An Action-Aware Combat Model for Efficient Video Compression of Massively Multiplayer Online Role-playing Games on Cloud Gaming Platforms

Sardar Basiri, Kaiwen Zhang, Stéphane Coulombe
Dept. of Software and IT Engineering
École de technologie supérieure
Montreal, Quebec, Canada
sardar.basiri.1@ens.etsmtl.ca, {kaiwen.zhang, stephane.coulombe}@etsmtl.ca

***Abstract*—Cloud gaming is a rising new trend for remote video gaming. Players send their commands using a thin-client device to a graphics rendering cloud server and receive a compressed video stream in response. However, video games with complex textures and motions, especially at high resolutions, require a substantial bitrate to deliver good visual quality. When the player's Internet connection is constrained or fluctuates, the visual quality may be significantly reduced, which negatively impacts the playing experience. In this paper, we present an Action-awaRe COmbat moDEl (ARCODE) for massively multiplayer online role-playing games (MMORPGs) running on cloud gaming platforms to improve compression efficiency. ARCODE captures different action data for different object types in the battle scene and determines the importance of each object relative to the player in each game state, considering the actions at the time. Based on the significance of each object to the player, the model determines how frequently its position should be updated. Reducing the number of motion updates in the scene leads to fewer bits needed to encode the video frames. Our experimental results on various test cases show that, for similar visual quality as that of the traditional approach, ARCODE can reduce the video bitrate from 9% to over 40%.**

***Index Terms*—cloud gaming, multiplayer cloud gaming, massively multiplayer online role-playing games, MMORPG**

I. Introduction

Cloud gaming [1] eliminates the need for powerful devices to play high-quality video games. Players access video games remotely over the Internet, wired or wireless, without downloading the game locally. The game itself is executed on cloud servers and the audiovisual (AV) data is sent over the Internet to the end user. As of 2021, the most recently launched cloud gaming platform is Amazon Luna [2].

Streaming video games, in essence, is similar to streaming movies. A significant difference is that in video game streaming, the control inputs are constantly sent over the Internet from the thin-client device to the cloud servers. After receiving the player's input, changes are applied to the game world. The modified scene is rendered by a graphics processing unit, compressed by the encoder, and sent to the player's device.

This work was funded by the Natural Sciences and Engineering Research Council of Canada (NSERC) Discovery Grant.

The video stream quality is an important factor affecting the player's quality of experience. Numerous works were proposed to improve the stream quality by either improving the encoder performance or by modifying the visual content so it would require fewer bits to encode at a certain fidelity [3].

Works belonging to the first category improve compression efficiency by focusing on the encoder. In [4], the rendering information is analyzed to generate a macroblock (MB) saliency map for each video frame. The encoder's quantization parameter (QP) values are determined based on the importance of each MB and the bitrate budget. The game information is also used to compute the motion vectors (MVs), which are then exploited in a fast mode selection algorithm to accelerate encoding. In [5], the authors propose a MB level rate control scheme, in the context of H.264 [6], where more bits are allocated to the region of interest (ROI) based on its importance. The same approach, but with a different spatial ROI-based weighting assignment, is proposed in [7] for the H.265 (aka high efficiency video coding (HEVC)) [8] video encoder. In both methods, after extracting the ROIs, different weights are assigned to blocks inside and outside the ROIs. In [9], the authors reduce the H.265 encoding time by exploiting the in-game objects information to pre-process the MVs instead of using the traditional diamond search. But these approaches are difficult to implement as they require specialized video coder-decoder (CODEC) expertise.

Works such as [10]–[12] belong to the second category, and their authors consider the network dynamics and manipulate the graphical content to make it suitable for video streaming. In [10], the authors investigate how different adaptive rendering parameters (e.g., realistic effect, texture and environment details, viewing distance) affect the communication and computation costs in cloud mobile gaming (CMG). They then propose an adaptive rendering method that changes these rendering settings based on the bandwidth constraints. In [11], Hemati et al. exclude the less important immobile objects from the game scene where the significance of each object in each activity in the game is provided by the game designer. Here, removing the immobile objects does not always reduce video bitrate significantly since excluding an object can expose

other objects in the scene. Their model also forces the game designer to be highly involved in the process. In [12], the authors propose a model inspired from [10] and [11] where the importance of each object in the scene is considered to either decide not to render it or to alter its texture in order to reduce the video bitrate.

In this paper, we present a new approach for the second category: an action-awaRe COmbat moDEl (ARCODE) for massively multiplayer online role-playing games (MMORPGs) on cloud gaming platforms. Our proposed model is aware of game semantics, specifically during battles. It captures different action data for different in-game object types and determines the importance of each object, considering the actions it can perform in the combat area. Based on the significance of each object to the player in each game state, the model adjusts its position update rate, which results in better compression efficiency. In our proposed model, no change is applied to the CODEC as the model merely generates visual content which can then be encoded more efficiently. This improves our solution's portability and compatibility. Furthermore, unlike other solutions, the method does not exclude objects from the game scene. To the best of our knowledge, this is the first model that adjusts the object position update rates for improving compression efficiency in cloud gaming.

The remainder of this paper is organized as follows. Section 2 is devoted to the background related to our work. In Section 3, we present our proposed model. We then present our experimental evaluation in Section 4. Finally, we conclude the work in Section 5.

II. Background

In this section, we briefly review the multiplayer cloud gaming (MCG) and MMORPGs, and present the concepts and techniques we consider to create our model.

A. Multiplayer Cloud Gaming

A multiplayer cloud gaming architecture is illustrated in Fig. 1 [13]. Each player is remotely connected to a rendering server using a thin-client device. An instance of the game runs on a rendering server for the connected player. The player sends the commands over the Internet to the rendering server. The rendering server is responsible for receiving the player's commands and apply them to the game world. The rendering servers are connected to the remote game server, which manages the updates and synchronizes the game states between all players. The remote game server should send any change in the game world to all the rendering servers before the AV data is sent back to the players. The rendering server is a server from a player's perspective and is a client of the remote game server.

B. Massively Multiplayer Online Role-playing games

Scalability is an important aspect of MMORPGs [14]. A large number of players should be able to interact in a vast virtual game world. Each player controls an avatar to interact with the other players' avatars, or non-playable characters

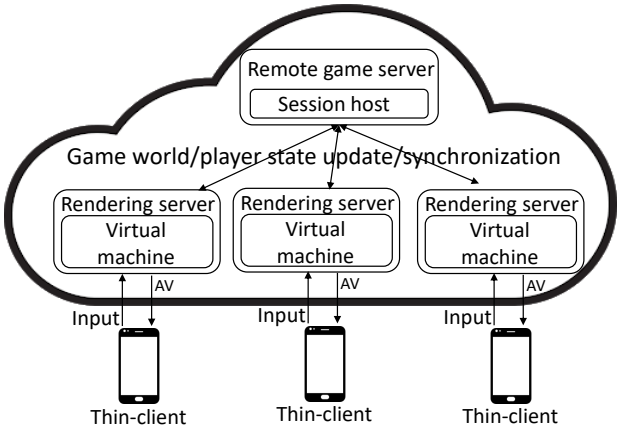

Fig. 1: MCG architecture

(NPCs) controlled by an artificial intelligence (AI) entity. In the combat area, each player and NPCs are constantly performing different *actions* (e.g, movements, using weapons and skills in the battle). Different skills and weapons are available to be associated with an avatar. Two types of weapons commonly used in MMORPGs are *melee weapons* for short-range attacks and *long-range weapons* to attack the targets from a distance. For an action performed using a weapon to be effective, the target position should be within the action's target area (e.g., the cone-shaped area for an axe-auto attack in Fig. 3). Apart from weapons, *skills* are playing an important role in combats. Each character can own various skills to use during battle. Many skills are used to affect the health of the in-game characters. *Orison of healing* is an example of a skill that heals any friendly object in its target area, while *firestorm* is a skill used to damage opponent objects in its target area (e.g., the yellow circle-shaped area in Fig.3). Usually, skills have a cooldown of a few seconds, which is the amount of time that the skill is unavailable to be used again. In the following subsections, we review the essential concepts and techniques commonly used in MMORPGs, which we take into account to conceive our proposed model.

1) Interest Management: Interest management (IM) is a technique commonly used in the traditional client-server architecture to make the MMORPGs more scalable. Different IM techniques are compared in [14]. A sequence of events is constantly happening in MMORPGs. Because of bandwidth constraints, a server cannot propagate all the state updates to all players. IM determines the eligible players to receive updates based on their avatar's area of interest (AoI). Fig. 2 illustrates a region-based IM where the game world is divided into sixteen regions of equal size. The blue edge rectangle around the player's green dot avatar specifies his AoI which is formed by regions overlapping with that rectangle (orange grids in the figure). The player only receives updates from the state of the objects within his avatar's AoI; not from the blue regions and blue dots, which are of no interest to him.

2) Area of Effect: Area of effect (AoE) [15] is a mechanism that specifies an area within which the state of an object can change. The AoE can be built on top of the AoI. In Fig. 2, the green circle-shaped area around the avatar represents the

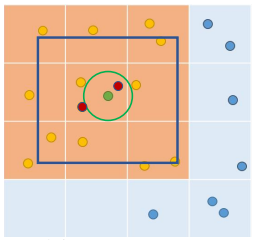

Fig. 2: Region-based interest management with an AoE built on top of it for the green dot avatar

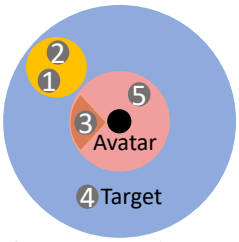

Fig. 3: Example of an avatar using an axe (weapon) and a skill to attack the opponent entities within its AoE

area within which an avatar's action can change the state of other objects. The concept of receiving different update rates based on the avatar's AoE is introduced in [16], where players receive high-frequency updates within or on the edge of their avatar's AoE and low-frequency updates outside it. Referring to Fig. 2, the player receives high-frequency position updates from the red dots while he receives low-frequency updates from the yellow dots.

For our model, we use the same concept of adjusting the position update rates using the AoE mechanism. However, the model of [16] adjusts these updates in traditional client-server architecture to limit bandwidth demand, while our model aims at improving video compression efficiency in cloud gaming platforms. Furthermore, in [16], only the player's AoE is considered when adjusting the update rates. In contrast, as presented in Section III-C, our model considers all in-game objects' AoE to adjust the update frequencies.

III. Proposed Model

In this section, we present our proposed model AR-CODE. The importance of each object in the combat area of MMORPGs is determined by the object type and the radius within which the object can change the state of other objects. ARCODE captures different action data, and based on the properties of each action associated with an object, the model determines the object's AoE. ARCODE considers two types of objects with motions in the combat area. The first type is the avatar of different players and NPCs, which we refer to as *character entities*. The second type is the *skill entities* spawned in the game world during the gameplay at the request of the character entities.

A. AoE of Character Entity

Each character entity can perform different actions during combat. Some actions, such as casting a skill, can change the state of other character entities if their position is within the range of the performed action. ARCODE considers the range of different actions the character entity can perform and forms circle-shaped areas of different sizes corresponding to the range of each action. The combination of these circle-shaped areas, each having the character entity as the center, constitute the character entity's AoE. Fig. 3 illustrates the case where a character entity owns an axe and a skill. Accordingly, its AoE consists of two circle-shaped areas. The red circle-shaped area is formed based on the range of the axe-auto attack. The blue circle-shaped area is formed based on the skill's range. In Section III-C, we explain the purpose of forming these areas and how our model manages different areas of the AoE when they overlap (e.g., the blue area covered the red area in Fig. 3).

B. AoE of Skill Entity

For an action performed by a character entity to be effective, the target position should be within the performed action's target area. Each skill's target area is considered as the corresponding skill entity's AoE. Each skill's target area size is equal to its skill entity size, which is only visual feedback to show that the skill can change this particular area's state.

In Fig. 3, the character entity has performed the *axe-auto attack* to attack the target #3. The target takes damage since its position is within the target area of axe-auto attack (cone-shaped area). The character entity also uses its skill to attack targets #1 and #2. These targets take damage since their position is within the skill's target area (yellow circle-shaped area). The targets #4 and #5 are safe in this combat state since their position is not within the target area of the performed actions.

C. Types of Situations Handled by ARCODE

The variety of weapons and skills with different ranges leads to each character entity having an AoE of different size areas. In each state of the game, the player's avatar can be within different character entities' AoE simultaneously without having these character entities inside its AoE. In [16], the player receives low-frequency updates from the position of these character entities. Since these character entities can attack the player's avatar, the player needs to receive enough updates from their position. ARCODE considers the AoE of all the objects besides the AoE of the player's avatar to address this asymmetric situation. Different situations in the combat area are taken into account by our model to decide how frequently each object's position should be updated in each game state. To explain each situation, the avatar (character entity) of the player from whose point of view the game is going on is referred to as the *client entity*, and by character entity, we mean the avatar of other players or NPCs. Please note that the player only receives video frames from the cloud gaming platform. By saying the client entity receives position updates, we mean the player's avatar in the game, which is placed on a rendering server (see Fig. 1).

1) Updates Based on the Client Entity's AoE: A client entity only receives updates based on its AoE, if it has one or more character entities within its AoE. If the character entities

are outside its AoE, it receives updates based on their AoE. In any state, the client entity receives updates based on the skill entities' AoE, as its actions do not affect them.

Each circle-shaped area of the client entity's AoE has a fixed update frequency. Depending on the area the target is within, the client entity receives updates from the target's position with the associated update rate. ARCODE considers the lowest interval between each update for the area that is formed based on the range of a weapon like axe (red area in Fig. 3). The target area of a weapon is usually small, and the effect of a performed action by a weapon *(e.g., axe-auto attack)* is applied instantaneously into the game world. Therefore, the player needs to have an accurate approximation of the target's position to attack it by his avatar's weapon. On the other hand, the skills' target area is usually big, and their effect is not applied instantaneously. Therefore, ARCODE considers lower frequency updates to be received by the client entity from the targets within the area formed by the range of a skill (blue area in Fig. 3) the client entity owns. If an action is on cooldown, the area formed based on its radius is not considered by the model until the action becomes available again. It is very likely for a target to be within different areas of the client entity's AoE (targets #3, #5 in Fig. 3) with different update frequencies. Here, the model always considers the area with the highest frequency (shortest interval between each update).

2) Updates Based on the Other Entities' AoE: ARCODE considers different parameters to calculate the update intervals at which the client entity receives from the position of other entities, based on their AoE. These parameters are the distance of the client entity to the edge of the other entity's AoE, the size of the other entity, the resolution, and the bitrate chosen by the player for the game streaming. The following formula is used to calculate the update intervals, in milliseconds, the client entity receives from the position of the given entity:

$$\mathsf{Inter}(\mathsf{E};\mathsf{CE}) = \Big(1 + \big[(w_{\mathsf{DI}} \times \mathsf{DI}(\mathsf{E};\mathsf{CE})) + (w_{\mathsf{PI}} \times \mathsf{PI})\big] \times \mathsf{RBD}\Big) \times \mathsf{Inter}_{\mathsf{min}} \quad (1)$$

where CE and E, respectively, denote the client entity and the other entity under consideration. We describe the other parameters in the following paragraphs:

- $\mathsf{DI}(\mathsf{E};\mathsf{CE})$: The *distance impact* parameter calculated in each game state based on the client entity's position with respect to the edge of other entity's AoE and calculated as:

$$\mathsf{DI}(\mathsf{E};\mathsf{CE}) = |\,\mathsf{D}(\mathsf{E};\mathsf{CE}) - \mathsf{Radius}(\mathsf{E})| \quad (2)$$

where $\mathsf{Radius}(\mathsf{E})$ is the radius of the other entity's AoE. If the other entity is a character entity, this radius is equal to the biggest range among all actions the character entity can perform. If the action is on cooldown, the next biggest range is considered by ARCODE. If the other entity is a skill entity, this radius is equal to the radius of the skill's target area. Moreover, the distance between E and CE is calculated as:

$$\mathsf{D}(\mathsf{E};\mathsf{CE}) = \sqrt{(\mathsf{E}_x - \mathsf{CE}_x)^2 + (\mathsf{E}_y - \mathsf{CE}_y)^2 + (\mathsf{E}_z - \mathsf{CE}_z)^2} \quad (3)$$

where $\mathsf{E}_x;\mathsf{E}_y;\mathsf{E}_z$ ($\mathsf{CE}_x;\mathsf{CE}_y;\mathsf{CE}_z$) denote the position of entity E, and client entity (CE) in x, y, and z, respectively. The frequency of updates increases when DI decreases. The most critical state is when the client entity is on the edge of the other entity's AoE, where enough updates from the other entity's position should be received so that the player can act rapidly. Therefore, DI increases outside and inside the AoE's range as we move further from the edge because, in these states, making an immediate decision is less critical.

- PI: The *pixel impact* is a parameter which value is determined based on the size of each object. The idea is that fewer position updates need to be received from larger entities in the scene than the smaller ones since, due to their big size, precision is less significant. It is essential that the values of PI and DI be on the same scale since we are using the weighted sum of the two factors in Equation 1. In our experiments, we considered the values 1 and 3 for small and large objects, respectively.
- w_{DI} and w_{PI}: *distance impact weight* and *pixel impact weight* are the weights assigned to the pixel impact and distance impact, respectively:

$$0 < w_{\mathsf{DI}} < 1 \quad \text{and} \quad w_{\mathsf{PI}} = (1 - w_{\mathsf{DI}}) \quad (4)$$

By changing the value of w_{DI} between 0 and 1, we modify each factor's impact on the frequency of updates in Equation 1.

- RBD: To compute the *relative bitrate difference* (RBD), the normalized difference between the standard bitrate (SB) needed by the cloud platform to render the best video quality, and the player's bitrate budget (BB) is calculated.

$$\mathsf{RBD} = 1 - (\mathsf{BB}/\mathsf{SB}) = (\mathsf{SB} - \mathsf{BB})/\mathsf{SB} \quad (5)$$

The frequency of updates is set to be inversely proportional to the relative bitrate difference.

- $\mathsf{Inter}_{\mathsf{min}}$: The shortest interval (in milliseconds) between each update. From Equation 1, when RBD is 0, then $\mathsf{Inter}(\mathsf{E};\mathsf{CE}) = \mathsf{Inter}_{\mathsf{min}}$. Otherwise, $\mathsf{Inter}(\mathsf{E};\mathsf{CE}) > \mathsf{Inter}_{\mathsf{min}}$ is obtained by multiplying $\mathsf{Inter}_{\mathsf{min}}$ by a factor depending on the above-defined parameters.

IV. Experimental Evaluation

In this section, we present our experimental setup, validation methodology and experimental parameters. We then show the experimental results and analyze them.

A. Experimental Setup

1) MMOnkey framework: We integrated our solution into MMOnkey [16], the adopted research framework for MMORPGs, which consists of two parts: server and client. The MMOnkey server is similar to the remote game server in Fig. 1. In the MMOnkey server, we implemented our model on top of its square tile interest management. The size of the interest area for each client entity is considered to be the size of the whole combat area; therefore, each player can see every entity in the scene. The MMOnkey client comprises our exemplar game (see Fig. 4), and can be seen as the rendering server in Fig. 1 which runs an instance of the game for the

connected player. The update rates for each client are managed by our model on the server.

2) Game stream setup: The game stream setup we use has the GeForce Experience software [17] installed on the game stream PC server comprising the MMOnkey framework. The GeForce Experience software streams the video frames, directly captured and encoded by the Nvidia GeForce graphics cards, to Moonlight [18] on the thin-client device. Moonlight is the open-source version of the Nvidia Gamestream [19] client and enables video game streaming from a PC having the GeForce Experience software installed to the devices where Moonlight is installed. To take advantage of video game streaming, using Moonlight and GeForce experience software, the host gaming PC requires to be equipped with an Nvidia GeForce graphics card [18]. The process of video encoding is performed by a section in the video card called Nvidia video encoding (NVENC). Our game stream PC server is equipped with a GeForce RTX 2070 graphics card, 16 GB RAM, and an Intel Core I7-9700 CPU 3.00 GHz.

B. Validation Methodology

To compare the performance of ARCODE against the unaltered game, which is referred to as the *baseline* approach, we use a metric inspired from the Bjøntegaard model [20]. The Bjøntegaard Delta (BD)-Rate measures the bitrate saving, for the same visual quality, provided by a method compared to a baseline while the BD-PSNR measures the improvement in visual quality for the same bitrate. Both metrics use the peak signal-to-noise ratio (PSNR) as the objective visual quality metric. Various bitrate and PSNR data points are needed by the Bjøntegaard model to compare the efficiency of two methods over a wide range of bitrates and qualities.

The Bjøntegaard model is widely used to compare the performance of two different compression methods on the same original video. However, in our case, we want to compare the performance of the same compression method, e.g. H.264 or H.265, on different original videos. The goal is to establish if ARCODE, which generates visually similar and comparably playable content as the baseline, permits the rendering of visually improved content under the same compression conditions.

We thus compare the visual quality of the content generated by the baseline and ARCODE after they are encoded with the same encoder and transmitted under four different bitrates, each one with respect to its own original video. The remainder of the evaluation process follows the Bjøntegaard model. Because of these differences in methodology, we call our evaluation methods *similar reference* (SR)-BD-Rate and SR-BD-PSNR. In each test, the original video is captured from the game stream PC server, and the reconstructed video frames are extracted from Moonlight.

C. Experimental Parameters

Different tests are performed to show the improvements of our model over the baseline. Our main scenario is the simulation of the common combat area in MMORPGs where many players in different regions of the combat area interact using their avatars. We implemented interactive behavior for the NPCs controlled by AI to simulate the battle and have a consistent gameplay in each round. Different in-game properties such as camera rotation and translation alongside different settings chosen for the game streaming such as the resolution and different video CODECs are considered to show the results for the main scenario. The second scenario is the boss fight, where the combat is concentrated in relatively small regions of the combat area, with all the NPCs fighting a few stronger enemies. Two sensitivity tests are also performed. For the first test, the NPCs are moving in the combat area without using their weapons and skills. For the second test, the NPCs are continuously using their skills witch results in many skill entities spawned and moving in the game world.

Each video sequence extracted from Moonlight contains 3600 frames generated from 60 seconds gameplay at 60 frames per second (FPS). For the baseline, the update interval in each state is 30 ms. In ARCODE, the position update intervals the client entity receives from the character entities within its AoE are between 30 ms to 60 ms. The update intervals are computed using Equation 1, when the client entity receives updates based on the other entities' AoE. The $\mathsf{Inter}_{\mathsf{min}}$ is equal to 30 ms. DI is calculated in each game state based on the distance of the client entity to the edge of the other entities' AoE. The PI value is 1 and 3, respectively, for small and large entities. We set $w_{DI} = 0{:}6$. The target bitrates for game streaming at 720P resolution are 1, 2, 3, 4 Mbps, while at 1080P, they are 2, 4, 6, 8 Mbps. The required bitrates for the best visual quality in 720P and 1080P resolutions are 10 and 20 Mbps, respectively.

D. Experimental Results and Analysis

The results for each test case are shown in Table I and a visual comparison is presented in Fig. 4 for H.264. More results are available in [21] (e.g., distortion curves and the image quality comparisons).

Table I demonstrates the superiority of our model over the baseline for both H.264 and H.265 video CODECs with bitrate savings ranging from 9% to over 40% depending on the test case. Based on the results of our sensitivity tests, it can be concluded that the more objects with motions in the scene, the better is the performance of our model compared to the baseline. Referring to the results of the main scenario (without camera movements) and boss fight scenario, our model offers a better performance if the motions and changes are distributed throughout the scene instead of a small portion of it. The performance of our model drops considerably when we have camera movements, including rotation or translation. The camera movements involve visual changes in video frames as new objects enter the scene and affect performance. However, the performance of our model is still superior to the baseline. Fig. 4 zooms in a part of the scene to better show the improved visual quality provided by ARCODE at the same bitrate.

V. Conclusion

The process of video encoding in cloud gaming platforms requires a high bitrate, especially for high-resolution video