GRAPHICS-TO-VIDEO ENCODING FOR 3G MOBILE GAME VIEWER MULTICAST USING DEPTH VALUES

Gene Cheung, Takashi Sakamoto, Wai-tian Tan

Hewlett-Packard Laboratories

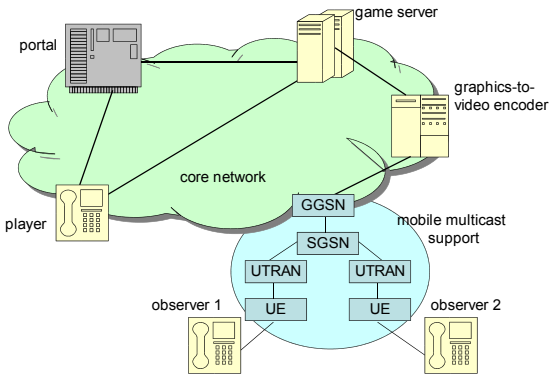


Fig. 1. Overview of GOVEM^2 Architecture

ABSTRACT

Towards the goal of streaming mobile network game sequences in standard compliant video to thousands of game observers, a specialized video encoder called *grencoder* that takes 3D game graphics as input is discussed. In particular, we present a framework that increases the visual quality of regions of interest (ROI) of the video by performing intelligent mode selection during h.263+ video encoding. The regions of interest are identified by extracting the *depth values* of pixels in the frame buffer made available when 3D objects are projected to a 2D surface during rasterization. Results show that by performing ROI mode selection, PSNR of test sequence slightly increases while subjective quality of nearby objects increases noticeably.

1. INTRODUCTION

While streaming video was believed to be the content that would occupy the expanded bandwidth of the 3G wireless networks, the fastest-growing applications, as seen in Korea and Japan, have instead been in the arena of mobile network games. Like their Internet counterparts, mobile network games require real-time interactivity, placing a stringent demand of volume timely deliverable data on the current 3G network real-time mechanism [1, 2]. Moreover, typical mobile terminals are low-powered light-weight devices with limited computing resources, making them impossible to render millions of triangles per second necessary for high-quality graphics [3]. The two reasons combined to result in today's mobile online games being limited in group size and interaction, and simplistic in visual quality.

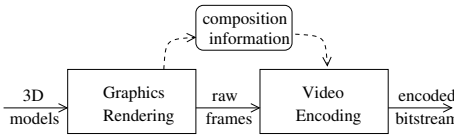


Fig. 2. Graphics-to-video Encoding

Given it is not foreseeable that either of these two inherent problems would be well solved until a fundamental advance in wireless network technology and a drastic speedup in mobile computing hardware take place, we instead focus on a different group of potential users who can flourish even within existing constraints — *game observers*. Like the Internet counterparts such as *Half-Life*, as games mature highly skilled players acquire fan base who loyally follow and observe their heroes in action en masse in multicast channels. As observers instead of active game participants, the network and hardware requirements to support the observer view are drastically different. First, the hard real-time nature of interactive games can be relaxed to a streaming scenario where an initial buffering delay up to several seconds can be tolerated. Second, if streaming *video* is delivered instead of graphics, then the burden of rendering triangles can be pushed back to a streaming server which converts graphics into standard-compliant video and then streams the encoded bitstream to interested observers. Using streaming video instead of graphics also has the added advantage of reachability: mobile handsets are much more likely to have a built-in streaming client available and ready to go than a vender-specific game client software.

Given the above observations, we have designed an architecture to support 3G mobile game viewing called GOVEM^2 (Game Observer Video Encoding and Mobile Network Multicast). The overview of GOVEM^2 is shown in Figure 1. *Game player* acquires permission and registers for an online game via the *portal*, then participates in the game in a server-client model via the *game server*. The game server sends updated game events to the player(s) as well as the *graphics-to-video encoder*. The graphics-to-video encoder, called *grencoder*, converts updated game events to encoded bitstream, then streams to interested observers using *mobile multicast support*. Among the many pieces in GOVEM^2, we choose to focus on the grencoder in this paper.

1.1. Graphics-to-video Encoder (Grencoder)

Schematically, the function of a grencoder can be separated into two parts as shown in Figure 2. A graphics rendering engine first renders 3D representation of objects onto a 2D plane in the frame buffer, a process called *rasterization* [3]. The raw frames in the frame buffer are then inputed to a standard compliant video en-

coder to be encoded into bitstream. The bitstream is subsequently packetized and sent to the interested observers.

Having the original 3D models that produce the 2D frames means the graphics rendering engine has *scene composition information* of the encoding source that are not typically available to a video encoder. In this paper, we exploit one particular type of composition information — *depth values* — to improve the visual quality of regions of interest (ROI). Depth values of objects are used so that one can discern which object is closer to the camera, and hence which objects are occluded and which are not. If we assume objects closer to camera are also objects of interest, then depth values also reveal regions of interest. In this paper, we propose to use depth values obtained during rasterization to identify regions of interest, then apply clever mode selection strategy to allocate more bits to the regions of interests to improve visual quality.

The outline of the paper is as follows. First, we discuss the framework in which we perform extraction of depth values and mode selection in Section 2. Preliminary results are presented in Section 3. We then briefly discuss related work in Section 4. Finally, we conclude in Section 5.

2. GRENCODING FRAMEWORK

The grencoding framework essentially needs to perform two tasks: i) extract depth values of objects during rasterization, ii) perform mode selection given extracted depth values during video encoding. We discuss them in order.

2.1. Depth Value Extraction

We begin with a brief discussion of the common representations of 3D graphics in the game industry. *OpenGL* [4], an industrial standard for graphics initiated by Silicon Graphics Inc., is a set of APIs (application programming interface) that enables graphics programmers to write software that can be easily compiled and optimized on a wide variety of computing platforms. Application of OpenGL is far-ranging: from medical imaging to virtual reality and CAD. In contrast, *OpenGL ES* (embedded system) [5] is a subset of OpenGL APIs, selected by a special interest industrial group *Khronos*, that is deemed essential for mobile network gaming. Using an essential subset instead of the full-size OpenGL lightens the burden of the hardware manufacturers to support a graphics specification, while enabling them to specialize in fewer APIs.

During rasterization when 3D objects are mapped to 2D plane, a depth value for each pixel mapped is calculated using Z-buffer algorithm [3] to determine object occlusion. Assuming 3D objects are expressed in OpenGL ES APIs, we write *API wrappers* for particular OpenGL ES APIs after rasterization to first extract depth value $d(j,k)$ for each pixel (j,k) from the frame buffer before calling the native APIs. This way, our technique can be easily adopted by any mobile network game developers that support OpenGL ES, and game developers do not need to make any alterations to their game software in order to reveal depth values for grencoding.

2.2. Coding Mode Selection

The coding mode selection problem in video coding is the problem of selection coding modes for a group of N macroblocks (MBs) such that the total distortion is minimized subject to a rate constraint. It has been proposed to model the inter-dependencies of a row of MBs in video standard h.263 version 2 (h.263+) linearly [6, 7], so that the rate and distortion of each MB_i, $R_i()$ and $D_i()$, depend only on mode m_i of MB_i and mode m_{i-1} of previous MB_{i-1}, if available. As such, the mode selection problem can be formalized as the following optimization:

$$\min_{m_i \in \mathcal{M}} \sum_{i=1}^{N} D_i(m_i, m_{i-1}) \quad \text{s.t.} \quad \sum_{i=1}^{N} R_i(m_i, m_{i-1}) \leq R_s, \quad (1)$$

where for h.263+ the possible mode set $\mathcal{M}$ for a P frame is: $\mathcal{M} = \{\text{INTRA}, \text{SKIP}, \text{INTER}, \text{INTER4}\}$, and R_s is the bitrate constraint for the N MBs.

Instead of solving the original constrained problem, it is common practice to solve the corresponding *Lagrangian* or unconstrained problem as follows:

$$\min_{m_i \in \mathcal{M}} \sum_{i=1}^{N} D_i(m_i, m_{i-1}) + \lambda_o R_i(m_i, m_{i-1}) \quad (2)$$

where λ_o is the Lagrange multiplier of a given value. It can be easily shown [8] that if there is a λ such that the optimal solution $\{m_i^o\}$ to (2) is such that $\sum_{i=1}^{N} R_i(m_i^o) = R_s$, then $\{m_i^o\}$ is also the optimal solution to (1). It has been shown that given the quantization parameter Q, the appropriate λ can be found empirically [9]. Given λ, (2) is typically solved by marching through a trellis and finding the shortest path within it [6, 7].

Given we have the available depth values $d(j,k)$ of each pixel (j,k), we can compute (to be discussed) the *weight* w_i of each MB_i, reflecting the level of interest for that MB. Given w_i's, we can then solve the following modified Lagrangian instead:

$$\min_{m_i \in \mathcal{M}} \sum_{i=1}^{N} D_i(m_i, m_{i-1}) + \lambda(w_i) R_i(m_i, m_{i-1}) \quad (3)$$

where the multiplier $\lambda(w_i)$, controlling the severity of the penalty function $\lambda(w_i)R_i()$, now depends on the weight w_i of MB_i. Two remaining problems need to be solve then: how to map pixel depth values $d(j,k)$'s to MB weight w_i's, and how to determine multiplier function $\lambda(w_i)$. We discuss them next.

2.2.1. Mapping Pixel Depths to MB Weights

Given we have the depth value of each pixel (j,k), $d(j,k)$, we need to calculate a weight w_i for each $16x16$ MB_i to reflect the level of interest of MB_i. We first define *anti-depth values* as the scalar difference of pixel depth from maximum depth value, i.e. $d_{\max} - d(j,k)$. We have already made one observation that the *surfaces* of objects closer to the camera (large anti-depth values) are likely to garner more viewer interest. Hence the *mean* of anti-depth values in a MB would be a good indicator of how close to the camera the surfaces of objects in the MB are likely to be. In fact, the square of the anti-depth value mean of pixels would be used to accentuate the importance of close-to-camera objects.

A secondary consideration is that the *edges* of objects are often important as viewers try to discern the shape of objects. Edges of objects in a MB would often be reflected in the *variance* of the pixel depth values. (This is not always the case. Consider a thin piece of paper on top of a desk.)

As an example, consider in Figure 3 a cone object whose bottom is closer to the camera than the top. MB_3 and MB_4 would

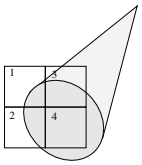


Fig. 3. Determining the Weight of MBs

have high anti-depth value mean, while MB_1 and MB_2 would have high variance of pixel depth values.

Given the two above considerations, we use the formula that w_i equals to the square of the anti-depth value mean in MB_i plus the variance of depth values in MB_i.

To control the extent to which we proportionally contribute more bits to ROIs at the expense of other MBs, we define $\gamma \geq 1$ to be the multiplicative factor such that no MB will receive more than $\frac{\gamma}{N}$ share of the bit budget. We accomplish that by defining *offset weights* $v_i = w_i + w_{off}$, with w_{off} being the offset parameter. On average MB_i will receive $\frac{v_i}{N\bar{v}}$ portion of the bit budget, where $\bar{v} = \frac{1}{N}\sum_{i=1}^{N} v_i$ is the mean of the N MB offset weights. By definition of γ, we have:

$$\frac{v_i}{N\bar{v}} \leq \frac{\gamma}{N} \qquad \forall i \in \{1, \ldots, N\} \tag{4}$$

We satisfy inequality (4) by defining offset parameter w_{off} as:

$$w_{off} \stackrel{def}{=} \frac{w_{\max} - \gamma\bar{w}}{\gamma - 1} \tag{5}$$

where $w_{\max} = \max_{i=1,\ldots,N}\{w_i\}$ and $\bar{w} = \frac{1}{N}\sum_{i=1}^{N} w_i$.

Notice that using this bit distribution strategy, we perfectly exhaust the budget R_s to the N MBs:

$$\begin{aligned} \sum_{i=1}^{N} \frac{v_i}{N\bar{v}} R_s &= \frac{R_s}{N\bar{v}} \sum_{i=1}^{N} w_i + w_{off} \\ &= \frac{R_s}{N\bar{v}} \left(N\,\bar{w} + N\,w_{off}\right) = R_s \end{aligned} \tag{6}$$

2.2.2. *Determining Multiplier Function $\lambda(w_i)$*

Suppose λ_o is selected *a priori* for original Lagrangian optimization (2) such that optimal solution $\{m_i^o\}$ has operational rate $R_s^o = \sum_{i=1}^{N} R_i(m_i^o, m_{i-1}^o)$ is the same or very close to R_s of original constrained optimization (1). The goal now is for each weight w_i of MB_i, find multiplier $\lambda(w_i)$ that will result in usage of proportion $\frac{v_i}{N\bar{v}}$ of the bit budget R_s when performing modified Lagrangian optimization (3). This means the solution $\{m_i^*\}$ to (3) will result in operational rate $R_s^* = \sum_{i=1}^{N} R_i(m_i^*, m_{i-1}^*) = R_s^o$. Having this requirement has the benefit that any previously derived formulas for λ such as [9] will have the same intended operational rate when our modified rate-distortion optimization is applied.

To derive the mappings $\lambda(w_i)$, we first need a theoretical characterization of λ and rate R. It is analyzed in [9] that the Lagrange multiplier λ corresponds to the negative slope of the distortion-rate function:

$$\lambda = -\frac{dD}{dR}. \tag{7}$$

We next assume a typical high-rate approximation curve for entropy-constrained scalar quantization can be written as:

$$R(D) = a \log\left(\frac{b}{D}\right), \tag{8}$$

where a and b are constants that parameterized the rate-distortion function. We can now see that λ is related to R exponentially:

$$\lambda = \left(\frac{b}{a}\right) e^{-\frac{R}{a}}. \tag{9}$$

One interpretation of (9) is that to achieve operational rate R_s^o for N MBs, the appropriate multiplier λ_o is found by (9). The problem is that we know neither parameters a and b, nor the intended rate R_s^o. However, we do know that $\lambda_o = \left(\frac{b}{a}\right) e^{-\frac{R_s^o}{a}}$ results in bit consumption of $\frac{1}{N}R_s^o$ per MB on average for N MBs. To achieve target usage $\frac{v_i}{N\bar{v}}R_s^o$ for MB_i then, we find $\lambda(w_i)$ that will result in operational rate $\frac{v_i}{\bar{v}}R_s^o$ and apply it to MB_i only as done in (3), so that it will consume $\frac{v_i}{N\bar{v}}R_s^o$ on average.

To find $\lambda(w_i)$ that consumes $\frac{v_i}{\bar{v}}R_s^o$ bits for N MBs, we solve for R_s^o in terms of λ_o and substitute in (9):

$$\lambda(w_i) = \left(\frac{b}{a}\right)^{1-\frac{v_i}{\bar{v}}} \lambda_o^{\frac{v_i}{\bar{v}}} \tag{10}$$

We know $\frac{b}{a} \geq \lambda_o$ from observing (9). If we let $\frac{b}{a} = \alpha\lambda_o$, where $\alpha \geq 1$, we get:

$$\lambda(w_i) = \lambda_o \alpha^{1-\frac{v_i}{\bar{v}}} = \lambda_o \alpha^{1-\left(\frac{w_i + w_{off}}{\bar{w} + w_{off}}\right)} \tag{11}$$

3. EXPERIMENTS

3.1. Implementation

To construct a testbed for grencoding, we employed `Mesa` release 5.1, an implementation of OpenGL version 1.4 API's on Linux. We wrote wrappers to extract RGB components from the frame buffer using the OpenGL (also OpenGL ES) API *glreadPixels()*, which were then converted to Berkeley YUV format as input to the h.263+ video encoder. Also using *glreadPixels()*, we extracted the depth value of each pixel from the frame buffer. The calculation in Section 2.2.1 is then performed for each MB and outputted to be used by the video encoder.

3.2. Numerical Results

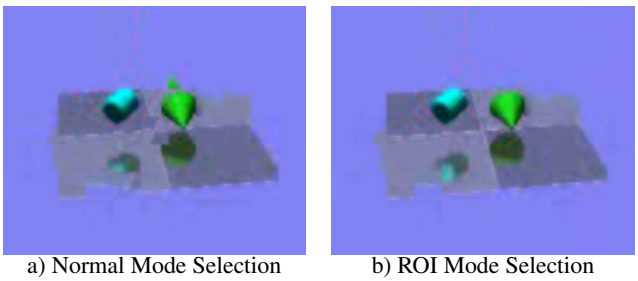
We generated three 100-frame demo sequences from `Mesa` package, `gears` and `reflect`, as our test sequences. We set the frame size at QCIF (176x144) and the frame rate at $30fps$. Using base mode selection algorithm [6] of (2), we encoded the sequences for given quantization parameters, resulting in bit-rate as shown in Table 1. Under the same parameter setting, we reran the video encoder using the ROI mode selection algorithm of (3), with γ and α set to 4 and 1.1, respectively. The Peak Signal-to-Noise Ratio (PSNR) performance of both mode selection algorithms are shown in Table 1.

From Table 1, we see that by using the depth values to adjust the multiplier value λ, the PSNR performance improves 0.36dB to 0.95dB, albeit a slight increase in bit-rate. One explanation can be that by investing in more bits in the near-camera objects, those objects are more likely to reoccur *and* not be occluded in future frames, resulting in better motion-compensation. Recall that base mode selection (2) is a frame-by-frame optimization and does not take into account this type of inter-frame dependencies.

The more telling improvement, however, can be seen visually as shown in Figure 4 and 5, where the 12th frame of sequence

sequence	Quant.	Opt	kbps	PSNR
gears	11	base	124	31.25dB
gears	11	ROI	132	31.66dB
reflect	9	base	124	32.71dB
reflect	9	ROI	134	33.07dB
gloss	5	base	123	30.86dB
gloss	5	ROI	137	31.81dB

Table 1. PSNR Comparison of Different Mode Selection Schemes



a) Normal Mode Selection b) ROI Mode Selection

Fig. 4. Visual Comparison of Sequence 'reflect'

reflect and 75th frame of sequence gloss are shown, respectively. In Figure 4, we see that on the right, the heavy bit allocation to the cone and the matt below of the ROI mode selection scheme resulted in a higher quality representation of the objects as compared to the base mode selection scheme. Similarly in Figure 5, the lid and mouth of the pot on the right is more detailed.

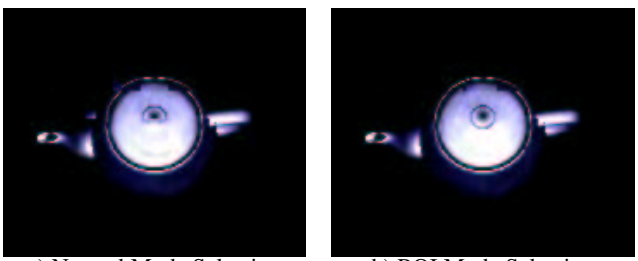
4. RELATED WORK

Intelligent bit allocation in video coding according to regions of interest has been studied previously in the literature [10]. Our work differs in that we are focusing on graphics-to-video encoding, where composition information such as depth values can be easily extracted as discussed in Section 2.1.

For compliance with the standardized packet streaming service (PSS) of 3GPP [1, 2], we specialize in the coding mode selection problem for video standard h.263 version 2 (h.263+) [11]. Mode selection for h.263+ has been extensively studied [6, 7, 12]. We leverage on these work as our starting point in Section 2.2.

One can interpret having available depth values of pixels in a frame as an improvement in source model over the basic raw video frame, and it has been shown [13] that having more informed source models does indeed improve coding efficiency. The difference in approach from [13] is that instead of using a parametric model based codec, we are constrained to have standard-compliant video as output, limiting our flexibility.

A related topic is light field coding [13, 3], outgrown from an image-based graphics rendering technique named light field rendering. To the best of the authors' knowledge, interactive game developers still use polygons in describing objects, and light field rendering is not currently supported in OpenGL specification [4].



a) Normal Mode Selection b) ROI Mode Selection

Fig. 5. Visual Comparison of Sequence 'gloss'

5. CONCLUSION & FUTURE WORK

In this paper, we have shown that using depth values available from 3D graphics rasterization, we can improve the visual quality of close-to-camera objects. We do so by writing wrappers to OpenGL ES APIs, hence requiring minimal code change to original software written by mobile game developers. The technique is sufficient general that it can be applied to other coding standards such as h.264. However, mode selection using depth values is only one concrete example of how composition information of 3D objects can be extracted for video encoding benefits. For future work, we are investigating other forms of composition information useful for compression efficiency and/or optimized mobile streaming.

6. ACKNOWLEDGMENT

The paper benefited from discussions with other members of the GOVEM^2 team in HP Labs Japan: Yasuhiro Araki and Takeaki Ota and Michael Sweeney.

7. REFERENCES

[1] *3GPP TS 26.233 Transparent End-to-End Packet Switched Streaming Services (PSS); General description (Release 4)*, ftp://ftp.3gpp.org/Specs/2001-03/Rel-4/26_series/26233-400.zip, March 2001.

[2] *3GPP TS 26.234 Transparent End-to-End Packet Switched Streaming Services (PSS); Protocols and codecs (Release 4)*, ftp://ftp.3gpp.org/Specs/2001-03/Rel-4/26_series/26234-400.zip, March 2001.

[3] T. Akenine-Moller and E. Haines, *Real-time Rendering*, AK Peters, 2002.

[4] "OpenGL: The industry's foundation for high performance graphics," http://www.opengl.org.

[5] "OpenGL ES: The standard for embedded 3D graphics," http://www.khronos.org/opengles/.

[6] T. Wiegand et. al., "Rate-distortion optimized mode selection for very low bit rate video coding and the emergin h.263 standard," in *IEEE Trans. on CSVT*, April 1996, vol. 6, no.2.

[7] D. Mukherjee and S. Mitra, "Combined mode selection and macroblock quantization step adaptation for the h.263 video encoder," in *ICIP*, 1997.

[8] Y. Shoham and A. Gersho, "Efficient bit allocation for an aibitrary set of quantizers," in *IEEE Trans. ASSP*, September 1988, vol. 36.

[9] T. Wiegand and Bernd Girod, "Lagrange multiplier selection in hybrid video coder control," in *IEEE International Conference on Image Processing*, Thessaloniki, Greece, October 2001.

[10] W.-S. Cheong, K. Kim, and G. H. Park, "A new scanning method for h.264 based fine granular scalable video coding," in *ACM Multimedia*, Berkeley, CA, November 2003.

[11] ITU-T Recommendation H.263, *Video Coding for Low Bitrate Communication*, February 1998.

[12] G. Cheung, "Directed acyclic graph based mode optimization for h.263 video encoding," in *IEEE International Conference on Image Processing*, Thessaloniki, Greece, October 2001.

[13] B. Girod et al., "3-D image models and compression — synthetic hybrid or natural fit?," in *International Conference on Image Processing*, October 1999.