Fast H.264 Mode Selection Using Depth Information for Distributed Game Viewing

Gene Cheung[†], Antonio Ortega[*], Takashi Sakamoto[†]

[†]HP Labs Japan, 3-8-13, Takaido-higashi, Suginami-ku, Tokyo, Japan 180–0022
[*]Signal and Image Processing Institute, University of Southern California, Los Angeles, CA 90089-2564

ABSTRACT

While H.264 has been known for its superior coding performance over its predecessors, its encoding complexity can be high, especially if an exhaustive search is performed for the rate-distortion optimal mode among many available coding modes for each macroblock. In this paper, we show that in the context of coding 3D graphically rendered frames, depth information—the distance between rendered pixels of objects and the virtual camera popularly used to determine object occlusion during graphics rendering—can be exploited to expedite the mode selection process at a negligible cost of decrease in coding performance. Our proposed techniques take into consideration that regions of similar depth are likely to correspond to regions of uniform motion, an observation that can be exploited to speed-up the selection of block sizes for motion compensation. In our experiments, we show that complexity of the full motion vector search can be reduced by 40% at the expense of small increase of less than 0.7% in bitrate and small decrease of less than $0.07dB$ in PSNR. Moreover, because our proposed schemes operate in the depth domain, we show that they can be easily combined with an existing pixel-domain mode selection algorithm in the literature leading to similar complexity reductions.

1. INTRODUCTION

In recent years we have witnessed phenomenal growth of networked games like `Counter-Strike`[1] and `Sauerbraten`[2] on the Internet, particularly in South Korea and Japan. Along the way, exceptionally skilled and charismatic game players have developed cult followings, and game watching has become a popular pastime for many. One example is Half-Life Television,[3] which allows passive viewing of a few selected real-time games for a large audience. Capitalizing on this trend, we have developed a networked video system called *ECHO*[4] *(Enabling Community of Hecklers and Observers)* that "transcodes" game graphics to standard compliant video like H.26x and streams the encoded bitstream to ECHO clients for game viewing and heckling. See Figure 1 for a system overview of ECHO. Unlike Half-Life Television,[3] which requires each client to possess a potentially computationally expensive graphics rendering engine to render thousands of triangles per second, ECHO viewers only need relatively low-complexity video decoders to view game content, while the burden of rendering graphics is pushed back upstream to the *Observer View Generator* (OVG) in the ECHO system. In so doing, an ECHO client can change game views of the same game or "channel-surf" among live games simply by requesting different encoded streams from OVG. A key determinant of performance for ECHO, therefore, is how well OVG can encode videos from graphically rendered game views in real-time. This is the focus of the paper.

It is well known that the recent video coding standard H.264[5] has significantly better coding performance as compared to previous standards like H.263, and hence enabling H.264 encoding in OVG is preferred for better video quality. However, complexity of real-time H.264 encoding is high, and a major source of complexity comes from the existence of many coding modes. As an example, motion compensation based on 16×16 blocks only in H.263 is replaced by compensation based on various possible block sizes (4×4, 4×8, 8×4, 8×8, 8×16, 16×8, in addition to 16×16), with the block size chosen for a specific part of a video frame typically based

Further author information:
Gene Cheung: `gene-cs.cheung@hp.com`, Antonio Ortega: `ortega@sipi.usc.edu`, Takashi Sakamoto: `takashi.sakamoto@hp.com`

on rate-distortion (RD) criteria. While today's more powerful workstations can now handle real-time H.264 encoding for smaller screen sizes like CIF (352×288) and QCIF (176×144), OVG needs to simultaneously generate multiple game views for each of multiple games, so a low-complexity real-time H.264 encoder is highly desirable.

Numerous techniques for efficient mode selection for H.264 have been proposed in the literature.[6–10] Most of these works construct heuristic decision trees that prune subsets of candidate modes based on metrics evaluated using information derived from the pixel domain. Unique to 3D graphically rendered frames, however, is the availability of *depth information*:[11] the distances from rendered pixels of objects to the virtual camera. Depth information is commonly used for 3D graphics rendering: when objects are drawn onto the frame buffer sequentially, the graphics renderer determines which pixels of objects are closer to the camera, and hence are occluding others, by the depth values of pixels of objects. Given the availability of depth maps for the frames to be coded, in this paper we propose to exploit such information—a unique source of side information beyond plain raw pixels—to speed up the H.264 encoding process. In particular, we show that by sequentially evaluating subsets of depth values within a 16×16 macroblock (MB) in a decision tree, we can quickly eliminate unlikely block sizes from the candidate list. In so doing, we can eliminate 40% of the full search complexity at the modest cost of less than 0.7% increase in bitrate and less than $0.07dB$ decrease in PSNR. The key intuition behind our work lies in the observation that regions of similar depth are more likely to correspond to regions of uniform motion; thus, if readily available depth information is indicative of uniform motion in a spatial region, it becomes possible to avoid considering smaller block-sizes for motion estimation, leading to complexity reductions in mode selection with small coding performance penalty.

The outline of the paper is as follows. We first discuss related work in Section 2. We then give an overview of the ECHO system in Section 3. Next we discuss how depth similarity can be quantified in Section 4. Our proposed techniques to optimize mode selection based on depth information are discussed in Section 5. We next discuss how to combine our approach with an existing fast mode decision algorithm in Section 6. Experimental results that quantify different tradeoffs and conclusions are provided in Sections 7 and 8, respectively.

2. RELATED WORK

In this paper we focus on video coding techniques to support online game viewing. Video coding for this application has been considered by other authors.[12] A key difference with respect to prior work is that we assume that the encoder has access to the frame buffer (including both the color buffer and the Z-buffer) but not the graphics rendering engine. That means that the encoder takes as input only rendered video frames and pixel-wise depth maps, both of which can be accessed independently from the frame buffer via 3D graphics APIs such as OpenGL.[13] In contrast, Bharambe *et al.*[12] assume that the encoder has complete knowledge of the rendering engine, including shapes and locations of various objects that appear on a scene. In practical terms, our game-independent system can be easily implemented with any game client, while previous game-dependent systems[12] need to be co-developed with every single implementation of a game client. Besides the obvious high development cost, it also raises copyright concerns as game software developers may be reluctant to provide source code of game clients to third parties.

In our previous system implementation,[4] we had used a simple H.263 video encoder in OVG for complexity reasons. Depth information was used to determine the relative importance of different regions in the scene, so that visual overlays would not obscure important objects. We also previously investigated how depth information can be used for bit allocation,[14] so that areas closer to the virtual camera are deemed more important and are allocated more encoding bits.

Due to its complexity and direct effect on coding performance, optimizing mode selection has been a popular topic for H.263[15–17] and now H.264.[6–10] As mentioned in the Introduction, unlike existing works that are based on pixel domain information, our proposed mode selection technique is based on depth information. We will discuss a hybrid approach that takes advantages of both pixel domain and depth information in Section 6.

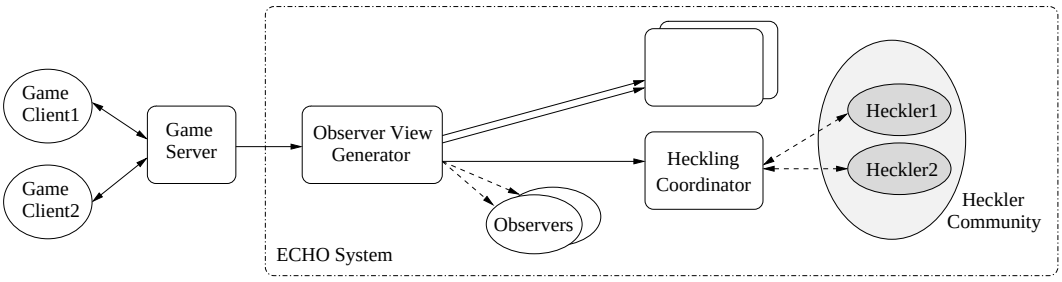


Figure 1. The ECHO system consists of an Observer View Generator (OVG) to render live game content into streaming video, and Heckling Coordinators (HC) to support customized visual overlays or heckling. A user can be a passive game "Observer" or an active "Heckler".

3. ARCHITECTURE OVERVIEW

A system overview of ECHO is shown in Figure 1. A networked game is hosted by a *game server*, and typically involves two or more game players who control the *game clients*. To support game viewing for non-players over a wide range of devices, it is desirable to convert the game content into standard-compliant video streams. This is achieved by the *Observer View Generator* (OVG). Inside OVG is a *ghost client* that receives real-time gaming data from the game server and draws frames of the game sequence from a chosen perspective. The generated frames are then encoded in real-time by a video encoder and are then passed to a video streaming server that supports RTP streaming.[18] An ECHO client interested in observing the game can then send an RTSP request to OVG, which upon request will setup an RTSP session and send two RTP streams, one each for video and audio, to the requesting observer.

Alternatively, an ECHO client can interact with other fellow observers in real-time via live-captured speech and visual overlays of avatars[19] or talking heads that are controllable by users' inputs, while observing the same game. This is called a *heckling session*, and the participants are called *hecklers*. In such case, the hecklers will request a heckling session from a *Heckling Coordinator*, who in turn will request a live stream from OVG on their behalf, overlay heckling characters on the encoded stream corresponding to the hecklers' inputs, and stream the modified video bitstream to hecklers in the session. This paper focuses only on video encoding at OVG; for further details about ECHO, see Cheung *et al.*[4]

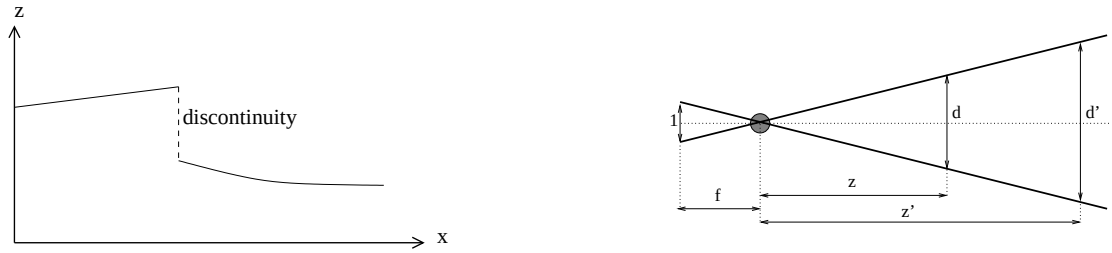
4. DEVELOPING METRICS BASEED ON DEPTH INFORMATION

Our goal is to use depth information generated by the 3D graphics rendering engine in order to facilitate video encoding. Primarily, we seek to use depth information to infer useful properties of the motion field. We can construe two intuitions based on depth information. First, we would expect regions that have "similar" depth to be more likely to exhibit uniform motion as well. Second, when performing motion estimation across frames, we would expect that two blocks in different frames will be more likely to be well matched after motion estimation if their depths are "similar". In this paper, we exploit the first intuition to reduce the complexity of mode selection. Exploiting the second intuition to speed up motion estimation is one of our planned future topics.

Note that in both cases we are invoking a notion of similarity between depth maps (either within a block or across blocks), so in what follows we first present two different arguments for absolute and relative value metrics that can potentially help quantify the similarity of regions at different depths. We will test both metrics in the experiments presented later. We then provide a specific formulation for the case when depth is obtained from a 3D graphics card.

4.1. Argument for Absolute Value Metric

Viewing from a camera, a singular object likely has gradually changing depth values on its surface. When there is an abrupt discontinuity in depth values along a spatial axis within a block (see Figure 2a)), there are likely two different objects across the discontinuity, each of which likely has its own unique motion. One way to discover these discontinuities is to look at the absolute difference in depth values between spatially neighboring pixels



a) Discontinuity of Depth Values b) Pinhole Camera Geometry

Figure 2. Arguments for Absolute and Relative Value Metrics

and/or blocks. Hence we advocate an *absolute value metric*, $z' - z$, to identify discontinuity between block A of depth z and block B of depth z'.

4.2. Argument for Relative Value Metric

Consider now instead two objects at different depths z and z' as shown in Figure 2b). When viewing frames displaying these two objects, a distance of one pixel in the video frame corresponds to different "real world" distances for each object. Denote d and d' the real distances corresponding to one pixel distance for the objects at depth z and z', respectively. From basic pinhole camera geometry[20] we have that:

$$f\frac{d}{z} = f\frac{d'}{z'} \tag{1}$$

where f is the depth of the focal plane and therefore

$$\frac{d}{z} = \frac{d'}{z'} \tag{2}$$

or

$$\frac{d'}{d} = 1 + \frac{\delta z}{z}, \tag{3}$$

where $z' = z + \delta z$. Note that (3) essentially implies that we should not focus on absolute depth differences in order to assess the effect of depth changes on pixel-domain matching, since the same absolute depth difference will have a greater impact in terms of pixel distortion when objects are close to the camera. Hence we advocate a *relative value metric* like (3) to evaluate "distance distortion" as an object appears at different depths. Clearly, metrics such as (3) are simplistic in that they do not take into consideration of the characteristics of the objects. For example, objects appearing at different depths but lacking texture may in fact lead to a good match*. Conversely, highly textured objects may be hard to match even if their depth is relatively similar.

4.3. Estimating Depth Similarity from Z-buffer Data

We now describe how to use directly depth information available in the Z-buffer of a typical graphics card. In a Z-buffer, depth information for each pixel is represented by a finite N-bit representation, with N typically ranging from 16 to 32 bits. Because of this finite precision limitation, instead of true depth values z, common implementations of Z-buffers use quantized depth values[11,21] z_b of N-bit precision:

$$z_b = 2^N \left(a + \frac{b}{z} \right) \tag{4}$$

where

$$a = \frac{zF}{zF - zN} \quad \text{and} \quad b = \frac{zF \cdot zN}{zN - zF}. \tag{5}$$

*As usual in the context of video coding we are only interested in how well a particular portion of a frame can be predicted from previously transmitted data, which does not require identifying the ground truth match.

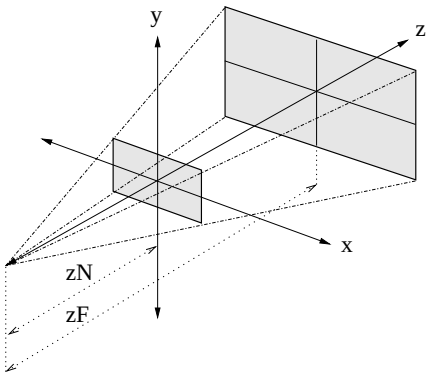


Figure 3. Projection Plane for Perspective Viewing

zN and zF are the z-coordinates of the near and far plane as shown in Figure 3 for perspective viewing.[11] Near plane is the projection plane, while far plane is the furthest horizon from which objects would be visible; zN and zF are carefully selected by graphics programmers to avoid erroneous object occlusion[21] due to rounding of true depth z to quantized depth z_b. (4) basically means that depth information is quantized non-uniformly; objects close to the camera have finer depth precision than objects far away, which is what is desired in most rendering scenarios. We can also define a normalized quantized depth:

$$z_0 = \frac{z_b}{2^N}, \tag{6}$$

with $z_0 \in [0, 1]$. Either the scaled integer version z_b or the normalized version z_0 of the quantized depth can typically be obtained directly from the graphics card.

Note that as z approaches zF (resp. zN), z_0 approaches 1 (resp. 0). Since typically $zF >> zN$ we can approximate

$$a \approx 1 \quad \text{and} \quad b \approx -zN, \tag{7}$$

and therefore

$$z = \frac{zN}{(1 - z_0)} \tag{8}$$

Thus in what follows, we will be using either an absolute or relative value metric to determine whether two depths should be considered significantly different.

5. MODE SELECTION BASED ON DEPTH INFORMATION

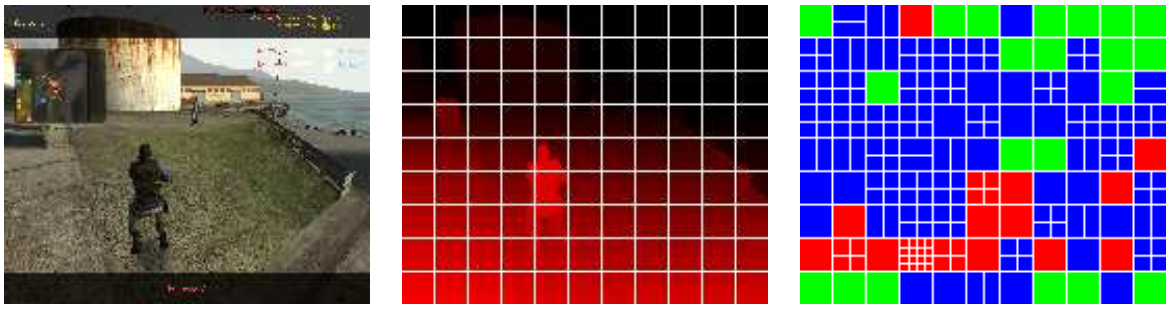


Figure 4. Original Image (left), Depth Values (center), and Selected Modes (right) of Frame 30 in `Counter-Strike` Sequence. For Selected Modes (right), green block indicates `Skip`, red block indicates `Intra` and blue block indicates `Inter`.

Given the discussion of metrics in the previous section, we now develop several schemes to *pre-prune* H.264 coding modes using depth information from the set of candidate modes, before starting a pixel-domain mode selection algorithm. To evaluate our system we use two representative pixel-domain mode selection algorithms: i) the full search algorithm implemented as low-complexity mode (LC) in the JM version reference software 12.4[22] and ii) the faster algorithm Yin *et al.*[7] (YTTB), which uses a specific decision tree based on expected properties of pixel-domain data. We consider these two algorithms to be representative of techniques proposed in the literature; we expect that similar results could be achieved by combining our proposed scheme with other pixel-domain mode decision tools. As will be seen, our proposed pre-pruning schemes can often obtain additional speedups on top of those provided by a given pixel-domain algorithm. This is because the depth information can capture scene structure that is difficult to obtain from the pixel domain. For example, using simple criteria based on depth, it may be easier to determine that a set of pixels belong to a single object, something that could be more challenging based on pixel intensity information alone.

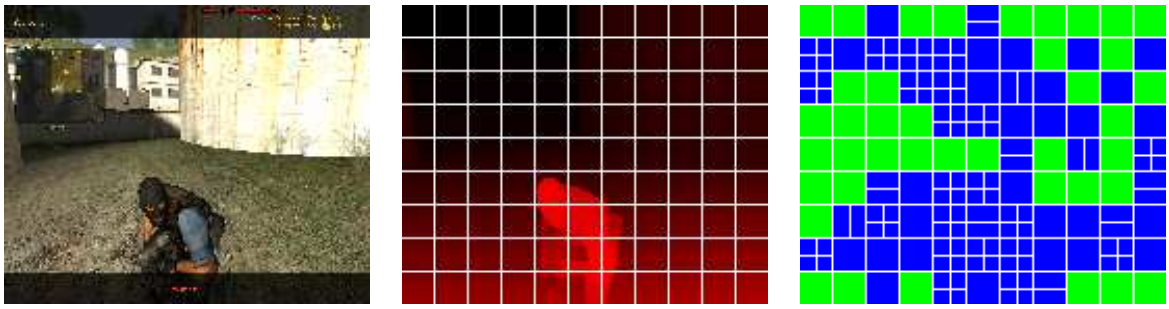
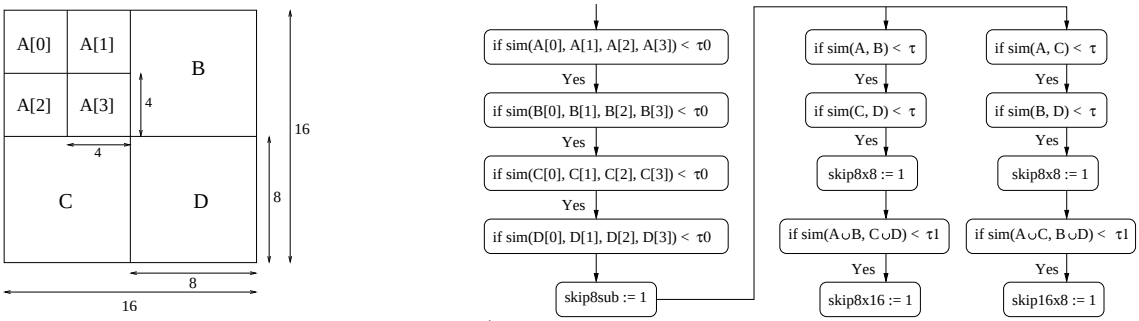


Figure 5. Original Image (left), Depth Values (center), and Selected Modes (right) of Frame 80 in `Counter-Strike` Sequence. For Selected Modes (right), green block indicates `Skip`, red block indicates `Intra` and blue block indicates `Inter`.

To motivate our pre-pruning schemes, consider Figure 4 and 5, each showing the original image in a captured `Counter-Strike`[1] sequence in QCIF (176×144), the quantized depth values z_o's obtained from the Z-buffer, and the corresponding "optimal" coding modes selected using LC in JM version 12.4. We see that among the selected INTER blocks with larger block sizes ($8 \times 16, 16 \times 8, 16 \times 16, SKIP$), they tend to have more uniform depth values. This validates our intuition discussed earlier that blocks with "similar" depth values tend to have uniform motion.



a) Coding Block Sizes of H.264 b) Decision Tree to Prune Candidate Block Sizes

Figure 6. Depth-based Decision Tree to Prune H.264 Coding Block Sizes from Candidate Set

To pre-prune coding blocks with dissimilar depth values, we construct a decision tree starting from small blocks to large blocks as shown in Figure 6. We first compare the depth values of four 4×4 blocks within each 8×8 block (say $A[0], \ldots, A[3]$ of 8×8 block A) using a similarity function $sim()$ (to be discussed). If they are sufficiently similar ($< \tau_0$), then we remove block sizes smaller than 8×8 from the candidate set ($skip8sub := 1$). We then compare depth values of the four 8×8 blocks in two parallel tracks: we check if the horizontal 8×8

blocks are sufficiently similar ($< \tau$), and we check if vertical 8×8 blocks are sufficiently similar. If either of them is true, we eliminate 8×8 block size from consideration ($skip8x8 := 1$). Finally, if the two 8×16 blocks (16×8 blocks) are similar ($< \tau_1$), then 8×16 block size (16×8 block size) is pruned as well ($skip8x16 := 1$ or $skip16x8 := 1$). Note that because 16×16 contains only one motion vector, it has low coding cost, and so it is never eliminated from the candidate set.

Obviously, the performance and complexity of the depth-based decision tree depends on how the similarity function $sim()$ is defined. In the following sections, we discuss how $sim()$ can be implemented in increasing complexity based on our discussion on metrics in Section 4.

5.1. Pre-pruning Scheme 1: Min-Max in Z-Buffer

In this low-complexity pre-pruning scheme, we find the maximum and minimum values of z_o from the Z-buffer in a given block. By (6), we know z_o is monotonically decreasing in z, so the maximum (minimum) value in z_o corresponds to the minimum (maximum) value in z. We then define the similarity of a block by applying either an absolute value or relative value metric using the maximum and minimum values of z_o. More precisely, given two blocks A and B, we compute:

$$\begin{aligned} z_{min}(A) &= \frac{zN}{1 - \max_{z_o \in A}(z_o)}, \qquad z_{max}(A) = \frac{zN}{1 - \min_{z_o \in A}(z_o)} \\ sim(A,B) &= z_{max}(A \cup B) - z_{min}(A \cup B) \quad \text{or} \quad \frac{z_{max}(A \cup B) - z_{min}(A \cup B)}{z_{max}(A \cup B) + z_{min}(A \cup B)} \end{aligned} \tag{9}$$

Given four blocks A, B, C and D, $sim(A, B, C, D)$ can be similarly defined as follows:

$$sim(A,B) = z_{max}(A \cup \ldots \cup D) - z_{min}(A \cup \ldots \cup D) \quad \text{or} \quad \frac{z_{max}(A \cup \ldots \cup D) - z_{min}(A \cup \ldots \cup D)}{z_{max}(A \cup \ldots \cup D) + z_{min}(A \cup \ldots \cup D)} \tag{10}$$

For this scheme we use the same threshold τ throughout the pre-pruning tree in Figure 6b):

$$\tau_0 = \tau_1 = \tau. \tag{11}$$

Notice that in this scheme, we avoid any direct conversion from z_o in the Z-buffer to true depth z. Considering only computation up to 8×8 block size in the depth-based decision tree in Figure 6 (computation in larger block sizes contributed marginally per pixel), the computation cost per pixel using the absolute value metric is:

$$\begin{aligned} C_1 &= \left(2 * \frac{63}{64}\right) * cost(\texttt{comp}) + \left(3 * \frac{1}{64}\right) * cost(\texttt{add}) + \left(2 * \frac{1}{64}\right) * cost(\texttt{mult}) \\ &\approx 2 * cost(\texttt{add}) \end{aligned} \tag{12}$$

where $cost(\texttt{comp})$, $cost(\texttt{add})$, and $cost(\texttt{mult})$ denote the estimated costs of comparisons, additions and multiplications, respectively. In the approximation, we assume $cost(\texttt{comp})$ is about as complex as $cost(\texttt{add})$.

5.2. Pre-pruning Scheme 2: Average Depth Values

In this mid-complexity pre-pruning scheme, we first convert all z_o-values from the Z-buffer to true depth z-values using (8), then compute the sum of the z-values. The similarity function $sim()$ using an absolute value metric is then the largest difference in sums between any two blocks. (Relative value metric can be similarly defined as done in previous pre-pruning scheme and hence is omitted.) More precisely, given two blocks A and B, $sim(A, B)$ is:

$$sim(A,B) = \sum(A) - \sum(B), \qquad \sum(A) = \sum_{z_o \in A} \frac{zN}{(1 - z_o)} \tag{13}$$

Similarly, given four blocks A, B, C and D, $sim(A, B, C, D)$ is:

$$sim(A, B, C, D) = \max\left\{\sum(A), \sum(B), \sum(C), \sum(D)\right\} - \min\left\{\sum(A), \sum(B), \sum(C), \sum(D)\right\} \quad (14)$$

Because of the different sizes of the cumulated sums, we scale the thresholds in the pre-pruning tree in Figure 6b) as follows:

$$\tau_0 = \tau/4, \qquad \tau_1 = 2\tau \quad (15)$$

The computational cost per pixel in this case is:

$$\begin{aligned} C_2 &= \frac{5}{64} * cost(\texttt{comp}) + \left(1 + \frac{60+1}{64}\right) * cost(\texttt{add}) + 1 * cost(\texttt{mult}) \\ &\approx 2 * cost(\texttt{add}) + 1 * cost(\texttt{mult}) \end{aligned} \quad (16)$$

5.3. Pre-pruning Scheme 3: Sobel Operator

In the high-complexity pre-pruning scheme, like the mid-complexity scheme we also first convert all z_o values from the Z-buffer to true depth using (8). For each pixel, we then apply the *Sobel operator*,[23] which is commonly used in image processing to detect edges in images. Wu *et al.*[9] had also proposed to use Sobel operator in the pixel domain to detect singular objects in coding blocks, but unlike their work, we apply the Sobel operator in the depth domain. Applying the Sobel operator in the depth domain instead of the pixel domain makes it easier to identify singular objects with complex texture. As done in Wu *et al.*,[9] the Sobel operator involves the following operations:

$$\begin{aligned} dx_{i,j} &= p_{i-1,j+1} + 2p_{i,j+1} + p_{i+1,j+1} - p_{i-1,j-1} - 2p_{i,j-1} - p_{i+1,j-1} \\ dy_{i,j} &= p_{i+1,j-1} + 2p_{i+1,j} + p_{i+1,j+1} - p_{i-1,j-1} - 2p_{i,j} - p_{i-1,j+1} \end{aligned} \quad (17)$$

$$Amp(\vec{D}_{i,j}) = |dx_{i,j}| + |dy_{i,j}|. \quad (18)$$

The similarity function $sim()$ is then simply defined as number of pixels with gradients $Amp(\vec{D}_{i,j})$'s greater than a pre-set gradient threshold θ:

$$sim(A, B) = \sum_{(i,j)\in A\cup B} \mathbf{1}(Amp(\vec{D}_{i,j}) > \theta) \quad (19)$$

where $\mathbf{1}(c) = 1$ if clause c is true, and $= 0$ otherwise. Similarly, given four blocks A, B, C and D, $sim(A, B, C, D)$ is:

$$sim(A, B, C, D) = \sum_{(i,j)\in A\cup B\cup C\cup D} \mathbf{1}(Amp(\vec{D}_{i,j}) > \theta) \quad (20)$$

Threshold τ in the pre-pruning tree is set to constant as done in pre-pruning scheme 1. The computational cost per pixel for this scheme is:

$$\begin{aligned} C_3 &= (2+1) * cost(\texttt{comp}) + \left(1 + 10 + 1 + \frac{63}{64}\right) * cost(\texttt{add}) + (1+4) * cost(\texttt{mult}) \\ &\approx 16 * cost(\texttt{add}) + 5 * cost(\texttt{mult}) \end{aligned} \quad (21)$$

6. COMBINING DEPTH-BASED & PIXEL-BASED MODE SELECTION ALGORITHMS

In this section, we show how our proposed pre-pruning schemes can be implemented in combination with an existing mode selection algorithm like YTTB[7] to further reduce computation at minimal bitrate increase. We first describe the YTTB mode selection, then discuss the combination of depth-based pre-pruning scheme with YTTB.

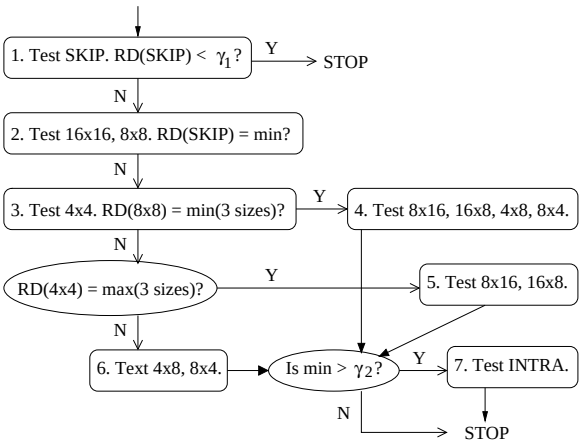


Figure 7. YTTB Mode Selection Algorithms

6.1. YTTB Mode Selection

Like other mode selection algorithms[6,8] that operate in the pixel domain, YTTB makes an astute assumption about the rate-distortion (RD) surface for different coding modes, and as points on the surface corresponding to some coding modes are generated, other points can be eliminated without actual computation of their corresponding coding modes. In YTTB, the property being exploited is *RD monotonicity with respect to block size*: if RD cost of coding blocks 4×4, 8×8 and 16×16 are strictly decreasing, then the remaining potential block candidates are only 8×16 and 16×8 that are between two best performing blocks thus far, 8×8 and 16×16. Conversely, if RD cost of coding blocks 4×4, 8×8 and 16×16 are strictly increasing, then remaining potential block candidates are only 4×8 and 8×4 that are between two best performing blocks thus far, 4×4 and 8×8. If RD costs of coding blocks 4×4, 8×8 and 16×16 are neither strictly increasing nor decreasing, then all coding blocks must be tried. The decision tree for YTTB is shown in Figure 7. There are two thresholds γ_1 and γ_2 in YTTB; for our experiments, we set both to zero.

6.2. Combined Pre-pruning Schemes & YTTB

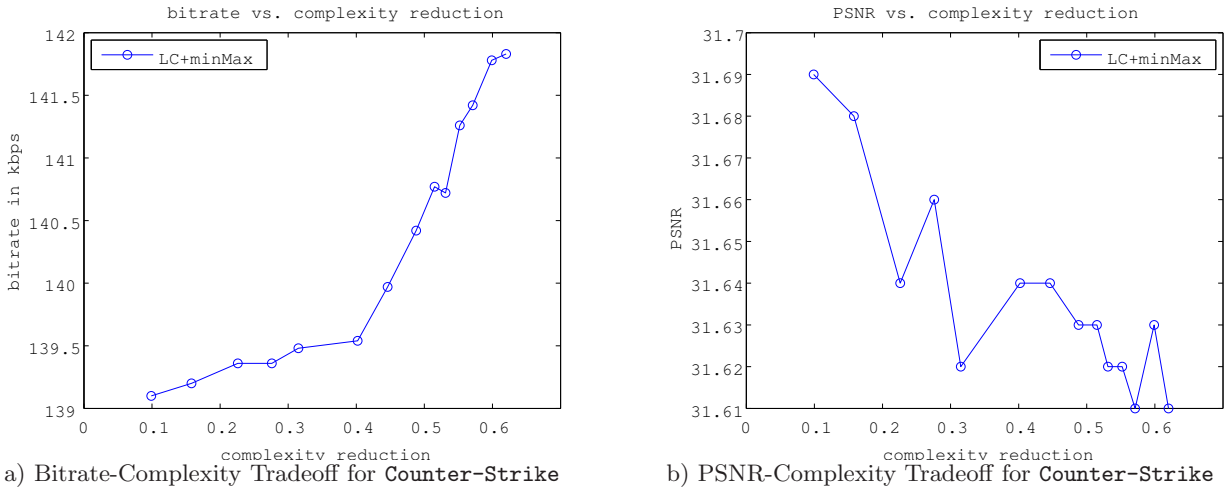
To combine one of our proposed pre-pruning schemes with YTTB, we first execute a chosen depth-based pre-pruning scheme to limit the potential coding block-size candidates to a smaller set, and set the RD costs of the pruned coding block sizes to ∞. We then execute YTTB as described previously. Notice that our depth-based pre-pruning tree always prunes smaller coding blocks before larger coding blocks. That means that RD costs are set to ∞ successively from smaller blocks to larger blocks, and we will never lead YTTB to eliminate block sizes erroneously if the original RD surface is itself not monotonic.

7. EXPERIMENTAL RESULTS

In the experimental results section, we test the effectiveness of our proposed pre-pruning schemes using a captured 100-frame game sequence from popular Internet game `Counter-Strike`.[1] Frames of a viewing client were rendered at 10 frames per second (fps) at resolution 640×480, then scaled down to QCIF 176×144 for H.264 encoding. Depth values of these frames were captured in the Z-buffer as well. With the exception of the first frame which was coded as I-frame, the rest of the frames were coded as P-frames. We used the available JM version 12.4[22] as the baseline codec. Quantization parameters were kept constant throughout the sequence.

7.1. Complexity and Coding Efficiency Tradeoff

We begin by examining the tradeoffs among bitrate, visual quality (Peak Signal-to-Noise Ratio (PSNR)) of encoded video and encoding complexity for low-complexity pre-pruning scheme `minMax` discussed in Section 5. By running `minMax` in combination with the low-complexity mode (LC) of JM 12.4 and varying the threshold τ, we obtained the bitrate-complexity reduction and PSNR-complexity reduction graphs shown in Figure 8a) and

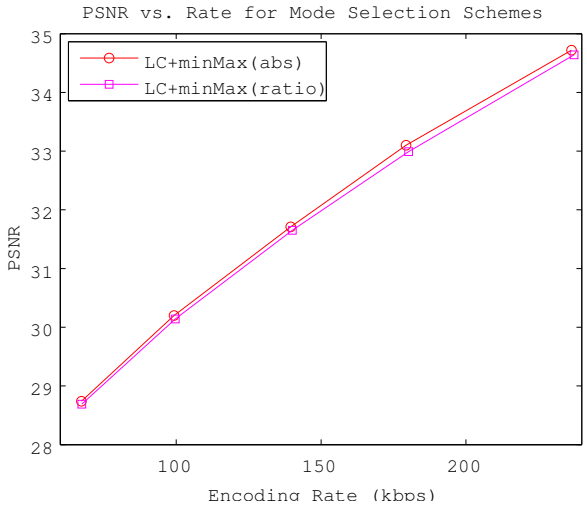


a) Bitrate-Complexity Tradeoff for `Counter-Strike` b) PSNR-Complexity Tradeoff for `Counter-Strike`

Figure 8. Rate-distortion-Complexity Tradeoffs for `minMax` with LC for `Counter-Strike` Sequence

Figure 8b), respectively. We see that as τ increases, the complexity reduction—the fraction of full motion search pre-pruned by `minMax`—increases, and correspondingly bitrate and PSNR increases and decreases respectively. We see the range of PSNR decrease is actually quite narrow. We also see that the rate of bitrate increase worsens after complexity reduction of 0.4. This seems to be a characteristics of all our proposed pre-pruning schemes. As a result, we adjust the thresholds in all our pre-pruning schemes so that the complexity reduction in each case is roughly at 40%.

7.2. Metric Selection



a) Distortion-Rate Tradeoff for `Counter-Strike`

Encoding Scheme	QP	PSNR	Bitrate
minMax(abs)	27	34.65	236.48
minMax(ratio)	27	34.64	237.33
minMax(abs)	29	33.00	179.35
minMax(ratio)	29	32.99	180.21
minMax(abs)	31	31.64	139.54
minMax(ratio)	31	31.65	140.07
minMax(abs)	33	30.15	99.26
minMax(ratio)	33	30.14	99.65
minMax(abs)	35	28.70	67.24
minMax(ratio)	35	28.69	67.41

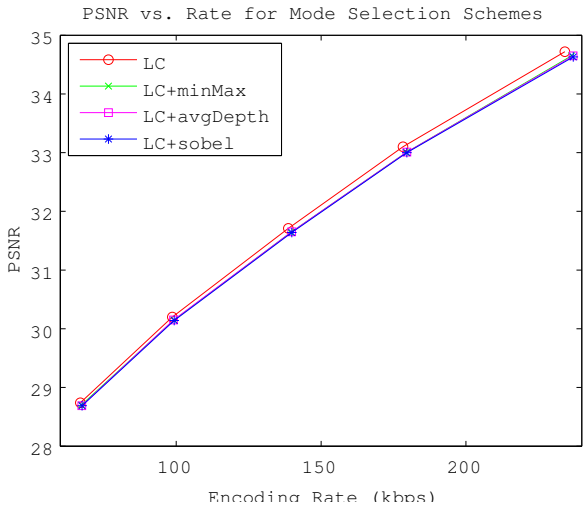
b) Numerical Comparison for `Counter-Strike`

Figure 9. Rate-distortion Performance for Different Metrics with `LC-minMax` for `Counter-Strike` Sequence

We next investigate the appropriate metric for our pre-pruning schemes. As discussed in Section 4, we can use either an absolute value metric or a relative value metric; we examine both metrics for `minMax` here. As discussed in Section 5.1, the absolute value metric (`abs`) for `minMax` simply finds the absolute difference between the maximum and minimum depth in a block. The relative value metric (`ratio`) divides the same absolute difference by the sum of the maximum and minimum depth. After the threshold in each case had been adjusted to have a complexity reduction of 40% from full motion search, we see rate-distortion performance of both metrics

in Figure 9. We see that `abs` is point-by-point better than `ratio`. We can therefore conclude that `abs` is a more suitable metric for our data set, and will henceforth use `abs` as our chosen metric for subsequent evaluation of pre-pruning schemes.

7.3. Coding Performance



a) Distortion-Rate Tradeoff for `Counter-Strike`

Encoding Scheme	QP	PSNR	Bitrate
LC	27	34.72	234.16
LC+minMax	27	34.65	236.48
LC+avgDepth	27	34.65	237.01
LC+sobel	27	34.64	237.07
LC	31	31.71	138.71
LC+minMax	31	31.64	139.54
LC+avgDepth	31	31.65	139.87
LC+sobel	31	31.64	139.89
LC	35	28.74	66.88
LC+minMax	35	28.70	67.24
LC+avgDepth	35	28.69	67.38
LC+sobel	35	28.69	67.47

b) Numerical Comparison for `Counter-Strike`

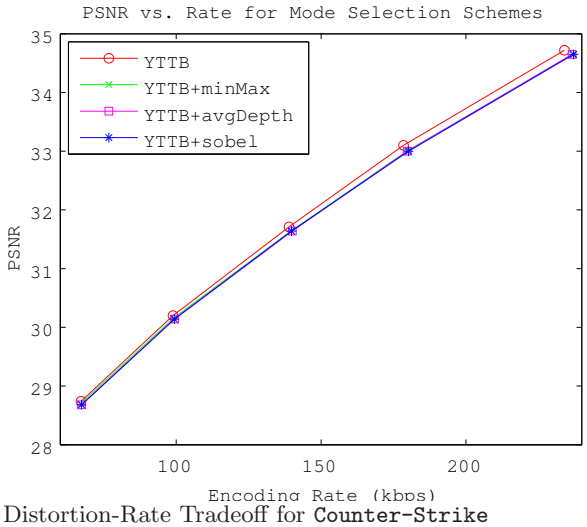
Figure 10. Rate-distortion Performance for Different Pre-Pruning Schemes with LC for `Counter-Strike` Sequence

To compare our proposed pre-pruning schemes, in Figure 10a) we plotted the RD performance of the discussed pre-pruning schemes when used in combination with `LC`: `LC+minMax`, `LC+avgDepth` and `LC+sobel` corresponding to the three discussed in Section 5. We again adjusted the threshold τ in each case so that the reduction in complexity from `LC` is roughly 40%. We see that there is a very slight drop in RD performance for the three pre-pruning schemes relative to the full search. Looking more closely at the numerical comparison in Figure 10b), we see that `LC+minMax` actually has the best average performance of the three. In particular, `LC+minMax`, `LC+avgDepth` and `LC+sobel` have average bitrate increase of 0.67%, 0.79% and 0.87% in bitrate relative to `LC`, respectively, and average PSNR decrease of $0.066dB$, $0.064dB$ and $0.074dB$, respectively. Given `minMax` has the lowest complexity, we can conclude `minMax` is the preferred pre-pruning scheme for this data set.

We conducted the experiment again, this time using YTTB in combination with each for the three pre-pruning schemes we have proposed. We plotted their RD performance in Figure 11a). Note that complexity reduction of close to 40% relative to YTTB is maintained in the experiment. We again see that there is a very slight drop in RD performance for the three pre-pruning schemes relative to YTTB. Looking more closely at the numerical comparison in Figure 11b), we again see that `YTTB+minMax` has on average the smallest increase in bitrate for about the same PSNR. In particular, `YTTB+minMax`, `YTTB+avgDepth` and `YTTB+sobel` have average bitrate increase of 0.44%, 0.71% and 0.75% relative to `YTTB`, respectively, and average PSNR decrease of $0.066dB$, $0.072dB$ and $0.072dB$, respectively. Again, given `minMax` has the lowest complexity, we can conclude `minMax` is the preferred pre-pruning scheme for this data set.

8. CONCLUSIONS

Depth information—the distance between rendered pixels of objects and the virtual camera—are popularly used by graphics renderer to determine object occlusion during image rendering. In this paper, we propose to use depth information to expedite MB's mode selection in H.264 encoding when the encoded source is graphically rendered images. We show that complexity of the full motion search can be decreased by 40% at the modest cost of less than 0.7% increase in bitrate and less than $0.07dB$ decrease in PSNR.



a) Distortion-Rate Tradeoff for `Counter-Strike`

Encoding Scheme	QP	PSNR	Bitrate
YTTB	27	34.73	234.02
YTTB+minMax	27	34.65	236.90
YTTB+avgDepth	27	34.65	236.68
YTTB+sobel	27	34.65	237.16
YTTB	31	31.71	138.95
YTTB+minMax	31	31.64	139.58
YTTB+avgDepth	31	31.64	140.08
YTTB+sobel	31	31.64	139.81
YTTB	35	28.74	67.09
YTTB+minMax	35	28.70	67.01
YTTB+avgDepth	35	28.68	67.29
YTTB+sobel	35	28.68	67.31

b) Numerical Comparison for `Counter-Strike`

Figure 11. Rate-distortion Performance for Different Pre-Pruning Schemes with YTTB for `Counter-Strike` Sequence

REFERENCES

1. "Counter-Strike." `http://www.counter-strike.net`.
2. "sauerbraten." `http://sauerbraten.org`.
3. "Half-Life TV." `http://www.hltv.org`.
4. G. Cheung, W. t. Tan, B. Shen, and A. Ortega, "ECHO: A community video streaming system with interactive visual overlays," in *SPIE/ACM Multimedia Computing and Networking, MMCN 2008*, (San Jose, CA), January 2008.
5. T. Wiegand, G. Sullivan, G. Bjontegaard, and A. Luthra, "Overview of the H.264/AVC video coding standard," in *IEEE Transactions on Circuits and Systems for Video Technology*, **13, no.7**, pp. 560–576, July 2003.
6. A. Chang, O. Au, and Y. Yeung, "A novel approach to fast multi-block motion estimation for H.264 video coding," in *IEEE International Conference on Multimedia and Expo*, (Baltimore, MD), July 2003.
7. P. Yin, H. Tourapis, A. Tourapis, and J. Boyce, "Fast mode decision and motion estimation for JVT/H.264," in *IEEE International Conference on Image Processing*, (Barcelona, Spain), September 2003.
8. Z. Zhou and M.-T. Sun, "Fast macroblock inter mode decision and motion estimation for H.264/MPEG-4 AVC," in *IEEE International Conference on Image Processing*, (Singapore), October 2004.
9. D. Wu, F. Pan, P. Lim, S. Wu, Z. Li, X. Lin, S. Rahardja, and C. Ko, "Fast intermode decision in H.264/AVC video coding," in *IEEE Transactions on Circuits and Systems for Video Technology*, **15, no.7**, pp. 953–958, July 2005.
10. T.-Y. Kuo and C.-H. Chan, "Fast variable block size motion estimation for H.264 using likelihood and correlation of motion field," in *IEEE Transactions on Circuits and Systems for Video Technology*, **16, no.10**, pp. 1185–1195, October 2006.
11. T. Akenine-Moller and E. Haines, *Real-time Rendering*, AK Peters, 2002.
12. A. Bharambe, V. Padmanabhan, and S. Seshan, "Supporting spectators in online multiplayer games," in *Prof. of third Workshop on Hot Topics in Networks, HotNets-III*, (San Diego, CA), November 2004.
13. "OpenGL: The industry's foundation for high performance graphics." `http://www.opengl.org`.
14. G. Cheung, T. Sakamoto, and W. t. Tan, "Graphics-to-video encoding for 3G mobile game viewer multicast using depth values," in *IEEE International Conference on Image Processing*, (Singapore), October 2004.
15. T. Wiegand, M. Lightstone, D. Mukherjee, T. Campbell, and S. Mitra, "Rate-distortion optimized mode selection for very low bit rate video coding and the emerging H.263 standard," in *IEEE Transactions on Circuits and Systems for Video Technology*, **6, no.2**, April 1996.
16. D. Mukherjee and S. Mitra, "Combined mode selection and macroblock quantization step adaptation for the H.263 video encoder," in *IEEE International Conference on Image Processing*, (Washington, DC), October 1997.
17. G. Cheung, "Directed acyclic graph based mode optimization for H.263 video encoding," in *IEEE International Conference on Image Processing*, (Thessaloniki, Greece), October 2001.
18. C. Zhu, "RTP payload format for H.263 video streams," September 1997. IETF RFC 2190.
19. "Yahoo Avatars." `http://avatars.yahoo.com`.
20. R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, Cambridge University Press, 2003.
21. S. Baker, "Learning to love your z-buffer." `http://www.sjbaker.org/steve/omniv/love_your_z_buffer.html`.
22. "The TML project web-page and archive." `http://kbc.cs.tu-berlin.de/ stewe/vceg/`.
23. "Sobel operator." `http://en.wikipedia.org/wiki/Sobel_operator`.