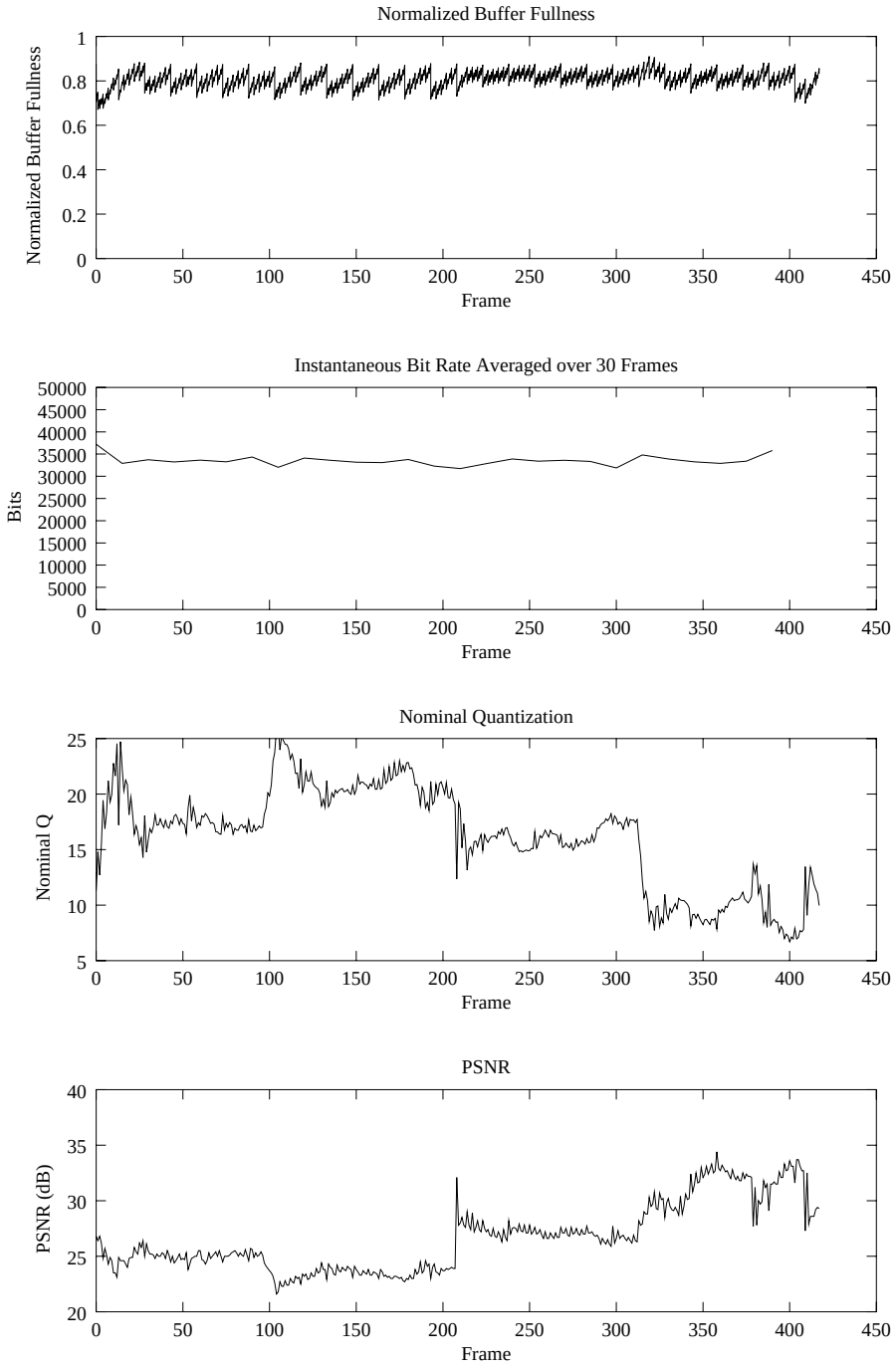



*Fig. 5.5* **Initial Simulation Results for TM5 CBR Coder.**

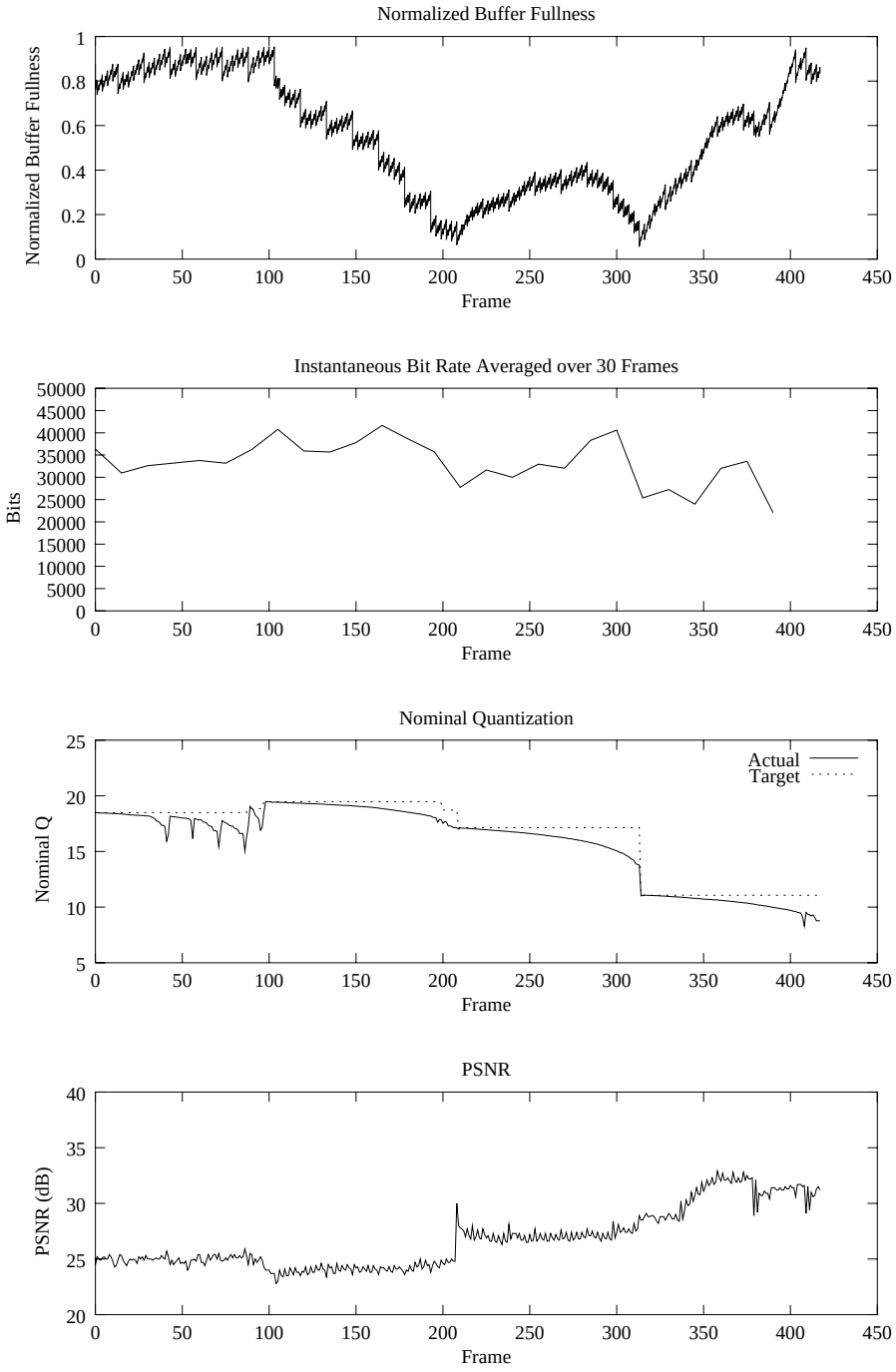



*Fig. 5.6* **Initial Simulation Results for Linear-Spline CBR Coder.**

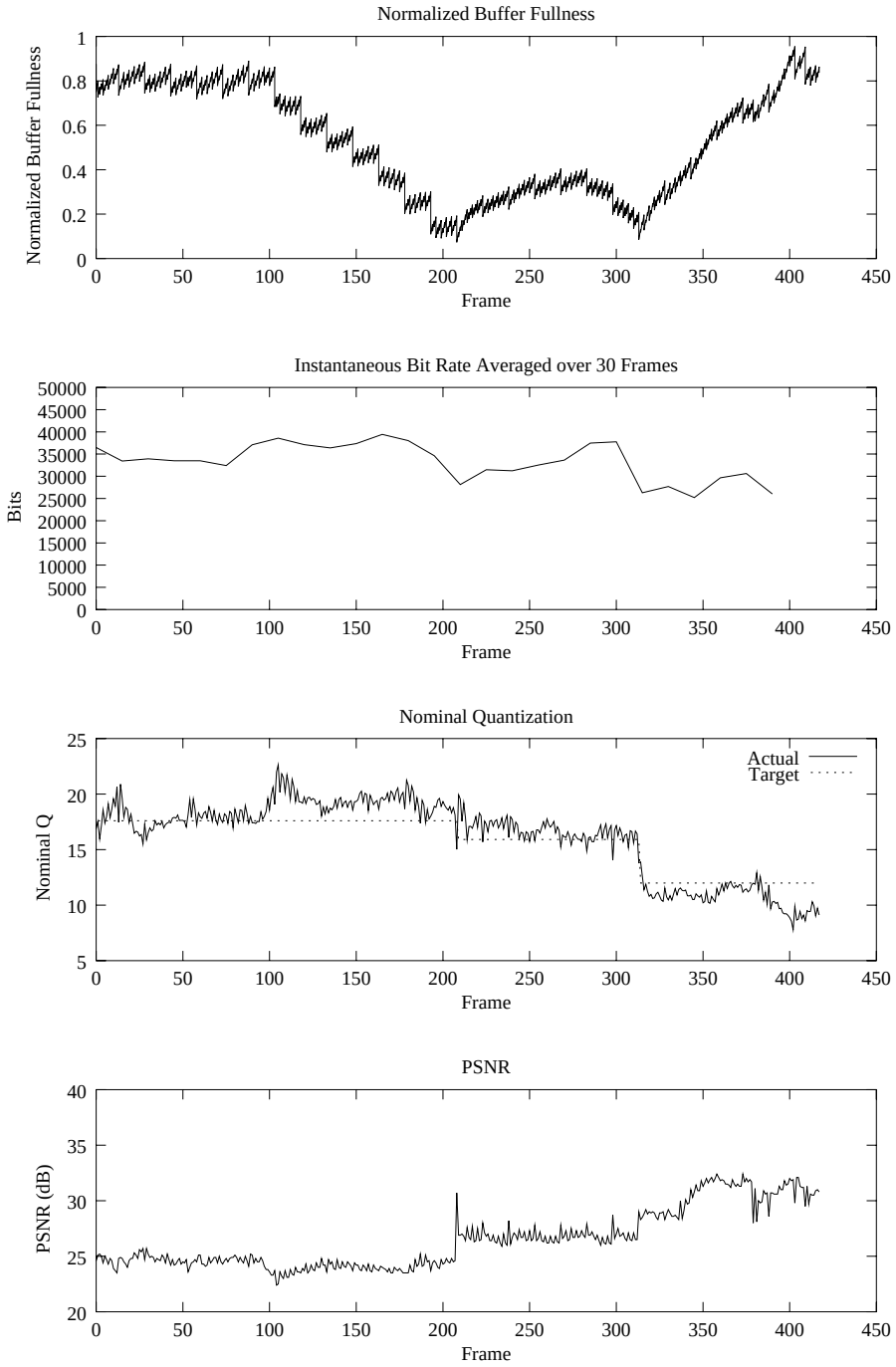



*Fig. 5.7* **Initial Simulation Results for Pass 1 of Hyperbolic CBR Coder.**

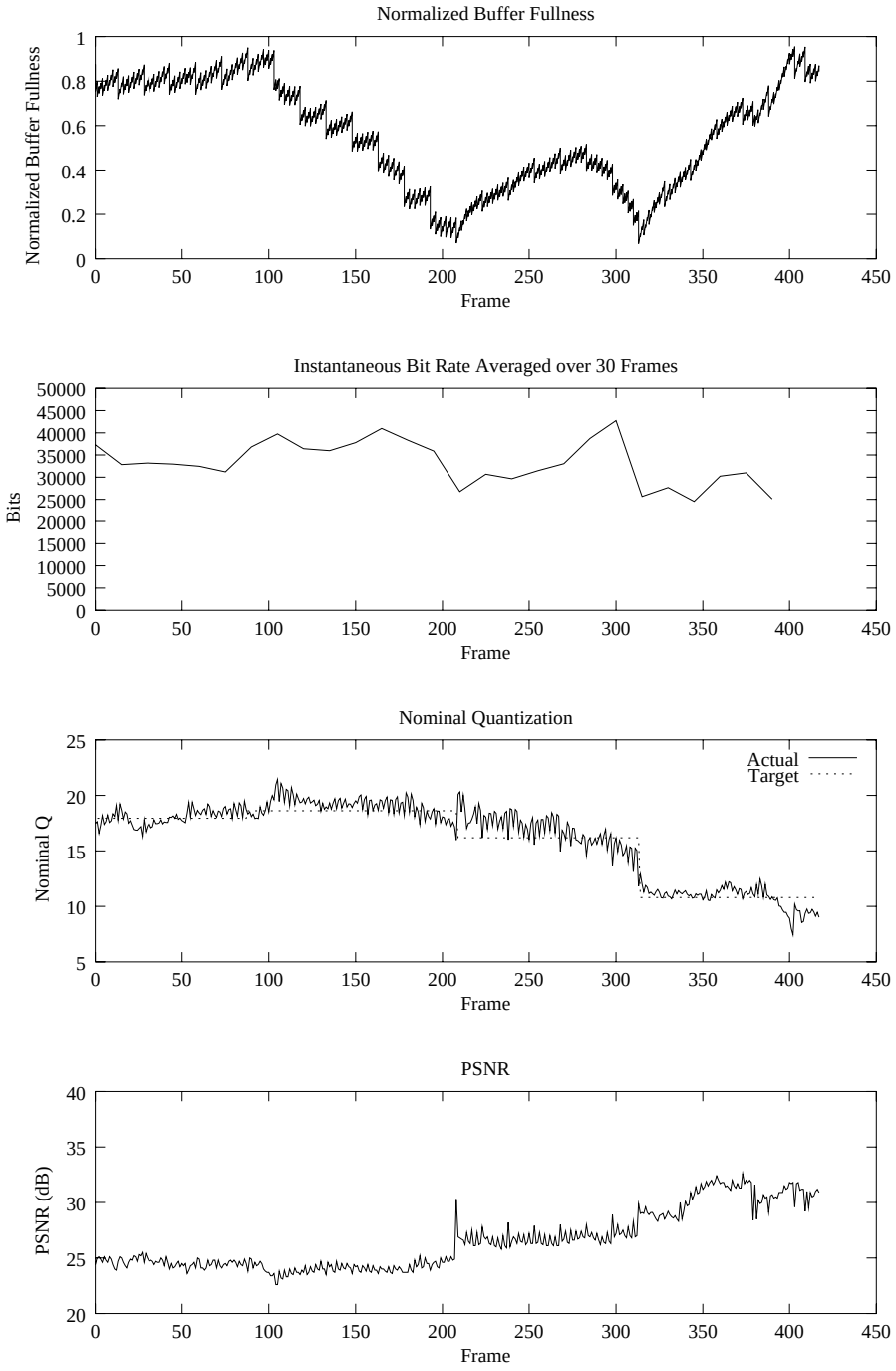



*Fig. 5.8* **Initial Simulation Results for Pass 2 of Hyperbolic CBR Coder.**

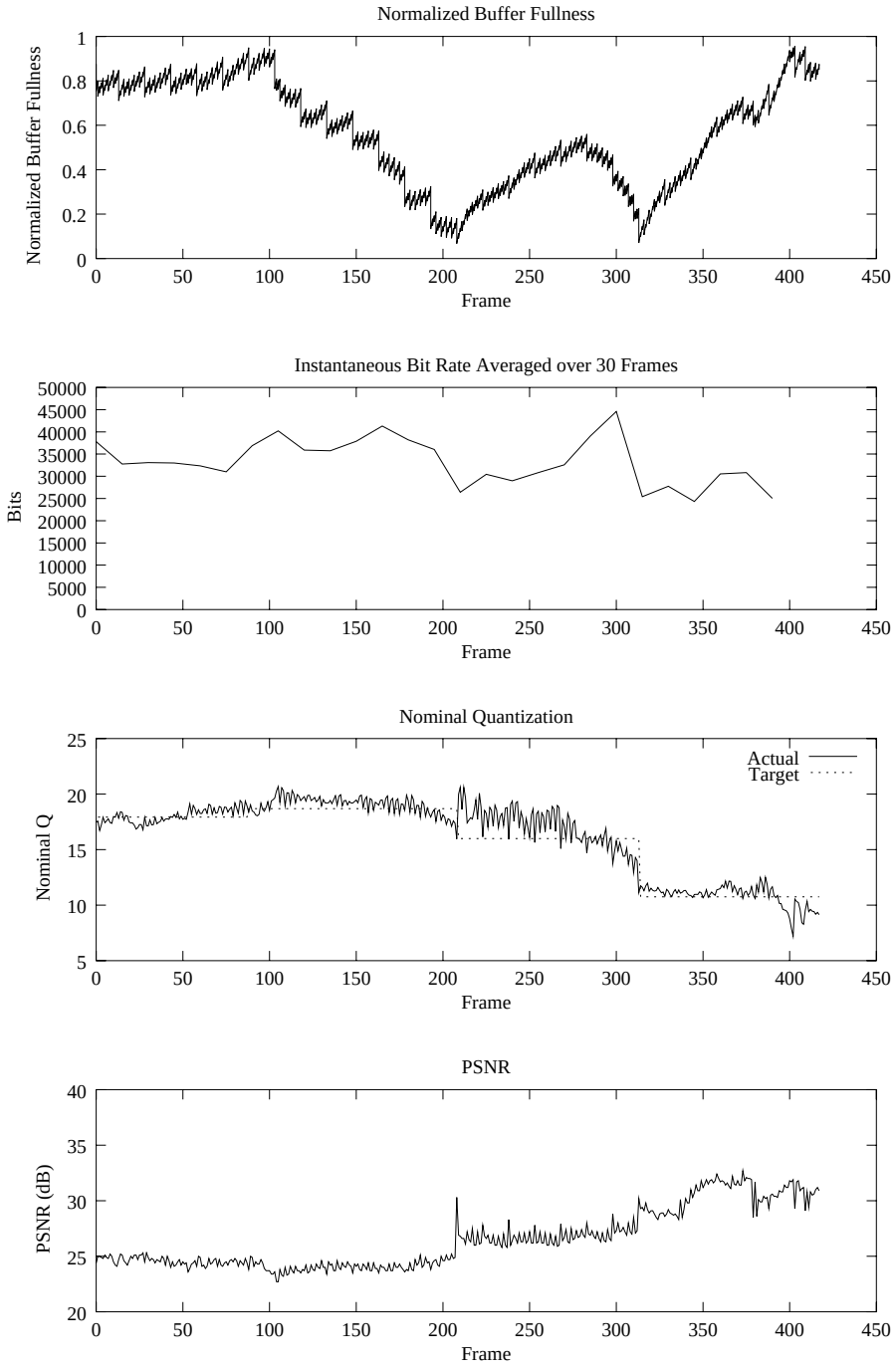



*Fig. 5.9* **Initial Simulation Results for Pass 3 of Hyperbolic CBR Coder.**

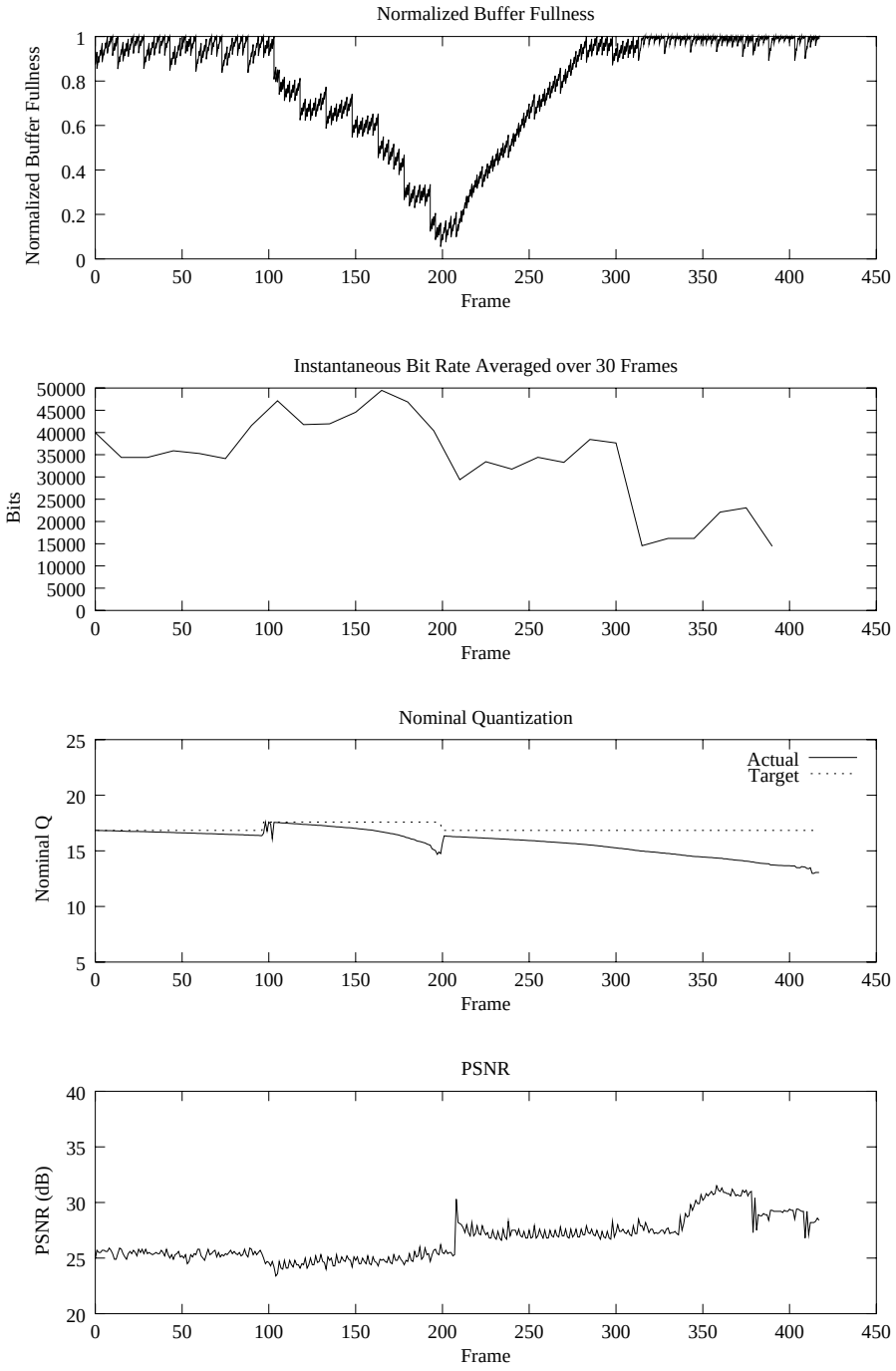



*Fig. 5.10* **Initial Simulation Results for Linear-Spline VBR Coder.**

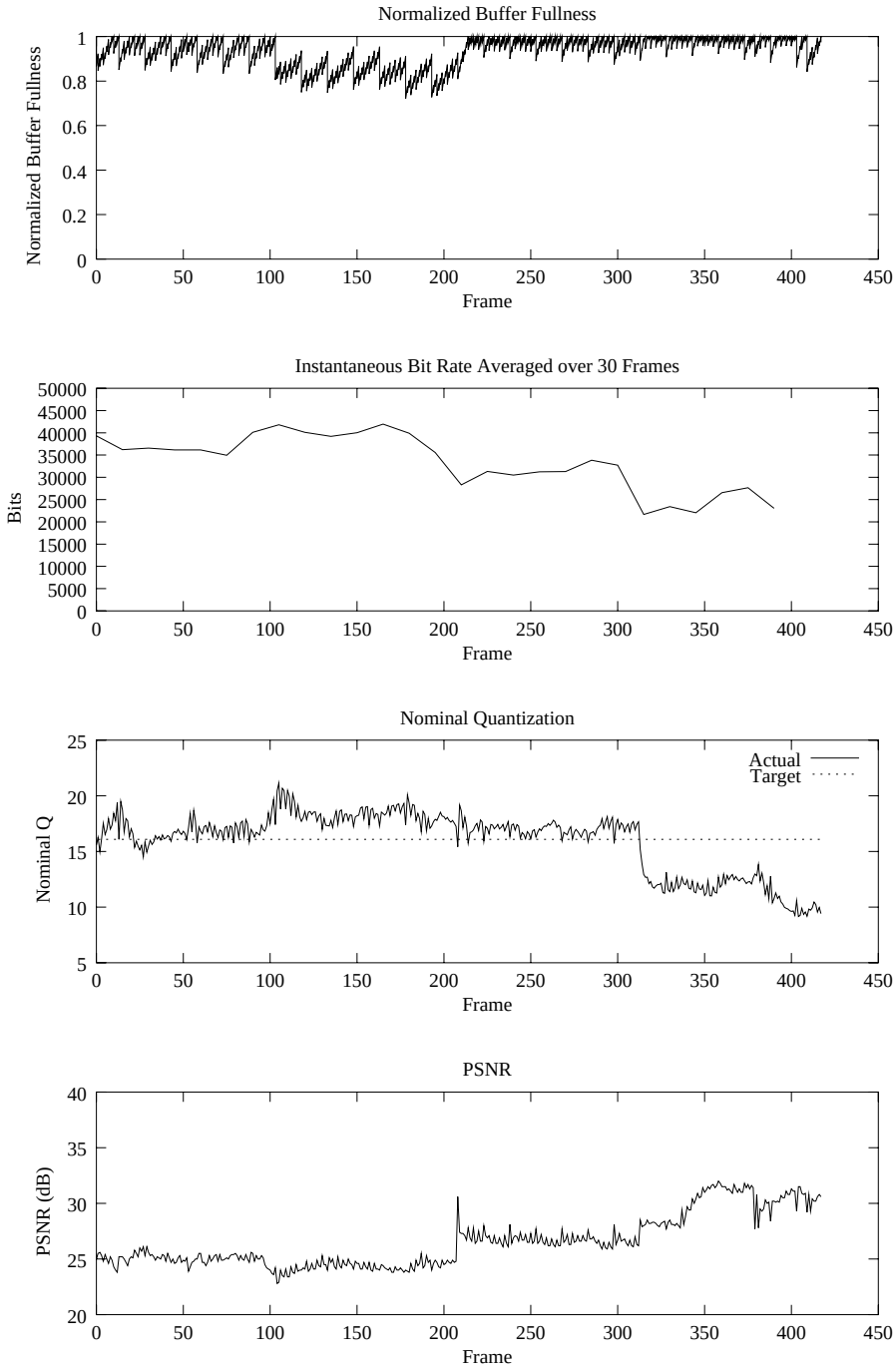



*Fig. 5.11* **Initial Simulation Results for Pass 1 of Hyperbolic VBR Coder.**

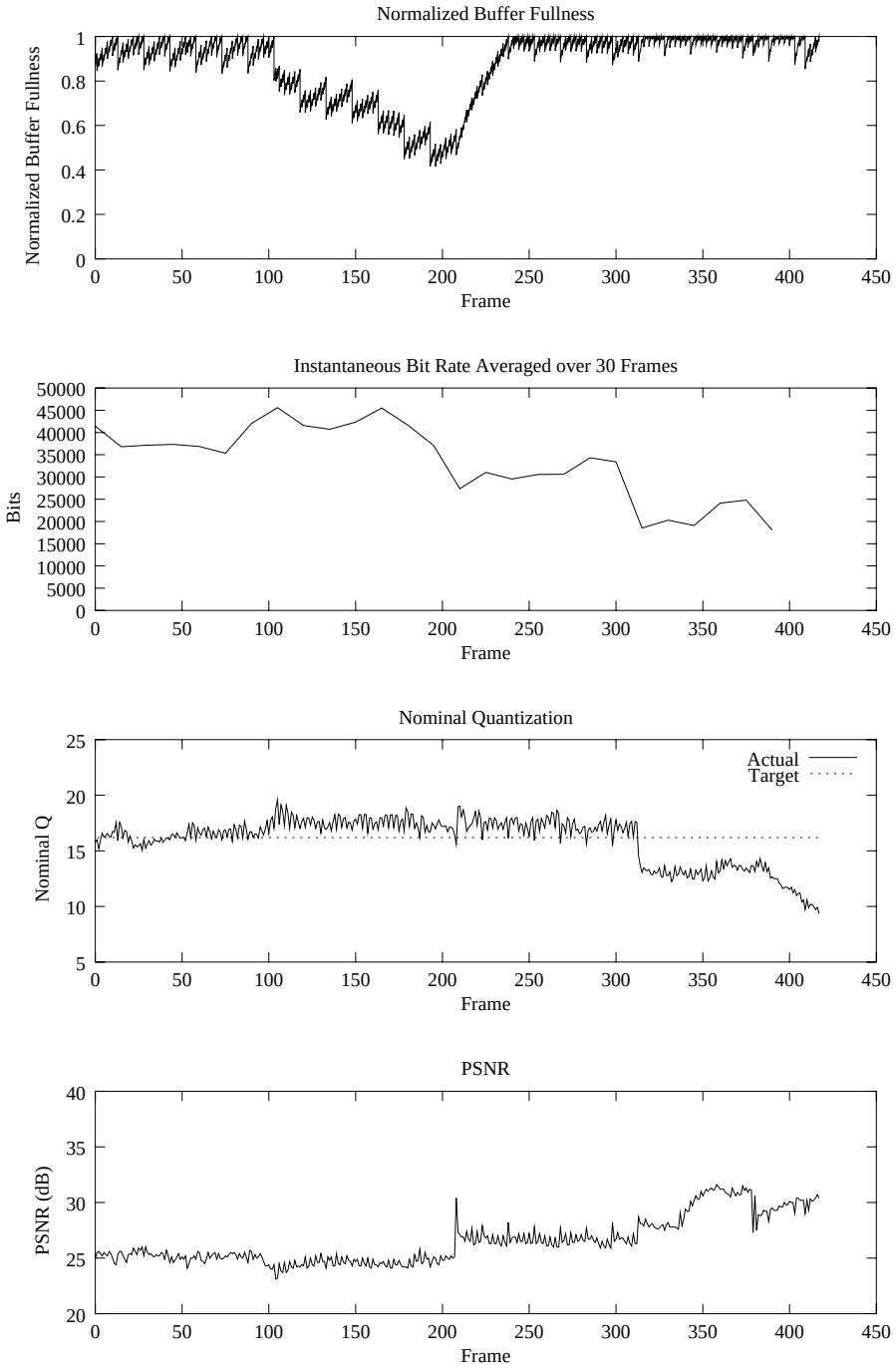



*Fig. 5.12* **Initial Simulation Results for Pass 2 of Hyperbolic VBR Coder.**

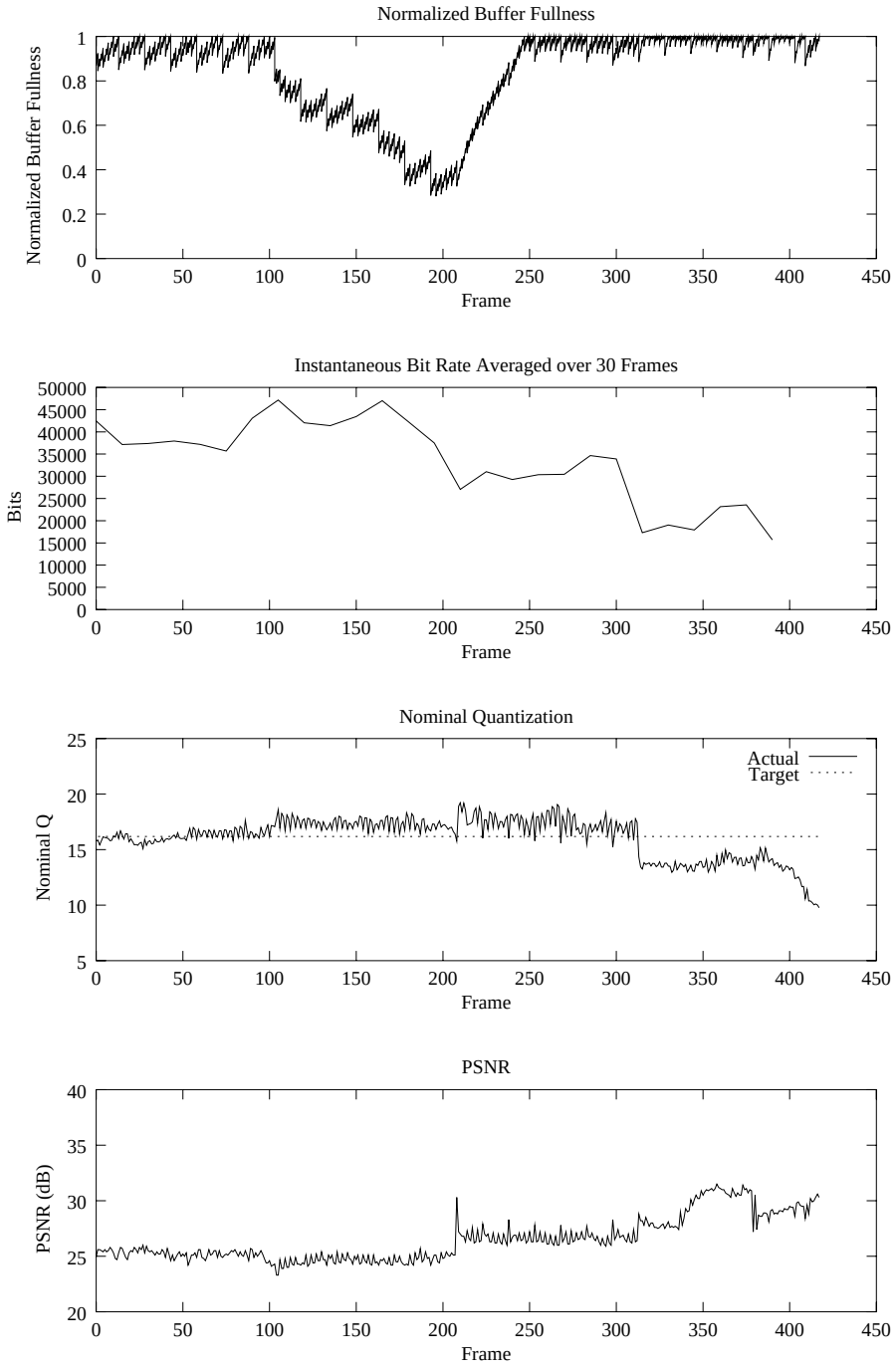



*Fig. 5.13* **Initial Simulation Results for Pass 3 of Hyperbolic VBR Coder.**

*Table 5.2* **Summary of Initial Simulations.** This table contains summary statistics for the initial coding simulations.

| Rate Control Algorithm | PSNR (dB) | | Nominal Q | | Nominal Q | |
|---|---|---|---|---|---|---|
| | Average | Std. Dev. | Average | Std. Dev. | Max | Min |
| TM5 CBR | 26.66 | 3.06 | 16.19 | 4.48 | 26.20 | 6.70 |
| Hyperbolic | | | | | | |
| CBR, Pass 1 | 26.48 | 2.67 | 16.19 | 3.44 | 22.63 | 7.78 |
| CBR, Pass 2 | 26.48 | 2.66 | 16.23 | 3.40 | 21.44 | 7.43 |
| CBR, Pass 3 | 26.48 | 2.67 | 16.28 | 3.41 | 20.68 | 7.20 |
| Linear-Spline | | | | | | |
| CBR, Hybrid | 26.73 | 2.68 | 15.80 | 3.36 | 19.48 | 8.29 |
| Hyperbolic | | | | | | |
| VBR, Pass 1 | 26.54 | 2.36 | 15.95 | 2.77 | 21.15 | 9.16 |
| VBR, Pass 2 | 26.52 | 2.10 | 16.00 | 2.14 | 19.60 | 9.34 |
| VBR, Pass 3 | 26.49 | 1.99 | 16.09 | 1.83 | 19.22 | 9.76 |
| Linear-Spline | | | | | | |
| VBR, Hybrid | 26.66 | 1.87 | 15.83 | 1.13 | 17.59 | 12.97 |

## 5.7 CODING A LONGER SEQUENCE

Since the video clips used in the initial experiments are short and we had to concatenate them to form somewhat artificial scene changes, we were not able to detect much perceptual difference between the different encodings. To assess the perceptual gains of lexicographically optimal bit allocation, we performed additional coding simulations using a longer video sequence with varied and dynamic content. The sequence is in NTSC CCIR-601 format and consists of 3,660 frames of a commercial produced by IBM to demonstrate its MPEG-2 encoding chipset. The clip starts with a fade-in to a spokeswoman standing in front of a slowing changing background. A block diagram in one corner of the picture then rotates and zooms to fill the screen. The diagram then remains static with some illumination changes before fading back to the spokeswoman. On one side of the picture, a collage of different video clips scroll up the screen. One of the clips zooms to occupy the full picture. The clips cycle through a variety of action-filled scenes from horses running to a skydiver rotating on a skateboard to a bicycle race and finally to highlights from a basketball game.

### 5.7.1 Independent Coding Simulations

With the `IBM Commercial` video sequence, we first evaluate the performance of the lexicographic bit-allocation algorithms using only I-frames so that the bit-production models are independent. For CBR mode, the constant bit rate is set to 10.5 Mbits/sec with guard zones at 5% and 95%. For VBR mode, the target average bit rate is set to 10.5 Mbits/sec with a peak bit rate of 13.5 Mbits/sec and a lower guard zone at 15%. The VBV buffer size is set to 1,835,008 bits. The coding parameters are listed in Table 5.3.

*Table 5.3* **Parameters for Independent-Coding Simulations.** This table lists the coding parameters used for the independent-coding simulations with the `IBM Commercial` sequence.

| Value | Description |
|---|---|
| 3660 | number of frames |
| 1 | N (# of frames in GOP) |
| 1 | M (I/P frame distance) |
| 0 | ISO/IEC 11172-2 stream |
| 0 | 0:frame pictures, 1:field pictures |
| 720 | horizontal_size |
| 480 | vertical_size |
| 2 | aspect_ratio: 1=square pel, 2=4:3, 3=16:9, 4=2.11:1 |
| 5 | frame_rate: 1=23.976, 2=24, 3=25, 4=29.97, 5=30 frames/sec |
| 10500000 | bit_rate (bits/sec) |
| 112 | vbv_buffer_size (in multiples of 16 kbit) |
| 0 | low_delay |
| 0 | constrained_parameters_flag |
| 1 | Profile ID: Simple=5, Main=4, SNR=3, Spatial=2, High=1 |
| 8 | Level ID: Low=10, Main=8, High 1440=6, High=4 |
| 0 | progressive_sequence |
| 2 | chroma_format: 1=4:2:0, 2=4:2:2, 3=4:4:4 |
| 2 | video_format: 0=comp,1=PAL,2=NTSC,3=SECAM,4=MAC,5=unspec |
| 720 | display_horizontal_size |
| 480 | display_vertical_size |
| 2 | intra_dc_precision: 0=8 bit, 1=9 bit, 2=10 bit, 3=11 bit |
| 1 | top_field_first |
| 0 0 0 | frame_pred_frame_dct (I P B) |
| 0 0 0 | concealment_motion_vectors (I P B) |
| 1 1 1 | q_scale_type (I P B) |
| 1 0 0 | intra_vlc_format (I P B) |
| 0 0 0 | alternate_scan (I P B) |
| 0 | repeat_first_field |
| 0 | progressive_frame |
| 0 | P distance between complete intra slice refresh |
| 0 | rate control: r (reaction parameter) |
| 0 | rate control: avg_act (initial average activity) |
| 0 | rate control: Xi (initial I frame global complexity) |
| 0 | rate control: Xp (initial P frame global complexity) |
| 0 | rate control: Xb (initial B frame global complexity) |
| 0 | rate control: d0i(initial I frame virtual buffer fullness) |
| 0 | rate control: d0p(initial P frame virtual buffer fullness) |
| 0 | rate control: d0b(initial B frame virtual buffer fullness) |
| 4 4 63 63 | P: forw_hor_f_code forw_vert_f_code search_width/height |

*Table 5.4* **Summary of Independent-Coding Simulations.** This table contains summary statistics for the independent-coding simulations with the `IBM Commercial` sequence.

| Rate Control Algorithm | PSNR (dB) | | Nominal Q | | Nominal Q | |
|---|---|---|---|---|---|---|
| | Average | Std. Dev. | Average | Std. Dev. | Max | Min |
| TM5 CBR | 35.03 | 3.53 | 9.93 | 3.81 | 28.97 | 0.48 |
| Linear-Spline CBR, Hybrid | 35.04 | 3.31 | 9.81 | 3.31 | 19.69 | 0.57 |
| Hyperbolic-Spline CBR, Hybrid | 35.04 | 3.29 | 9.80 | 3.29 | 19.11 | 0.74 |
| Linear-Spline VBR, Hybrid | 34.98 | 2.04 | 9.64 | 0.97 | 14.31 | 7.98 |
| Hyperbolic-Spline VBR, Hybrid | 35.01 | 2.03 | 9.58 | 0.91 | 14.18 | 7.99 |

Summary encoding statistics are listed in Table 5.4. The buffer fullness, smoothed instantaneous bit rate, nominal quantization, and PSNR plots are shown in Figures 5.14 to 5.18. The differences between the different coders are more pronounced with these simulations than with the previous ones. The lexicographic CBR coders are able to control the quantization to a narrower range than the TM5 coder, with a resulting increase in PSNR. The lexicographic VBR coders sacrifice quality in earlier pictures in order to code better the later more complex pictures. The result is that the nominal quantization is nearly constant and the PSNR plot is more even. Among the lexicographic coders, those using the hyperbolic-spline model slightly outperform those using the linear-spline model. Because of the better match between the actual and target nominal quantization, we conclude that the hyperbolic-spline model is slightly more accurate than the linear-spline model.

The lexicographic VBR coders produce near constant-quality video with few noticeable coding artifacts. In contrast, the CBR coders produce noticeable blocking artifacts in scenes with high motion, especially in the basketball scene. However, the lexicographic CBR coders fare noticeably better than TM5 at maintaining constant quality through scene changes and reducing artifacts during complex scenes of short duration.

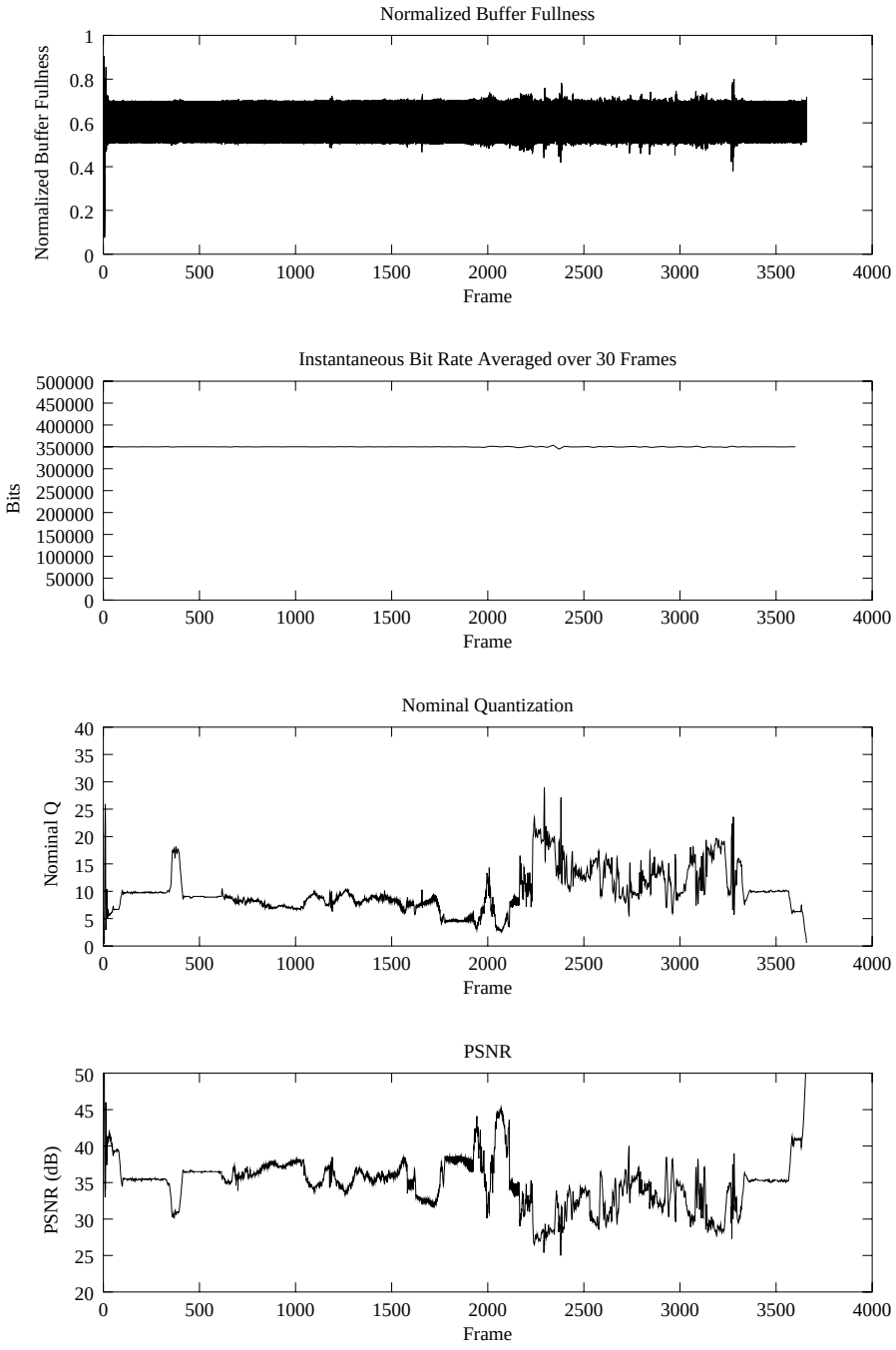



*Fig. 5.14* **Independent-Coding Results for TM5 CBR Coder.**

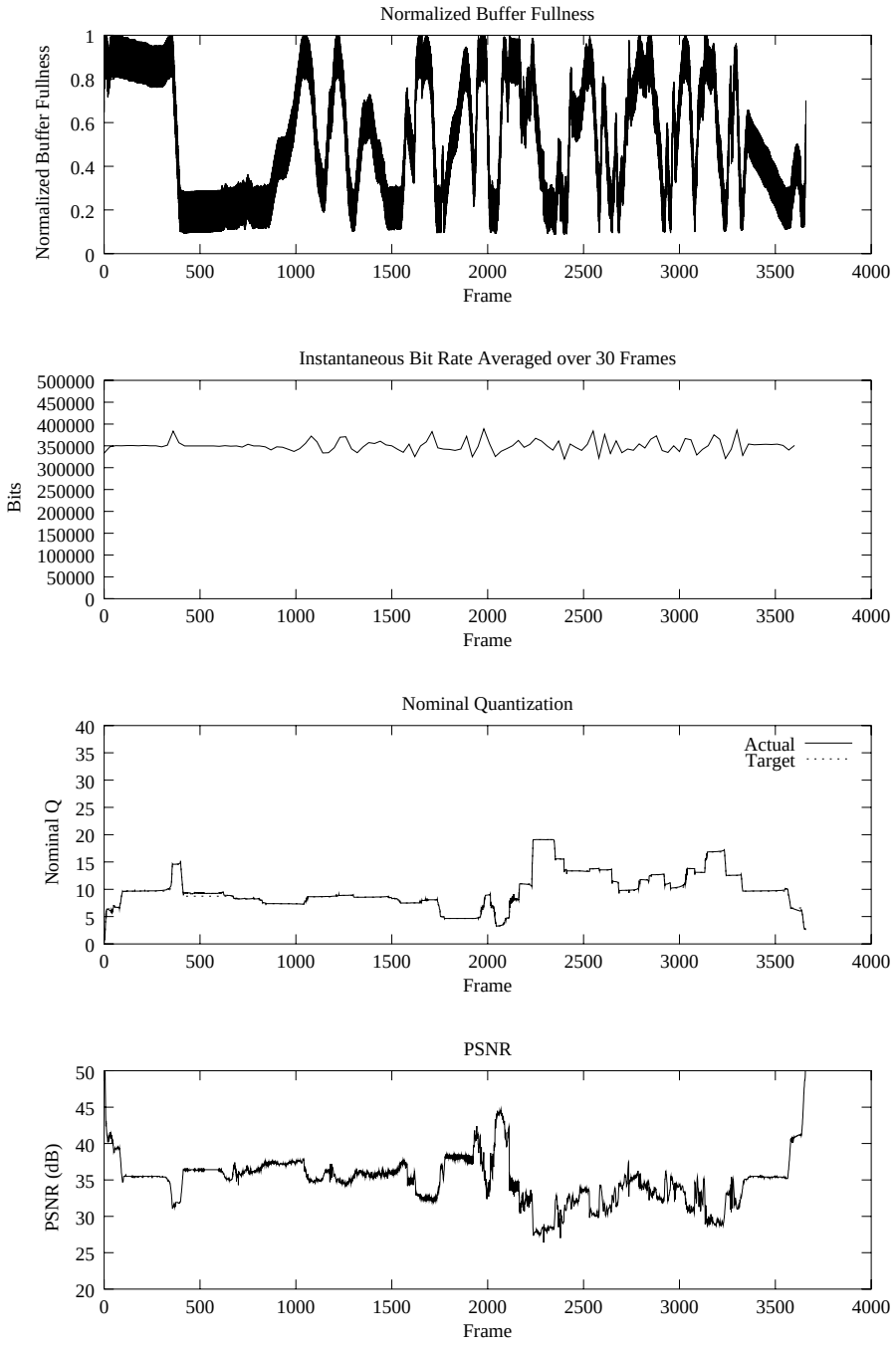



*Fig. 5.15* **Independent-Coding Results for Hyperbolic-Spline CBR Coder.**

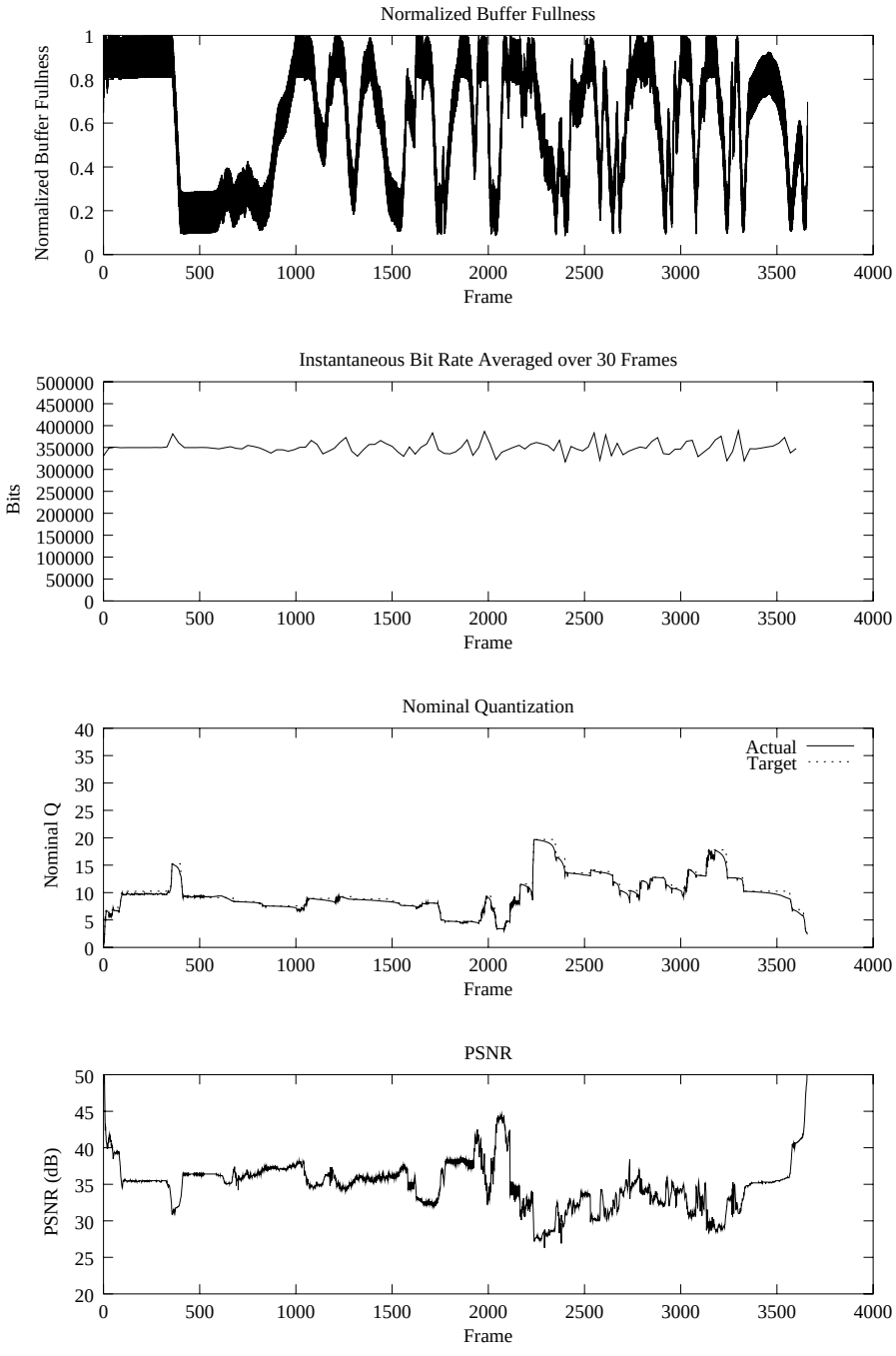



*Fig. 5.16* **Independent-Coding Results for Linear-Spline CBR Coder.**

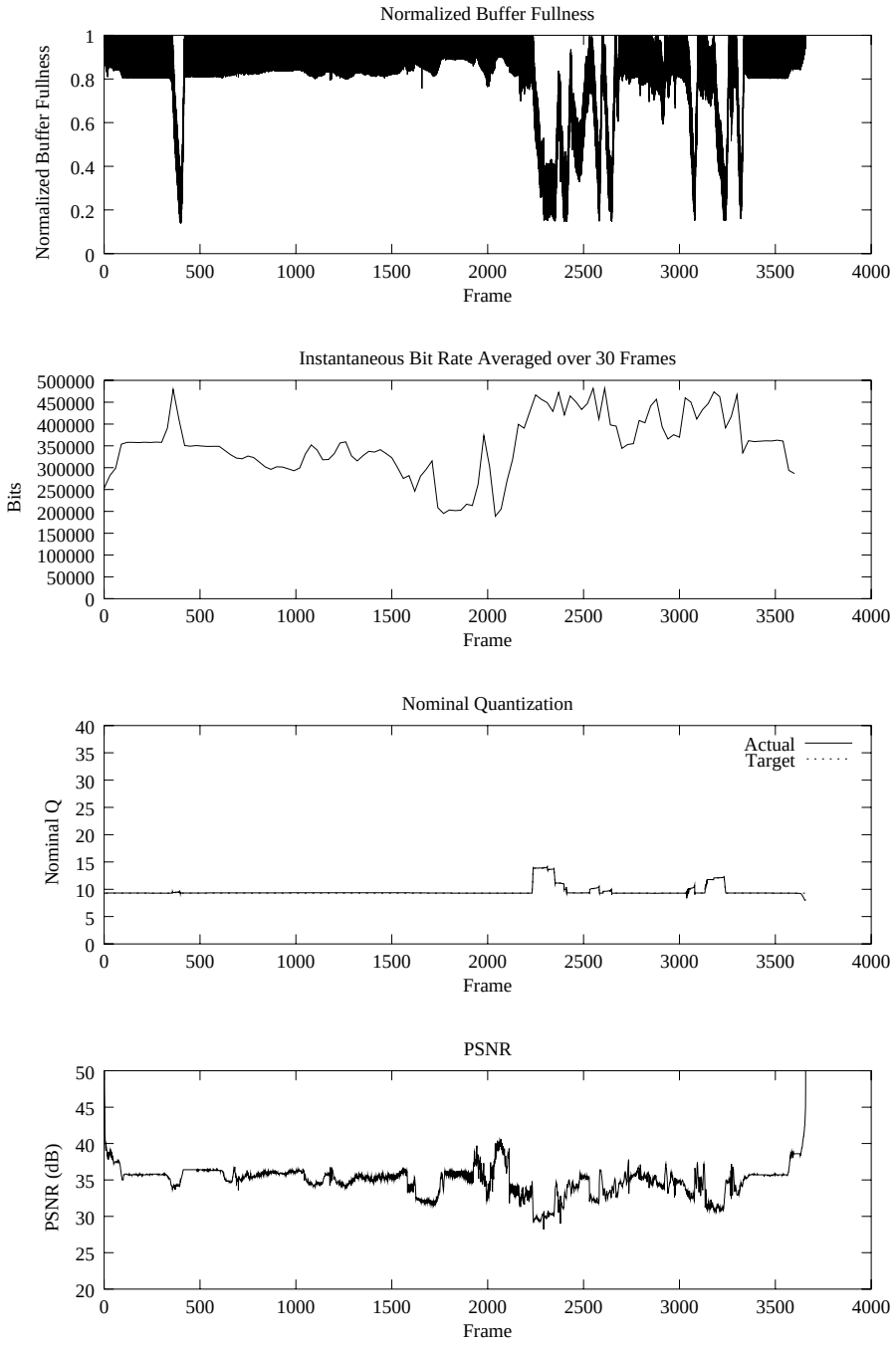



*Fig. 5.17* **Independent-Coding Results for Hyperbolic-Spline VBR Coder.**

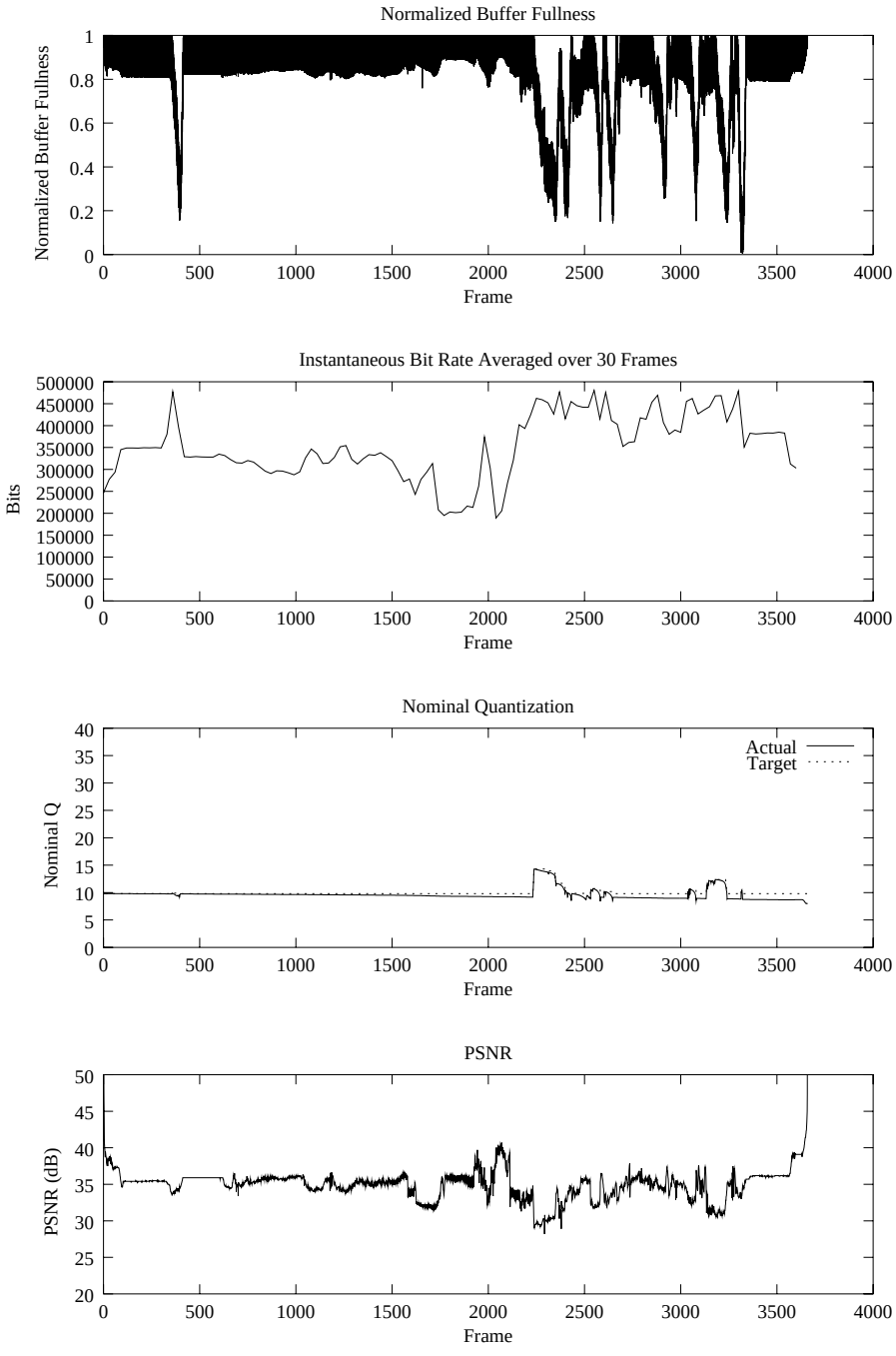



*Fig. 5.18* **Independent-Coding Results for Linear-Spline VBR Coder.**

### 5.7.2 Dependent Coding Simulations

In the final set of encoding simulations, we evaluate the performance of the lexicographic bit-allocation algorithms using I-frames, P-frames, and B-frames. The aim of these simulations is to determine the effects of using independent bit-production models with dependent coding. As with the independent-coding simulations, we use the `IBM Commercial` sequence. For CBR mode, the constant bit rate is set to 3.0 Mbits/sec with guard zones at 5% and 95%. For VBR mode, the target average bit rate is set to 3.0 Mbits/sec with a peak bit rate of 4.5 Mbits/sec and a lower guard zone at 10%. The VBV buffer size is set to 1,835,008 bits. The coding parameters are listed in Table 5.5.

Encoding statistics are listed in Table 5.6, with the buffer fullness, nominal quantization scale $Q$, and PSNR plots shown in Figures 5.19 to 5.23. The differences between the actual and target nominal quantization are more noticeable for the dependent-coding results compared to the independent-coding results. This indicates that the models are less accurate for dependent coding. The differences are more pronounced for the linear-spline model than for the hyperbolic-spline model. With the hyperbolic-spline model, the errors introduced by using independent bit-production models is quite small. This suggests that independent bit-production models can be used effectively for dependent coding with hybrid rate control.

## 5.8 LIMITING LOOKAHEAD

The above rate control algorithms compute an allocation sequence for the entire video sequence. This may not be feasible when the video sequence consists of many pictures, as in a feature-length movie, for example. One way to deal with this is to partition the sequence into blocks consisting of a small number of consecutive pictures. Optimal allocation can then be performed on the blocks separately. In order to do this, the starting and ending buffer fullness must be specified for each block for the CBR case. For the VBR case, the bit budget must also be specified for each block. This approach is globally suboptimal; however, it is easy to parallelize since the block allocations are independent of each other.

Another approach is to use limited lookahead in conjunction with hybrid rate control. Using a lookahead window of size $W$ and a step size $S \leq W$, the procedure is as follows:

1. Compute a bit allocation sequence for the next $W$ pictures not yet coded by computing a reverse dynamic programming table.

2. Code the next $S$ pictures using hybrid rate control, using the dynamic programming table to recover from model errors.

3. Repeat Step 1.

*Table 5.5* **Parameters for Dependent-Coding Simulations.** This table lists the coding parameters used for the Dependent-coding simulations with the `IBM Commercial` sequence.

| Value | Description |
|---|---|
| 3660 | number of frames |
| 15 | N (# of frames in GOP) |
| 3 | M (I/P frame distance) |
| 0 | ISO/IEC 11172-2 stream |
| 0 | 0:frame pictures, 1:field pictures |
| 720 | horizontal_size |
| 480 | vertical_size |
| 2 | aspect_ratio: 1=square pel, 2=4:3, 3=16:9, 4=2.11:1 |
| 5 | frame_rate: 1=23.976, 2=24, 3=25, 4=29.97, 5=30 frames/sec |
| 4500000 | bit_rate (bits/sec) |
| 112 | vbv_buffer_size (in multiples of 16 kbit) |
| 0 | low_delay |
| 0 | constrained_parameters_flag |
| 1 | Profile ID: Simple=5, Main=4, SNR=3, Spatial=2, High=1 |
| 8 | Level ID: Low=10, Main=8, High 1440=6, High=4 |
| 0 | progressive_sequence |
| 2 | chroma_format: 1=4:2:0, 2=4:2:2, 3=4:4:4 |
| 2 | video_format: 0=comp,1=PAL,2=NTSC,3=SECAM,4=MAC,5=unspec |
| 720 | display_horizontal_size |
| 480 | display_vertical_size |
| 2 | intra_dc_precision: 0=8 bit, 1=9 bit, 2=10 bit, 3=11 bit |
| 1 | top_field_first |
| 0 0 0 | frame_pred_frame_dct (I P B) |
| 0 0 0 | concealment_motion_vectors (I P B) |
| 1 1 1 | q_scale_type (I P B) |
| 1 0 0 | intra_vlc_format (I P B) |
| 0 0 0 | alternate_scan (I P B) |
| 0 | repeat_first_field |
| 0 | progressive_frame |
| 0 | P distance between complete intra slice refresh |
| 0 | rate control: r (reaction parameter) |
| 0 | rate control: avg_act (initial average activity) |
| 0 | rate control: Xi (initial I frame global complexity) |
| 0 | rate control: Xp (initial P frame global complexity) |
| 0 | rate control: Xb (initial B frame global complexity) |
| 0 | rate control: d0i(initial I frame virtual buffer fullness) |
| 0 | rate control: d0p(initial P frame virtual buffer fullness) |
| 0 | rate control: d0b(initial B frame virtual buffer fullness) |
| 4 4 63 63 | P: forw_hor_f_code forw_vert_f_code search_width/height |
| 4 4 63 63 | B1: forw_hor_f_code forw_vert_f_code search_width/height |
| 4 4 63 63 | B1: back_hor_f_code back_vert_f_code search_width/height |
| 4 4 63 63 | B2: forw_hor_f_code forw_vert_f_code search_width/height |
| 4 4 63 63 | B2: back_hor_f_code back_vert_f_code search_width/height |

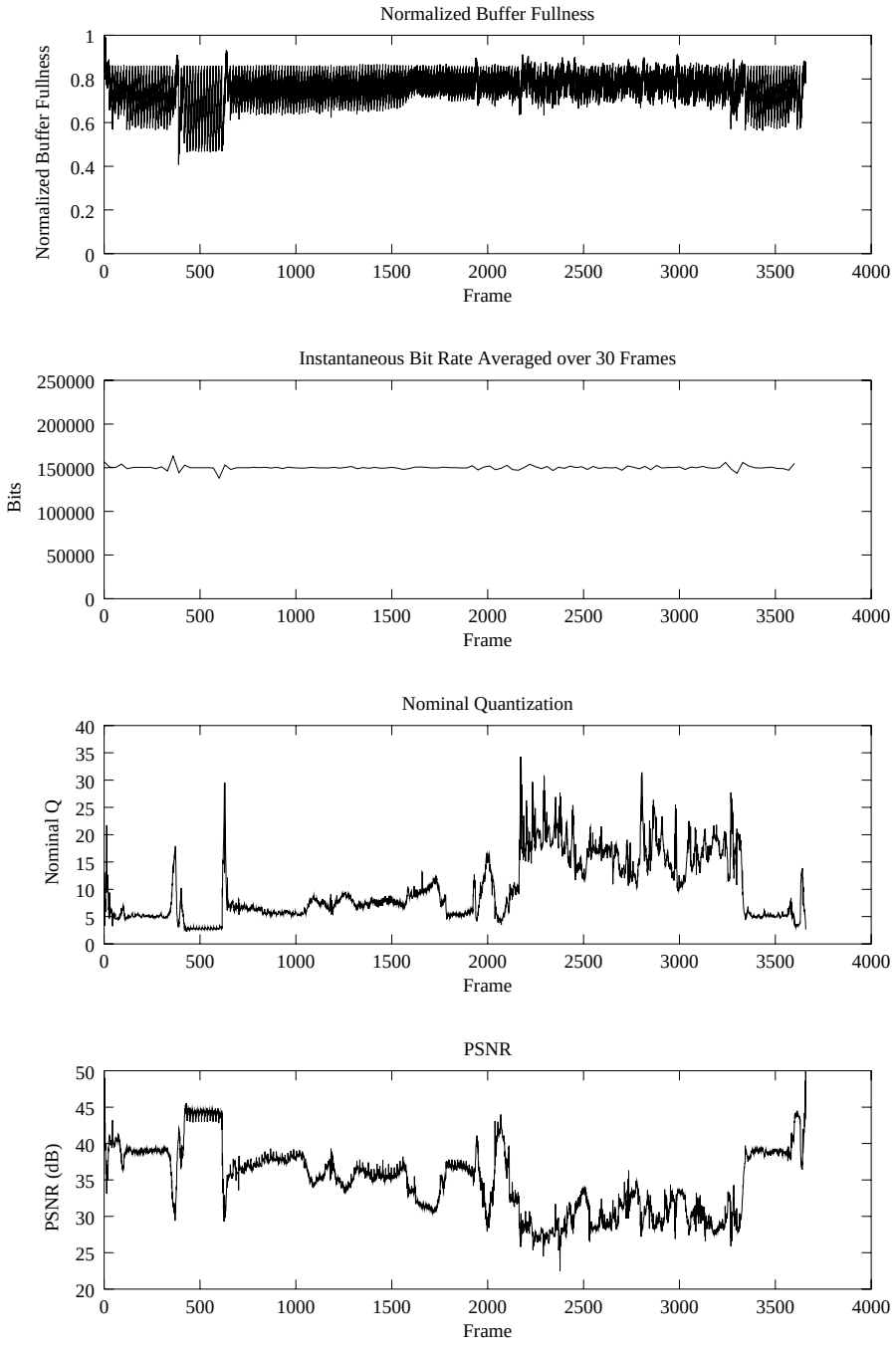



*Fig. 5.19* **Dependent-Coding Results for TM5 CBR Coder.**

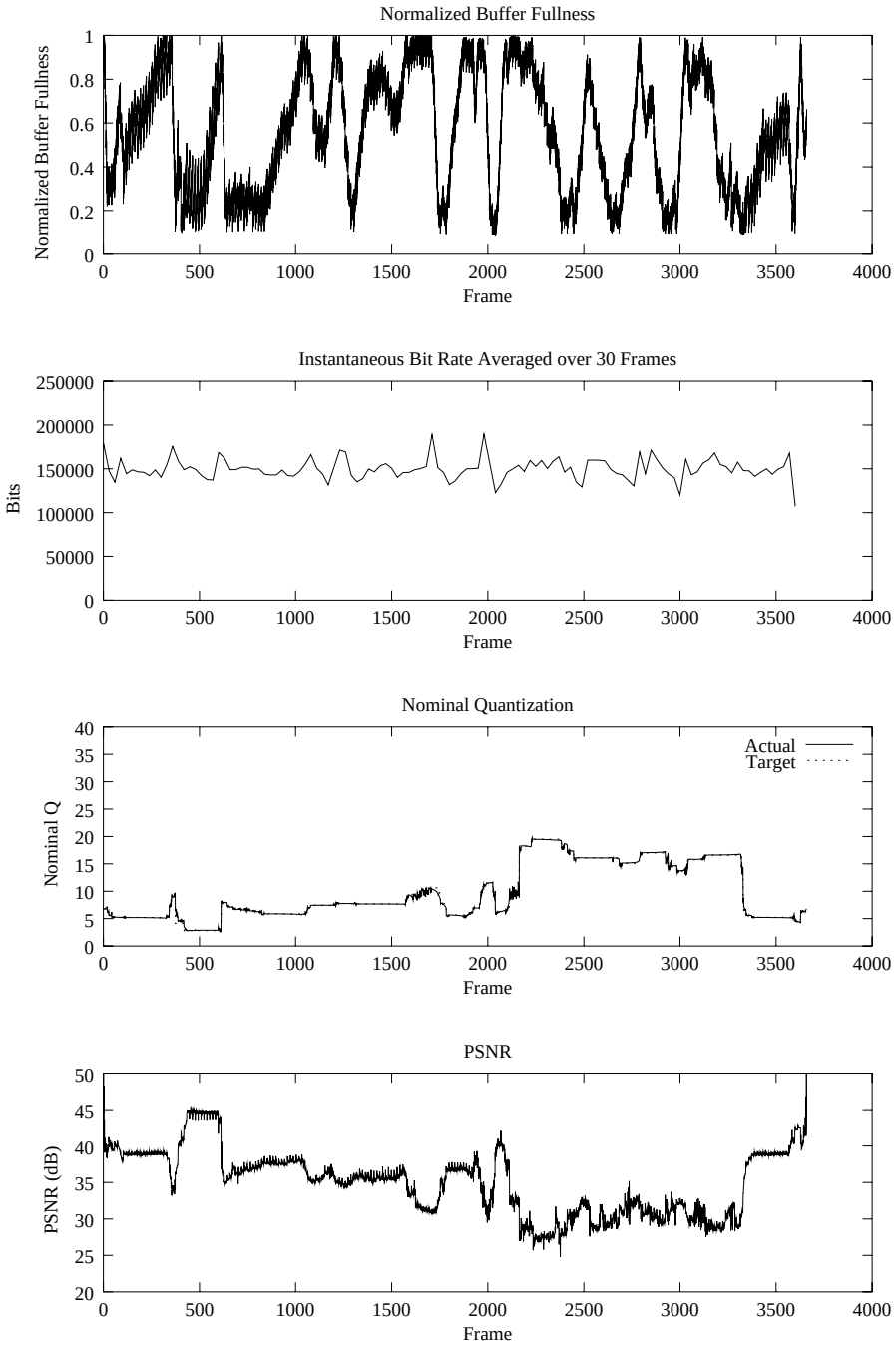



*Fig. 5.20* **Dependent-Coding Results for Hyperbolic-Spline CBR Coder.**

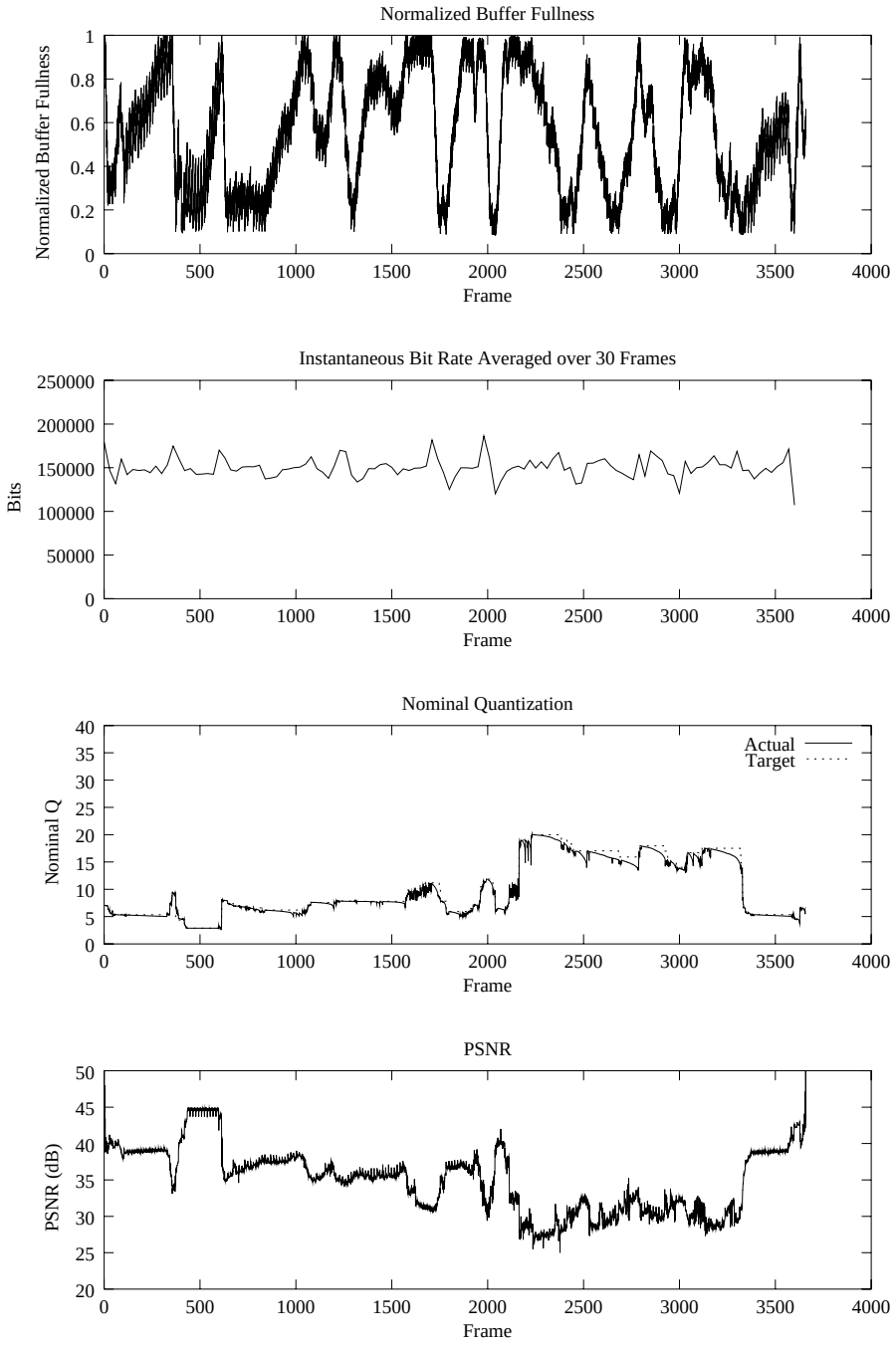



*Fig. 5.21* **Dependent-Coding Results for Linear-Spline CBR Coder.**

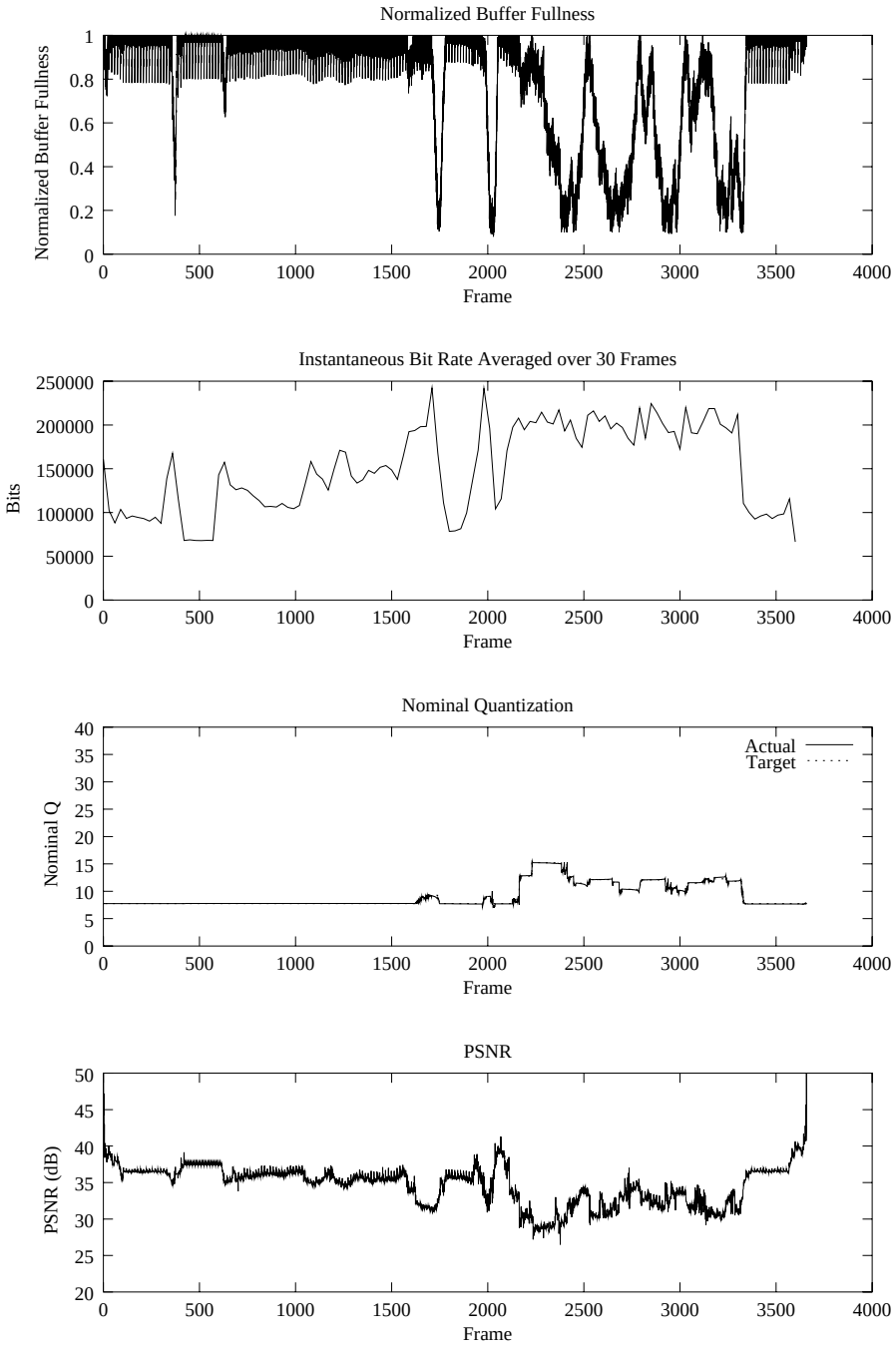



*Fig. 5.22* **Dependent-Coding Results for Hyperbolic-Spline VBR Coder.**

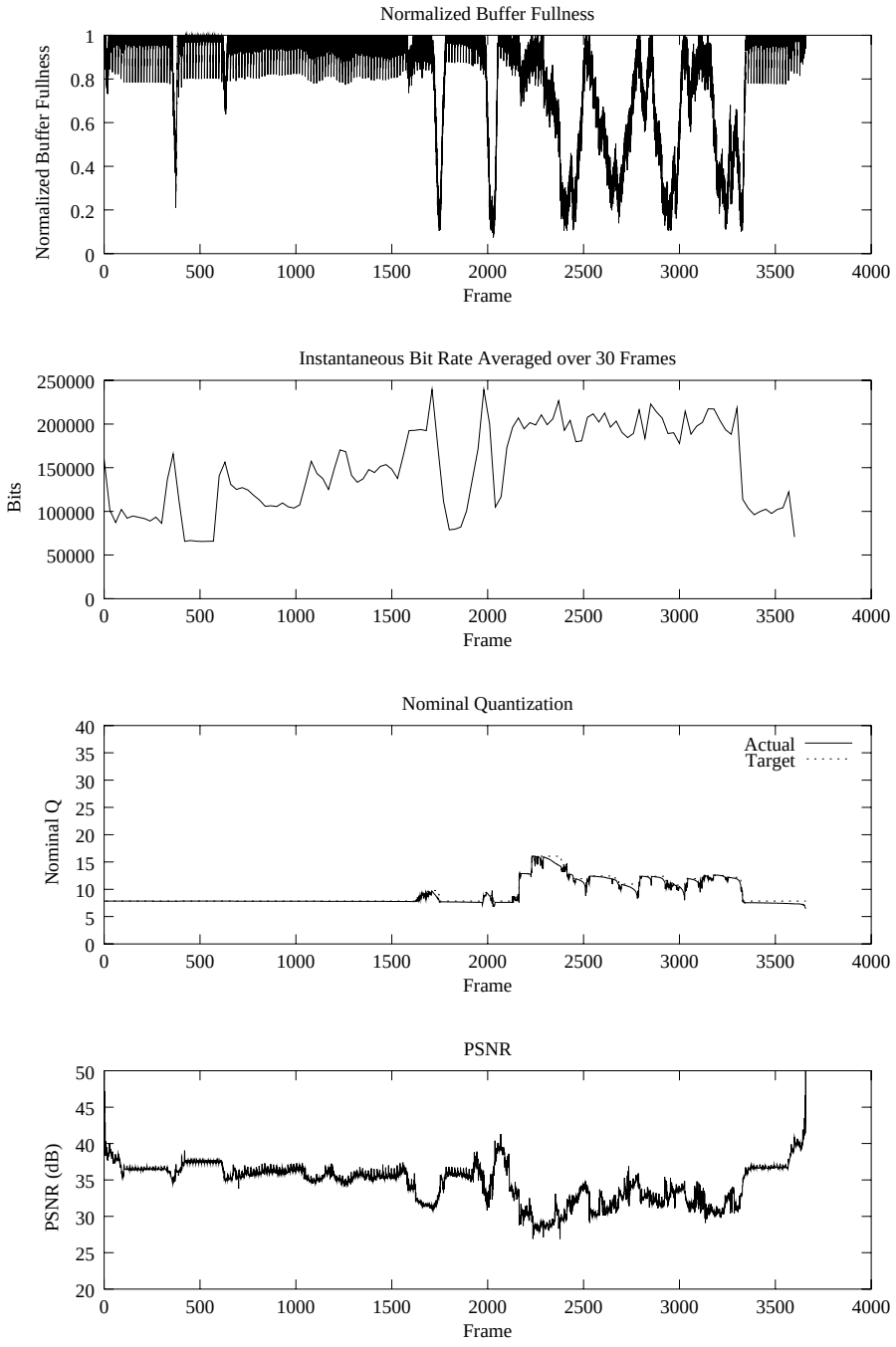



*Fig. 5.23* **Dependent-Coding Results for Linear-Spline VBR Coder.**

*Table 5.6* **Summary of Dependent-Coding Simulations.** This table contains summary statistics for the dependent-coding simulations with the `IBM Commercial` sequence.

| Rate Control Algorithm | PSNR (dB) | | Nominal Q | | Nominal Q | |
|---|---|---|---|---|---|---|
| | Average | Std. Dev. | Average | Std. Dev. | Max | Min |
| TM5 CBR | 34.99 | 4.62 | 10.07 | 5.80 | 34.30 | 2.32 |
| Linear-Spline CBR, Hybrid | 35.05 | 4.47 | 9.74 | 5.07 | 20.17 | 2.76 |
| Hyperbolic-Spline CBR, Hybrid | 35.06 | 4.46 | 9.73 | 5.05 | 19.72 | 2.80 |
| Linear-Spline VBR, Hybrid | 34.68 | 2.74 | 9.20 | 2.22 | 16.09 | 6.52 |
| Hyperbolic-Spline VBR, Hybrid | 34.68 | 2.72 | 9.20 | 2.19 | 15.43 | 7.04 |

This procedure can be thought of as performing lookahead with a sliding window.

Another approach similar to the hybrid rate control method is to use the allocation sequence computed from a given model and only recompute the allocation sequence when the buffer fullness breaches preset buffer boundaries, such as 10% and 90% of buffer fullness. As with hybrid rate control, reverse dynamic programming can be used to speed up the reallocation, so that the next optimal nominal quantization scale can be determined in $O(N)$ time. A more efficient option is to use the approach of Chapter 6 so that the time per picture can be reduced from $O(N)$ time to constant time in practice and to $O(\log N)$ time in the worst case.

## 5.9 RELATED WORK

Ortega, Ramchandran, and Vetterli [60] propose a suite of heuristic methods to reduce the complexity of bit allocation from that of the Viterbi algorithm. They apply a Lagrangian optimization technique to recompute an allocation sequence incrementally for each picture, similarly to the technique described in Section 5.3.2. In addition, the Lagrangian optimization is performed with a finite window size. In essence, this method implements limited lookahead with a sliding window, as we did in Section 5.8. The authors also describe the heuristic of recomputing an allocation only when the buffer reaches predefined threshold levels.

In this chapter, we have considered a simple hyperbolic model, a linear-spline model, and a hyperbolic-spline model of bit production. Chen and Hang [5] derive a bit-production model for block transform coders based upon rate-distortion theory and assuming a stationary Gaussian process.

They apply the model to VBR coding with motion JPEG and H.261 coders. Cheng and Hang [6] propose an adaptive tree-structured piecewise linear bit-production model and apply it to MPEG video coding using a one-pass encoding strategy. Lin, Ortega, and Kuo [47, 48] develop a cubic-spline model of rate and distortion for use with their gradient-based rate control algorithm that attempts to minimize MSE. The model takes into account the temporal dependencies introduced by predictive coding.

## 5.10 DISCUSSION

In this chapter, we have described an implementation of the bit allocation algorithms of Chapters 3 and 4 within an MPEG-2 software encoder. In addition to evaluating three types of bit-production models, we have developed robust techniques for recovering from errors in the bit-production models. Since we can view coding dependencies as contributing to errors in the (independent) bit-production models, these error-recovery techniques effectively allow us to apply the bit allocation framework to predictive video coders. In the next chapter we introduce more sophisticated (and optimal) methods for handling errors in the bit-production model.

# 6

# A More Efficient Dynamic Programming Algorithm

In practice, the bit-production model described in Section 2.3, which is at the heart of the relation between the bit allocation and the nominal quantization scale, is only *approximate*. The value $f_k(Q)$ is the predicted number of bits used to encode picture $k$ with nominal quantization scale $Q$, but the actual encoding length for the $k$th picture will typically be different. Therefore, after the $(j-1)$st picture is encoded, when it is time to assign a nominal quantization scale for the $j$th picture, the actual buffer fullness level is generally not the same as what it was predicted to be. In order to assign the nominal quantization scale for the $j$th picture, we need to recompute the optimal solution from the current buffer fullness level, using the bit-production model for the remaining pictures $j, j+1, \ldots, N$.

Our modified DP algorithm of Section 5.3.2, in which we compute the dynamic programming table *in reverse order*, allows the recomputation for each $1 < j \leq N$ to be done in linear time, so the total running time remains $O(N^2)$. If the bit-production model is accurate, the approach of Salehi et al. [69], appropriately modified as described in Section 3.3, computes the entire allocation sequence in linear time, using the Lee and Preparata [42] algorithm for computing shortest paths in polygonal channels. But with an inaccurate bit-production model, as is typically the case, rerunning the algorithm for each $1 < j \leq N$ results in an $O(N^2)$-time algorithm, as with our reverse approach.

In this chapter we develop an improved algorithm for CBR (and thus VBR) bit allocation that is robust against inaccurate bit-production models and is faster than those mentioned above. Our new algorithm uses a modification of the Lee-Preparata approach for finding shortest paths within polygonal

boundaries [42]. We run the algorithm on the pictures of the sequence *in reverse order*, but rather than construct a full dynamic programming table for the reversed sequence as in Section 5.3.2, we maintain a sparse representation of the table, which we call the *reverse structure*. After we encode the $(j-1)$st picture and determine the actual buffer fullness level, we use the reverse structure to compute the nominal quantization for the $j$th picture, and then we update the reverse structure for the subsequent picture. Each picture typically takes only constant time and in the worst case takes $O(\log N)$ time. The net result is that the entire video sequence can be encoded in linear time, in practice, and always in linear space; in the worst case, the running time is $O(N \log N)$.

In the next section, we show how to determine the nominal quantization scale based upon the reverse structure. In Section 6.2, we describe the preprocessing needed for the reverse structure, and in Section 6.3, we show how to update the reverse structure incrementally as each picture is encoded. We discuss related work in Section 6.4.

## 6.1 ENCODING THE NEXT PICTURE

For simplicity in discussion, let us assume CBR rate control with hybrid picture-level control, as explained in Section 5.3. The particulars can be modified easily for various control scenarios and buffer guards as well as for VBR.

In the general scenario, after we have just finished encoding the $(j-1)$st picture, the actual buffer fullness level may not be not what was predicted, because of the inaccuracies in the bit-production model. If we are in the middle of a constant-$Q$ segment, and if we continue coding the subsequent pictures $j$, $j+1$, ... in that segment using nominal quantization scale $Q$, then rather than arrive as desired at a full or empty buffer (at which point the nominal quantization scale changes), we may instead arrive at a buffer that is partially full, which will be nonoptimal according to the switching conditions of Section 3.1, or worse yet, the buffer may overflow or underflow.

Therefore, we need to recompute the optimal allocation sequence for pictures $j$, $j+1$, ..., $N$ based upon the actual buffer fullness level after the $(j-1)$st picture is processed. The key tool we use for the recomputation is the *reverse structure*, which is a sparse representation of the reverse dynamic programming table described in Section 5.3.2 for pictures $j$, $j+1$, ..., $N$. Figure 6.1 gives an example of a reverse structure for $j = 6$, which we will use as a running example in the remainder of this section to explain our algorithm.

To describe the reverse structure, it helps to make use of our revised notation in Section 5.3.2 for dynamic programming on the sequence of pictures $j$, $j+1$, ..., $N$, for decreasing $j$. We define **Final** to be the final buffer state, which corresponds by (2.11) to a buffer fullness level of $B_1 + \sum_{j=1}^{N-1} B_{\mathrm{inc}}(j) - B_{\mathrm{tgt}}$ bits. For $1 < j \le N$, we use **Top**$[j]$ to repre-

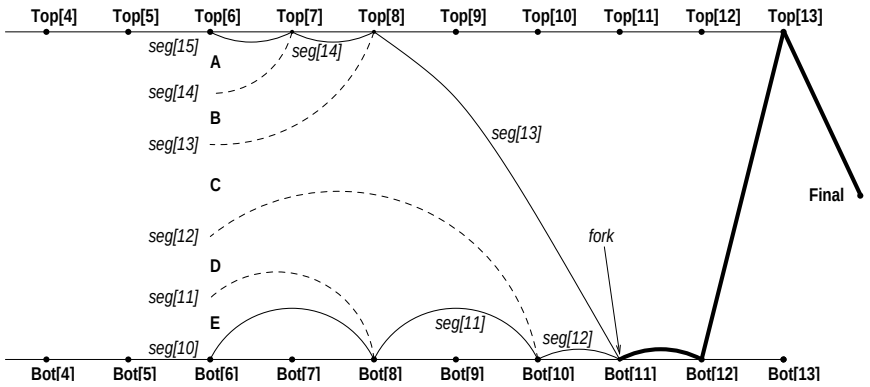



*Fig. 6.1* **Reverse Structure for Determining Nominal Quantization Scale.** This figure illustrates the various legal choices for the end destination of the constant-$Q$ segment starting with picture 6. Which destination is chosen depends upon the actual buffer fullness after picture 5 is processed.

sent the legal allocation sequence for pictures $j$, $j+1$, ..., $N$ that starts with a full buffer immediately before picture $j$ is removed (i.e., $B_{\rm f}(\boldsymbol{s}^*, j) = B_{\rm vbv}$, or equivalently $B^*_{\rm f}(\boldsymbol{s}^*, j-1) = B_{\rm vbv} - B_{\rm inc}(j)$) and ends up in **Final**. Similarly, we define **Bot**$[j]$ to be the legal allocation sequence for pictures $j$, $j+1$, ..., $N$ that starts with an empty buffer immediately after picture $j-1$ is removed (i.e., $B_{\rm f}(\boldsymbol{s}^*, j-1) = s^*_{j-1}$, or equivalently, $B^*_{\rm f}(\boldsymbol{s}^*, j-1) = 0$ or $B_{\rm f}(\boldsymbol{s}^*, j) = B_{\rm inc}(j)$) and ends up in **Final**. We use **Initial** to denote the optimal allocation sequence $\boldsymbol{s}^*$ for all $N$ pictures that begins with initial buffer fullness level $B_1$ and ends up in **Final**.

With a slight abuse of notation, we use **Top**$[j]$ (resp., **Bot**$[j]$) to denote not only the legal allocation sequence for pictures $j$, $j+1$, ..., $N$ as defined above, but also its beginning state at picture $j$. Similarly, we refer to **Initial** and **Final** as the initial and final states for the entire allocation sequence of $N$ pictures. We say that **Top**$[j]$ corresponds to a "full buffer," and **Bot**$[j]$ corresponds to an "empty buffer." We refer to **Initial**, **Final**, and **Top**$[j]$ and **Bot**$[j]$, for $1 < j \le N$, as *boundary states*. The switching conditions of Section 3.1 state that the nominal quantization scale for an optimal allocation sequence with an exact bit-rate model can change only at the boundary states.

The reverse structure consists of the allocation sequences **Top**$[j]$ and **Bot**$[j]$. In the example in Figure 6.1, where $j = 6$, **Top**[6] starts with constant-$Q$ segments

$$\begin{array}{ll} seg[15] & \text{for picture 6 (ending at picture 7),} \\ seg[14] & \text{for picture 7 (ending at picture 8),} \\ seg[13] & \text{for pictures 8–10 (ending at picture 11),} \end{array} \tag{6.1}$$

and **Bot**[6] starts with constant-$Q$ segments

$$\begin{array}{ll} seg[10] & \text{for pictures 6–7 (ending at picture 8),} \\ seg[11] & \text{for pictures 8–9 (ending at picture 10),} \\ seg[12] & \text{for picture 10 (ending at picture 11).} \end{array} \tag{6.2}$$

The *fork point* occurs at **Bot**[11], where the two sequences **Top**[$j$] and **Bot**[$j$] merge for pictures 11, 12, and 13 (ending at **Final**).

Let us denote the nominal quantization scale for segment $seg[i]$ by $seg[i].q$. By the switching conditions of Section 3.1, the nominal quantization scales for the constant-$Q$ segments in (6.1) and (6.2) are strictly decreasing when considered in counterclockwise order:

$$seg[10].q > seg[11].q > seg[12].q > seg[13].q > seg[14].q > seg[15].q. \tag{6.3}$$

In the reverse structure, we explicitly store only a subset of the boundary states **Top**[$j$], **Top**[$j+1$], ..., **Top**[$N$] and **Bot**[$j$], **Bot**[$j+1$], ..., **Bot**[$N$], namely, the ones that are suffixes of the optimal allocation sequences **Top**[$j$] and **Bot**[$j$], up to and including the fork point. In the example in Figure 6.1, for $j = 6$, the constant-$Q$ segments of (6.1) and (6.2) lead from

$$\textbf{Top}[6] \text{ to } \textbf{Top}[7] \text{ to } \textbf{Top}[8] \text{ to } \textbf{Bot}[11], \tag{6.4}$$

and from

$$\textbf{Bot}[6] \text{ to } \textbf{Bot}[8] \text{ to } \textbf{Bot}[10] \text{ to } \textbf{Bot}[11]. \tag{6.5}$$

It is convenient to list the boundary states of the reverse structure in the counterclockwise order of (6.3):

$$\textbf{Bot}[6],\ \textbf{Bot}[8],\ \textbf{Bot}[10],\ \textbf{Bot}[11],\ \textbf{Top}[8],\ \textbf{Top}[7],\ \textbf{Top}[6]. \tag{6.6}$$

The corresponding constant-$Q$ segments that connect them in counterclockwise order are

$$seg[10],\ seg[11],\ seg[12],\ seg[13],\ seg[14],\ seg[15]. \tag{6.7}$$

Our algorithm is based upon the following key observation: In order to recompute the optimal allocation sequence starting with picture $j$, only the states (6.6) in the reverse structure (other than **Bot**[$j$] and **Top**[$j$]) are eligible as possible boundary destinations for the constant-$Q$ segment starting at picture $j$, since by the switching conditions, the allocation sequence cannot cross over the constant-$Q$ segments in (6.7).

Before we encode picture $j$, we can imagine partitioning the possible buffer fullness levels into at most a linear number of contiguous intervals, where each interval corresponds to a particular boundary destination for the constant-$Q$ segment starting at picture $j$. In Figure 6.1, we denote these intervals as **A**, **B**, **C**, **D**, and **E**. The corresponding boundary destinations are

| *Region* | *Boundary Destination* |
|---|---|
| **A** | **Top**[7] |
| **B** | **Top**[8] |
| **C** | **Bot**[11] |
| **D** | **Bot**[10] |
| **E** | **Bot**[8] |

To encode the $j$th picture, we determine which interval the actual buffer fullness level lies in, which, in turn, determines the boundary destination. Given the boundary destination, we can compute in constant time the nominal quantization scale needed to reach that destination, using the formulas in Section 5.2.

Conceptually we can form the contiguous intervals by continuing *in reverse* the constant-$Q$ segments in (6.7) all the way back to picture $j$. For example, in Figure 6.1, if we extend segment *seg*[14] backwards from **Top**[7] to picture 6, then by (6.3) it dips below the path for segment *seg*[15], forming the interval **A**. We do the same for the other segments in (6.7) and end up with a set of contiguous intervals **A**, **B**, **C**, **D**, and **E** and their corresponding boundary destinations, as specified in (6.1).

We do not actually compute the buffer fullness intervals explicitly. Rather, for a given buffer fullness level, we search to determine the legal $Q$ to use for the $j$th picture. Given a candidate boundary destination, say, **Top**[$k$], let $q'$ and $q''$ be the two nominal quantization scales in counterclockwise order in (6.3) whose segments touch the boundary at **Top**[$k$]. Given the current buffer fullness level, we compute the nominal quantization scale $Q$ to arrive at **Top**[$k$], using the formulas for the bit-production model in Section 5.2. If $q' \geq Q \geq q''$, then $Q$ meets the switching conditions of Section 3.1, and we assign $Q$ to encode the $j$th picture. Otherwise if $Q > q'$, then we choose a new candidate boundary destination that is earlier in the counterclockwise order (6.6). If $q'' > Q$, we choose a new candidate boundary destination that is later in the counterclockwise order (6.6).

The above conditions suggest a natural binary finger search strategy for finding the legal $Q$: We can start with the target boundary destination predicted by the model before the previous picture was actually encoded. We can then use the search strides 1, 2, 4, 8, ... in the appropriate direction until we overshoot the correct target boundary destination, after which a normal binary search on the last stride interval will narrow in on the legal destination. For the $j$th picture, if there are $k_j$ buffer fullness intervals between the predicted destination and the legal destination, the search takes $O(\log k_j)$ time, which is $O(\log N)$ in the worst case. If there are no errors, then each picture can obviously be encoded in constant time. When the bit models are fairly accurate, or when the number of buffer fullness intervals in the reverse structure is small, the total running time $\sum_{j=1}^{N} \log k_j$ is $O(N)$. In practice, a

linear search may be faster than a binary finger search. In the worst case, the total running time is $O(N \log N)$.

## 6.2 INITIAL PREPROCESSING OF THE REVERSE STRUCTURE

In order to recompute the nominal quantization scale for the $j$th picture, the algorithm we developed in the last section makes use of the reverse structure for pictures $j$, $j+1$, ..., $N$, which is essentially an encoding of $\mathbf{Top}[j]$ and $\mathbf{Bot}[j]$. We can compute that reverse structure in an efficient way by "rolling back" the reverse structure for pictures $j-1$, $j$, ..., $N$, so as to avoid computing it from scratch for each $j$. We defer to the next section how to do the rollback. In this section we show how to precompute the reverse structure for pictures 1, 2, ..., $N$ from scratch, which we use to start the encoding process.

The reverse sequence for pictures $j$, $j+1$, ..., $N$, for the cases $j = 6$, $j = 5$, and $j = 4$ are pictured in Figures 6.2, 6.3, and 6.4. We represent the reverse structure by an array called *seg*. The maximum number of constant-$Q$ segments in the reverse structure at any one time is $2N$ (at most $N$ on the top boundary and $N$ on the bottom boundary), so we index the array from 1 to $2N$. The current constant-$Q$ segments on the boundary are stored consecutively in *counterclockwise order* (6.7) in $seg[first]$, $seg[first+1]$, ..., $seg[last]$. Each entry in the *seg* array has a member (or field) $q$ that stores the nominal quantization scale for the segment as well as members *start* and *end* that store the starting boundary state and the ending boundary state for the segment, according to the counterclockwise order. For each $first \le i < last$, we have $seg[i].end = seg[i+1].start$. Because of the counterclockwise order, if $seg[i]$ is on the top boundary, then $seg[i].start$ is a higher-numbered boundary state than is $seg[i].end$.

The algorithm below computes the entire reverse structure for pictures 2, 3, ..., $N$. The idea is to construct, for decreasing values of $j$, the reverse structure for pictures $j$, $j+1$, ..., $N$ from the reverse structure for pictures $j+1$, $j+2$, ..., $N$. In particular, we need to add a constant-$Q$ segment starting from $\mathbf{Top}[j]$ and a constant-$Q$ segment starting from $\mathbf{Bot}[j]$ so as to maintain the counterclockwise order of constant-$Q$ segments. In the process, some of the previous constant-$Q$ segments get deleted from the counterclockwise order. The fork point changes when one of the new constant-$Q$ segments goes from the top boundary to the bottom boundary, or vice versa.

1. [Initialize counterclockwise order.] Assign $first \leftarrow N$, $last \leftarrow N+1$, $seg[first] \leftarrow$ the constant-$Q$ segment that connects the bottom boundary at picture $N$ to the final state, and $seg[last] \leftarrow$ the constant-$Q$ segment that connects the top boundary at picture $N$ to the final state. Initialize the fork point to be the final state.

2. [Process pictures in reverse.] For each picture $j = N-1$, $N-2$, ..., 2, do Steps 3–8:

3. [Determine where to connect the new constant-$Q$ segment from the top boundary at picture $j$.] Set the dummy sentinel value $seg[first-1]$ to be the constant-$Q$ segment that connects the bottom boundary at picture $j$ to the bottom boundary at picture $j+1$. Decrement *last* zero or more times until $seg[last].q$ is larger than the constant nominal quantization scale $Q$ that connects the top boundary at picture $j$ to $seg[last].end$.

4. [Update the fork point?] If $seg[last].end$ is on the bottom boundary and is not the fork point, then make it the new fork point, and add to the optimal allocation sequence the boundary segments between the previous fork point and the new fork point.

5. [Insert the top constant-$Q$ segment.] Increment *last* and assign $seg[last] \leftarrow$ the constant-$Q$ segment that connects the top boundary at picture $j$ to $seg[last-1].end$.

6. [Determine where to connect the new constant-$Q$ segment from the bottom boundary at picture $j$.] Increment *first* zero or more times until $seg[first].q$ is less than the constant nominal quantization scale $Q$ that connects the bottom boundary at picture $j$ to $seg[first].start$.

7. [Update the fork point?] If $seg[first].start$ is on the top boundary and is not the fork point, then make it the new fork point, and add to the optimal allocation sequence the boundary segments between the previous fork point and the new fork point.

8. [Insert the bottom constant-$Q$ segment.] Decrement *first* and assign $seg[first] \leftarrow$ the constant-$Q$ segment that connects the bottom boundary at picture $j$ to $seg[first+1].start$.

Strictly speaking, in order to avoid consecutive constant-$Q$ segments with the same $Q$ value, the "larger than" condition in Step 3 should be replaced by "larger than or equal to" when $seg[last]$ is part of the bottom boundary. Similarly, the "less than" condition in Step 6 should be replaced by "less than or equal to" when $seg[first]$ is on the top boundary.

The process to extend the reverse structure to all $N$ pictures (i.e., including $j = 1$) is a straightforward modification of Steps 3–5. The only difference is that we use the starting point **Initial** rather than the top boundary starting point **Top**$[j]$.

To explain how the algorithm works, let us consider the case $j = 5$, in which we convert the reverse structure for pictures 6, 7, ..., $N$ in Figure 6.2 to the reverse structure for pictures 5, 6, ..., $N$ in Figure 6.3. In Steps 3–5, we determine that the constant $Q$ segment from **Top**[5] should end at **Top**[7] in order to maintain the counterclockwise decreasing order of nominal quantization scales. The old value of $seg[15]$ in Figure 6.2 is thus replaced by the new value of $seg[15]$ in Figure 6.3; the value of *last* remains 15. Similarly, in Steps 6–8, we find that the constant-$Q$ segment from **Bot**[15] should end at

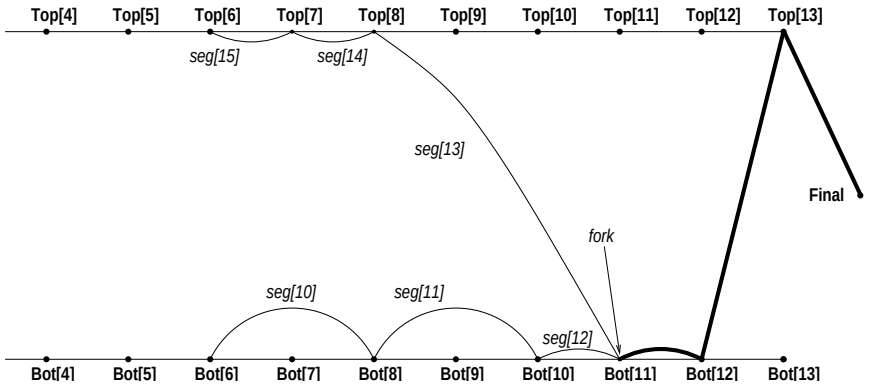



*Fig. 6.2* **Bottom and Top Boundaries Starting at Picture 6.** This figure shows the reverse structure for pictures 6, 7, ..., $N$. The six constant-$Q$ segments on the boundaries are stored in $seg[10]$, ..., $seg[15]$; that is, the current values of *first* and *last* are 10 and 15, respectively. We have, for example, $seg[10].start = \mathbf{Bot}[6]$, $seg[10].end = \mathbf{Bot}[8]$, and $seg[13].start = \mathbf{Bot}[11]$, $seg[13].end = \mathbf{Top}[8]$.

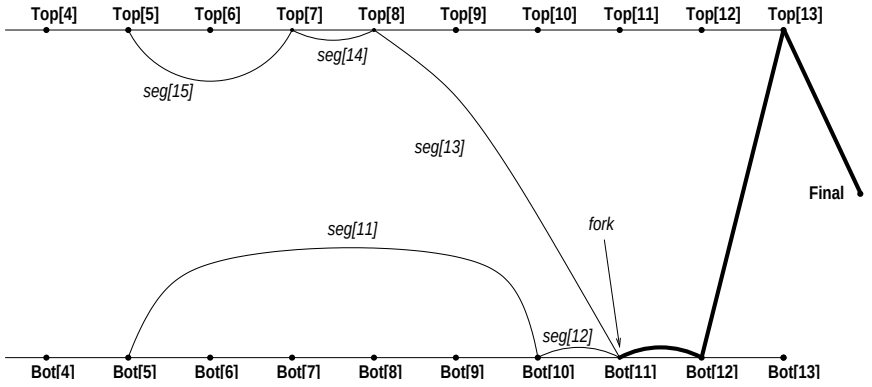



*Fig. 6.3* **Bottom and Top Boundaries Starting at Picture 5.** This figure shows the reverse structure for pictures 5, 6, ..., $N$. The six constant-$Q$ segments on the boundaries are stored in $seg[11]$, ..., $seg[15]$; that is, the values of *first* and *last* are 11 and 15, respectively.

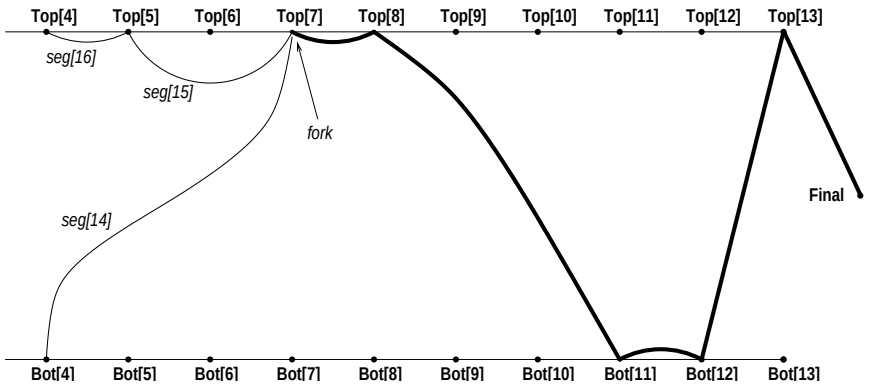



*Fig. 6.4* **Bottom and Top Boundaries Starting at Picture 4.** This figure shows the reverse structure for pictures 4, 5, ..., $N$. The six constant-$Q$ segments on the boundaries are stored in $seg[14]$, ..., $seg[16]$; that is, the values of *first* and *last* are 14 and 16, respectively.

**Bot**[10]. The former segments $seg[10]$ and $seg[11]$ in Figure 6.2 are replaced by the new segment $seg[11]$ in Figure 6.3; the value of *first* is changed from 10 to 11. The new counterclockwise order of segments in Figure 6.3, in decreasing order of nominal quantization scale, is

$$seg[11],\ seg[12],\ seg[13],\ seg[14],\ seg[15]. \tag{6.8}$$

The case $j = 4$, in which we convert from Figure 6.3 to Figure 6.4, involves an update of the fork point. The new constant-$Q$ segment added to the top boundary is $seg[16]$ from **Top**[4] to **Top**[5], and the value of *last* is changed from 15 to 16. The more interesting update involves the bottom boundary, in which the constant-$Q$ segment $seg[14]$ is added from **Bot**[4] to **Top**[7], the value of *first* is changed from 11 to 14, and the fork point shifts from **Bot**[11] to **Top**[7]. The new counterclockwise order of segments in Figure 6.4, in decreasing order of nominal quantization scale, is

$$seg[14],\ seg[15],\ seg[16]. \tag{6.9}$$

The full trace of all segments ever written during the preprocessing of Figure 6.4 appears in Figure 6.5.

### 6.2.1 Time and Space Complexity

In order to determine the total time used by the algorithm in order to construct the full reverse sequence for all $N$ pictures, let us consider the work done in Steps 3–8 for each value of $j$. We add a new segment to the top boundary and a new segment to the bottom boundary. In the process we walk through the counterclockwise list of segments from both ends, discarding segments until we find the proper endpoints for the newly added segments.

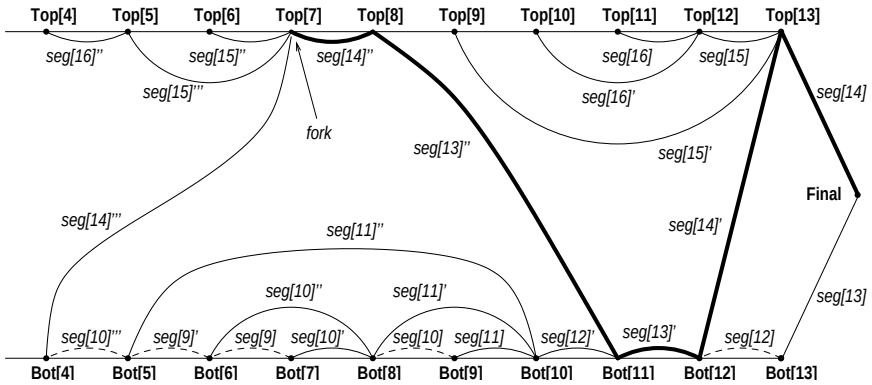



*Fig. 6.5* **Trace of Execution.** This figure shows all the segments ever written during the course of the construction of the reverse sequence for pictures 4, 5, ..., $N$, as shown in Figure 6.4. To avoid confusion, we add a prime $'$ every time a segment in the *seg* array gets modified. For example, the segment $seg[15]'''$ from **Top**[5] to **Top**[7] means that it was the fourth segment stored (i.e., the third modified) in position 15 during the course of the preprocessing algorithm. The dashed segments are those whose sole purpose was to serve as dummy sentinels for the linear search in Step 3.

We spend constant time plus the time for the discarded segments, which we never process again. Therefore, the amortized time for each new value of $j$ is $O(1)$, and the total time over the course of the algorithm is $O(N)$. The time for each $j$ can be made $O(\log N)$ in the worst case by use of binary search to determine where to insert the new constant-$Q$ segment. The total time remains $O(N)$. As noted earlier, the size of the *seg* array is at most $2N$.

## 6.3 INCREMENTAL UPDATE OF THE REVERSE STRUCTURE

In order to complete our algorithm for buffer-constrained bit allocation in the presence of an inaccurate bit-production model, all that remains is to show how to efficiently "roll back" the reverse structure for pictures $j-1, j, \ldots, N$ in order to obtain the the reverse structure for pictures $j, j+1, \ldots, N$.

We can easily accomplish this rollback task if we record all updates made during during the course of the preprocessing algorithm of Section 6.2. We noted in Section 6.2.1 that the total number of updates to the *seg* array was linear. If we record the history of updates to the *seg* array and also to *first* and *last*, then in order to perform the roll backs, we can replay the history in reverse order and "undo" the updates. The extra space required to store the history is $O(N)$, which increases the space by a constant factor. The total time to do all the rollbacks is therefore $O(N)$, since each item in the history is processed once and takes constant time to undo.

## 6.4 RELATED WORK

The problem of preprocessing the reverse structure for all $N$ pictures in Section 6.2 is equivalent to computing the shortest path in a polygonal channel in reverse order. In the former case, the switching conditions require that the nominal quantization scales in the counterclockwise list of segments on the bottom and top boundaries are strictly decreasing. Beyond the fork point, the bottom and top boundaries coalesce into a common path. The same phenomenon occurs in computing the shortest path in a polygonal channel, except that in this case, the switching conditions apply to the *slopes* of the segments along the boundary paths: The slopes of the segments on the bottom and top boundaries are strictly decreasing in counterclockwise order. Figures 6.6 and 6.7 illustrate the correspondence between the two problems.

With this reduction in mind, the particular approach we used for the preprocessing algorithm in Section 6.2 corresponds to the algorithm of Lee and Preparata [42] for finding shortest paths in polygonal channels. Salehi et al. [69] used the same approach in a more direct setting to reduce variability in the transmission rate for stored video. Their setting corresponded exactly to finding shortest paths in polygonal channels. However, in buffer-constrained bit allocation, there is no fixed relationship between the bit allocation and the nominal quantization scale (or equivalently, distortion), which is the object of optimization. Instead, we require one extra level of indirection; for each video sequence of interest, we precompute the relationship between bit allocation and nominal quantization scale in linear time and space by the approach of Section 3.2.5. One other difference to note in our approach over that of Salehi et al. [69] is the conceptual simplification we gain by merging the bottom and top boundaries into a single counterclockwise order. The searches for which constant-$Q$ segments to add become simple linear scans from both ends of the list; the decision of updating the fork point follows directly and does not require a separate scan.

In the shortest path problem, the switching conditions on the slopes imply that the bottom boundary leading up to the fork point is an upper convex hull, and similarly the top boundary is a lower convex hull. (See Figures 6.6 and 6.7. E.g., the segments $c$, $d$, $e$ on the top boundary in Figure 6.6 form a lower convex hull.) Melkman [54], Friedman et al. [20], and Hershberger and Suri [28] consider the problem of how to maintain upper and lower convex hulls when vertices are inserted and deleted. The rollback technique we used for updating the reverse structure is akin to a technique developed by [20] in a more general setting.

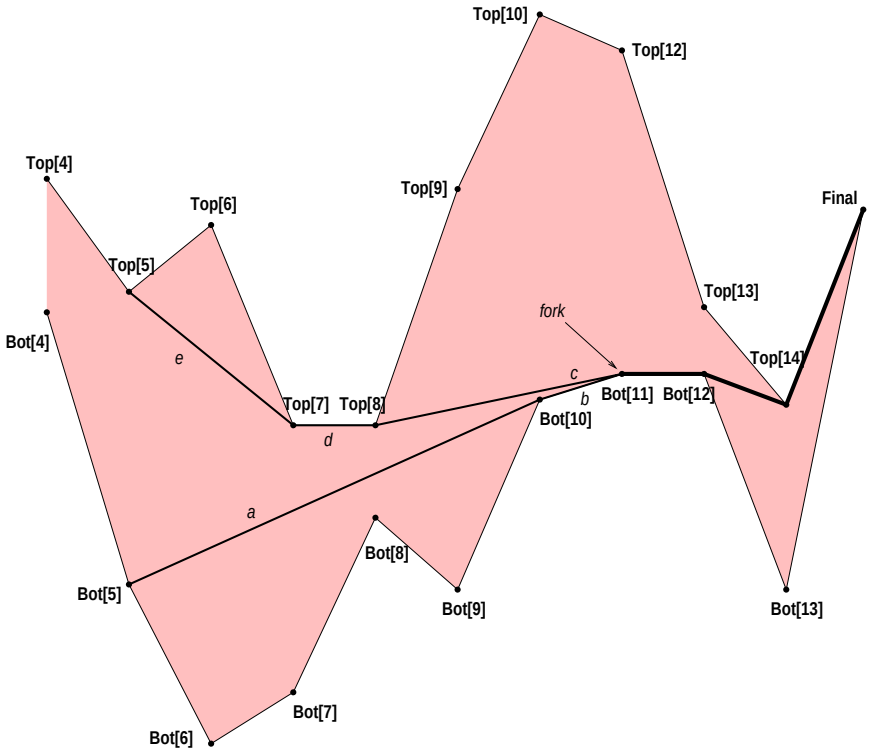



*Fig. 6.6* **Correspondence with Shortest Paths in Polygonal Channels.** Finding shortest paths in this polygonal channel from **Top**[5] to **Final** and from **Bot**[5] to **Final** corresponds to the problem of bit allocation in the reverse structure in Figure 6.3. The slopes of segments $a$, $b$, $c$, $d$, $e$ appear in decreasing order, as do the nominal quantization scales of the corresponding segments $seg[11]$, $seg[12]$, $seg[13]$ $seg[14]$, $seg[15]$ in Figure 6.3. The fork point in both cases is **Bot**[11].

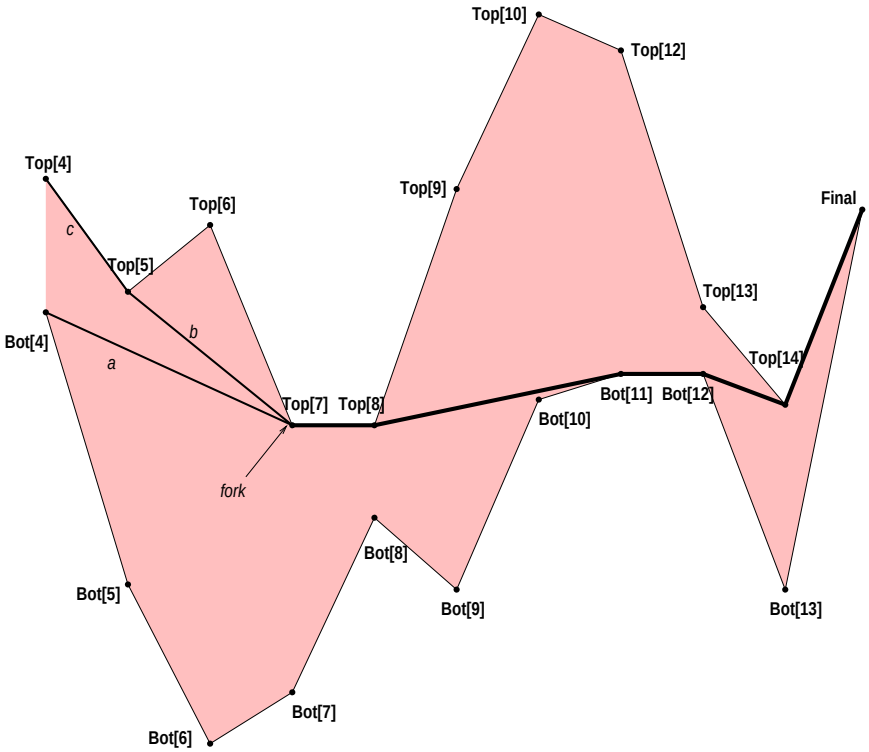



*Fig. 6.7* **Further Correspondence with Shortest Paths in Polygonal Channels.** Finding shortest paths in this polygonal channel from **Top**[4] to **Final** and from **Bot**[4] to **Final** corresponds to the problem of bit allocation in the reverse structure in Figure 6.4. The slopes of segments $a$, $b$, $c$ appear in decreasing order, as do the nominal quantization scales of the corresponding segments $seg[14]$, $seg[15]$, $seg[16]$ in Figure 6.4. The fork point in both cases is **Top**[7].

# 7

# Real-Time VBR Rate Control

The lexicographic bit allocation framework presented in Chapter 2 and the optimal algorithms of Chapters 3 and 4 assume *a priori* knowledge of the rate-distortion characteristics of the input video sequence. In practice, this assumption is met by processing the video sequence off-line with three computational passes. In the first pass, the rate-distortion characteristics of each picture are measured and modeled. In the second pass, an optimal global bit allocation sequence is computed. In the third and final pass, the computed bit allocation sequence is used to encode the video sequence. For some video applications, off-line processing is impractical. Examples of these applications include live broadcasting, digital VCR, and video-telephony. In these applications, all processing must be performed in real time and with low delay.

In this chapter, we address VBR encoding in a real-time environment requiring low encoding delay. Specifically, we present a new real-time VBR rate control algorithm that is based upon the optimal algorithm of Chapter 4. In the real-time algorithm, we skip the first two passes and perform bit allocation "on-the-fly" in a single encoding pass, while maintaining some of the desirable properties of the optimal algorithm.

In Section 7.1, we review the optimal VBR algorithm of Section 4.2. We then show and then show how to modify this algorithm to work in a single pass in Section 7.2. We present simulation results in Section 7.3 and discuss related work in Section 7.4.

## 7.1 OPTIMAL VBR BIT ALLOCATION ALGORITHM

Below is the algorithm of Section 4.2 for computing an optimal VBR allocation sequence that uses a total of $B_{\text{tgt}}$ bits.

1. Mark all pictures as *easy*. Let $B_{\text{easy}} \leftarrow B_{\text{tgt}}$.

2. Allocate $B_{\text{easy}}$ bits to easy pictures using a constant nominal quantizer. Let $Q_{\min}$ be the nominal quantizer used.

3. Simulate the VBV to identify *hard* segments of pictures. A hard segment leads to a buffer underflow when $Q_{\min}$ is used and consists of pictures that follow the most recent virtual overflow up to and including the picture the caused the underflow. After identifying a hard segment, reduce the bit allocation to the picture that caused the underflow to just prevent underflow. Reset the buffer fullness to empty and continue the simulation, adding new pictures to the existing hard segment if the buffer continues to underflow.

4. Allocate bits to each newly identified hard segment according to the optimal CBR algorithm, with a bit budget such that the underflow is just prevented. By preventing underflow in the hard segments, we are left with extra unallocated bits.

5. Let $B_{\text{hard}}$ be the total number of bits allocated to the hard pictures. Let $B_{\text{easy}} \leftarrow B_{\text{tgt}} - B_{\text{hard}}$.

6. If a new hard segment has been identified in Step 3, goto Step 2.

We make the following observations about the algorithm. The algorithm loops when a new hard segment has been detected. Since the number of hard segments is bounded by the length $N$ of the sequence, the algorithm terminates after at most $N$ iterations. With each iteration, $Q_{\min}$ is reduced. Halting the algorithm at the end of an iteration results in an optimal allocation sequence for the bit budget used. In effect, additional iterations only refine the allocation sequence to meet the original bit budget $B_{\text{tgt}}$.

## 7.2 SINGLE-PASS ALGORITHM

To transform the above algorithm to operate in a single pass, we need to remove any looping and off-line processing. This means that we can perform only one iteration. We cannot perform Step 2 which assumes knowledge of the rate-quantization characteristics of the entire sequence to be able to compute the value required for $Q_{\min}$. The identification of hard pictures in Step 3 requires looking into the future to determine whether the current picture

belongs to a hard segment. Finally, the optimal CBR algorithm in Step 4 is an off-line algorithm and needs to be replaced with an on-line algorithm.

The design of efficient algorithms for on-line CBR rate control has been taken up by many researchers, for example [7, 9, 61, 76]. We do not propose yet another CBR algorithm, but show how to modify an existing one to use in our VBR algorithm.

The limitation of a single iteration and the lack of knowledge of rate-quantization characteristics impact the bit budget and the choice of $Q_{\min}$. One approach is not to impose a bit budget and to set $Q_{\min}$ based upon the desired quality. This approach results in a final bit budget (and average bit rate) that depends upon the complexity of the input sequence. Another approach is to monitor and control the average bit rate to meet a specified target. Which approach is best to use depends upon the particular application. For example, storage applications that are capacity-limited would require the latter approach, whereas quality-conscious applications would favor the former. We first describe an algorithm to deliver video at a desired base quality and later show how to modify it to meet a desired bit budget.

### 7.2.1 Basic VBR Algorithm

The single-pass VBR algorithm below removes the off-line processing identified above and is suitable for low-complexity encoding with a specified base quality.

1. Initially the VBV buffer is set to full. In addition to the VBV buffer size $B_{\text{vbv}}$, peak input bit rate $R_{\max}$, and base quantization scale $Q_{\min}$, the algorithm has three extra parameters: a CBR trigger threshold $t_{\text{cbr}}$, a VBR trigger threshold $t_{\text{vbr}}$, and a target buffer fullness $B_{\text{tgt}}$, with $B_{\text{vbv}} > t_{\text{vbr}} \geq t_{\text{cbr}} > B_{\text{tgt}} > 0$.

2. The encoder simulates operation of the VBV and keeps track of the VBV buffer fullness $B_{\text{f}}$. The encoder operates in VBR mode and encodes the input pictures using $Q_{\min}$ until $B_{\text{f}} \leq t_{\text{cbr}}$. With this event, the encoder switches to CBR mode and allocates bits to the next $K$ pictures so that the target fullness $B_{\text{tgt}}$ would be reached after those pictures have been encoded. The parameter $K$ may correspond to the number of pictures in a small number of GOPs, for example. In CBR mode, the peak rate $R_{\max}$ is used as the target rate.

3. A CBR rate control algorithm, such as the one specified in TM5, can be used in CBR mode with one modification: the perceptually adjusted nominal quantization scale used to code any picture cannot be lower than $Q_{\min}$. The VBV buffer is to be operated near the target fullness $B_{\text{tgt}}$.

4. The encoder switches from CBR mode to VBR mode when $B_{\text{f}} > t_{\text{vbr}}$. When this occurs the base quantizer $Q_{\min}$ is again used.

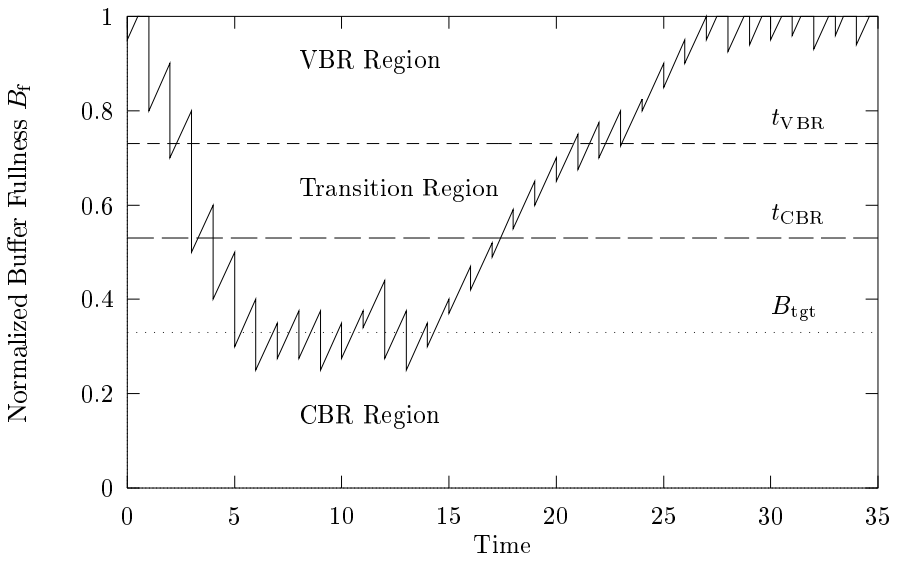



*Fig. 7.1* **Illustration of the Single-Pass VBR Algorithm.** This figure illustrates key points in the operation of the single-pass VBR algorithm. The algorithm initially operates in VBR mode, switches to CBR mode when $B_{\mathrm{f}} \leq t_{\mathrm{cbr}}$ and switches back to VBR mode when $B_{\mathrm{f}} > t_{\mathrm{vbr}}$.

The operation of the algorithm is illustrated with the aid of Figure 7.1. In the VBR region above $t_{\mathrm{vbr}}$, the algorithm operates in a constant-quality VBR mode. The CBR region below $t_{\mathrm{cbr}}$ marks where the algorithm operates in CBR mode at the peak input rate. The transition region provides hysteresis for transitions between VBR and CBR modes.

The single-pass VBR algorithm does not attempt to model the complexity of the coded pictures. Instead, it *reacts* to changes in complexity of pictures that are manifested as changes in the VBV buffer level. One implication of this reactive nature is a delay in switching between VBR and CBR modes compared to the optimal algorithm. Whereas the optimal algorithm switches to CBR mode when the buffer is full, the single-pass algorithm must wait until the buffer reaches $t_{\mathrm{cbr}}$ before switching. This leaves less buffering for the CBR-coded pictures. Another difference is that for segments with short spikes in complexity, the single-pass algorithm may switch to CBR coding when the optimal algorithm would continue to operate in VBR mode. The switching from CBR mode to VBR mode is less problematic. Because the CBR algorithm in Step 3 enforces a minimum nominal quantization scale of $Q_{\mathrm{min}}$, the switching to VBR mode can actually take place before the buffer reaches $t_{\mathrm{vbr}}$. In fact, it is this enforcement of the minimum nominal quantization scale that enables the buffer to fill up as the complexity of coded pictures decreases.

### 7.2.2 VBR Algorithm with Bit Budget

As presented above, the single-pass VBR algorithm does not attempt to control the coding rate to meet a specified bit budget. In a sense, meeting a total bit budget is a long-term process in which a target rate is specified over a long interval, whereas the basic VBR algorithm above can be viewed as a short-term process in which coding decisions are made in response to local variations in buffer state. A similar approach is taken by Ding [15]. Enforcing a bit budget is akin to specifying a constant bit rate with the rate averaged across the entire sequence, or a large portion of the sequence. In this context, a budget-constrained VBR algorithm can be viewed as performing the function of a CBR algorithm in which the encoder buffer is sized large enough to sufficiently smooth the variation in coding rate across a large portion of the sequence. In a practical VBR encoder, the size of the encoder buffer is fixed and limited. However, we can use this intuition to design a VBR rate control algorithm that incorporates a bit-budget constraint.

In the TM5 encoder model, a simple buffer-feedback mechanism is used to perform CBR rate control. In this scheme, a base quantizer scale is computed as a function of the fullness of a "virtual" encoder buffer that empties at a constant rate. Denoting the fullness of the virtual buffer as $B_{\mathrm{v}}$, the buffer-feedback function takes the form $Q = \mathit{clip}(31B_{\mathrm{v}}/r)$, where $r$ is a normalization factor and the function $\mathit{clip}(\ )$ clips the quantization scale to the range of $[1, 31]$. The base quantizer scale is then modulated with an *activity factor* that attempts to compensate for the difference in visibility of quantization errors among blocks with different levels of spatial detail.

For simulation, we use the TM5 buffer-feedback mechanism to perform the long-term rate control. However, instead of controlling the quantizer scale directly, we control $Q_{\min}$ with the feedback function: $Q = \mathit{clip}(31B_{\mathrm{v}}/r)$. The size of virtual buffer determines how quickly $Q_{\min}$ can change, and it also affects the accuracy of the rate control. Conceptually, the size of the virtual buffer need not be constant. For example, to come close to the specified bit budget, we can reduce the size of the buffer gradually to constrain the variance in rate near the end of the sequence. In the simulation results presented below, the virtual buffer size is held constant. The constant emptying rate of the virtual buffer is computed as the total bit budget divided by the number of pictures in the sequence.

## 7.3 SIMULATION RESULTS

We use the same simulation environment as described in Section 5.5. For the test sequence, we use the 3,660-frame video clip described in Section 5.7. We report results here for the following four algorithms: 1) Lexicographic VBR with hyperbolic-spline model, 2) TM5 CBR, 3) single-pass VBR without long-

*Table 7.1* **Summary of Single-Pass VBR Simulations.** This table summarizes the results of encoding simulations comparing the TM5 CBR, Lexicographic VBR, Open-Loop Single-Pass VBR, and Controlled Single-Pass VBR Rate Control Algorithms.

| Rate Control | PSNR (dB) | | Nominal Q | | Nominal Q | |
|---|---|---|---|---|---|---|
| Algorithm | Average | Std. Dev. | Average | Std. Dev. | Max | Min |
| TM5 CBR | 34.99 | 4.62 | 10.07 | 5.80 | 34.30 | 2.32 |
| Lexicographic VBR | 34.68 | 2.72 | 9.20 | 2.19 | 15.43 | 7.04 |
| Single-Pass VBR | 34.86 | 2.91 | 8.44 | 2.38 | 21.96 | 7.00 |
| Controlled VBR | 35.15 | 4.34 | 8.97 | 4.33 | 17.34 | 2.22 |

term rate control, and 4) single-pass VBR with long-term rate control. The simulation parameters are as described in Section 5.7.2.

For both single-pass VBR algorithms, the initial base quantization parameters were adjusted to give roughly the same encoded file size as TM5 and the following thresholds were used: $t_{\mathrm{vbr}} = 0.85B_{\mathrm{vbv}}$, $t_{\mathrm{cbr}} = 0.7B_{\mathrm{vbv}}$ and $B_{\mathrm{tgt}} = 0.3B_{\mathrm{vbv}}$. For the single-pass VBR algorithm with long-term rate control, the size of the virtual encoder buffer was set to $20B_{\mathrm{vbv}}$.

A summary of the simulation results is listed in Table 7.1. Plots of the buffer fullness, smoothed instantaneous bit rate, nominal quantization, and PSNR are shown in Figures 7.2–7.5. The results show that the VBR algorithms produce less variance in PSNR and nominal quantization compared with TM5 CBR. However, the average PSNR is actually better for TM5 CBR than for the VBR algorithms. We observe that TM5 gives better PSNR for scenes in the first half of the sequence where the the VBR algorithms operate in constant-quality mode and worse PSNR on for complex scenes in the second half. Since the majority of the pictures operate in constant-quality mode, the average PSNR is biased toward TM5. Visually however, the VBR encodings have more even quality and better scene transitions. Between the optimal VBR algorithm and the single-pass VBR algorithm without long-term rate control, the major differences can be attributed to the use of on-line versus off-line CBR algorithms. The single-pass VBR algorithm with rate control shows some similarity with TM5 CBR in long-term variations in nominal quantization, but with smoother local variations in nominal quantization and PSNR.

## 7.4 RELATED WORK

Reibman and Haskell [68] study encoder rate constraints in the context of an ATM network with a leaky-bucket policing function and propose a coding system in which the selections of channel rate and encoder rate are performed jointly. The proposed encoder rate control is based upon the Reference Model 8 (RM8) simulation encoder [4]. In order to prevent the use of pro-

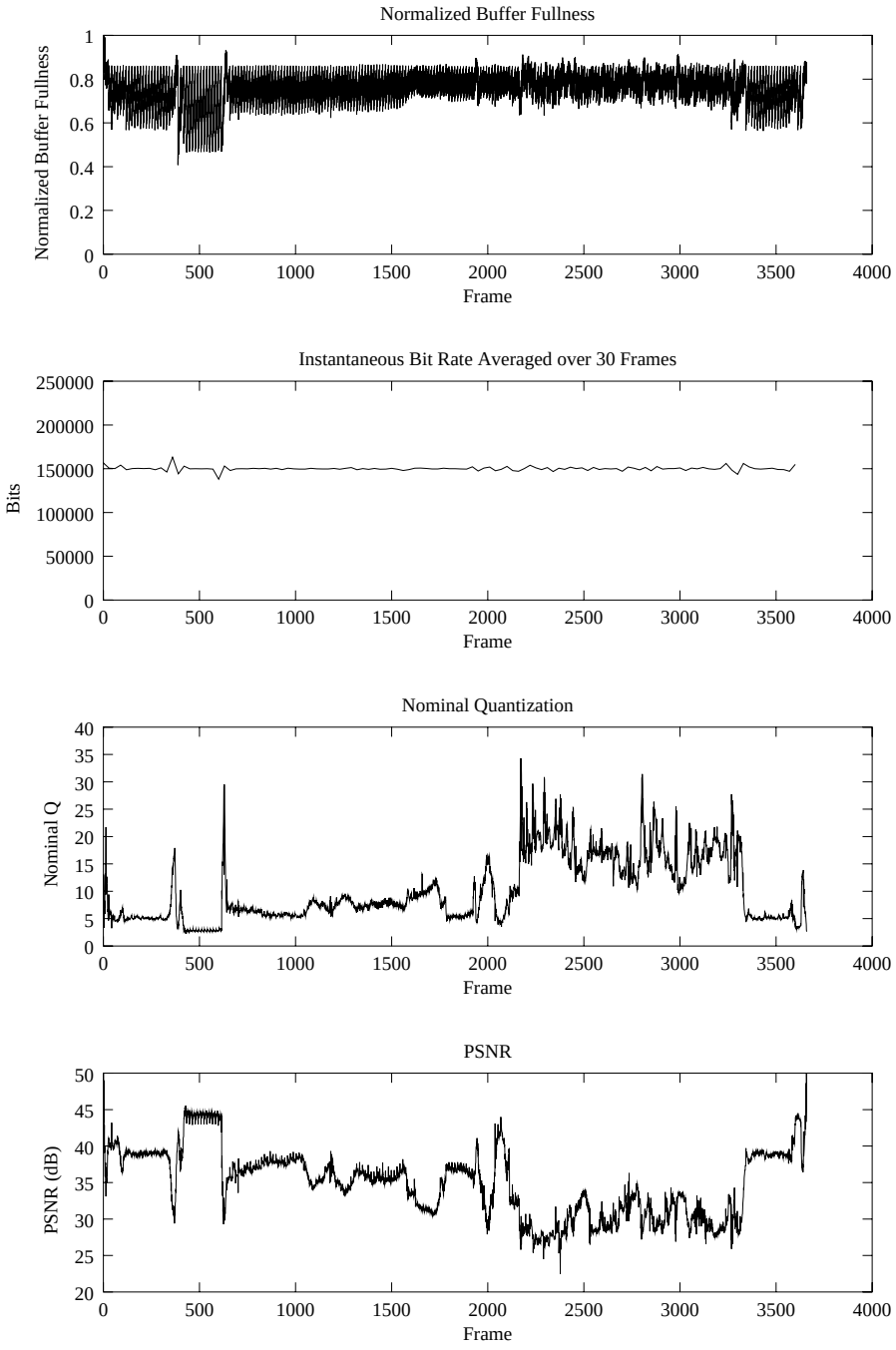



*Fig. 7.2* **Simulation Results for TM5 CBR Coder.**

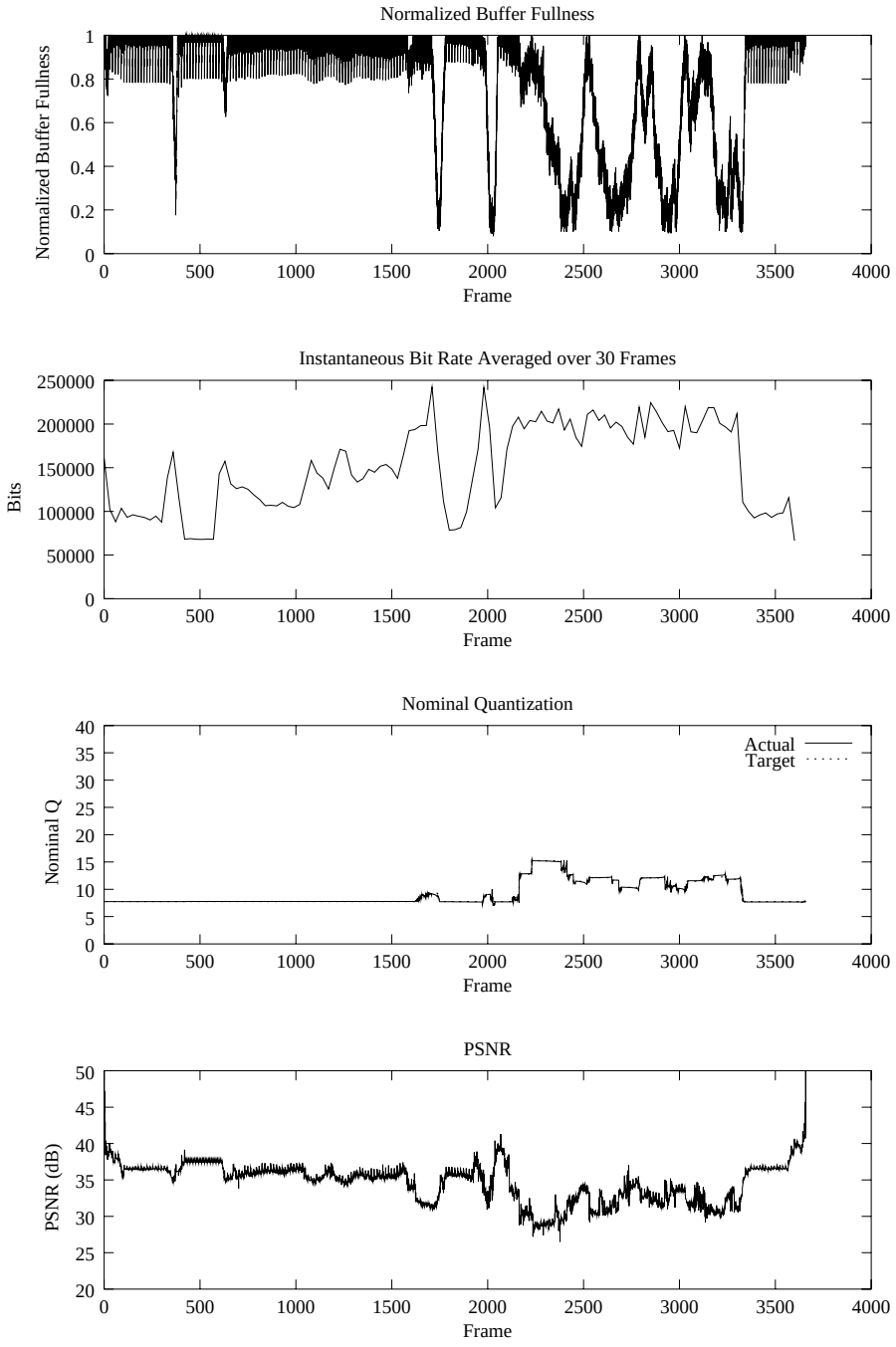



*Fig. 7.3* **Simulation Results for Lexicographic VBR Coder.**

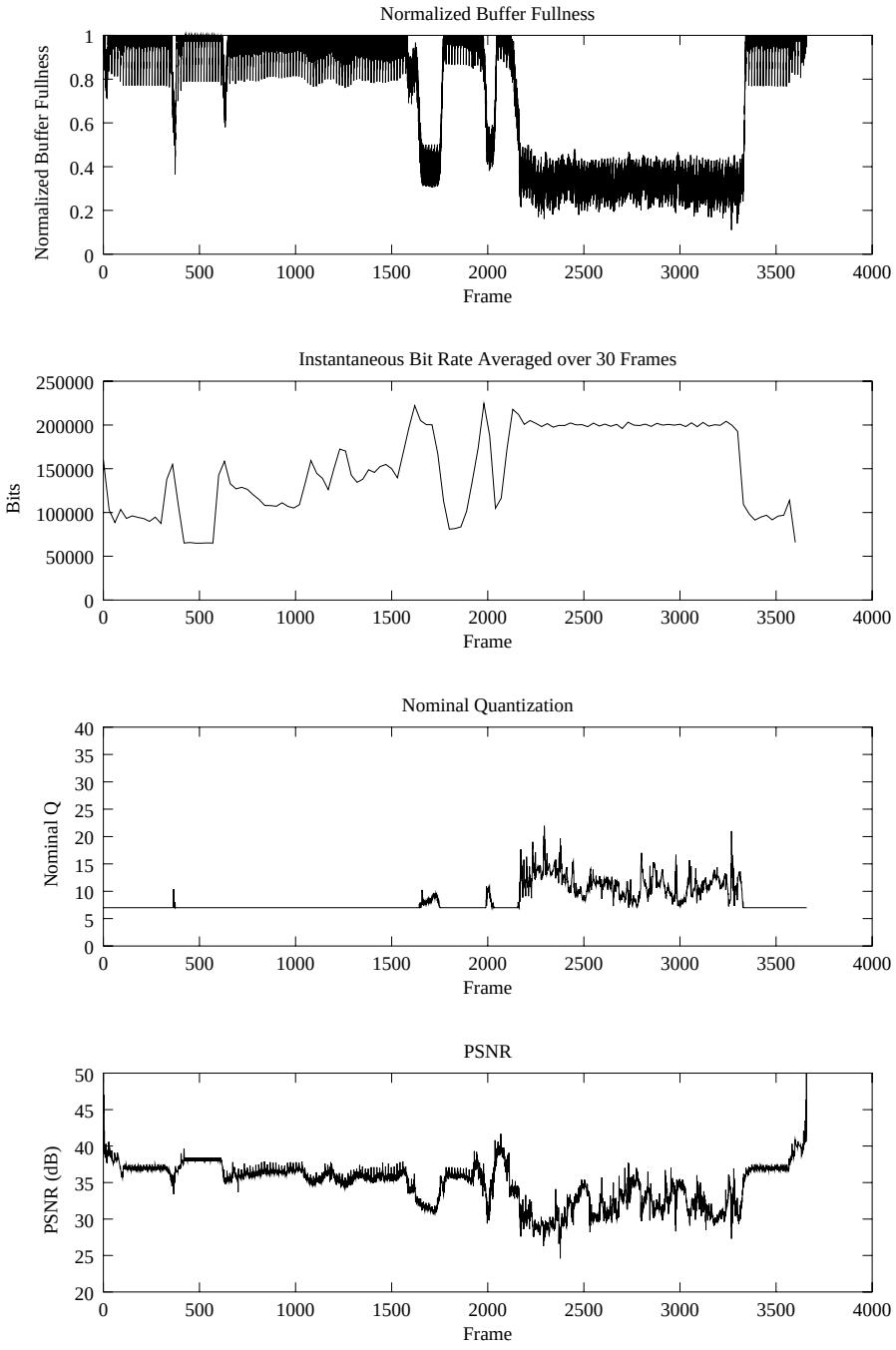



*Fig. 7.4* **Simulation Results for Open-Loop Single-Pass VBR Coder.**

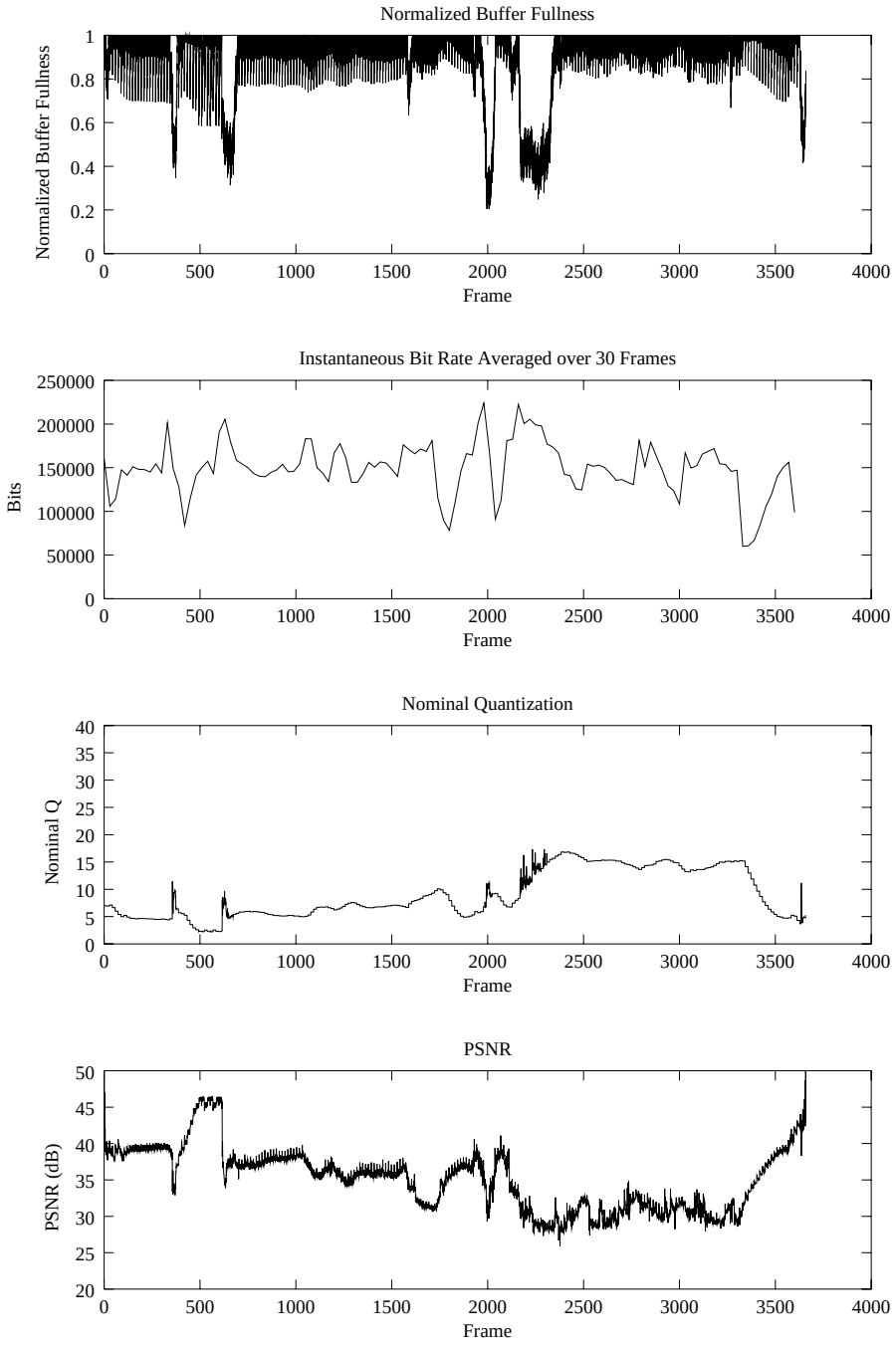



*Fig. 7.5* **Simulation Results for Controlled Single-Pass VBR Coder.**

gressively smaller quantizer scales for low-complexity segments of video, a minimum quantizer scale is enforced, assuming that the user has selected a maximum desired quality setting. The motivation for this modification is to save some rate for future peaks. The modified RM8 encoder rate control is similar to that specified in Step 3 of the single-pass VBR algorithm of Section 7.2.1.

The leaky-bucket policing mechanism ensures that the channel rate cannot be sustained above a specified average rate for a period of time determined by the size of the bucket. Such constraints are not considered in Equation (2.9). The "Greedy Leaky-Bucket Rate Control Algorithm" of [68] selects the highest allowed transmission rate in order to empty the encoder buffer as fast as possible. In the absence of leaky-bucket constraints, this policy corresponds to the MPEG-2 VBR mode described by Equation 2.9, where the decoder is filled as quickly as possible with the maximum transmission rate.

If we disregard the leaky-bucket constraints, the similarities and differences between the encoder rate control algorithm in [68] and the single-pass VBR algorithm of Section 7.2.1 become more evident. Since RM8 uses a feedback rate control mechanism whereby the quantizer scale is determined as a monotonically increasing function of the fullness of the encoder buffer, there is a buffer fullness $B^{\mathrm{e}}_{\mathrm{min}}$ that corresponds to $Q_{\mathrm{min}}$. When the encoder's buffer fullness falls below $B^{\mathrm{e}}_{\mathrm{min}}$, the modified RM8 algorithm is essentially operating in VBR mode with constant quantizer scale of $Q_{\mathrm{min}}$. Since the encoder's buffer mirrors the decoder's buffer in the case under consideration, the modified RM8 algorithm corresponds roughly to our single-pass VBR algorithm with $t_{\mathrm{vbr}} = t_{\mathrm{cbr}} = B_{\mathrm{vbv}} - B^{\mathrm{e}}_{\mathrm{min}}$. The modified RM8 algorithm does not have a corresponding $B_{\mathrm{tgt}}$.

Ding [15] also considers joint encoder and channel rate control over ATM networks. The proposed channel rate control basically performs bitstream smoothing, where the channel rate is determined as an average of the encoding rate of some number of past pictures. Encoder rate control is separated into two processes: *encoder instantaneous-rate control* and *encoder sustainable-rate control*. Encoder instantaneous-rate control also uses the concept of a minimum quantization parameter, labeled sequence $Q_{\mathrm{s}}$ in [15]. The encoder instantaneous-rate control increases the encoding quantization parameter above $Q_{\mathrm{s}}$ only when the upper bound on encoder bit rate is violated. The violation can be determined either by estimating the bit rate of the current picture or by performing a two-pass encoding. Again, if we disregard constraints imposed by ATM policing function, the encoder instantaneous-rate control roughly corresponds to our single-pass VBR algorithm with $t_{\mathrm{vbr}} = t_{\mathrm{cbr}} = E_i$, where $E_i$ is the estimated or computed number of bits to encode the current picture with $Q_{\mathrm{s}}$.

The encoder sustainable-rate control of [15] adjusts the sequence $Q_{\mathrm{s}}$ to adapt to the changing local statistics of the video sequence. The sequence $Q_{\mathrm{s}}$ is adjusted in discrete increments by monitoring changes in the average fullness

of the virtual buffer, which is defined to be the sum of the fullness of the encoder buffer and the leaky bucket.

Considering the previous works described above, our single-pass VBR algorithm, with or without long-term control of $Q_{\min}$, seems eminently suitable for encoder rate control in an ATM setting when coupled with an appropriate channel rate control algorithm.

## 7.5 CONCLUSION

We have presented a low-complexity VBR rate control algorithm suitable for low-delay, real-time encoding applications. The development of the algorithm is motivated by the lexicographic bit allocation framework. Although the algorithm is best suited to provide a desired quality level and can be used for short-term encoder rate control, it can also be applied with a suitable long-term rate control strategy to meet bit budget constraints. Results from a challenging encoding simulation shows that the new algorithm retains advantages of the more complex optimal algorithm in equalizing quality, especially for scene transitions.

# 8

# *Extensions of the Lexicographic Framework*

In Chapters 2–4, we laid a theoretical foundation for lexicographic bit allocation, and in Chapter 5 we demonstrated that the framework works well in practice. In this chapter, we provide evidence that the framework is also flexible and general by showing how it can be readily applied to other domains, extended to perform statistical multiplexing, and used in a discrete optimization setting.

## 8.1 APPLICABILITY TO OTHER CODING DOMAINS

While the lexicographic bit allocation framework was originally motivated and formulated for MPEG video coding, it can be applied equally well in other lossy coding domains for which buffer-constrained bit allocation is a valid problem and where a perceptual distortion measure needs to be equalized among coding units. With the increasing popularity and reach of the Internet, buffer-constrained bit allocation becomes relevant for compressed data that are *streamed* across a network. Obvious examples include lossy image coding (such as specified by the JPEG standard [62]), where the coding unit would logically be a block of pixels, and audio coding, where a coding unit might correspond to half a second of sampled sound.

## 8.2 MULTIPLEXING VBR STREAMS OVER A CBR CHANNEL

### 8.2.1 Introduction

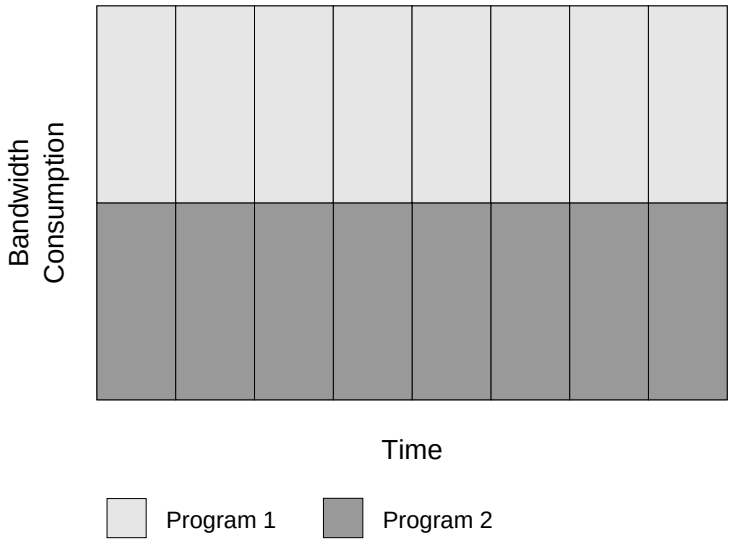
There are many scenarios where multiple compressed video streams are to be transmitted through a common channel. Two obvious examples are networked video and digital video broadcasting. In these types of applications, the transmission channel is typically bandwidth-limited. With the available bandwidth, we would like to provide as much video programming as possible without having to sacrifice quality.

An often-cited motivation for VBR video encoding is that VBR encoding can potentially allow for the simultaneous transmission of more video streams over a common channel than CBR encoding at the same quality level. The main reasoning is provided through a concept called *statistical multiplexing*. Statistical multiplexing is based upon the observation that the bit rate of constant-quality video is highly variable from frame to frame. In order to achieve image quality that is *not less* lexicographically than that of a constant-quality VBR encoding, a CBR encoding would require a bit rate that would correspond to the peak rate of the VBR encoding. Since a VBR encoding typically requires the peak rate for only a small percentage of time, it uses less bandwidth on average than a comparable CBR encoding. Furthermore, assuming that the VBR streams have independent bit rate characteristics, we can transmit more VBR streams than CBR streams over a common channel with a low probability that the combined instantaneous bit rate would exceed the channel rate.

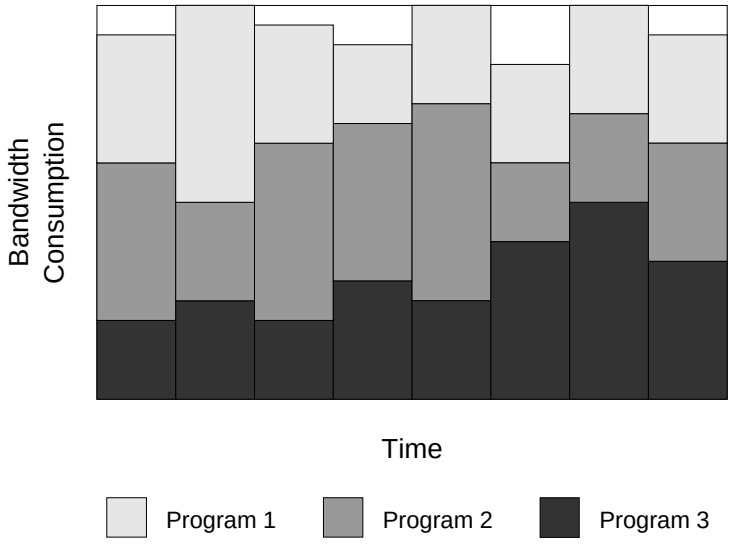
As an example, consider a channel with a bandwidth of 100 Mbits/sec. Suppose that for a desired level of quality, a peak rate of 10 Mbits/sec is required for coding a suite of video programming. Using CBR encoding, up to 10 sequences can be transmitted through the channel simultaneously. Since the peak rate is required only a small percentage of the time, suppose that the actual average rate is only 5 Mbits/sec. Then using VBR encoding with a peak rate of 10 Mbits/sec and average rate of 5 Mbits/sec, we can *potentially* transmit 20 simultaneous sequences. This would correspond to a *statistical multiplexing gain* of 2.

In order to transmit the 20 VBR sequences simultaneously, however, the instantaneous bit rate for the 20 sequences must not exceed the channel capacity for an extended period of time, which is determined by the amount of buffering present. Assuming that the bit rates of the different video streams are uncorrelated in time, there is a low probability that the channel capacity would be exceeded in any time interval. Quantifying and minimizing this probability are central themes of research.

The advantage of VBR encoding over CBR is illustrated through a simple example in Figure 8.1. In this example, three VBR encodings are shown multiplexed using at most the same bandwidth required by a CBR encoding of only two of the sources.




(a) Multiplexing of 2 CBR Bitstreams




(b) Multiplexing of 3 VBR Bitstreams

*Fig. 8.1* **Illustration of Statistical Multiplexing.** This figure shows an example of how three VBR bitstreams can be multiplexed into the same channel as two CBR bitstreams, yielding a statistical multiplexing gain of 1.5.

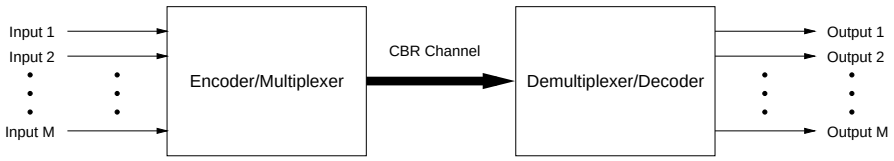



*Fig. 8.2* **Multiplexing Model.** This figure shows a block diagram of system for transmitting multiple sequences over a single channel. At the encoder, the video sources are encoded, multiplexed, and transmitted over a common channel. At the decoder, the compressed video streams are received, demultiplexed, and decoded.

In the remainder of this section, we will show how our basic lexicographic bit allocation framework can be readily extended to handle the multiplexing of multiple VBR bitstreams over a CBR channel. However, in contrast to typical statistical multiplexing techniques, as exemplified in Figure 8.1, our method allocates bits to the VBR bitstreams in a *deterministic* manner, making full use of all the available channel bandwidth.

In related work, a buffered rate control scheme for multiplexing VBR sources onto a CBR channel is described in [61]. This work is based upon the rate-distortion framework of [60] and [7] and uses a multiplexing model very similar to the one we are about to present. As described in the paper, the basic allocation unit is taken to be a GOP.

### 8.2.2 Multiplexing Model

We first elaborate a model for multiplexing multiple VBR bitstreams onto a CBR channel. Since our bit allocation framework is deterministic and uses lookahead, we assume that complete statistics of the multiple video sources are available to the bit allocation algorithm. This requirement can be met by providing a centralized encoder for the multiple sources, as depicted in Figure 8.2. In the figure, $M$ video sources enter a encoder/multiplexer that produces a single multiplexed stream for transport over a CBR channel. On the receiving end, a demultiplexer/decoder performs demultiplexing and decoding to reproduce the $M$ video sequences. This multiplexing model is similar to that proposed in [27].

This model is applicable to applications such as a video server where the video sequences to be multiplexed are known in advance. An especially noteworthy case is that of near-video-on-demand (NVOD), where a single sequence is to be transmitted simultaneously with different starting times. For example, 20 copies of a two-hour movie can be multiplexed so that the viewing of the movie can begin every six minutes.

The encoder/multiplexer block is expanded in Figure 8.3. As shown, the input video sources are encoded individually and time-division multiplexed and stored in a buffer before being output to the channel at a constant bit

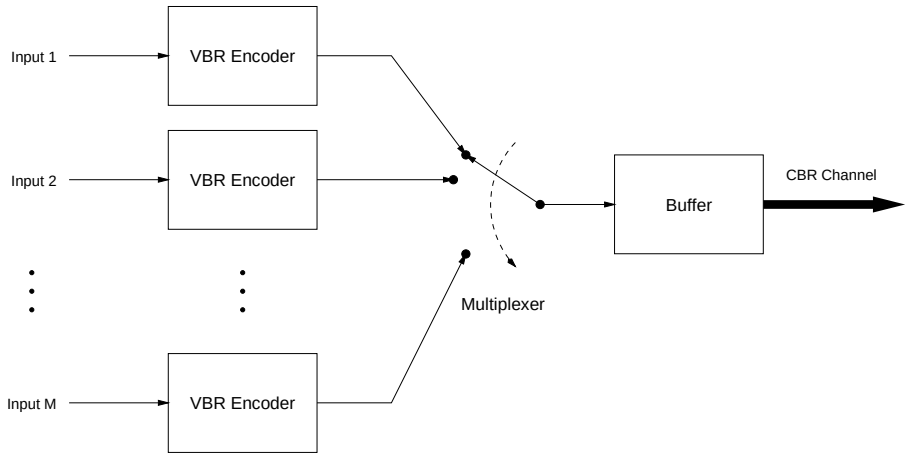



*Fig. 8.3* **Block Diagram of Encoder/Multiplexer.** This block diagram shows the time-division multiplexing of the output of several encoders, with the multiplexed output being stored in a single channel buffer.

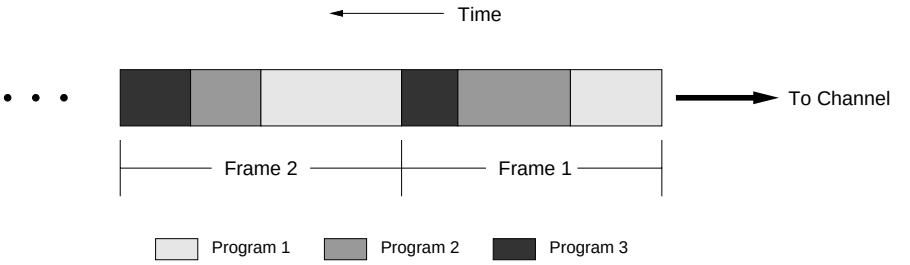



*Fig. 8.4* **Operation of Multiplexer.** This diagram illustrates the operation of a simple multiplexing model. In this multiplexing model, the compressed frames of video from the multiple sources are time-division multiplexed onto the CBR channel.

rate. The encoders are synchronized so that they output the encoding of a picture at the same time every $T$ seconds. The multiplexer then concatenates the multiple encodings in order as shown in Figure 8.4.

The demultiplexer/decoder block is expanded in Figure 8.5. The demultiplexer/decoder mirrors the operation of the encoder/multiplexer. Incoming bits from the channel are stored in a decoding buffer. Every $T$ seconds, the demultiplexer instantaneously removes from the buffer all bits needed to decode the next picture of all sequences and routes the bitstreams to the appropriate decoders, which then output the reconstructed video.

The multiplexing model described above resembles the operation of the single-stream encoder and decoder system implied by the MPEG Video Buffering Verifier. If we view the different input sources as providing "slices" of the

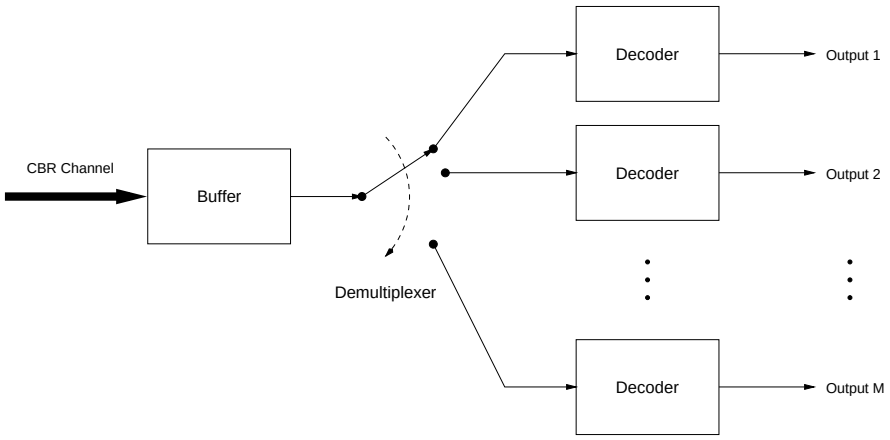



*Fig. 8.5* **Block Diagram of Demultiplexer/Decoder.** The demultiplexer and decoders shown in this diagram are used to separate and decode the multiplexed bitstreams produced by the encoder/multiplexer of Figure 8.3.

same picture, the resemblance would be very close indeed. This construction is intentional and allows us to apply the lexicographic framework to allocate bits optimally to the multiple VBR bitstreams.

### 8.2.3 Lexicographic Criterion

Before we can apply our lexicographic bit allocation framework, we need to define an optimality criterion. Since there are multiple sources, a lexicographical criterion based upon the encoding of a single video sequence is certainly not appropriate. We need to consider the quality of all the sequences. A simple way to do this is to consider the concatenation of all the sequences and define a lexicographic criterion on the concatenated video stream. By doing this, we are putting equal weight to each picture of each video sequence. We can consider extending the lexicographic criterion to compare vectors of length $M$ instead of scalar quantities. However, as we will see, this is equivalent to just considering the concatenation of the $M$ sequences.

### 8.2.4 Equivalence to CBR Bit Allocation

In this section, we show for the multiplexing model put forth above that the problem of optimal bit allocation for multiple VBR streams reduces to a CBR bit allocation problem for a single stream.

The multiplexing model follows the operation of the MPEG Video Buffering Verifier (VBV). We can view the buffer, the demultiplexer, and the bank of $M$ decoders in Figure 8.5 as comprising a single VBV. The lexicographic

framework of Chapter 2 can then be applied. Since the transmission channel operates at a constant bit rate, the CBR constraints of Chapter 3 would apply.

For display interval $i$, we need to consider the nominal quantization scales used to code picture $i$ of each of the $M$ video sequences. Given a fixed bit budget for coding picture $i$ of each video sequence, it is easy to show that the same nominal quantization scale must be used to code picture $i$ of all sequences to achieve lexicographic optimality; if the nominal quantization scales differ, we can always shift bits around to reduce the highest nominal quantization scale by increasing a lower nominal quantization scale. This result also holds if we formulate the lexicographic criterion using vectors of nominal quantization scales.

By using a combined bit-production model that is the sum of the bit-production models for the individual sequences, we can then allocate bits jointly to the sequences using the CBR algorithm of Chapter 3.

While the above technique guarantees that the buffer in Figure 8.5 does not overflow or underflow, it should be noted that doing so *does not* guarantee MPEG VBV compliance for the individual sequences, except when the individual VBV buffers are at least the size of the buffer in Figure 8.5. A multiplexing model that explicitly includes individual decoder buffers is certainly possible. However, analysis of this situation is not as straightforward as the above model and remains an open problem.

## 8.3 BIT ALLOCATION WITH A DISCRETE SET OF QUANTIZERS

One of the assumptions made in Chapter 2 is that there is a continuous relationship between quantization (distortion) and coding rate (number of bits used). As shown in Chapters 3 and 4, this assumption facilitates rigorous analysis of the buffer-constrained bit allocation problem under the lexicographic optimality criterion and results in an elegant characterization of the optimal solution. In order to apply directly the results of the analysis, we need to construct a continuous model of the relationship between quantization and coding rate. As demonstrated in Chapter 5, this can be done by gathering statistics during multiple encoding passes and fitting these to a chosen functional form. Because of the inevitable error in the modeling, some form of error recovery is needed, such as the scheme proposed in Chapter 5.

In most practical coders, however, both the set of available quantizers and the number of bits produced are discrete and finite. The problem of buffer-constrained bit allocation under these conditions has been examined by Ortega, Ramchandran, and Vetterli [60]. They provide a dynamic programming algorithm to find a CBR allocation sequence that minimizes a sum-distortion metric. In this section, we briefly describe their algorithm and show how it can be readily extended to perform lexicographic minimization.

### 8.3.1 Dynamic Programming

The dynamic programming algorithm described in [60] is based upon the Viterbi algorithm outlined in Section 1.8.4 for solving the budget-constrained bit allocation problem. To handle the additional buffer constraints, the buffer fullness is recorded at each state instead of the total number of bits used so far; for CBR coding, the number of bits used can be determined from the buffer fullness. We can use the recurrence equations in Section 2.4.1 to update the buffer fullness and create a trellis. Instead of pruning states that exceed a given bit budget, we instead prune states that overflow or underflow the buffer. At each stage in the construction of the trellis, we compare the current sum distortion associated with edges that enter a new state and record the minimum distortion along with a pointer to the source state. At the last stage of trellis construction, we identify the state with the minimum sum distortion and backtrack through the stored pointers to recover an optimal bit allocation sequence. Since an integral number of bits is generated, the maximum number of states that can be generated at each stage is equal to the size of the buffer. Therefore, with $M$ quantizers, $N$ pictures, and a buffer of size $B$, the dynamic programming algorithm of [60] requires $O(MBN)$ time to compute an optimal bit allocation sequence.

### 8.3.2 Lexicographic Extension

It is straightforward to modify the dynamic programming algorithm of [60] to perform lexicographic minimization. Instead of keeping track of a minimum sum distortion value, a scalar, we keep track of a lexicographic minimum, a vector. A naive implementation would store a vector of length $k$ for a state at the $k$th stage in the trellis, where the vector records the quantizers used for coding the first $k$ pictures. However, since the set of quantizers is finite and we are only concerned with the number of times a given quantizer is used and not with the order in which the quantizers are used, we only need to store $M$ values at each state, where $M$ is the number of quantizers. Each of these $M$ values count the number of times a given quantizer has been used to code the first $k$ pictures in an optimal path ending at the given state. Given two vectors of quantizer counts, a lexicographic comparison can be performed in $O(M)$ time. With this modification, we can find a lexicographically optimal bit allocation sequence in $O(M^2BN)$ time.

# *References*


1. V. R. Algazi, Y. Kato, M. Miyahara, and K. Kotani. Comparison of image coding techniques with a picture quality scale. In *Proceedings of the SPIE International Symposium on Electronic Imaging Science and Technology—Visual Communications and Image Processing*, volume 1771, pages 396–405, San Diego, July 1992.

2. V. Bhaskaran and K. Konstantinides. *Image and Video Compression Standards*. Kluwer Academic Publishers, Boston, 1995.

3. ITU-R Recommendation BT.601-5. Studio encoding parameters of digital television for standard 4:3 and wide-screen 16:9 aspect ratios, 1995.

4. CCITT. Description of reference model 8 (RM8), June 1989. Study Group XV—Document 525.

5. J.-J. Chen and H.-M. Hang. A transform video coder source model and its application. In *Proceedings of the IEEE International Conference on Image Processing*, volume 2, pages 967–971, 1994.

6. J.-B. Cheng and H.-M. Hang. Adaptive piecewise linear bits estimation model for MPEG based video coding. In *Proceedings of the IEEE International Conference on Image Processing*, volume 2, pages 551–554, 1995.

7. J. Choi and D. Park. A stable feedback control of the buffer state using the controlled lagrange multiplier method. *IEEE Transactions on Image Processing*, 3(5):546–557, September 1994.

8. P. A. Chou, T. Lookabaugh, and R. M. Gray. Entropy-constrained vector quantization. *IEEE Transactions on Signal Processing*, 37(1):31–42, January 1989.

9. K.-W. Chow and B. Liu. Complexity based rate control for MPEG encoder. In *Proceedings of the IEEE International Conference on Image Processing*, volume 1, pages 263–267, Austin, TX, November 1994.

10. K. W. Chun, K. W. Lim, H.D. Cho, and J. B. Ra. An adaptive perceptual quantization algorithm for video coding. *IEEE Transactions on Consumer Electronics*, 39(3):555–558, August 1993.

11. T.-Y. Chung, K.-H. Jung, Y.-N. Oh, and D.-H. Shin. Quantization control for improvement of image quality compatible with MPEG2. *IEEE Transactions on Consumer Electronics*, 40(4):821–825, November 1994.

12. G. Cicalini, L Favalli, and A. Mecocci. Dynamic psychovisual bit allocation for improved quality bit rate in MPEG-2 transmission over ATM links. *Electronic Letters*, 32(4):370–371, February 1996.

13. W. Ciciora, J. Farmer, and D. Large. *Modern Cable Television Technology: Video, Voice, and Data Communications.* Morgan Kaufman, San Francisco, 1999.

14. T. M. Cover and J. A. Thomas. *Elements of Information Theory.* Wiley-Interscience, New York, 1991.

15. W. Ding. Joint encoder and channel rate control of VBR video over ATM networks. *IEEE Transactions on Circuits and Systems for Video Technology*, 7(2):266–278, April 1997.

16. W. Ding and B. Liu. Rate control of MPEG video coding and recording by rate-quantization modeling. *IEEE Transactions on Circuits and Systems for Video Technology*, 6(1):12–20, February 1996.

17. H. Everett. Generalized langrange multiplier method for solving problems of optimum allocation of resources. *Operation Research*, 11:399–417, 1963.

18. G. D. Forney. The Viterbi algorithm. *Proceedings of the IEEE*, 61:268–278, March 1973.

19. J. E. Fowler and S. C. Ahalt. Differential vector quantization of real-time video. In *Proceedings of the IEEE Data Compression Conference*, pages 205–214, Snowbird, UT, March 1994. IEEE Computer Society Press.

20. J. Friedman, J. Hershberger, and J. Snoeyink. Efficiently planning compliant motion in the plane. *SIAM Journal on Computing*, 25(3):562–599, June 1996.

21. A. Gersho and R. M. Gray. *Vector Quantization and Signal Compression.* Kluwer Academic Press, Boston, 1992.

22. J. B. Ghosh. Siting facilities along a line when equity of service is desirable. *Journal of Operation Research Society*, 47(3):435–445, March 1996.

23. ITU-T Recommendation H.261. Video codec for audiovisual services at $p \times 64$ kbit/s, 1990. Revised at Helsinki, March 1993.

24. ITU-T Recommendation H.262 | ISO/IEC 13818-2. Generic coding of moving pictures and associated audio information: Video, 1995.

25. ITU-T Recommendation H.263. Video coding for low bit rate communication, 1996. Revised 1998.

26. B. G. Haskell, A. Puri, and A. N. Netravali. *Digital Video: An Introduction to MPEG–2.* Chapman & Hall, New York, 1997.

27. B. G. Haskell and A. R. Reibman. Multiplexing of variable rate encoded streams. *IEEE Transactions on Circuits and Systems for Video Technology*, 4(4):417–424, August 1994.

28. J. Hershberger and S. Suri. Off-line maintenance of planar configurations. *Journal of Algorithms*, 21(3):453–475, November 1996.

29. D. T. Hoang, E. Linzer, and J. S. Vitter. Lexicographic bit allocation for MPEG video. *Journal of Visual Communication and Image Representation*, 8(4):384–404, December 1997. Special issue on high-fidelity media processing.

30. D. T. Hoang and E. N. Linzer. *Motion Video Compression System with Buffer Empty/Fill Look-Ahead Bit Allocation.* United States Patent No. 5,719,632, IBM, February 17, 1998.

31. D. T. Hoang and J. S. Vitter. Multiplexing VBR video sequences onto a CBR channel with lexicographic optimization. In *Proceedings of the IEEE International Conference on Image Processing*, volume 4, pages 101–110, Santa Barbara, CA, October 1997.

32. C.-Y. Hsu, A. Ortega, and A. R. Reibman. Joint selection of source and channel rate for VBR video transmission under ATM policing constraints. *IEEE Journal on Selected Areas in Communications*, pages 1016–1028, August 1997.

33. T. Ibaraki and N. Katoh. *Resource Allocation Problems.* MIT Press, Cambridge, MA, 1988.

34. ISO-IEC/JTC1/SC29/WG11/N0400. Test model 5, April 1993. Document AVC-491b, Version 2.

35. ISO/IEC 11172-2. Coding of moving pictures and associated audio for digital storage media at up to about 1.5 mbits/s, 1993.

36. ISO/IEC 14496-2. Generic coding of audio-visual objects: Part 2—Visual, 1995.

37. A. K. Jain. *Fundamentals of Digital Image Processing*. Prentice-Hall, Englewood Cliffs, NJ, 1989.

38. J. R. Jain and A. K. Jain. Displacement measurement and its application in interframe coding. *IEEE Transactions on Communications*, COM-29(12):1799–1808, 1981.

39. N. S. Jayant, J. Johnson, and R. Safranek. Signal compression based on models of human perception. *Proceedings of the IEEE*, 81:1385–1422, October 1993.

40. R. S. Klein, H. Luss, and D. R. Smith. Lexicographic minimax algorithm for multiperiod resource allocation. *Mathematical Programming*, 55(2):213–234, June 1992.

41. D. Le Gall. MPEG: A video compression standard for multimedia applications. *Communications of the ACM*, 34(4):46–58, April 1991.

42. D. T. Lee and F. P. Preparata. Euclidean shortest paths in the presence of rectilinear barriers. *Networks*, 14(3):393–410, 1984.

43. D. W. Lin and J.-J. Chen. Efficient bit allocation under multiple constraints on cumulated rates for delayed video coding. In J. Biemond and E. J. Delp, editors, *Proceedings of the SPIE International Symposium on Electronic Imaging Science and Technology—Visual Communications and Image Processing*, volume 3024, pages 1370–1381, February 1997.

44. F.-H. Lin and R. M. Mersereau. An optimization of MPEG to maximize subjective quality. In *Proceedings of the IEEE International Conference on Image Processing*, volume 2, pages 547–550, 1995.

45. L.-J. Lin, A. Ortega, and C.-C. J. Kuo. Gradient-based buffer control techniques for MPEG. In *Proceedings of the SPIE International Symposium on Electronic Imaging Science and Technology—Visual Communications and Image Processing*, Taipei, Taiwan, May 1995.

46. L.-J. Lin, A. Ortega, and C.-C. J. Kuo. A gradient-based rate control algorithm with applications to MPEG video. In *Proceedings of the IEEE International Conference on Image Processing*, volume 2, Washington, D.C., October 1995.

47. L.-J. Lin, A. Ortega, and C.-C. J. Kuo. Cubic spline approximation of rate and distortion functions for MPEG video. In V. Bhaskaran, F. Sijstermans, and S. Panchanathan, editors, *Proceedings of the SPIE International Symposium on Electronic Imaging Science and Technology—Digital Video Compression*, volume 2668, pages 169–180, February 1996.

48. L.-J. Lin, A. Ortega, and C.-C. J. Kuo. Rate control using spline-interpolated R-D characteristics. In *Proceedings of the SPIE International Symposium on Electronic Imaging Science and Technology—Visual Communications and Image Processing*, 1996.

49. M. Liou. Overview of the $p \times 64$ kbit/s video coding standard. *Communications of the ACM*, 34(4):60–63, April 1991.

50. M. Luptacik and F. Turnovec. Lexicographic geometric programming. *European Journal of Operational Research*, 51(2):259–269, March 1991.

51. H. Luss and S. K. Gupta. Allocation of effort resources among competitive activities. *Operation Research*, 23:360–366, 1975.

52. E. Marchi and J. A. Oviedo. Lexicographic optimality in the multiple objective linear programming. The nucleolar solution. *European Journal of Operational Research*, 57(3):355–359, March 1992.

53. A. W. Marshall and I. Olkin. *Inequalities: Theory of Majorization and its Applications*. Academic Press, New York, 1979.

54. A. A. Melkman. On-line construction of the convex hull of a simple polyline. *Information Processing Letters*, 25(1):11–12, April 1987.

55. J. L. Mitchell, W. B. Pennebaker, C. E. Fogg, and D. J. LeGall, editors. *MPEG Video Compression Standard*. Chapman & Hall, New York, 1997.

56. MPEG Software Simulation Group. MPEG-2 encoder/decoder version 1.2, July 19 1996. URL: `http://www.mpeg.org/MSSG`.

57. A. N. Netravali and B. G. Haskell. *Digital Pictures: Representation, Compression, and Standards*. Plenum Press, New York, second edition, 1995.

58. W. Ogryczak. On the lexicographic minimax approach to location–allocation problems. Technical Report IS - MG 94/22, Université Libre de Bruxelles, December 1994.

59. L. A. Olzak and J. P. Thomas. Seeing spatial patterns. In K. Boff, L. Kaufman, and J. Thomas, editors, *Handbook of Perception and Human Performance*. Wiley, New York, 1986.

60. A. Ortega, K. Ramachandran, and M. Vetterli. Optimal trellis-based buffered compression and fast approximations. *IEEE Transactions on Image Processing*, 3(1):26–40, January 1994.

61. D. Park and K. Kim. Buffered rate-distortion control of MPEG compressed video channel for DBS applications. In *Proceedings of the International Conference on Communications*, volume 3, pages 1751–1755, 1995.

62. W. B. Pennebaker and J. L. Mitchell. *JPEG—Still Image Data Compression Standard*. Van Nostrand Reinhold, New York, 1993.

63. M. R. Pickering and J. F. Arnold. A perceptually efficient VBR rate control algorithm. *IEEE Transactions on Image Processing*, 3(5):527–532, September 1994.

64. A. Premoli and W. Ukovich. Piecewise lexicographic programming. A new model for practical decision problems. *Journal of Optimization Theory and Applications*, 72(1):113–142, January 1992.

65. A. Puri and R. Aravind. Motion-compensated video coding with adaptive perceptual quantization. *IEEE Transactions on Circuits and Systems for Video Technology*, 1(4):351–361, December 1991.

66. K. Ramachandran, A. Ortega, and M. Vetterli. Bit allocation for dependent quantization with applications to multiresolution and MPEG video coders. *IEEE Transactions on Image Processing*, 3(5):533–545, September 1994.

67. R. K. Rao and Z. S. Bojkovic. *Packet Video Communications over ATM Networks*. Prentice Hall PTR, Upper Saddle River, NJ, 2000.

68. A. R. Reibman and B. G. Haskell. Constraints on variable bit-rate video for ATM networks. *IEEE Transactions on Circuits and Systems for Video Technology*, 2(4):361–372, December 1992.

69. J. D. Salehi, Z.-L. Zhang, J. F. Kurose, and D. Towsley. Supporting stored video: Reducing rate variability and end-to-end resource requirements through optimal smoothing. In *Procedings of ACM SIGMETRICS Joint International Conference on Measurement and Modeling of Computer Systems*, pages 222–231, Philadelphia, May 1996.

70. G. M. Schuster and A. K. Katsaggelos. *Rate-Distortion Based Video Compression: Optimal Video Frame Compression and Object Boundary Encoding*. Kluwer Academic Publishers, Boston, 1997.

71. C. E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423, July 1948.

72. Y. Shoham and A. Gersho. Efficient bit allocation for an arbitrary set of quantizers. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 36(9):1445–1453, September 1988.

73. M.-T. Sun and A. R. Reibman, editors. *Compressed Video over Networks*. Marcel Dekker, New York, 2001.

74. D. Turaga and T. Chen. Fundamentals of video compression: H.263 as an example. In M.-T. Sun and A. R. Reibman, editors, *Compressed Video over Networks*. Marcel Dekker, New York, 2001.

75. K. M. Uz, J. M. Shapiro, and M. Czigler. Optimal bit allocation in the presence of quantizer feedback. In *Proceedings of the IEEE International Conference on Acoustics Speech and Signal Processing*, volume 5, pages 385–388, 1993.

76. E. Viscito and C. Gonzales. A video compression algorithm with adaptive bit allocation and quantization. In *Proceedings of the SPIE International Symposium on Electronic Imaging Science and Technology—Visual Communications and Image Processing*, volume 1605, pages 205–216, November 1991.

77. A. J. Viterbi and J. K. Omura. *Principles of Digital Communication and Coding*. McGraw-Hill, New York, 1979.

78. X. Wang, S. M. Shende, and K. Sayood. Online compression of video sequences using adaptive VQ codebooks. In *Proceedings of the IEEE Data Compression Conference*, pages 185–194, Snowbird, UT, March 1994. IEEE Computer Society Press.

79. S. J. P. Westen, R. L. Lagendijk, and J. Biemond. Perceptual optimization of image coding algorithms. In *Proceedings of the IEEE International Conference on Image Processing*, volume 2, pages 69–72, 1995.

# *About the Authors*

**Dzung Tien Hoang** was born on April 20, 1968, in Nha Trang, Vietnam. He immigrated to the United States in 1975 with his family and settled in New Orleans, Louisiana.

After graduating in 1986 from the Louisiana School for Math, Science and the Arts, a public residential high school in Natchitoches, Louisiana, he attended Tulane University in New Orleans with a Dean's Honor Scholarship and graduated in 1990 with Bachelor of Science degrees in Electrical Engineering and Computer Science, both with Summa Cum Laude honors.

He joined the Department of Computer Science at Brown University in Providence, Rhode Island, in 1990 under a University Fellowship and later under a National Science Foundation Graduate Fellowship. He received a Master of Science in Computer Science from Brown in 1992 and a Doctor of Philosophy in Computer Science from Brown in 1997. From 1993 to 1996, he was a visiting scholar and a research assistant at Duke University in Durham, North Carolina. From 1991 to 1995, he spent summers working at the Frederick National Cancer Research Facility, the Supercomputing Research Center, and the IBM T. J. Watson Research Center.

In August 1996, he joined Digital Video Systems, in Santa Clara, California, as Senior Software Engineer, where he developed algorithms for video compression. From October 1997 to May 2000, he served as Senior Software Systems Engineer and System-On-Chip Software Manager in the Semiconductor Business Division of Sony Electronics, Inc. He is currently Video Algorithm Development Manager at iCompression, a subsidiary of GlobeSpan.

**Jeffrey Scott Vitter** was born on November 13, 1955, in New Orleans, Louisiana. He received a Bachelor of Science degree with Highest Honors in Mathematics from the University of Notre Dame in 1977, and a Doctor of Philosophy degree in Computer Science from Stanford University in 1980. He was on the faculty at Brown University from 1980 until 1993. He is currently the Gilbert, Louis, and Edward Lehrman Professor of Computer Science at Duke University, where he served during 1993–2001 as Chair of the Department of Computer Science. He is also Co-Director and a Founding Member of the Center for Geometric and Biological Computing at Duke.

Prof. Vitter is a Guggenheim Fellow, an ACM Fellow, an IEEE Fellow, an NSF Presidential Young Investigator, a Fulbright Scholar, and an IBM Faculty Development Awardee. He is co-author of the book *Design and Analysis of Coalesced Hashing*, co-editor of the collections *External Memory Algorithms* and *Algorithm Engineering*, and is co-holder of patents in the areas of external sorting, prediction, and approximate data structures. He has written numerous articles and has consulted frequently. He serves or has served on the editorial boards of *Algorithmica*, *Communications of the ACM*, *IEEE Transactions on Computers*, *Theory of Computing Systems* (formerly *Mathematical Systems Theory: An International Journal on Mathematical Computing Theory*), and *SIAM Journal on Computing*, and has been a frequent editor of special issues.

Prof. Vitter is on the Board of Directors of the Computing Research Association, where he co-chairs the Government Affairs Committee. He has served as Chair of ACM SIGACT, the Special Interest Group on Algorithms and Computation Theory of the world's largest computer professional organization, the Association for Computing Machinery, and as a member of the Executive Council of the European Association for Theoretical Computer Science (EATCS). He has spent sabbaticals at the the Mathematical Sciences Research Institute in Berkeley, at Institut National de Recherche en Informatique et en Automatique (I.N.R.I.A.) in Rocquencourt, France, at Ecole Normale Supérieure in Paris, at Lucent Technologies Bell Laboratories in Murray Hill, New Jersey, and most recently at I.N.R.I.A. in Sophia Antipolis, France. He is an associate member of the Center of Excellence in Space Data and Information Sciences and an adjunct professor at Tulane University in New Orleans.

# *Index*