A VIDEO ENCODING SPEED-UP ARCHITECTURE FOR CLOUD GAMING

Mehdi Semsarzadeh, Mahdi Hemmati, Abbas Javadtalab, Abdulsalam Yassine, and Shervin Shirmohammadi

Distributed and Collaborative Virtual Environment Research Laboratory, University of Ottawa, Canada
{msemsarzadeh | javadtalab | ayassine | shervin} @discover.uottawa.ca, mhemmati@uOttawa.ca

ABSTRACT

In cloud-based video gaming systems, game engines are hosted in the cloud, and rendered gaming scenes are streamed to players over the Internet. In such systems, the tasks of rendering graphics and video encoding impose huge computational complexity on cloud servers. Therefore, speeding up the encoding process to meet the stringent requirements of the game becomes a critical issue in cloud-based video gaming systems. In this paper, we analyze the feasibility of developing a mechanism to accelerate the power-intensive process of video encoding, by using available game objects information in game engines. Specifically, we utilize the game engine’s information about the motion of the objects within the scene in order to bypass the time-consuming procedure of Motion Estimation (ME) in conventional video encoders like H.264/AVC. Based on our analysis, the game engine’s information could be usable inside a video encoder if an interface is involved to re-shape object information and make them compatible for the video encoder. Our experiments show that our approach accelerates the motion estimation process by 14.32% on average for two specific games, when object’s information is taken into account during the encoding phase.

Index Terms— Cloud gaming, Video encoding, Game video motion estimation, Computational complexity

1. INTRODUCTION

Over the past few decades, games have become an important part of our society [1]. In 2013, the market revenue of the gaming industry reached a total of $93 billion worldwide [2]. Online and mobile games are experiencing the most rapid growth among different segments of the gaming industry. However, a large part of the future growth will come from cloud gaming, which is also referred to as the paradigm of “Game as Video” (Gav) [3]. While cloud gaming offers many advantages to both game players and developers [4], it also raises new challenges, especially in terms of computational complexity. This is because the tasks of running the game logic, updating and rendering the scenes, and encoding and streaming the resultant video are all carried in cloud servers. Also, the interactions between players bring more challenges in the design of game servers. In 2012, it was officially reported that OnLive’s financial troubles were due to their inability to sustain the numerous servers needed in order to run the service (which averaged a mere 1600 concurrent users per server). This suggests that any slight improvement in server-side processing requirements would really matter for cloud gaming service providers to maintain the profitability of their business.

The above challenge motivates us to investigate a mechanism for video encoding speed up in cloud gaming applications. The aim of the proposed mechanism is to accelerate the power-intensive process of video encoding, by using available game objects information from the game engines. Specifically, we utilize the game engine’s information about the motion of the objects within the scene in order to bypass the time-consuming procedure of motion estimation in conventional video encoders like H.264/AVC. In this paper, we investigate the feasibility of skipping the ME step of the video encoding process, using game engine’s information. In this feasibility study, an abstract representation of game engine and video encoder are taken into account. In addition, we propose a generic interface which receives the game engine’s information and makes them useable for the video encoder. Based on our analysis, a minor modification has to be made at the game engine side to be capable to provide proper information for the interface, while the same thing is true for the video encoder. We have evaluated the performance of the proposed method for a practical game engine and video encoder, where we gained up to 19.85% speed-up.

The rest of this paper is organized as follows. Section 2 provides a review of the existing work about accelerating the process of rendering game scenes and encoding video sequences. Our main idea for taking advantage of game engine’s information about objects movements to speed up the video encoding process is described in section 3 in details. To examine the feasibility of this idea, a number of simulations are presented in section 4. Finally section 5 concludes the paper.
2. RELATED WORK

Running the game engine on cloud servers imposes huge computational complexity, mainly due to intensive graphic rendering and video encoding tasks. Simultaneously, with each new video coding standard comes more complexity, due to providing almost double compression ratio in comparison to their former standards. For example, HEVC is roughly twice as efficient as H.264/AVC which itself is twice as efficient as MPEG-2, in terms of quality and/or compression ratio. However, the complexity of HEVC is considerably more than H.264/AVC which itself is more complex than MPEG-2. In this section, we first explore some of the existing works on reducing the complexity of video encoders. Then, we briefly describe the works that have accelerated the rendering process using the game engine information.

By the introduction of the H.264/AVC standard, real-time video compression became a challenge for researchers. Several studies (e.g. [6]-[10]) have focused on reducing the computational complexity of H.264/AVC standard as well as the new emerging HEVC standard. Beyond various steps of the encoding process, the motion estimation and mode decision steps have been more interesting due to their considerable complexity contribution in the encoding operation. In this regard, some methods have focused on reducing the search points of motion estimation process [6], while many others, such as [7] and [8], have proposed fast motion estimation approaches. There are also methods, which have focused on skipping mode decision operation in certain conditions, such as in [10]. All of these methods are specifically designed for video encoding part, and without considering specific characteristic of gaming video coding. Hence, they have no information about incoming stream and the potential of using the side information that may come with it.

Using game engine’s information to speed-up the rendering process has been explored by several studies; e.g. [3], [11]-[14]. In these methods some information from the game side is sent to the encoder side to improve the encoding time by bypassing some complex procedures in the encoder such as motion estimation. In [11], using the run-time graphics rendering context, the video encoding process is sped up. The authors of [12] and [13] have tried to bypass the motion estimation step by using the depth map extracted from the game engine. The authors of [14] have also used depth information to accelerate the mode selection step of H.264/AVC encoding process. Based on their observation, the regions of similar depth are likely to correspond to regions of uniform motion. They have engaged this observation in selecting the block size and in consequence speeding up the mode selection process. Finally, the authors of [3] have used their object prioritization method [15][16] to reduce the encoding complexity by removing less-important objects from the scene.

In this paper, we provide an abstract level method for speeding up the game scene encoding process by introducing an interface between the game engine and the video encoder. Unlike existing methods, such as [12]-[14], in our proposed scheme, neither a special game type (i.e. 2D or 3D), nor a specific rendering technology (e.g. OpenGL) is assumed. This makes our proposed method more generic and applicable for any type of game engine and video encoder. Our approach targets the video encoding part only, and is complementary to other approaches such as [3] which manipulate the game scene itself before encoding it.

3. SYSTEM DESIGN

Normally the game engine runs the game logic, renders the game scenes, and sends frame-based scenes to the video encoder. The video encoder treats these frames as video frames and performs compression to produce video which is then streamed to the player. The video encoder does the above without having any knowledge about the frame sources or the game engine characteristics. In our system, we propose an interface between the game engine and the video encoder, as shown in Fig. 1. This interface is responsible for collecting some information from the game engine and pre-processing them to be understandable by the video encoder. Having this information, the video encoder will be able to skip the motion estimation operation for certain blocks in the frame, leading to faster encoding.

In order to provide a practical interface, which can be used for vast combinations of game engines and video encoders, some common information has to be extracted from the game engine, modified by the interface, and fed into the video encoder. Such information would be produced by any conventional game engine and consumed by any video encoder with only slight modifications at both sides. In this regard, we have considered the object location and angle, referred as object info, as the main information passing throughout the interface.

As shown in Fig. 1, in addition to object info, the interface has to receive some other data from the game, game engine, and the video encoder, named game info, game engine info and video encoder info. The game info includes the number of objects within the game, width and length of each object, as well as the size of the game frame. The game engine info can be considered as the characteristics of the game engine, such as its coordination system. The video encoder info is the setup of the video encoder including video frame size, video frame coordination system and number of coded frames per second.

Some of the above information is sent to the interface, in an initial phase, to consider the specification of each side and be able to convert information appropriately. After performing the initialization phase, the game engine starts sending the object info to the interface for each of the game scenes. This process continues as long as the user continues playing the game. The interface receives the scene based
info from the game engine, makes them appropriate for the video encoder and sends them to the video encoder. Since the whole process has to be performed in a real-time manner, the interface has to impose only minimal latency. The interface has to perform several tasks including the scale conversion between the game and video encoder, converting scene based object info to frame based ones and providing the Motion Vectors (MV) of each MacroBlock (MB). In the following sub-section the architecture of the interface is discussed in more details.

![Diagram](image.png)

**Fig. 1.** Block diagram of the proposed method

### 3.1. Interface architecture

A detailed architecture of the interface is shown in Fig. 2. As the figure shows, in the initialization phase, the video coordination system is converted to the Cartesian system. More importantly, the game frame size and the video frame size are compared and a scaling factor is derived. Since the video encoding process is performed on a frame basis, the object info has to be reported to the video encoder only one time per frame. Whereas, the game engine reports the object info as long as the scene changes due to player and objects interactions. The interface receives the scene based object info, including object location, object angle and scene center point from the game side, as well as the frame rate of the video encoder and converts them to frame based ones. The next step is converting the frame based object info from the game engine coordination system to the Cartesian system.

In the game engine side, the position of the object is calculated by considering its distance from the left and top corners of the game, while in video encoder the positioning is made based on distance between the object and left and top corners of the frame. To make game engine’s object info compatible with the video encoder, we extract the center point of each scene and found the relative location of each object within its current scene. This process is performed in the object location adjustment block of Fig. 2. Having the current location of the object in video compatible format, and comparing it with its previous location, the movement of the object can now be extracted easily. Using the object location and its angle, the boundary of the object and the macroblocks inside that boundary can be detected as well. The motion vector of the object and the list of macroblocks within the objects would be the output of the interface to the video encoder. The video encoder receives the interface data at the start of the encoding process of each frame. For each macroblock of the current frame, the video encoder examines if the macroblock is within the object area. If so, the motion of the object will be used as the motion vector of the current MB and hence the motion estimation process will be bypassed. Otherwise, the conventional ME process would be performed for that MB. This process will be repeated for all of the MBs and frames of the video sequence.

Using the information of the game engine accelerates the compression process, while imposing some quality degradation due to the sub-optimality of MVs, in comparison with the conventional ME process. In other words, since the conventional ME process searches for the best candidate within a certain search area, it finds the best match for the current MB, while in this method only one MV will be considered as the final MV of each MB. In the next section, we will explore these tradeoffs and the cons and pros of this method for various test scenarios.

### 4. EXPERIMENTAL RESULTS

In this section, we provide the results of our experiments to evaluate the feasibility of our proposed method. We used Torque2D [17] as our game engine and selected the TruckToy and DeathBallToy games, as shown in Fig. 3. For simplicity, in both games the information of only one object is reported by the game engine, as depicted in Fig. 3. The latest reference software of H.264/AVC [18] was used as the video encoder. Along with the above tools, we also used FRAPS [19] to capture the game scenes.

In order to evaluate our proposed method, we have considered two scenarios. In the first scenario, we consider the conventional scheme, where the user plays the game, the rendering application renders raw video frames and the video encoder compresses the video frames. In the second scenario, the object’s information is reported to the interface during the gaming period. The video encoder receives the object movement info, from the interface, as well as the raw video frame, from the rendering application, and performs the compression process. For both scenarios, the setting provided in Table 1 is used for video encoding.
We have performed both scenarios for various quantization parameters (QP) and on some portions of the TruckToy and DeathBallToy games. The ME speed up as well as the total encoding time speed up of the proposed methods is shown in Fig. 4. As the figure shows, the speed up for both games and for most of the quantization parameters is almost the same. On average, using our method, the ME time and total encoding time are reduced by 14.32% and 8.86% respectively. We can also see that TruckToy has a higher speed up than DeathBallToy. The reason is that the truck object in TruckToy is larger, in terms of its dimensions compared to the dimensions of the whole fame, than the death ball object in DeathBallToy. In other words, the larger the moving object, the more speed up can be achieved by our method. This is intuitive, because a larger object comprises more MBs, and since our method skips the ME process for those MBs, a higher number of MBs means more ME skipping using our method, and more ME skipping leads to faster encoding. By the same token, we can also say that our method will gain more speed when there are more objects moving in the game scene, because in our method more objects moving also means more ME skipping and therefore faster encoding.

But as mentioned, this faster encoding time is achieved at the expense of a slight quality loss, since instead of examining all of the motion vector candidates within an area of 256×256 pixels, only the movement of the object is considered as the final motion vector of each object. To determine the amount of quality loss, the Rate-Distortion (R-D) curves for both games are extracted using MATLAB.

**Table 1. Parameter setting of video encoder**

<table>
<thead>
<tr>
<th>Profile</th>
<th>Baseline</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level</td>
<td>3</td>
</tr>
<tr>
<td>Number of coded frames</td>
<td>100</td>
</tr>
<tr>
<td>Number of reference frames</td>
<td>1</td>
</tr>
<tr>
<td>Search range</td>
<td>256</td>
</tr>
<tr>
<td>RDO</td>
<td>On</td>
</tr>
<tr>
<td>Quantization parameters</td>
<td>28-36</td>
</tr>
<tr>
<td>Motion estimation precision</td>
<td>¼ pixel</td>
</tr>
</tbody>
</table>
software, and shown in Fig. 5, where we can see that quality loss is negligible for DeathBallToy, while it is less than 1 dB for TruckToy game. Technically, the quality loss in TruckToy is the result of selecting the same integer motion vector for all portions of blocks within the macroblock, whereas in the conventional ME process, each sub-block can have a different MV value. This affects the ideal mode distribution of the encoder and may force the encoder to code some macroblocks as intra instead of inter blocks. In addition, for the objects which are overlapped or partially covered by the truck object, an incorrect motion vector may be generated, since the other objects may move toward a different direction than the truck object. This especially happens at the border area of the object. In addition, when the orientation of the object changes or it stretches or expands, the motion provided by the game engine may not be as accurate as the conventional motion estimation process. All of these shortcomings lead to having a lower quality, in comparison with the conventional video encoding method.

In general, we can see that using the object information provided by the game engine does speed up the encoding process, but more research is needed to reduce the imposed quality degradation. One potential solution could be performing a partial motion estimation process around the object location provided by the game engine. Detecting the overlap between various objects and disabling our method within these areas is another possible solution.
5. CONCLUSIONS

The feasibility of using game engine information for accelerating the video encoding process in cloud gaming applications was studied in this paper. We showed that speeding up the video encoding process using information from the game engine is indeed feasible, without any assumptions about the game at hand, and using only the interface we proposed in Fig. 2. In our method, the motion estimation process is skipped for the macroblocks which are located inside the moving object’s borders. Based on our experiments, the motion estimation process was accelerated by up to 19.85% using our method, while the quality loss was less than 1 dB. Finally, we also showed that, the higher the number of moving MBs, the more speed up our method can gain.

For future work, we will work on the problem of reduced quality, and will design and test solutions, such as those reported at the end of section 4, to improve the quality of the resulting video.

6. REFERENCES