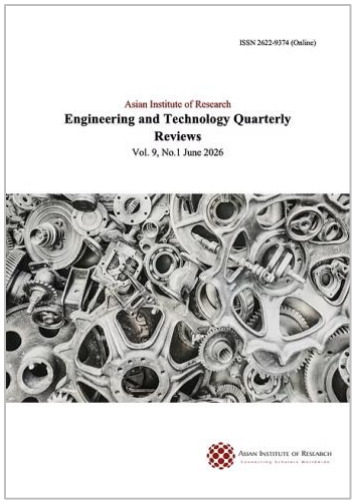

Engineering and Technology Quarterly Reviews

Gunawardhana, L. K. P. D. (2026), Enhancing Video Compression Efficiency for Low-Bandwidth Environments with H.265/HEVC. In: *Engineering and Technology Quarterly Reviews*, Vol.9, No.1, 26-35.

ISSN 2622-9374

The online version of this article can be found at:
https://www.asianinstituteofresearch.org/

Published by:
The Asian Institute of Research

The Asian Institute of Research *Engineering and Technology Quarterly Reviews* is a peer-reviewed International Journal. The journal covers scholarly articles in the fields of Engineering and Technology, including (but not limited to) Civil Engineering, Informatics Engineering, Environmental Engineering, Mechanical Engineering, Industrial Engineering, Marine Engineering, Electrical Engineering, Architectural Engineering, Geological Engineering, Mining Engineering, Bioelectronics, Robotics and Automation, Software Engineering, and Technology. As the journal is Open Access, it ensures high visibility and an increase in citations for all research articles published. The *Engineering and Technology Quarterly Reviews* aims to facilitate scholarly work on recent theoretical and practical aspects of Education.

ASIAN INSTITUTE OF RESEARCH
Connecting Scholars Worldwide

ASIAN INSTITUTE OF RESEARCH
Connecting Scholars Worldwide

The Asian Institute of Research
Engineering and Technology Quarterly Reviews
Vol.9, No.1, 2026: 26-35
ISSN 2622-9374

Enhancing Video Compression Efficiency for Low-Bandwidth Environments with H.265/HEVC

L. K. Pulasthi Dhananjaya Gunawardhana[1]

[1] Department of Information & Communication Technology, University of Sri Jayewardenepura, Pitipana, Sri Lanka. ORCID ID: 0000-0003-3486-7844

Abstract
This paper explores the functionality of the H.265/HEVC (High Efficiency Video Coding) standard in low-bandwidth scenarios. We provide an overview of H.265's key features and mechanisms that make it suitable for lower bandwidth environments. H.265, also known as High-Efficiency Video Coding (HEVC), is renowned for delivering superior video quality at lower bitrates. We investigate the critical features of H.265 and its application in low-bandwidth scenarios, providing insights into its efficiency, performance, and practical implementation. We present experimental results demonstrating the performance improvements and benefits of H.265 regarding video quality and bandwidth utilisation. The paper discusses potential applications and directions for optimising video compression in constrained network conditions.

Keywords: H.265, Low Latency, Constant Bit Rate, Variable Bit Rate

1. Introduction

The primary objective of this research is to investigate and develop algorithms that minimise latency during video streaming by employing innovative techniques. This research is not just theoretical but also has significant practical implications for real-world applications. It explores multiple aspects: streaming size minimisation, video quality preservation, and mobile device constraints. The research aims to provide a holistic approach to enhancing the real-time streaming experience by addressing these factors.

Applications such as live streaming, video conferencing, and mobile multimedia services require low-latency transmission while maintaining acceptable visual fidelity. The H.265/High Efficiency Video Coding (HEVC) standard was developed to address these challenges, offering approximately 50% bitrate reduction compared to H.264 for equivalent perceptual quality (Sullivan et al., 2012). Despite these advantages, latency and bandwidth optimisation remain critical challenges in real-world deployments.

The proposed algorithm aims to preserve the quality of the output video in addition to reducing streaming size. It is essential to ensure that the frame concealment technique has a minimal negative impact on the video, although certain assumptions may need to be made to achieve this objective. Furthermore, this research acknowledges the computational complexity limitations of mobile devices and considers them while developing the algorithm.

The first compression seen in the H.26x generation is the H.261 compression scheme. H.261 is the International Telecommunication Union (ITU) T video coding standard approved in November 1988. H.261 was initially developed for transmission over ISDN lines, and the data rate is a multiple of 64 kbps. The encoding algorithm is designed to work with video bitrates from 40 kbps to 2 Mbps. H.261 supports two video frame sizes: CIF (352x288 luminance with 176x144 chroma) and QCIF (176x144 with 88x72 chroma) with 4:2:0 sampling. There is also a backwards-compatible trick to transfer still images at 704x576 luminance resolution and 352x288 chrominance resolution. The H264 standard was developed by the ITU-T Video Coding Expert Group (VCEG) of the 16th Study Group with ISO / IEC JTC1 Moving Picture Experts Group (MPEG)

2. Bandwidth-Adaptive Streaming

H.265's adaptive streaming capabilities enable it to adjust the video bitrate in real time based on the available bandwidth. The H.26x family of video compression standards has evolved to address increasing demands for compression efficiency and transmission quality. Early standards such as H.261 were designed for ISDN-based communication, supporting limited resolutions and bitrates (Hanzo et al., 2007). Subsequent standards introduced improved motion compensation and entropy coding techniques, culminating in the H.265/HEVC standard. HEVC incorporates advanced features such as larger coding tree units, improved intra-prediction, and enhanced entropy coding, enabling superior compression performance in bandwidth-constrained environments (Sze et al., 2014). This is particularly crucial for users with limited access to high-speed networks.

- **Encoding Latency**: The time taken to encode a video frame using the H.265 codec can significantly impact overall latency. The research focuses on developing techniques to minimise the encoding latency while ensuring efficient and high-quality compression. This can involve exploring parallel processing, algorithmic optimisations, or hardware acceleration to expedite the encoding process.
- **Transmission Latency**: The time taken to transmit video frames from the server to the client over the network introduces additional latency. The research investigates methods to reduce transmission latency, such as optimising network protocols, leveraging adaptive streaming techniques, or implementing efficient data compression and packetisation strategies.
- **Decoding and Playback Latency**: The time taken to decode and render the received video frames on the client side also contributes to overall latency. The research considers approaches to streamline the decoding and playback process, including efficient buffer management, synchronisation techniques, or hardware-accelerated decoding to achieve real-time playback with minimal delay.
- **End-to-End Latency Optimisation**: Latency reduction in video streaming involves considering the entire end-to-end pipeline, from the source video to its final display on the viewer's device. The research focuses on holistic approaches that minimise latency across all stages, including encoding, transmission, decoding, and rendering. This may involve designing and implementing innovative algorithms, protocols, or mechanisms that optimise the end-to-end latency while maintaining video quality.

3. Video Quality Preservation

Ensuring high video quality is a critical objective in video streaming applications as it directly influences viewer engagement and satisfaction. This research focuses on developing an algorithm that effectively preserves video quality while minimising the streaming size through innovative frame concealment techniques. The proposed algorithm aims to conceal lost or dropped frames without introducing noticeable artefacts or compromising the overall visual fidelity of the streamed video content.

To achieve the preservation of video quality, the algorithm incorporates several essential considerations:

- **Minimisation of Artefacts**: The frame concealment process minimises introducing artefacts or distortions that may degrade the perceived video quality. Artefacts such as blocking, blurring, or flickering can arise from frame concealment techniques. Advanced concealment methods are tailored specifically for the H.265 codec to mitigate these artefacts. By carefully selecting frames for concealment and employing adaptive concealment strategies, the algorithm strives to maintain high visual fidelity.

- **Accurate Motion Representation**: Accurate motion representation is crucial for preserving video quality. Using advanced motion estimation and compensation techniques, the algorithm conceals frames with significant motion. This ensures that concealed frames accurately capture the underlying motion of the video content, minimising motion-related artefacts and preserving the smoothness and fluidity of the video.
- **Perceptual Quality Assessment**: Objective and subjective quality assessment methods are employed to evaluate the perceptual impact of the frame concealment process. Objective metrics such as structural similarity index (SSIM), peak signal-to-noise ratio (PSNR), or video quality metrics (VQM) provide quantitative measurements of the concealed video's visual quality compared to the original video. Additionally, subjective evaluations involving human participants are conducted to gather feedback on the perceived video quality. The algorithm is refined based on the results of these assessments to optimise video quality preservation.

4. Mobile Device Constraints

In video streaming, it is crucial to consider the computational constraints of mobile devices, which have limited processing power, memory, and battery life. This research finds the hardware limitations. It aims to develop an algorithm that can effectively operate within these constraints while achieving low latency and high-quality video streaming using the H.265 codec.

- **Computational Efficiency**: The algorithm optimises computational efficiency to ensure smooth video streaming performance on mobile devices. This involves developing efficient data structures, algorithms, and encoding techniques that minimise video processing tasks' computational complexity and resource requirements. By leveraging hardware acceleration features available on modern mobile devices, such as GPUs (Graphics Processing Units) or specialised video encoding/decoding units, the algorithm can efficiently process H.265-encoded video streams while minimising the impact on device performance.
- **Power Consumption Optimisation**: Mobile devices are powered by batteries with limited capacity, and power consumption optimisation is crucial to prolong battery life. The algorithm considers the power consumption implications of video processing tasks and strives to minimise energy-intensive operations. By utilising power-efficient algorithms, minimising unnecessary computations, and leveraging hardware acceleration capabilities, the algorithm aims to reduce power consumption without compromising video quality or latency.
- **Adaptive Streaming**: The algorithm incorporates adaptive streaming techniques to mitigate the impact of mobile device constraints. Adaptive streaming dynamically adjusts the video quality and streaming bitrate based on network conditions and device capabilities. The algorithm can provide a smooth streaming experience on mobile devices by adapting the video quality in real time, even in challenging network conditions or resource-constrained environments. This ensures that the video streaming remains responsive and optimised for the specific capabilities of the mobile device.

5. Random Frame Dropping

Random frame dropping is a technique employed to reduce the streaming size of H.265-encoded video files while maintaining an acceptable level of video quality. The algorithm selectively drops frames randomly during the streaming process to achieve compression without significantly impacting the visual experience.

Frame Selection Strategy: The algorithm employs a frame selection strategy to determine which frames to drop. Random frame dropping involves randomly selecting frames from the video stream and discarding them. The frame selection process considers the desired compression ratio, target streaming size, and acceptable video quality degradation. By adopting a random approach, the algorithm avoids bias towards specific frames or sequences, ensuring a more balanced distribution of dropped frames throughout the video.

- **Keyframe Preservation:** Keyframes, also known as intra-frames, are essential for maintaining video coherence and ensuring accurate decoding. The algorithm focuses on retaining keyframes while selectively dropping non-key frames to preserve video quality. By preserving keyframes and dropping non-key frames,

the algorithm minimises the impact on video quality and ensures the ability to decode and display subsequent frames accurately.

- **Compression and Bitrate Adjustment:** Random frame dropping inherently contributes to video compression by reducing the number of frames transmitted during streaming. This compression effect reduces the streaming size, as fewer frames need to be transmitted over the network. Additionally, bitrate adjustment techniques can be employed to optimise the streaming size further while considering network bandwidth limitations. By dynamically adjusting the bitrate based on available network resources, the algorithm optimises the compression while minimising the impact on video quality.
- **Quality Assessment and Thresholds:** The algorithm employs quality assessment techniques to ensure that the dropped frames do not significantly degrade the visual experience. Objective metrics such as PSNR (Peak Signal-to-Noise Ratio) or SSIM (Structural Similarity Index) may be utilised to evaluate the impact of random frame dropping on video quality. The algorithm sets thresholds for acceptable quality degradation, ensuring that the dropped frames do not fall below a certain quality threshold. The algorithm can balance compression and video quality preservation by incorporating these quality thresholds.

5.1. Importing Libraries

Successful implementation of the random frame-dropping algorithm heavily relies on leveraging relevant libraries and frameworks for video processing, random number generation, and image manipulation.

```
import cv2
import random
```

Figure 1: Frame Dropping Code - Importing Libraries

The following libraries are imported to support the development of the algorithm:

- **OpenCV (cv2)** - OpenCV is a popular computer vision library that provides various functions and tools for video processing, image manipulation, and computer vision tasks. This research uses OpenCV to read video frames, write frames to disk, and manipulate images. The cv2 module from OpenCV is imported to access its functions and objects.
- **Random** - The random module is a built-in Python library that provides various functions for generating random numbers and making random selections. In this research, the random library is imported to create random numbers for frame dropping. The random module's function generates a random integer within a specified range.

5.2. Open Video Capture

The video capture process is a fundamental step in video processing and analysis. This research uses the OpenCV library to open and access the video file for further frame processing and analysis—the cv2. The VideoCapture() function creates a video capture object, allowing the algorithm to retrieve individual frames from the video source.

```
vidcap = cv2.VideoCapture('PreviewVideo.mp4')
```

Figure 2: Frame Dropping Code - Open Video Capture

- **Video Source Selection**—The cv2.The VideoCapture() function takes the video file path as its parameter. Depending on the location of the video file in the system, the path can be absolute or relative. Specifying the correct path to the video file is essential to ensuring successful video capture.
- **Video Capture Initialisation**—The cv2.VideoCapture() function initialises the video capture object by associating it with the specified video file. After the video file is opened, the capture object is ready to read frames from the video source.

- **Retrieving Video Properties—**Various video properties can be accessed and examined after creating the video capture object. These properties include the number of frames in the video, frame rate, resolution, duration, and more. These properties provide essential information about the video that can be used for further processing and analysis.
- **Frame Retrieval** - Once the video capture object is initialised, individual frames can be retrieved from the video source. The read() method of the video capture object is employed to read the next frame in the video sequence. The returned values include a Boolean indicator representing the success of the frame retrieval operation and the actual frame data as an image array.

6. Bit Depth Compression

Bit depth compression is a fundamental aspect of multimedia data processing that aims to reduce the number of bits required to represent the colour information in images and videos. Decreasing the bit depth can significantly reduce the data size, enabling more efficient storage and transmission. However, the challenge lies in achieving this compression while preserving the visual quality of the content. Bit-depth compression techniques are crucial in reducing file sizes and maintaining perceptually acceptable visual fidelity. This section explores various bit-depth compression techniques, including uniform quantisation, dithering, error diffusion, non-uniform quantisation, adaptive quantisation, hybrid compression, and the potential application of machine learning and deep learning. Understanding these techniques is essential for optimising compression efficiency and ensuring high-quality multimedia content in applications such as image and video compression, storage, and transmission.

The focus is on reducing latency by utilising the H.265 codec, known for its efficient video compression capabilities. The objectives include minimising streaming size by concealing lost or dropped frames, preserving video quality, considering hardware computational limitations of mobile devices, and exploring innovative techniques such as frame duplication and dropping.

The methodology section outlines the research design and approach to achieving the objectives. It includes details on the data collection process, experimental setup, and algorithm development using Python. The research methodology ensures rigorous analysis and evaluation of the proposed techniques, providing reliable results and insights.

7. Bit Depth Compression Techniques

- **Uniform Quantisation**: Uniform quantisation is a straightforward technique that can reduce the bit depth by dividing the range of colour values into more minor levels. While simple, it may result in visible quantisation artefacts.
- **Dithering**: Dithering can be applied during the quantisation process to distribute the quantisation error and reduce artefacts. Algorithms like Floyd-Steinberg, Jarvis-Judice-Ninke, or Stucki can be utilised for dithering.
- **Error Diffusion**: Error diffusion techniques distribute the quantisation error across neighbouring pixels, reducing its visibility and preserving more details. Algorithms such as Floyd-Steinberg, Sierra, or Stucki can be employed for error diffusion.
- **Non-Uniform Quantisation**: Non-uniform quantisation techniques allocate more bits to visually important colour values and fewer bits to less essential values, considering human visual perception. Logarithmic quantisation or perceptual quantisation can be used for non-uniform quantisation.
- **Adaptive Quantisation**: Adaptive quantisation adjusts the quantisation step size dynamically based on the image or video content. It allocates more bits to areas with detail and fewer to areas with low detail, enhancing compression efficiency. Algorithms for adaptive quantisation analyse the content and determine optimal quantisation parameters for each frame or scene.
- **Hybrid Compression Techniques**: Hybrid compression approaches combine bit depth compression with other methods, such as spatial or temporal. Integration of techniques like H.264 or H.265 (HEVC), which

incorporate bit depth and spatial and temporal compression, can achieve higher compression ratios while maintaining visual quality.

- **Machine Learning and Deep Learning**: Machine learning techniques, such as convolutional neural networks (CNNs) and generative adversarial networks (GANs), can be explored to improve compression efficiency and reduce artefacts. These models learn complex mappings between high-precision colour values and their compressed representations, enhancing compression while preserving visual quality.

7.1. Defining the compress_bit_depth function

The compress_bit_depth function compresses the bit depth of the video frames while preserving the visual quality. It takes three parameters: video_path (the path of the input video file), output_path (the path where the compressed video will be saved), and target_bit_depth (the desired bit depth for compression).

```
[ ] def compress_bit_depth(video_path, output_path, target_bit_depth):
        vidcap = cv2.VideoCapture(video_path)
        fps = vidcap.get(cv2.CAP_PROP_FPS)
        frame_width = int(vidcap.get(cv2.CAP_PROP_FRAME_WIDTH))
        frame_height = int(vidcap.get(cv2.CAP_PROP_FRAME_HEIGHT))
        num_frames = int(vidcap.get(cv2.CAP_PROP_FRAME_COUNT))

        fourcc = cv2.VideoWriter_fourcc(*'mp4v')
        out = cv2.VideoWriter(output_path, fourcc, fps, (frame_width, frame_height), isColor=True)

        for _ in range(num_frames):
            success, frame = vidcap.read()

            if success:
                compressed_frame = apply_bit_depth_compression(frame, target_bit_depth)
                out.write(compressed_frame)

        vidcap.release()
        out.release()
```

Figure 3: Bit Depth Compress Code -Compress_bit_Depth Function

Open the video file.	The function uses cv2.VideoCapture to open the input video file specified by video_path. It retrieves the frame rate, width, height, and total number of frames from the video using the get functions of the video capture object (vidcap).
Create the output video file.	The function creates a VideoWriter object using cv2.VideoWriter. The output_path, fourcc (the four-character code representing the video codec), fps, (frame_width, frame_height), and isColor=True parameters are passed to initialise the VideoWriter object.
Iterate through each frame and compress:	The function utilises a loop that iterates num_frames times, where num_frames is the total number of frames in the video. Within the loop, it reads each frame of the video using vidcap.read() and assigns it to the frame variable.
Apply bit depth compression:	If the frame is successfully read (success = True), the function calls the apply_bit_depth_compression function to compress the bit depth of the frame. The compressed frame is then stored in the compressed_frame variable.
Write the compressed frame:	The compressed frame is written to the output video file using out. write(compressed_frame).

Table 1: Defining the apply_bit_depth_compression function

7.2. Buffer Management

Buffer management is crucial to achieving low latency in video streaming. It involves efficient control and synchronisation of buffers at both the encoder and decoder sides. Buffer management strategies aim to minimise buffering delays, maintain synchronisation between the encoder and decoder, and enable adaptive buffering to handle variable network conditions.

- **Buffer Occupancy Control**: The algorithm employs strategies to control the buffer occupancy at the encoder and decoder sides. The incoming video frames are buffered at the encoder side to ensure a steady supply of frames for encoding. The buffer occupancy is controlled to prevent excessive delay and maintain a continuous frame flow to the decoder. The output frames are buffered at the decoder side to smooth out network latency variations and ensure a consistent playback experience.
- **Synchronisation between Encoder and Decoder**: To achieve low latency, synchronisation between the encoder and decoder is essential. The algorithm ensures that the encoder and decoder maintain a synchronised operation by managing the buffer sizes and handling frame transmission and reception. This synchronisation minimises the delay introduced by buffering and ensures smooth video playback without noticeable latency.
- **Adaptive Buffering Techniques**: Adaptive buffering techniques handle varying network conditions and optimise buffer occupancy. These techniques dynamically adjust the buffer sizes and control the transmission rate to adapt to available network bandwidth and latency changes. The algorithm can dynamically adjust buffer occupancy and transmission rates by monitoring network conditions in real-time to maintain optimal video streaming performance.
- **Buffer Management Algorithms**: These algorithms consider factors like network bandwidth, latency, and video complexity to adjust the buffer sizes and transmission rate dynamically for optimal streaming performance. The rate control algorithms aim to prevent buffer overflows or underflows and maintain a consistent video streaming experience with minimal latency.
- **Error Resilience Mechanisms**: These mechanisms employ techniques like forward error correction (FEC) or retransmission to recover lost or corrupted video frames. By incorporating error resilience techniques, the algorithm can mitigate the impact of packet loss on buffer occupancy and maintain the overall streaming quality and low latency.

Overall, efficient buffer management, synchronisation between the encoder and decoder, and adaptive buffering techniques are vital in achieving low latency in video streaming. The algorithm carefully controls the buffer occupancy, adjusts transmission rates, and employs error resilience mechanisms to optimise video streaming performance, reduce buffering delays, and deliver end-users a seamless and responsive streaming experience.

7.3. Bit Rate Adjustment

The bit rate adjustment is an essential technique in video streaming to optimise the trade-off between video quality and bandwidth consumption. By adjusting the bit rate, the amount of data transmitted per unit of time can be modified, thereby impacting the quality and size of the video stream. This technique aims to achieve efficient video delivery while maintaining an acceptable level of visual quality. There are several approaches to adjust the bit rate of a video stream, including:

- Constant Bit Rate (CBR)
- Variable Bit Rate (VBR)

7.3.1. Constant Bit Rate (CBR)

Constant Bit Rate (CBR) is a bit rate adjustment technique commonly used in video streaming and encoding. The video is encoded and transmitted in CBR at a fixed bit rate throughout the entire stream. Each frame is allocated a predetermined number of bits, ensuring a consistent data rate.

CBR is often suitable for applications or systems where network bandwidth is stable and consistent, and maintaining a constant data rate is a priority. It can also benefit video-on-demand services or offline video encoding, where the encoding parameters are fixed, and bandwidth constraints are known in advance.

By implementing and evaluating CBR encoding techniques and comparing them with other bit rate adjustment methods, the research aims to identify the most suitable approach for achieving efficient video streaming with low latency and reduced bandwidth consumption.

```
[ ] import cv2

    # Set the input and output file paths
    input_file = 'input_video.mp4'
    output_file = 'output_video_cbr.mp4'

    # Set the target bit rate (in kbps)
    target_bit_rate = 2000

    # Open the input video file
    input_video = cv2.VideoCapture(input_file)

    # Get the video properties
    frame_width = int(input_video.get(cv2.CAP_PROP_FRAME_WIDTH))
    frame_height = int(input_video.get(cv2.CAP_PROP_FRAME_HEIGHT))
    fps = input_video.get(cv2.CAP_PROP_FPS)

    # Create the video writer object
    fourcc = cv2.VideoWriter_fourcc(*'mp4v')
    output_video = cv2.VideoWriter(output_file, fourcc, fps, (frame_width, frame_height))

    # Calculate the target bit rate per frame
    target_bit_rate_per_frame = target_bit_rate / fps
```

Figure 4a: CBR Code

```
    # Calculate the target bit rate per frame
    target_bit_rate_per_frame = target_bit_rate / fps

    # Read and process each frame from the input video
    while input_video.isOpened():
        ret, frame = input_video.read()

        if not ret:
            break

        # Perform CBR encoding by adjusting the frame's quality
        encoded_frame = cv2.imencode('.jpg', frame, [int(cv2.IMWRITE_JPEG_QUALITY),
                                                     target_bit_rate_per_frame])[1]

        # Decode the encoded frame
        decoded_frame = cv2.imdecode(encoded_frame, cv2.IMREAD_COLOR)

        # Write the decoded frame to the output video
        output_video.write(decoded_frame)

    # Release the video capture and writer objects
    input_video.release()
    output_video.release()

    # Display completion message
    print("CBR encoding complete. Output video saved as:", output_file)
```

Figure 4b: CBR Code

7.3.2. Variable Bit Rate (VBR)

Variable Bit Rate (VBR) is a bit rate adjustment technique commonly used in video encoding and streaming. Unlike Constant Bit Rate (CBR), which maintains a fixed bit rate throughout the video stream, VBR dynamically adjusts the bit rate based on the content complexity. It allocates more bits to complex scenes and fewer to less complex scenes, resulting in more efficient bandwidth utilisation and improved video quality.

The main characteristics and considerations of VBR are as follows:

- **Dynamic Bit Rate**: VBR allows for varying the bit rate frame-by-frame. It analyses the content complexity and allocates various bits to each frame. This adaptive approach ensures that more bits are earmarked for scenes with high motion or intricate details while fewer bits are assigned to scenes with low motion or simple backgrounds.
- **Improved Video Quality**: By allocating more bits to complex scenes, VBR can achieve higher video quality compared to CBR. It reduces compression artefacts and preserves more details in visually demanding portions of the video, resulting in a better viewing experience for the audience.
- **Efficient Bandwidth Utilisation**: VBR optimises bandwidth utilisation by adjusting the bit rate dynamically. It reduces the overall file size by allocating fewer bits to less complex scenes, requiring fewer bits for faithful reproduction. This enables efficient streaming over networks with varying bandwidth capacities.
- **Variable File Size**: The file size can fluctuate since the bit rate varies throughout the video. VBR-encoded videos may have larger file sizes compared to CBR-encoded videos. However, this variation in file size allows for a better balance between video quality and storage/bandwidth requirements.

- **Transcoding Challenges**: VBR-encoded videos may present challenges during transcoding or further processing. Since the bit rate varies, it can affect the synchronisation of audio and video streams, requiring additional synchronisation adjustments during post-processing.
- **Complexity and Compatibility**: Implementing VBR encoding requires more computational resources than CBR due to the analysis and allocation of bits on a frame-by-frame basis. Additionally, VBR-encoded videos may have compatibility issues with some playback devices or streaming platforms that only support specific video encoding formats.

VBR is widely used in various video streaming applications, such as online video platforms, video-on-demand services, and live streaming. It offers a flexible and efficient approach to maintaining high-quality video while adapting to the dynamic nature of content complexity.

```python
import cv2

# Set input and output file paths
input_file = 'input_video.mp4'
output_file = 'output_video_vbr.mp4'

# Set target bit rate range (in kbps)
min_bit_rate = 500
max_bit_rate = 3000

# Open input video
cap = cv2.VideoCapture(input_file)

# Get video properties
width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
fps = cap.get(cv2.CAP_PROP_FPS)

# Create video writer object
fourcc = cv2.VideoWriter_fourcc(*'mp4v')
out = cv2.VideoWriter(output_file, fourcc, fps, (width, height))

# Calculate target bit rate per frame range
min_bit_rate_per_frame = min_bit_rate / fps
max_bit_rate_per_frame = max_bit_rate / fps

# Read and process each frame
while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        break
```

Figure 5a: VBR Code

```python
    # Apply VBR encoding based on frame complexity
    # Calculate frame complexity metric (e.g., motion analysis, visual details)
    # Adjust the bit rate per frame based on the complexity metric
    # Perform encoding on the frame using the adjusted bit rate

    # Write the encoded frame to the output video
    out.write(frame)

    # Display the frame (optional)
    cv2.imshow('VBR Encoding', frame)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# Release resources
cap.release()
out.release()
cv2.destroyAllWindows()
```

Figure 5b: VBR Code

8. Conclusion

In conclusion, this research proposal addresses the challenge of low-latency video streaming using the H.265 codec. Throughout the research, several key findings and contributions have emerged. Firstly, the random frame-dropping technique proved to be ineffective in reducing latency. However, bit depth compression demonstrated the potential to reduce the streaming size without significantly compromising video quality. The implemented algorithm successfully adjusted the bit depth of the video frames, leading to improved compression efficiency. Furthermore, the research explored the concepts of constant bit rate (CBR) and variable bit rate (VBR). It was found that CBR maintained a consistent bit rate, ensuring a predictable streaming experience but potentially resulting in wasted bandwidth. On the other hand, VBR adjusted the bit rate dynamically, optimising streaming efficiency while considering the video content's complexity.

Another significant finding of this research project is the effectiveness of the frame duplication-dropping technique in reducing latency during video streaming. Latency refers to the delay or lag between the time a frame is captured or encoded and the time it is displayed on the viewer's screen. High latency can negatively impact the real-time streaming experience, causing delays and synchronisation issues. To address this challenge, the research implemented a frame duplication-dropping algorithm. This technique identifies consecutive frames that contain identical visual information, typically resulting from minimal scene changes or motion. The research improved real-time streaming performance by selectively dropping these duplicate frames, effectively reducing latency. The frame duplication dropping algorithm intelligently analyses the video frames and determines which frames can be safely dropped without significantly impacting the overall video quality. By eliminating redundant frames, the algorithm optimises the streaming process, allowing for smoother and faster delivery of the video content.

Moreover, the research considered mobile device constraints, acknowledging the computational limitations of these devices. This consideration led to the development of optimised algorithms for mobile platforms, ensuring efficient video streaming even on resource-constrained devices. In conclusion, this research project contributes to low-latency video streaming by exploring various techniques to minimise streaming size, preserve video quality, and reduce latency. The implemented Python algorithms, combined with quantitative data analysis, provide valuable insights into the performance and effectiveness of different strategies. The findings of this research have implications for applications in areas such as live video streaming, online gaming, and real-time video communication. The developed algorithms and insights can guide the development of more efficient video streaming systems, improving the user experience by minimising latency and optimising bandwidth utilisation. This research project highlights the potential of using H.265 video codec with innovative algorithms to achieve low-latency video streaming. The research findings contribute to the body of knowledge in this domain and open avenues for further exploration and refinement of video streaming techniques.

Funding: Not applicable.

Conflict of Interest: The authors declare no conflict of interest.

Informed Consent Statement/Ethics Approval: Not applicable.

Declaration of Generative AI and AI-assisted Technologies: This study has not used any generative AI tools or technologies in the preparation of this manuscript.

References

Farooq, V. M. V., & Mahammad, S. (2017). A study on H.26x family of video streaming compression techniques. *International Journal of Pure and Applied Mathematics, 117*(10, Special Issue), 63–72.

Hanzo, L., Cherriman, P., & Streit, J. (2007). *Video compression and communications*. John Wiley & Sons.

International Telecommunication Union. (n.d.). *ITU-T recommendation database*. https://www.itu.int/ITU-T/recommendations/rec.aspx?rec=13189.

Kemp, S. (2020, February 18). *Digital 2020: Sri Lanka*. DataReportal. https://datareportal.com/reports/digital-2020-sri-lanka.

Sullivan, G. J., Ohm, J.-R., Han, W.-J., & Wiegand, T. (2012). Overview of the high-efficiency video coding (HEVC) standard. *IEEE Transactions on Circuits and Systems for Video Technology, 22*(12), 1649–1668. DOI: 10.1109/TCSVT.2012.2221191

Sze, V., Budagavi, M., & Sullivan, G. J. (2014). *High-efficiency video coding (HEVC): Algorithms and architectures*. Springer.

Sze, V. (2014). Entropy coding in HEVC. In *Integrated circuits and systems* (pp. 209–274). Springer.