ZGaming: Zero-Latency 3D Cloud Gaming by Image Prediction

Jiangkai Wu[1], Yu Guan[1], Qi Mao[2], Yong Cui[3], Zongming Guo[1], Xinggong Zhang[1]
[1]Peking University, [2]Communication University of China, [3]Tsinghua University,

ABSTRACT

In cloud gaming, interactive latency is one of the most important factors in users' experience. Although the interactive latency can be reduced through typical network infrastructures like edge caching and congestion control, the interactive latency of current cloud-gaming platforms is still far from users' satisfaction.

This paper presents ZGaming, a novel 3D cloud gaming system based on image prediction, in order to eliminate the interactive latency in traditional cloud gaming systems. To improve the quality of the predicted images, we propose (1) a quality-driven 3D-block cache to reduce the "hole" artifacts, (2) a server-assisted LSTM-predicting algorithm to improve the prediction accuracy of dynamic foreground objects, and (3) a prediction-performance-driven adaptive bitrate strategy which optimizes the quality of predicted images. The experiment on the real-world cloud gaming network conditions shows that compared with existing methods, ZGaming reduces the interactive latency from 23 *ms* to 0 *ms* when providing the same video quality, or improves the video quality by 5.4 *dB* when keeping the interactive latency as 0 *ms*.

CCS CONCEPTS

• **Networks** → **Application layer protocols**;

KEYWORDS

Cloud gaming, interactive latency, cache, adaptive bitrate.

ACM Reference Format:
Jiangkai Wu[1], Yu Guan[1], Qi Mao[2], Yong Cui[3], Zongming Guo[1], Xinggong Zhang[1]. 2023. ZGaming: Zero-Latency 3D Cloud Gaming by Image Prediction. In *ACM SIGCOMM 2023 Conference (ACM SIGCOMM '23), September 10–14, 2023, New York, NY, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3603269.3604819

1 INTRODUCTION

Cloud gaming is a promising way for next-generation 3D video games [4, 7, 9, 10, 14, 27, 32, 60]. It runs and renders games on the cloud server, and feedbacks the gameplay video to end devices. End devices do not need to download and install the game locally, and offload the heavy computation and rendering task to the remote cloud server. This enables a 'thin' client (*e.g.,* low-end PC, browser, smartphone) to run complex 3D games. However, one still unresolved problem of cloud gaming is interactive latency (*a.k.a.* response latency, the time between the user making an action and the corresponding game frame being played on the client device). Usually, the expected interactive latency should be consistently less than 60 ms in order to obtain the same user experience of instant feedback as a local game. But due to network congestion and delay variations, the Round Trip Time (RTT) of most networks, even in 5G, is far from this requirement. It becomes the main obstacle to deploying cloud gaming widely.

Is that possible to eliminate interactive latency while the complex computation and game rendering are still running on the cloud? This vision would enable a cheap device to experience a higher rendering quality but with a similar latency as the local game. This work tries to provide a solution for zero-latency 3D cloud gaming.

Various pioneer efforts have been devoted to reducing the interactive latency of cloud gaming [21–23, 31, 44, 53, 55, 66, 67], such as real-time video codec [55, 66], low-latency congestion control [21, 31], edge computing [67] or fog computing [44], etc. Due to network physical delay and video coding delay, they still suffer from at least one RTT interactive latency. Action-based video prefetching [42] is the first effort to zero interactive latency. To eliminate the interactive latency, it predicts multiple potential actions at a time and renders a video frame for each predicted action on the server. These frames are then prefetched to the client. On the client, once the user plays an action locally, the corresponding prefetched frame is displayed immediately. However, user actions are usually very random. It is hard to predict them accurately, especially for the long-term period. Besides, the logic of most games is to render a single frame at a time based on the real action of the user. So it needs to modify the game's logic to enable rendering multiple frames for different actions at a time [42]. This is infeasible for commercial cloud-gaming platforms with thousands of games.

Unlike action-based video prefetching, image-based prediction is another promising and straightforward solution to eliminate interactive latency. Instead of predicting actions and then rendering video frames, it directly predicts future frames based on the color and 3D geometry of past frames. The most widely used prediction method is DIBR (Depth Image Based Rendering) [34, 46, 47], which is a 3D Image Warping algorithm, as illustrated in Figure 1. In DIBR-based cloud gaming, once the end user makes a new action (*e.g.,* movement), according to a recently received RGBD (RGB and Depth) image and the viewpoint movement, the end device directly predicts the video frame under the moved viewpoint instead of waiting for receiving it from the server. In this way, end users do not experience any interactive latency. In addition, different from action-based prefetching, all the information required by DIBR can

Jiangkai Wu (jiangkai.wu@stu.pku.edu.cn), Yu Guan (shanxigy@pku.edu.cn), Zongming Guo (guozongming@pku.edu.cn) and Xinggong Zhang (zhangxg@pku.edu.cn) are with Wangxuan Institute of Computer Technology, Peking University. Qi Mao (qimao@cuc.edu.cn) is with the State Key Laboratory of Media Convergence and Communication, Communication University of China. Yong Cui (cuiyong@tsinghua.edu.cn) is with Department of Computer Science and Technology, Tsinghua University.

ACM SIGCOMM '23, September 10–14, 2023, New York, NY, USA

ACM ISBN 979-8-4007-0236-5/23/09…$15.00
https://doi.org/10.1145/3603269.3604819

be hooked from the rendering pipeline of the original games [2, 13], without modifying the games' logic.

We are motivated by the question: How accurately an image-based prediction method can respond to user actions in a cloud gaming system? Generally, we face three technical challenges:

First, there are artifacts (missing pixels) in the predicted image, as illustrated in Figure 2(c). For example, due to occlusion or out-of-view, some contents are not sampled in the reference view and DIBR is unable to generate them correctly, causing artifacts. For simplicity, we use "hole" to refer to these artifacts in this paper. To solve this problem, some works on multiple reference views have been proposed [46, 58], aiming at enriching the reference content. However, these methods require modifications to the game's code to render images from multiple viewpoints at a time.

Second, DIBR does not work for dynamic objects in the reference frame. For example, as illustrated in Figure 2(c)(d), the predicted image of the moving character is very different from the ground truth (such as the part circled in red). This is because the character is in motion, its real position, pose and appearance are constantly changing, but DIBR can only predict images by projecting its outdated and static pixels stored in the reference frame. So the predicted images hardly match the real images.

Third, there is a trade-off between video bitrate and prediction performance when streaming. On the one hand, increasing the video bitrate can reduce the image distortion caused by encoding, so that the reference frame has a higher quality, resulting in a better prediction performance. On the other hand, under constrained bandwidth, increasing the bitrate [25, 26] potentially leads to an increase in interactive latency, resulting in a longer prediction period, which in turn reduces prediction performance.

Addressing these challenges, this paper explores the design space of the cloud gaming system to optimize the performance of image-based prediction and proposes ZGaming, a zero-latency 3D cloud gaming system. In ZGaming, a frame is divided into the static background and dynamic foreground. For the static background, the client stores multiple frames that have already been played and uses these history frames along with the most recently received frames as reference frames for DIBR. As the history frames provide more sufficient 3D information, the artifacts on the predicted frames are recovered. For the dynamic foreground, instead of DIBR, an LSTM (Long Short-Term Memory) model is proposed to predict images of dynamic objects after deformation or movement. Finally, a prediction-performance-driven Adaptive Bitrate (ABR) algorithm is proposed that adaptively selects the optimal video bitrate based on current user actions, game contents, and network conditions. Overall, the system design of ZGaming entails three contributions:

Quality-driven 3D-block cache. ZGaming proposes to store history frames on the client to recover the "hole" artifacts, however, the storage overhead of history frames is huge and the limited storage capacity of the client constrains the performance of hole filling. To store history frames efficiently, ZGaming further proposes a utility-based block-caching strategy. The proposed strategy splits history frames into blocks and computes the utility value of hole filling for each history block. In real-time cloud gaming, the client only caches the blocks with the highest utility value.

Server-assisted LSTM-predicting foreground. ZGaming proposes an LSTM model that estimates the motion and deformation trends of foreground objects from history frames, and therefore accurately predicts the images of these dynamic objects after the interactive latency. In order to adapt to client devices with limited computing power, the computational overhead of the LSTM is offloaded to the server side and the prediction results are pre-fetched to the client side. To further enhance performance, ZGaming proposes to improve the priority of foreground objects in packet sending and apply Forward Error Correction (FEC), in order to ensure they can be completely received by the client in time.

Prediction-performance-driven adaptive bitrate. ZGaming first analyzes the relationship between prediction performance and interactive latency, reference frame quality, user action, and video content, proposing a relationship model. Then, ZGaming analyzes the relationship between interactive latency and video bitrate, network conditions, proposing another relationship model. According to these models, ZGaming chooses the video bitrate that maximizes the prediction performance under certain network conditions.

We implemented a prototype of ZGaming and evaluated it using a real-world dataset of Grand Theft Auto V [3], a popular 3D game. The result shows that compared with existing methods, ZGaming reduces the interactive latency from 23 ms to 0 ms when providing the same video quality, or improves the video quality by 5.4 dB when maintaining the interactive latency as 0 ms. While achieving these gains, ZGaming has some limitations, such as being weak in responding to sudden actions, poor prediction quality when the interactive latency is large, being time-consuming in image prediction, and increased bandwidth usage, etc., as detailed in § 8.

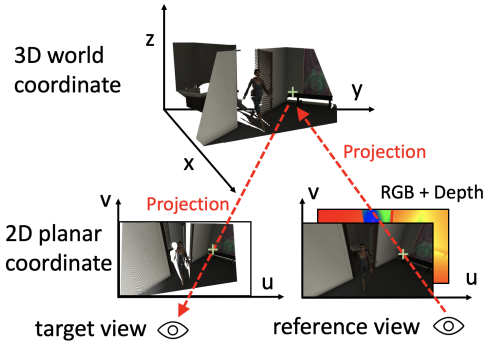


Figure 1: The definition of DIBR. To predict the image of the target view, based on the depth map, DIBR projects each pixel in the reference view from 2D planar coordinates into the 3D world coordinates and then projects these 3D points into the imaging plane of the target view.

2 MOTIVATION

We begin by setting up the background of cloud gaming (§ 2.1) and the DIBR-based cloud gaming architecture (§ 2.2). Then we present the gap between the real-world performance and the potential performance of zero-latency cloud gaming. Finally, we point out the main insights (§ 2.3).

2.1 Cloud gaming

Advanced 3D games usually require high-performance hardware support. For example, for Cyberpunk 2077 [5], if someone wants to

Some components of ZGaming are open-source at: https://pku-netvideo.github.io/zgaming/

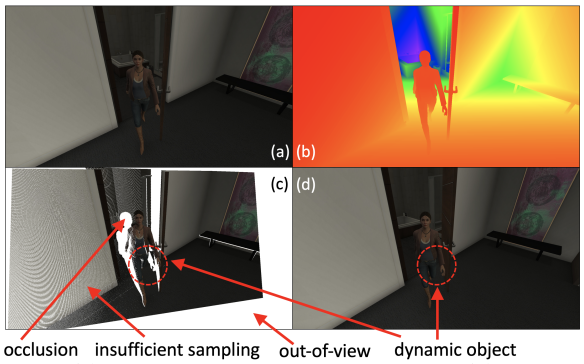


Figure 2: (a) The image in the reference view. (b) The depth map in the reference view. (c) The predicted image in the target view. There are some artifacts. (d) The ground truth image in the target view.

experience high rendering quality (*e.g.,* enabling ray tracing at 4K resolution), NVIDIA Geforce GTX 3090 is required. The cost of a GPU alone can be thousands of dollars. However, users hope to play games with lightweight devices, such as laptops or smartphones, which are far away from supporting advanced 3D games.

Cloud gaming is a promising solution to satisfy users' requirements. The workflow of cloud gaming is similar to that of remote desktops. The user's action is sent to the cloud server, where the game software is stored and executed on. The cloud server takes the user's action as input, then generates the gameplay video. Finally, the gameplay video is streamed back to the user for play. In this way, the huge computation overhead is offloaded to the cloud server, while the end device only needs to act as a video player.

The biggest challenge of cloud gaming is interactive latency. Unlike local gaming which directly renders the gameplay on the end device, the cloud gaming system needs to collect the player's actions, transmit them to the cloud server, process the actions, render the result frame, encode the video, and finally stream it back to the player. After all these processes, the resulting frame is played on the user's screen. State-of-the-art cloud-gaming platforms can provide around 120 ms interactive latency in good network conditions. If network congestion happens or the cloud server is overburdened, the latency can be several hundreds of milliseconds. However, for advanced 3D games, the interactive latency is required to be consistently less than 60 ms [29, 30]. There is still a big gap between current cloud gaming systems and users' expectations.

2.2 Cloud gaming with DIBR

Is it possible to make the user not experience the interactive latency, even if the network transmission, video encoding&decoding are still working? In 3D cloud gaming, the video frames are rendered by 3D models. Adjacent frames are usually generated by the same set of 3D models. So there are always strong spatial correlations between adjacent frames. Given this opportunity, DIBR is a promising solution to eliminate interactive latency in cloud gaming.

DIBR utilizes the 3D Image Warping algorithm (Figure 1) to predict the new frame based on an existing frame. In DIBR-based cloud gaming [24, 28, 58], the cloud server not only transmits the color images to the end device but also transmits the depth maps and the viewpoints. If a frame does not arrive at the expected time, the end device will use DIBR to predict the expected frame based on the latest received frame. Therefore, users will not experience any interactive latency, which greatly improves the user experience.

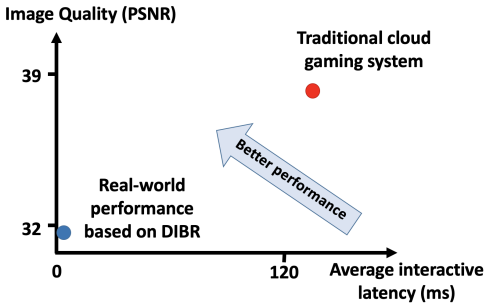


Figure 3: Compared with the traditional cloud gaming system, enabling DIBR greatly reduces the interactive latency while degrading the image quality.

Two advantages of DIBR make it suitable for cloud gaming. First, since DIBR only needs to compute the coordinate projection, it can run in real time on embedded CPU/GPU [69] or on custom hardwares [48]. Second, it does not require modifications to the logic of games. All the information required is the intermediate data that exists during the running of the original game and can be fetched through Graphics API hooking (more details are in §A.1).

2.3 Potential gains and insights

Figure 2 presents an example of DIBR performance in cloud gaming. We can see although most regions are accurately predicted, there are still many artifacts and mismatches in the image. We further evaluate the performance of DIBR in a cloud gaming system [57], as shown in Figure 3. We use Peak Signal-to-Noise Ratio (PSNR) to measure the quality of the predicted image. The higher the value, the more similar the predicted image is to the ground truth image (refer to Equation 4). The result shows DIBR leads to a 7.0 dB drop in PSNR. It indicates that although the DIBR method guarantees zero latency, it significantly degrades the image quality.

Based on the above analysis, in the zero-latency 3D cloud gaming system, the core challenge is how to improve the performance of image prediction. Specifically, we have the following three insights that have potential gains:

Prediction performance for DIBR can be improved by history gameplay frames. A major drawback of DIBR is that it introduces many "hole" artifacts due to a lack of 3D information. So how to get more 3D information? Based on the real-world dataset [3], we observe that in a 3D game, most scenes are visited at least twice by the user. Some special scenes may even be visited frequently in a short period of time, such as the location of an important Non-Player Character (NPC). Usually, gameplay frames from adjacent viewpoints are highly similar because there are many identical 3D contents. Therefore, we can make full use of these adjacent frames in the gameplay history to enrich the 3D information for DIBR.

To analyze how much the history frames can improve the prediction performance for DIBR, we evaluate the performance of DIBR with different numbers of reference frames on the dataset [3], as shown in Figure 4. Specifically, if only the latest received frame is referenced, 97.5% pixels on the predicted image can be covered when the reference distance (the difference between the frame numbers of the reference frame and the target frame, used to measure the interactive latency) is 1, but it drops to 79.6% when the reference

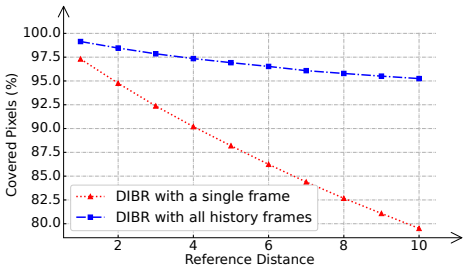


Figure 4: When DIBR references all history frames, pixel coverage increases significantly. In this way, the prediction performance is improved because the "hole" artifacts from uncovered pixels are reduced.

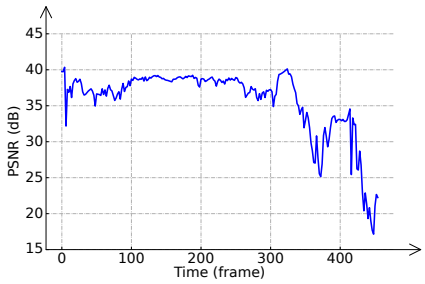


Figure 5: An example of how prediction performance varies during the video playback. Between the 100th and 300th frames, the in-game character moves relatively smoothly. After the 300th frame, the character moves violently.

distance is 10. The uncovered pixels on the predicted image will become "hole" artifacts. On the other hand, in addition to the latest received frame, if all the history frames are referenced, when the reference distance is 1 and 10, the pixel coverage of the predicted image can reach 99% and 95.1% respectively. It can be seen that adding the history frames can provide more 3D information for DIBR, so it can achieve better prediction performance.

Prediction performance for foreground objects can be improved by LSTM. Since the foreground objects are usually moving or deforming, the 3D information recorded in the reference frame of DIBR is outdated, resulting in mismatches between the predicted image and the ground truth image. Fortunately, for most 3D games, the mask and depth map obtained through API Hooking (§A.1) can be used to separate the dynamic foreground and static background. This provides the prerequisites for predicting the dynamic foreground with a method different from DIBR. Based on the dataset [3], we observe that foreground object changes continuously between adjacent frames and are correlated in time. Is it possible to learn this trend of change from the sequence of previous frames? So that we can directly predict the image of the object after the change. Nowadays, LSTM based models are commonly used to process such sequence data. Some existing works [59, 63–65] also demonstrate their excellent performance on video frame prediction. So it is promising to leverage an LSTM model in our cloud gaming system to improve the prediction performance for dynamic objects.

Prediction performance can be optimized by an adaptive streaming strategy. According to our analysis above, there is a tradeoff between video bitrate and prediction performance. Existing ABR algorithms [25, 26] tend to increase the bitrate under the constrained bandwidth, which limits the prediction performance. To demonstrate this, we select a sequence of frames from the dataset [3] and evaluate its DIBR performance in the cloud gaming system, as shown in Figure 5. Specifically, between the 100th and 300th frames, the in-game character moves relatively smoothly and regularly, so the prediction performance is good and stable. During this period, increasing the video bitrate has the potential to further improve the prediction performance because the longer-term prediction may still work well and the quality of the reference frame dominates the prediction performance. After the 300th frame, the character moves violently, so the prediction performance is poor and fluctuates. At this point, prediction becomes difficult. Increasing the bitrate will lead to increased latency, subsequently further degrading the prediction quality. Conversely, reducing the video bitrate to reduce the interactive latency can keep the prediction performance within an acceptable range. Therefore, it is promising to design a quality-driven adaptive bitrate algorithm that takes into account various influencing factors (such as user behavior).

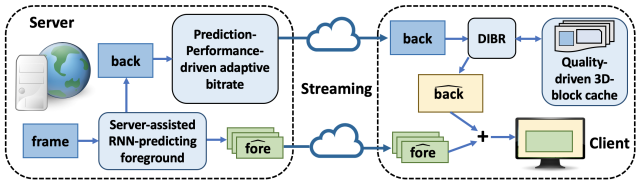


Figure 6: ZGaming overview

3 OVERVIEW

In response to the above challenges and insights, we design three modules for ZGaming: Quality-driven 3D-block cache (§ 4), Server-assisted LSTM-predicting foreground (§ 5), and Prediction-performance-driven adaptive bitrate(§ 6). The architecture of ZGaming is shown in Figure 6 and its workflow is as follows:

Server-assisted LSTM-predicting (SL) foreground reads in the full frames rendered by the game software and divides each frame into static background and dynamic foreground objects. For each foreground object, the LSTM model predicts its images after the time of current interactive latency based on its history images. These predicted foreground images will be sent to the client with a higher priority than the background image.

Prediction-performance-driven adaptive (PPA) bitrate reads in the background images from the SL foreground module and sends these images to the client at an adaptive bitrate. To choose the bitrate, it first identifies the current network conditions. Then, based on the proposed relationship models, it calculates the expected prediction quality at different video bitrates. Finally, the bitrate that maximizes the prediction quality is chosen.

Quality-driven 3D-block (Q3B) cache provides additional 3D information for DIBR to recover artifacts. Specifically, DIBR first uses the newly received background image to predict an image with artifacts. Then, DIBR fetches 3D blocks from the Q3B cache to recover these artifacts. The recovered predicted image will be combined with the received foreground images to form a full frame for playback. Finally, the Q3B cache will be updated according to the recovery performance of each 3D block. Besides, the newly received background image will be added to the Q3B cache.

To avoid ambiguity, the term 'server' in this paper refers to the cloud gaming server (used for rendering game frames and streaming

to end devices), rather than the game backend server (used for maintaining and switching gaming data); and the term 'client' in this paper refers to the cloud gaming client on end devices (used for video receiving and playing), rather than the game software.

4 QUALITY-DRIVEN 3D-BLOCK CACHE

In this section, we first describe how ZGaming efficiently stores history frames and propose the Q3B cache. Then, we evaluate the prediction performance gains from the Q3B cache.

4.1 Quality-driven 3D-block caching strategy

Although the user's gameplay history can significantly improve prediction performance, the end device needs huge memory to store the history frames. For instance, the full information of a single history frame (including RGB, depth, and viewpoints, with a resolution of 1920*1080) roughly occupies 11.9 MB of storage space. If every frame is cached directly, a second of gameplay history at 30 FPS would incur a storage overhead of 357 MB, which is prohibitive. A feasible solution is to reduce redundancy between frames, storing contents at the same 3D location only once. However, even then, a 100-second gameplay history still occupies up to 451.7 MB, putting severe strain on the storage systems. Compression can further reduce storage occupancy, but real-time encoding and decoding will impose significant computational pressure on lightweight devices.

To cache the continuously generated gameplay history in limited storage, cache replacement is inevitably required. Traditional methods generally replace the infrequently used contents when the cache storage is full. However, we find that the history frames generated by users at different times and different 3D locations have different effects on the improvement of prediction quality. Ignoring these differences and merely replacing them based on frequency would limit the performance gains of the cache. Specifically, it is influenced by the following factors:

The lighting effect of the 3D world. For example, in outdoor scenes, affected by sunlight, the color and the luminance of content generated at noon are highly different from the same content generated at night. As a result, if we use the content at noon to predict a night scene, there will be a significant error. Although some adaptive luminance methods can adjust this (our implementation about this is in § A.2), there is still a big gap with the ground truth image. In contrast, in indoor scenes with fixed lighting conditions, the luminance of contents remains unchanged at different times, so as to provide high-accuracy 3D information for image prediction.

The competition of inpainting algorithms. For hole artifacts, apart from using history contents, some simple inpainting algorithms can be used for hole filling, such as nearest neighbor interpolation. Sometimes, the performance of inpainting even surpasses that of history contents. For instance, for the holes on the wall in Figure 2(c) caused by insufficient sampling or out-of-view issues, the nearest neighbor interpolation can almost perfectly fill them due to the wall's solid color and lack of texture. For history contents, affected by lighting, their luminance might differ from the current frame, resulting in a weaker recovery performance compared to the luminance-consistent nearest neighbor interpolation. Besides, history contents are always first utilized for hole-filling, and if there are any remaining holes, the inpainting is invoked for further filling. Once the holes have been filled with history contents, inpainting will no longer be applied. Therefore, caching the contents of this wall area is not only useless but could even have a negative impact.

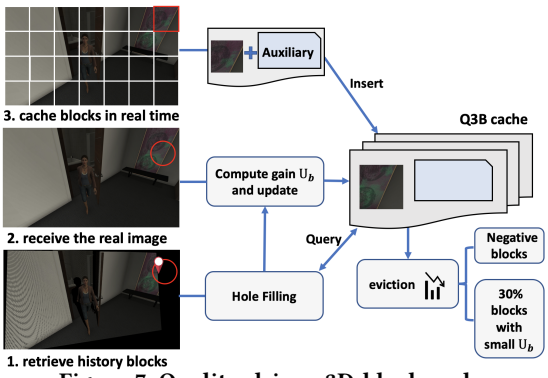


Figure 7: Quality-driven 3D-block cache

The influence of user behavior. There will be some high-frequency access areas in the game, such as scenes with important missions, which users will repeatedly appear in. Contents in these areas will be used more frequently in subsequent predictions. Therefore, caching the contents in these areas has more potential to improve prediction performance compared to areas that users seldom visit.

Considering all the above factors, we present the quality-driven 3D-block (Q3B) caching strategy (Figure 7). The caching strategy inputs all the video frames generated by the users' gameplay, computes the utility value of caching for each area in each frame, then replaces the low-value contents in real time. First, in order to remove the redundant information (*e.g.*, the adjacent two frames usually have many redundant pixels representing the same contents), we split each history frame into 64*64 blocks (the block size is a hyper-parameter). Thus for the same content, we only need to cache one block. Then, for each block b, we define the utility function U_b to quantify the long-term prediction performance gain from caching this block:

$$U_b = \frac{\sum_{i=1}^{N_b}(PSNR_{b,i} - PSNR'_{b,i})}{T_{now} - t_b} \quad (1)$$

where N_b represents the times of being used since the block b entered the cache. $PSNR_{b,i}$ represents the PSNR of b's covered area in the ith use, $PSNR'_{b,i}$ represents the PSNR of the corresponding area if b is not used in image prediction (which means the area is repaired by inpainting algorithms). As a result, $PSNR_{b,i} - PSNR'_{b,i}$ can be regarded as the prediction performance gain in this use of b, thus $\sum_{i=1}^{N_b}(PSNR_{b,i} - PSNR'_{b,i})$ is the total prediction performance gain since b entered the cache. T_{now} is the current time stamp, and t_b is the time when b entered the cache, so $T_{now} - t_b$ is the span of time b is stored in the cache. In all, the ratio of b's prediction performance gain and the duration b is cached, is defined as b's utility value in unit time.

Finally, given the above definition of prediction performance gain of each block, once the end device storage is full, the caching strategy evicts 30% blocks (the eviction rate is a hyper-parameter) to free the storage space. In addition, when the utility value of a block is less than zero, it indicates that its performance is inferior to that of inpainting. To prevent these negative blocks to make the prediction performance worse, the caching strategy periodically evicts them, even though the cache is not full.

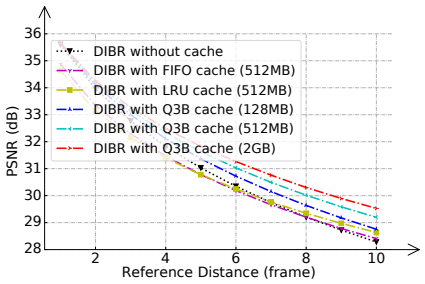


Figure 8: Performance gains from Q3B cache. First, the Q3B cache can significantly enhance prediction performance. Second, the larger the cache capacity, the more pronounced the performance improvement. Third, the effect of the Q3B cache greatly surpasses the strawman strategies LRU and FIFO.

4.2 Improved performance for background

We evaluate the performance of the Q3B cache, as shown in Figure 8. Firstly, compared to DIBR without cache, Q3B cache can enhance prediction performance by up to 1.25 dB. Additionally, the improvement becomes more pronounced as the reference distance increases, indicating that the advantage of the Q3B cache is more noticeable under poorer network conditions. Secondly, when the cache capacity increases from 128 MB to 2 GB, the performance of the Q3B cache improves by 0.77 dB (when the reference distance is 10). This again proves the gain of long-term gameplay history on image prediction. Thirdly, compared to the strawman caching strategies Least Recently Used (LRU) and First In, First Out (FIFO), Q3B can improve prediction performance by 0.56 dB and 0.79 dB respectively (when the reference distance is 10). This demonstrates the huge differences in the effects of different history content. Ignoring these differences and replacing them based only on usage frequency or time will weaken cache performance. Fourthly, the performance of DIBR with LRU cache and DIBR with FIFO cache is even poorer than DIBR without cache when the reference distance is small. This proves that some history contents can have a negative effect on prediction performance.

5 SERVER-ASSISTED LSTM-PREDICTING FOREGROUND

In this section, we propose the SL foreground module. We first describe how ZGaming separates the foreground from the background. Second, we present the LSTM model. Third, we evaluate the performance gains from the proposed LSTM. Finally, we introduce a server-assisted prediction architecture to offload the overhead.

5.1 Separating foreground from background

In order to predict the foreground using LSTM instead of DIBR, it is necessary to separate the foreground from the background. Noting that ZGaming considers an object as the foreground only when it is dynamic, irrespective of its in-game logic. For instance, in the case of interactive objects, if they remain stationary throughout the gameplay, ZGaming still classifies them as part of the background, despite they are designed to be movable.

To identify dynamic objects, first, ZGaming obtains per-frame depth maps, mask maps, and viewpoint information of the gameplay by employing API Hooking. The mask map indicates to which object each pixel belongs, enabling the segmentation of the full image. At

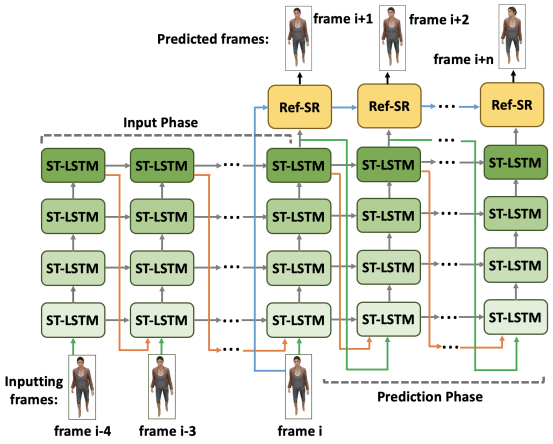


Figure 9: The architecture of the proposed LSTM model

this stage, the image is divided into objects but dynamic and static objects have not been distinguished. To distinguish them, the most intuitive solution is to calculate the centroid of the object on the 2D image and compare the 3D coordinate difference of the centroids between two adjacent frames. If the difference is large, it is classified as a dynamic object. However, due to changes in viewpoint, the areas of the object on adjacent frames may vary, causing some static objects to have a large shift in centroid coordinates. Therefore, ZGaming warps the RGB image of each object in the current frame onto the RGB image of the previous frame by DIBR and calculates the PSNR for the overlapping areas. If the mean PSNR of an object over the last $\eta = 10$ frames is less than $\varepsilon = 29\ dB$ (both η and ε are hyperparameters), this object is considered a dynamic foreground object; otherwise, it is classified as part of the static background. This makes use of the property of DIBR, whose prediction accuracy is low for dynamic objects and high for static objects when the reference distance is 1.

5.2 LSTM-based foreground prediction

When designing the LSTM model for predicting dynamic objects, two main factors need to be addressed:

Motion. To tackle the motion, deformation, and appearance changes of foreground objects, we predict images for these dynamic objects with Spatiotemporal Long Short-Term Memory (ST-LSTM) [64] rather than DIBR. Specifically, we use four stacked ST-LSTM units, which can predict the changing trend of the image sequence in the temporal and spatial domains simultaneously. Therefore, the ST-LSTM stack can directly predict future frames for dynamic objects by inputting their history frames.

Blur. Although the ST-LSTM stack successfully reduces mismatches in dynamic object prediction, its predicted images are relatively blurry, which greatly impacts visual quality. This is due to the fact that the neural network tends to learn the low-frequency signal (smooth areas in the image) [50], resulting in the predicted image losing many texture details (high-frequency) and being blurred. To address the issue of blur, we append a Reference-Based Super-Resolution (RefSR) unit after the ST-LSTM stack. In early attempts, we use SRCNN [33], a data-driven image super-resolution method to directly refine the blurry predicted images. However, we find that these purely prior-based methods have a limited ability to increase the resolution (only 2x to 4x) and therefore cannot compensate for

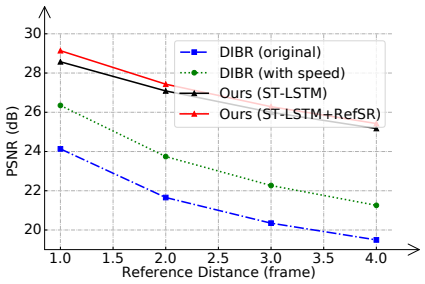


Figure 10: Performance gains from our LSTM model, which improves the accuracy of dynamic object prediction in two aspects: using ST-LSTM units to learn changing trends, and using a RefSR unit to address the blurring issue.

the blurring degradation in our predicted images. Fortunately, the inputting history images always overlap greatly in content with the predicted images, providing a good reference for super-resolution. So we instead use RefSR [45] to transfer high-frequency details from the history images to the blurry predicted images.

Combining the above two factors, the architecture of our LSTM is shown in Figure 9. At the input phase, 5 image frames are fed into the ST-LSTM stack in turn. In each ST-LSTM unit, the information of the current frame is extracted, memorized, and transferred to the step for the next frame. At the prediction phase, the fourth ST-LSTM unit will output a predicted image, which will be fed into the RefSR unit along with an inputting image. In the RefSR unit, the inputting image is used as the reference information to repair the blurred areas of the predicted image. Finally, the output of the RefSR unit is the prediction result.

5.3 Improved performance for foreground

We compare our LSTM model with three baselines. The first baseline uses the original DIBR to predict foreground objects. The second baseline not only uses DIBR but also moves the foreground object in 3D space according to its current speed of movement, so as to model the changes of objects preliminarily. The third baseline solely uses the ST-LSTM stack to predict foreground objects, without the RefSR unit to handle blurriness.

According to Figure 10, it can be seen that our method significantly improves the performance of foreground prediction. Firstly, compared with the original DIBR, DIBR with speed can improve the image quality by 2.21 dB, while ST-LSTM can improve it by 4.44 dB. This suggests that modeling the changes of foreground objects can enhance image prediction. Additionally, it indicates that the LSTM-based nonlinear algorithm has a higher accuracy when modeling these changes. Secondly, after adding the RefSR unit, our method further improved by 0.57 dB, for a total improvement of 5.01 dB, demonstrating that the accuracy of image prediction is further increased after addressing the blurring issue.

5.4 Server-assisted foreground object prediction

Despite our LSTM model greatly improving the accuracy of image prediction for foreground objects, it requires intensive computation, which lightweight end devices cannot satisfy.

To solve the problem, we propose a server-assisted foreground object prediction architecture. Since the server is usually located in the data center, which has sufficient computing resources and capacity, it is reasonable to offload the inference of LSTM there. First, the server separates each foreground object from the complete image. For a separated foreground object, the server takes the latest images as input to the LSTM and predicts its future images. Given that the server cannot know in advance the precise interactive latency at the moment the client receives the predicted images, it is necessary to predict multiple future images (in our setup, it is 3 frames around the current interactive latency) to increase the probability of covering the actual interactive latency. Then, the predicted images of foreground objects will be streamed to the client. To enable clients to receive foreground objects more promptly, the foreground part is transmitted as a separate stream with a higher priority than the background, and an FEC packet is inserted for every 3 packets. When the client receives the predicted foreground images, it will select the predicted images that match the currently expected frame id. The selected foreground images and the background generated by DIBR make up the complete image for playback.

6 PREDICTION-PERFORMANCE-DRIVEN ADAPTIVE BITRATE

In this section, we present the PPA bitrate module. We first analyze in cloud gaming, which factors can influence image prediction performance on the client side. Based on these factors, we present two relationship models, respectively concerning prediction performance and video streaming performance. Based on these models, we propose an adaptive bitrate algorithm and test its performance.

6.1 What factors influence the performance of image prediction in cloud gaming?

In the image prediction based cloud gaming system, there are three factors that influence the prediction performance:

Video content: As analyzed in § 2.3 and shown in Figure 5, in the former part of the sequence, the prediction performance is consistently high, but in the latter part, due to the user's actions, too many new contents enter the user's view, thus prediction performance is consistently limited. Based on the same analysis of all frame sequences in the dataset, we have two conclusions. First, in different segments of the frame sequence, prediction performance is highly different. Second, for sequential frames in a short period of time (*e.g.*, the continuous 10 frames), prediction performance is similar, especially when user behavior is relatively smooth. The above two conclusions will be considered important insights in the design of the following bitrate adaptation algorithm.

Reference frame quality & reference distance: Figure 11 shows the average prediction performance under different reference frame quality and reference distance. The result shows that as reference distance increases, the prediction performance decreases. Besides, when the reference distance is small, the prediction performance relies more on the quality of the reference frame, so a good choice of reference frame quality can greatly improve prediction performance. In contrast, when the reference distance is large, prediction performance relies more on whether the reference frame can cover most pixels of the predicted frame after DIBR. In this case, the quality of the reference frame has a smaller impact on prediction.

6.2 Modeling the performance of image prediction in video streaming

Based on the analysis in the last section, prediction performance relies on video contents, reference frame quality, and reference

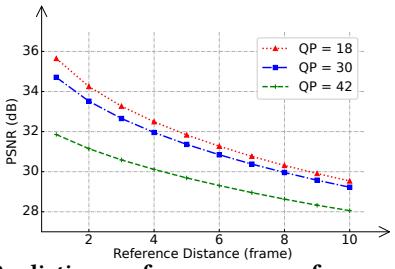


Figure 11: Prediction performance v.s. reference frame quality & reference distance

distance. Thus, if all these variations are known, we can make the decision to optimize prediction performance. From the perspective of a cloud gaming system, video content is decided only by the game software itself and the user's action, thus it can be regarded as a constant input in server-side bitrate adaptation. The reference frame quality is the output of the server (which can decide the bitrate of each frame). The reference distance is different, which is neither the input nor the output of the server. Reference distance is related to both the video streaming logic and the network condition. As a result, modeling the reference distance is the key to modeling the performance of image prediction.

We present a measurement study to build the relationship model of how the video streaming logic (the video bitrate) and network condition (including network delay, bandwidth, and loss rate) influence reference distance. We randomly vary the bitrate, network delay, and packet loss rate per second at different network bandwidths, measure the actual reference distance of each frame, and plot the probability distribution of the reference distance under different conditions. See § 7.1 for the setup of this experiment.

The results at different bandwidths indicate that when the network bandwidth exceeds the video bitrate, further increasing the network bandwidth has a limited influence on the reference distance. When the network bandwidth is less than the video bitrate, the reference distance is continuously accumulated to an extremely large value. Therefore, to analyze the relationship between the reference distance and bitrate, network latency, and loss rate, we show the results with sufficient bandwidth.

Figure 12(a) shows the probability distribution of different reference distances at different video bitrates when the video bitrate is less than the network bandwidth. In video streaming, each network packet has an upper limit of packet size (around 1400 bytes), thus when the video bitrate is increased, each frame will be allocated in more network packets. In a fixed network condition (*e.g.,* a fixed packet loss rate), the probability of all packets arriving at the client side at one time is decreased, thus the server has to resend the lost packets. So increasing video bitrate has a positive relationship with reference distance. Figure 12(b) shows the probability distribution of different reference distances at different loss rates. The result indicates that when the packet loss rate increases, the probability of larger reference distances increases and the probability of smaller reference distances decreases. Figure 12(c) shows the probability distribution of different reference distances at different network delays. The result indicates that as the network delay increases, the similar probability distribution shifts to larger reference distances.

Based on all the above results, the probability distribution of different reference distances is defined as a function of reference

Table 1: Dataset summary

Number of test videos	101
Total duration (second)	12353
Resolution	1920 x 1080
Frame rate	30

distance $r \in \{0, \ldots, R\}$ (where R is the maximum reference distance), video quality q, network bandwidth B, network delay d and loss rate l, denoted by $P(r, q, B, d, l)$. Again, based on the conclusion in §6.1, since the video content has continuity, given the prediction quality model based on current video content, the model will basically keep unchanged in a short period of time, thus the expected prediction quality of the current frame is defined as a function of reference distance r, video quality q and frame id i, denoted by $Q(r, q, i)$. Therefore, for the ith frame, the expected prediction quality under video bitrate q is:

$$E(q, B, d, l, i) = \sum_{r \in \{0,\ldots,R\}} P(r, q, B, d, l) * Q(r, q, i) \tag{2}$$

In the real-world implementation, since the function P cannot be presented by a mathematical formula, we use the random sampling results of P in different r, q, B, d, and l mentioned above as a function. Because q, B, d, and l are all floats, if the variables are between two adjacent samples, we apply linear interpolation to get the value of P under the specific input. Similarly, Q is calculated by conducting a linear extrapolation based on the history prediction quality.

6.3 Server-side bitrate adaptation

On the server side of cloud gaming, the adaptive bitrate algorithm assumes the client side always keeps the interactive latency as zero, which means once the user makes an action, the client side does image prediction based on DIBR immediately. Thus, the server will adjust the bitrate in real time to optimize prediction performance under this assumption.

Suppose the server is streaming the ith frame, the current network bandwidth is B, the loss rate is l and the network delay is d. The server will choose the bitrate of the next frame as follows:

$$q_{chosen} = \arg\max_{q} E(q, B, d, l, i) \tag{3}$$

Figure 13 shows an example of adaptive bitrate performance. Compared with the existing solution which directly chooses the closest bitrate with the currently available bandwidth, our proposed algorithm tries to model and optimize client-side prediction performance, thus getting a better result.

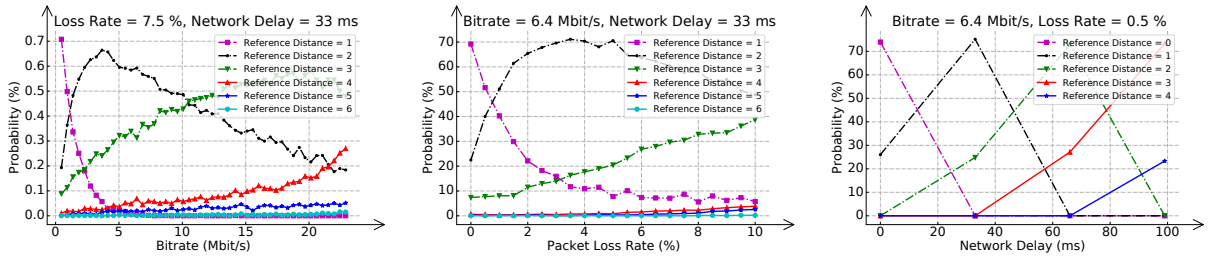
7 EVALUATION

Our three main evaluation results are as follows: First, the video quality of ZGaming is better than existing methods in 92% of cases. Second, compared with existing methods, ZGaming reduces the interactive latency from 23 ms to 0 ms or improves the video quality by 5.4 dB. Third, each module in this paper substantially contributes to the performance gain.

7.1 Implementation and Setup

We implement a prototype of ZGaming by C++, Pytorch, and CUDA. Here, we describe implementation details in the experimental setup.

Baseline. We compare ZGaming with two existing cloud gaming systems, GamingAnyWhere [35] and DIBR [58]. GamingAnyWhere



(a) Reference distance v.s. bitrate (b) Reference distance v.s. loss rate (c) Reference distance v.s. delay

Figure 12: Mathematical modeling of reference distance

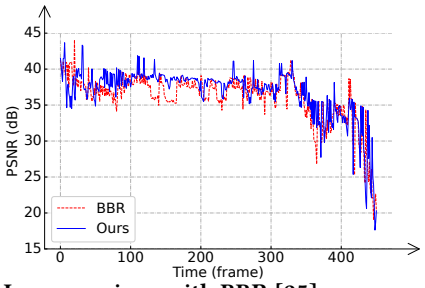


Figure 13: In comparison with BBR [25], our proposed adaptive bitrate algorithm can give the client better prediction performance in most cases.

is a well-known traditional cloud gaming system without prediction. Based on WebRTC [12], GamingAnyWhere utilizes basic RTP/RTCP to stream the game video. The advantage is it always provides the best video quality to the user since the client side always displays the real image. The problem is it does not optimize for the interactive latency. DIBR is GamingAnyWhere integrated with the traditional image-based prediction (single reference DIBR). DIBR significantly reduces the interactive latency but decreases the video quality because of artifacts. On the other hand, for comparison with the proposed ABR algorithm in § 6.3, all baseline systems adopt GCC [26] and BBR [25] for bitrate adaptation.

Dataset. We use Grand Theft Auto V [3] dataset, as summarized in Table 1. In order to make the evaluation more objective, we do not customize our own dataset but use a public dataset. To the best of our knowledge, this dataset is the only public one that meets our requirements: long-duration game recording, depth maps, mask maps, and viewpoints. In our experiment, the dataset is divided into 57 training videos and 44 testing videos. The training videos are only used for the training of the LSTM model, while the testing videos are used for the testing of the LSTM, Cache, and ABR models, as well as for the end-to-end performance evaluation. To prevent the LSTM from overfitting to specific scenes, the training and testing videos are recorded in different game scenes, and the appearances of the foreground objects in the test videos are different from that in the training videos. When streaming, we apply x264 [1] to encode the test video in 5 quality levels (QP=18, 24, 30, 36, 42).

Network trace. For the realism of evaluation, we collect network traces from a real-world popular commercial cloud gaming platform [17]. A WiFi-connected mobile phone is used to collect network traces in 3 different locations. We collect traces every hour from 9 a.m. to 23 p.m., obtaining a total of 14 network traces. Each trace is about 10 minutes long, including the longitude&latitude of the collecting location, bandwidth usage per second, network delay per second, and packet loss rate per second.

Network simulation. We utilize WebRTC (a peer-to-peer real-time communication framework) to send & receive packets. Specifically, we instantiate two WebRTC peers, one as the server side that sends the game frames and one as the client side that receives the game frames. To test ZGaming in dynamic and reproducible network conditions, we implement a trace-based network emulator. This network emulator acts as a relay for communication, receiving, and forwarding packets between the server and the client. In the emulator, NetLimiter [19] is called to limit the network bandwidth, and packets are dropped or delayed with a specific distribution (the specific parameters are read from the trace). With this emulator, we dynamically adjust the bandwidth, network delay, and loss rate according to the collected real-world network traces.

Metrics of user experience. We apply video quality and interactive latency as two dimensions of user experience.

- **Video Quality.** For video quality, PSNR is used as the metric, which can be expressed as follows:

$$PSNR = 10 \times \log_{10} \frac{255^2}{MSE} \tag{4}$$

$$MSE = \frac{1}{w \times h} \sum_{i=0}^{w-1} \sum_{j=0}^{h-1} \|p(i,j) - \hat{p}(i,j)\|^2 \tag{5}$$

  where w, h denote the width and height of images, $p(i,j)$ and $\hat{p}(i,j)$ denote the pixel at (i,j) of the ground truth image and that of the predicted image respectively. Intuitively, when PSNR increases k dB, the mean squared error between the predicted image and the ground truth image will drop to $\frac{1}{10^{k*0.1}}$. So the higher the PSNR, the better the quality. Note that all PSNR values in this paper are calculated against lossless raw images.
- **Interactive latency.** Interactive latency is the time between the user making an action and the corresponding game frame being played on the client device. We compute the average interactive latency during the whole gameplay as the metric.

7.2 Bandwidth Usage

This section shows the bandwidth usage of different systems. Compared with GamingAnyWhere, ZGaming, and DIBR require additional bandwidth to send the extra information. For GamingAnyWhere, only the RGB (24 bits per pixel) of the full image is streamed. For DIBR, in addition to RGB, pixel-wise depth (24 bits per pixel) is also transmitted. For ZGaming, it streams the depth and RGB of the background and the predicted RGB for the foreground.

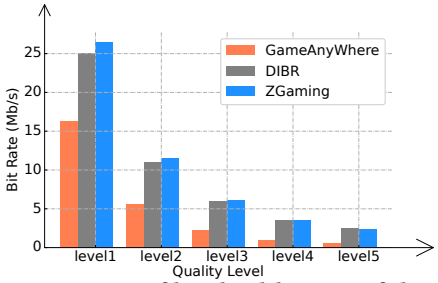


Figure 14: Comparison of bandwidth usage of the three systems in different video qualities

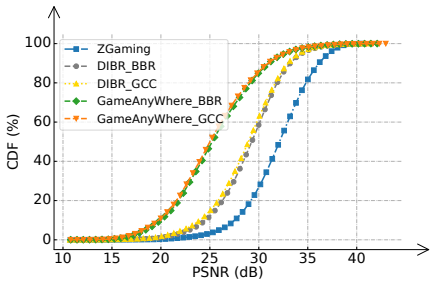


Figure 15: The CDF of video quality while maintaining zero latency. It indicates that ZGaming can significantly improve the video quality experienced by users.

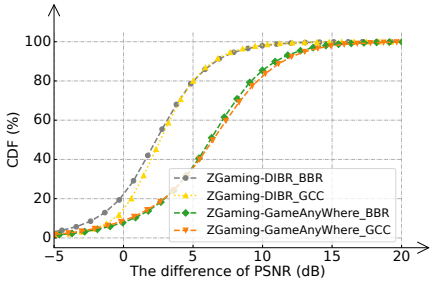
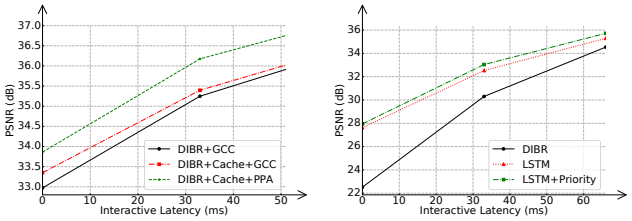


Figure 16: The CDF of the frame-wise quality difference between ZGaming and baselines. It shows the video quality of ZGaming is better than existing methods in 92% of cases and indicates ZGaming is robust to prediction errors.



(a) background (b) foreground

Figure 17: Ablation study. The result shows each module proposed in this paper contributes to the performance gain.

7.3 Overall Performance

This section shows in the real-world cloud gaming network conditions, the video quality when each system maintains interactive latency as zero. Since GamingAnyWhere cannot compensate for interactive latency, its latency is always the real value and cannot be zero. For comparison, we calculate the image similarity (in PSNR) between its latest received frame and the desired frame as the video quality when GamingAnyWhere keeps the latency as zero.

Video Quality. Figure 15 shows the CDF of video quality. It indicates that ZGaming can significantly improve the video quality experienced by users. For example, ZGaming's video quality is above 33 dB in 40.0% of cases, compared with only 9.5% for DIBR_GCC and 4.0% for GameAnyWhere_GCC. On the other hand, ZGaming can significantly reduce the probability of users experiencing low video quality. For example, ZGaming has only 3.0% of cases where the video quality is below 25 dB, while DIBR_GCC has 14.5% and GameAnyWhere_GCC has 50.5%. The above improvement is reasonable. Instead of using outdated images, DIBR responds to user input with predicted images, so it can achieve better performance than GameAnyWhere. ZGaming further improves the performance of image prediction, so it achieves better performance than DIBR.

Robustness. ZGaming contains some prediction-based models (*e.g.,* LSTM in § 5.2, P and Q models in § 6.2), which may have prediction errors. To quantify the impact of prediction errors, we calculated the frame-wise quality difference between ZGaming and the baselines, as shown in Figure 16. The results show that ZGaming achieves better video quality in the vast majority of frames. For example, the video quality of ZGaming is better than DIBR_GCC in 85.5% of the frames and better than GameAnyWhere_BBR in 92.0% of the frames. On the other hand, even if prediction errors occur, the degradation of image quality is very limited. For example, ZGaming's video quality is 5 dB lower than DIBR_GCC's and GameAnyWhere_BBR's in only 1.3% and 1.4% of frames respectively. It shows that ZGaming is robust to prediction errors.

7.4 Ablation study

The ablation study is presented in this section. Based on DIBR, we add the modules presented in this paper one by one, to show the performance gain of each module.

Background. For the background, Zgaming improves DIBR in two aspects: (1) Q3B cache, and (2) PPA bitrate. As shown in Figure 17(a), based on DIBR, when applying Q3B cache, we improve video quality by 0.38 dB, or reduce 31% interactive latency (at the $PSNR = 34dB$ line). Then, PPA bitrate improves video quality by 0.51 dB, or reduces 80% interactive latency (at the $PSNR = 34dB$ line). Put the above modules all together, when maintaining the interactive latency as zero, ZGaming improves video quality by 0.89 dB, or reduces interactive latency from 13 ms to zero.

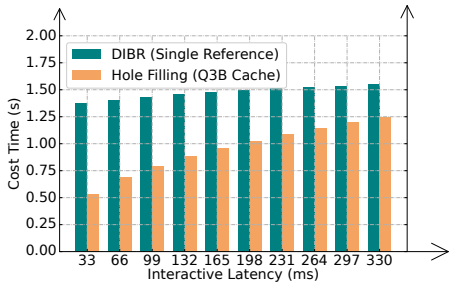


Figure 18: Client-side time cost on a single CPU core.

Foreground. For the foreground, ZGaming improves DIBR in two aspects: (1) SL foreground's LSTM prediction, and (2) SL foreground's priority streaming. As shown in Figure 17(b), LSTM prediction improves video quality by 5.1 dB, or reduces 89% interactive latency (at the $PSNR = 28dB$ line). Then, priority streaming improves video quality by 0.32 dB, or reduces 16% interactive latency (at the $PSNR = 30dB$ line). In total, when maintaining the interactive latency as zero, ZGaming improves video quality by 5.4 dB, or reduces interactive latency from 23 ms to zero.

7.5 Overhead

In this section, we analyze the overhead of ZGaming on the client and server side, including time consumption and storage occupancy. For most existing commercial cloud-gaming platforms [4, 7, 10, 14], the server side runs in the cloud data center with abundant GPU resources. Therefore, we evaluate the server overhead on high-end GPUs (NVIDIA GeForce RTX 4090 [8], 24GB VRAM) and the CPU (AMD Ryzen9 7950X). On the other hand, the client side usually runs on mediocre PC, browsers, and mobile devices with common characteristics of weak graphic computing power and limited storage space. So we evaluate the client overhead on a low-end GPU (NVIDIA Quadro M2000 [8], 4GB VRAM) and the CPU (Intel Xeon E5-2630v4) to simulate the target devices. In ZGaming, the Q3B cache runs on the client side, while the SL foreground and the PPA bitrate run on the server side. The SL foreground, which requires the execution of a computation-intensive LSTM, incurs significant overhead, while the overhead of the PPA bitrate is negligible in comparison. Therefore, for the server overhead, we focus on the LSTM in the SL foreground.

Server Side. The overhead of the LSTM consists of two parts: training overhead and inference overhead. First, the training of the LSTM is conducted on two RTX 4090 GPUs, taking 32.7 hours and occupying 17.8 GB of VRAM. We only train the ST-LSTM stack, while for the RefSR unit, the pre-trained parameters are retained. It is important to note that the LSTM model has been trained offline before being integrated into the cloud gaming system, and no more training is needed while the game is running, so the training overhead does not affect the runtime performance of ZGaming. Second, the inference overhead of the LSTM is related to the interactive latency that needs to be compensated. This is due to the ST-LSTM stack operating in a step-by-step manner. As illustrated in Figure 9, to predict the i+n frame, the ST-LSTM stack needs to be executed n+4 times. Fortunately, some intermediate results in the ST-LSTM can be reused for the next prediction and the RefSR unit can be computed in parallel. In the experiment, the inference of the ST-LSTM stack and the RefSR unit is separately conducted on two RTX 4090 GPUs. To predict frame i+1 to i+n for a foreground object of 1024*1024 pixels, the ST-LSTM stack takes (n+1)*13.9 ms, the RefSR unit takes 224.8 ms, totaling (n+1)*13.9+224.8 ms. In terms of storage, they occupy a total of 41.0 GB of VRAM. Although the LSTM model cannot run in real time, we believe that future optimizations in implementation and advances in hardware will greatly enhance its efficiency, as clarified in § 8.

Client Side. Predicting an image on the client side involves two main steps: performing DIBR with the most recently received frame and filling holes with the Q3B cache. To evaluate the computational overhead of the hole filling, we implement DIBR and hole filling in C++, and measure the respective time cost of these two steps on a single core of the CPU, as shown in Figure 18. As the interactive latency increases, the time cost of DIBR increases slightly (around 1.5 s), while the time cost of hole filling varies significantly (increasing from 0.53 s to 1.25 s). This is because the cost time for DIBR mainly depends on the resolution of the reference image but the cost time for hole filling depends on the area of holes in the predicted images. With the increase of interactive latency, the area of the holes becomes larger, so the number of blocks required to fill the holes increases, resulting in an increase in cost time. Noting that the time cost of hole filling also takes into account the updating of the Q3B cache. The time cost of the hole filling is less than that of DIBR, thereby demonstrating the efficiency of both the hole-filling process and the Q3B cache. Although the time cost of both DIBR and hole filling is large on a single CPU core, it can be greatly reduced through parallelization. As a reference, we implement a CUDA version of DIBR and test it on the GPU. The result shows that when the interactive latency is 330 ms, the time cost of DIBR is only 16 ms. This also suggests that hole filling has the potential to achieve real time through parallelization. On the other hand, the Q3B cache does not take up too much storage. As shown in Figure 8, with any storage capacity (even as little as 128 MB), the Q3B cache can obtain a stable performance improvement.

8 LIMITATION

LSTM prediction. The LSTM model has the following three limitations: First, it is weak in response to sudden actions. According to Figure 9, the LSTM model predicts based on a sequence of 5 continuous frames, so it struggles to accurately predict sudden actions that last less than 5 frames, such as a sudden jump. Second, the LSTM model exhibits poor prediction performance when interactive latency is high. This is because the ST-LSTM stack uses the predicted result of the previous frame as the input for the next frame's prediction, leading to an accumulation of blurring degradation until even the RefSR unit can no longer handle it. Specifically, when the interactive latency exceeds 6 frames, the images predicted by the LSTM model become considerably blurry. Third, the efficiency of the LSTM model requires optimization. According to the overhead evaluation in § 7.5, the inference time of the LSTM model is substantial. We do not include this inference time in the evaluations of § 7.3 and § 7.4. However, this inference time is also part of the interactive latency in practice, and sometimes the latency reduced by its prediction may be less than the latency introduced by its inference. So its efficiency optimization is inevitable in the future.

Server-side prediction. Offloading the LSTM inference to the server introduces the following limitations: First, the server-side prediction can lead to extra bandwidth usage. Given that for each

foreground object, the server needs to predict its frames at multiple time points and prefetch them to the client. The multiplying of the foreground frames can lead to extra bandwidth usage during transmission. The reason this issue is not reflected in the bandwidth usage comparison between ZGaming and DIBR in § 7.2 is because the proportion of the foreground in the evaluated dataset [3] is small (approximately 10%). For some games where the foreground occupies a large proportion of the frame, such as Splatoon [20], this could result in a dramatic increase in bandwidth usage. Second, server-side prediction can cause client-side stuttering when the expected foreground frame is not received. For instance, when the interactive latency changes drastically (more than ±2 frames), predicting 3 frames around the current interactive latency cannot always cover the actual interactive latency, leading to stuttering when the client does not receive the expected foreground frame. Similarly, if some predicted foreground frames are lost during transmission, it would also potentially result in client-side stuttering.

Foreground separation. The algorithm for separating dynamic foreground described in § 5.1 has some failure cases in practice. For example, for a static object, if there is a huge color difference between two frames due to lighting effects, it would be misidentified as a dynamic object because of the low PSNR. For a dynamic object, if it lacks texture and its color is similar to the background, it would have a high PSNR and be misidentified as a static object.

Generality. First, ZGaming is not general to some games. Specifically, ZGaming performs Graphics API Hooking by replacing DLLs (Dynamic Link Library). However, some games verify these DLLs before launch to identify whether they have been replaced by fake ones, such as Genshin Impact [18]. As the hooking fails, ZGaming is unable to acquire the data, thus it cannot work. Second, the pre-trained LSTM model performs well on videos from the game it has been trained on, but it may not necessarily perform well on videos from other unseen games. So it requires fine-tuning for new games.

9 RELATED WORK

Image prediction. Image prediction methods can generally be classified into 3D-based prediction and 2D-based prediction. The most commonly used 3D-based method is DIBR [47], which projects pixels according to known 3D information to form a novel view image. DIBR is simple and efficient, so there have been continuous improvements, such as using Splatting [54, 61, 72] to handle insufficient sampling, Layered Depth Images (LDI) [28, 37, 62] to reduce occlusions, and Blending to enable view-dependent colors [38, 43, 61, 70]. DIBR is only applicable to static scenes where everything remains unchanged. However, there are many dynamic objects in 3D games. To cope with these dynamic objects, some 3D-based prediction methods [49] estimate the 3D motion of the pixels, calculate the 3D coordinates of the pixels after the motion, and then project them onto the target view. In this way, although the motion is handled, it cannot deal with deformations, or appearance changes (*e.g.*, affected by lighting) of dynamic objects. To solve this, ZGaming proposes to separate dynamic objects from the full image and predict them separately with a 2D-based method. 2D-based prediction [59, 63–65] does not require 3D information, it uses LSTM to directly predict the future color of each pixel based on a sequence of history images. Thanks to the powerful fitting ability of LSTM, it can handle various types of changes in dynamic objects. However, while these methods successfully predict changes, they can lead to a blurring degradation of the predicted images. To address this issue, ZGaming proposes the use of RefSR [45] for post-processing to further enhance the prediction quality.

Prediction in cloud gaming. There are other existing works aimed at achieving low-latency cloud gaming through prediction. Shi et al. [56–58] also use DIBR to predict images at the client. Differently, the server needs to additionally render some specified viewpoints, which will be used together in DIBR to reduce holes. Outatime [42] has the server render images with a larger Field of View (FoV) to reduce the holes. In addition, it also enumerates user inputs, rendering images following these enumerated inputs at the server and prefetching them to the client. Furion [40] divides the scene into foreground interactions and a background environment. The foreground interactions are rendered at the client based on their 3D models, while the background is pre-rendered at the server according to the predicted viewpoint and then prefetched to the client. The above works all require customized modifications to the game logic in order to render images under specified viewpoints or user inputs, which are not natively supported by games.

Caching strategy for DIBR. There exist works aiming to improve DIBR performance by designing caching strategies. For example, Lee et al. [41, 68] design a Markov Decision Process based caching strategy for multi-view 3D videos, enabling the cache on the mobile proxy to respond to user requests not only through cache hits but also through the DIBR synthesis. For this problem, Zhang et al. [71] propose a proactive caching strategy that prefetches images into the cache by predicting user movements, further enhancing the ability of DIBR synthesis to respond to user requests. However, these strategies are used for caching 3D multi-view videos, most of which have only a small number of views (usually 16 or 32) [71]. In 3D games, an infinite number of views are created as the player moves through the scene. A large number of views will lead to the failure of such strategies or unacceptable computation.

10 CONCLUSION

In this paper, we propose a novel 3D cloud gaming system based on image prediction, in order to eliminate the interactive latency in the traditional cloud gaming systems. To improve the quality of the predicted images, we make three contributions. First, we present a quality-driven 3D-block cache to reduce the "hole" artifacts. Second, we improve the prediction quality of dynamic foreground objects by LSTM-based model. Third, we design a prediction-performance-driven adaptive bitrate algorithm that tries to optimize the performance of image prediction. The experiment shows that compared with existing methods, ZGaming reduces the interactive latency from 23 ms to 0 ms when providing the same video quality, or improves the video quality by 5.4 dB when maintaining zero latency.

This work does not raise any ethical issues.

Acknowledgements. We sincerely thank our shepherd Junchen Jiang, and anonymous reviewers for their valuable feedback. This work is sponsored in part by the National Natural Science Foundation of China (No.U21B2012, 62201526, 62132009, and 62221003). We gratefully acknowledge the support of the State Key Laboratory of Media Convergence Production Technology and Systems, Key Laboratory of Intelligent Press Media Technology. Xinggong Zhang is the corresponding author.

REFERENCES

[1] 2013. X264. (2013). https://www.videolan.org/developers/x264.html.
[2] 2018. gamehook. (2018). https://github.com/philkr/gamehook.
[3] 2020. GTA-IM-Dataset. (2020). https://github.com/ZheC/GTA-IM-Dataset.
[4] 2022. Amazon Luna. (2022). https://www.amazon.com/luna/landing-page.
[5] 2022. Cyberpunk 2077. (2022). https://www.cyberpunk.net/.
[6] 2022. DirectX. (2022). https://en.wikipedia.org/wiki/DirectX.
[7] 2022. Google Stadia. (2022). https://stadia.google.com.
[8] 2022. gpu. (2022). https://developer.nvidia.com/cuda-gpus.
[9] 2022. Microsoft Xbox Cloud Gaming. (2022). https://www.xbox.com/en-US/xbox-game-pass/cloud-gaming.
[10] 2022. NVIDIA GeForce Now. (2022). https://www.nvidia.com/en-us/geforce-now/.
[11] 2022. opengl. (2022). https://www.opengl.org/.
[12] 2022. Razor. (2022). https://github.com/yuanrongxi/razor.
[13] 2022. RenderDoc. (2022). https://renderdoc.org/.
[14] 2022. Sony PlayStation Now. (2022). https://www.playstation.com/en-us/ps-now/.
[15] 2022. ue. (2022). https://www.unrealengine.com/.
[16] 2022. unity. (2022). https://unity.com/.
[17] 2023. Cloud Gaming for Genshin Impact. (2023). https://mhyy.mihoyo.com.
[18] 2023. Genshin Impact. (2023). https://genshin.hoyoverse.com/.
[19] 2023. NetLimiter. (2023). https://www.netlimiter.com.
[20] 2023. Splatoon. (2023). https://splatoon.nintendo.com.
[21] Alberto Alós, Francisco Morán, Pablo Carballeira, Daniel Berjón, and Narciso García. 2019. Congestion Control for Cloud Gaming Over UDP Based on Round-Trip Video Latency. IEEE Access 7 (2019), 78882–78897. https://doi.org/10.1109/ACCESS.2019.2923294
[22] Maryam Amiri, Hussein Al Osman, Shervin Shirmohammadi, and Maha Abdallah. 2015. An SDN controller for delay and jitter reduction in cloud gaming. In Proceedings of the 23rd ACM international conference on Multimedia. 1043–1046.
[23] Maryam Amiri, Hussein Al Osman, Shervin Shirmohammadi, and Maha Abdallah. 2016. Toward delay-efficient game-aware data centers for cloud gaming. ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM) 12, 5s (2016), 1–19.
[24] Paul Bao and Douglas Gourlay. 2004. Remote walkthrough over mobile networks using 3-D image warping and streaming. IEE Proceedings-Vision, Image and Signal Processing 151, 4 (2004), 329–336.
[25] Neal Cardwell, Yuchung Cheng, C Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. 2017. BBR: congestion-based congestion control. *Commun. ACM* 60, 2 (2017), 58–66.
[26] Gaetano Carlucci, Luca De Cicco, Stefan Holmer, and Saverio Mascolo. 2016. Analysis and design of the google congestion control for web real-time communication (WebRTC). In Proceedings of the 7th International Conference on Multimedia Systems. 1–12.
[27] Marc Carrascosa and Boris Bellalta. 2020. Cloud-gaming:Analysis of Google Stadia traffic. (2020). arXiv:cs.NI/2009.09786
[28] Chun-Fa Chang and Shyh-Haur Ger. 2002. Enhancing 3D graphics on mobile devices by image-based rendering. In Pacific-Rim Conference on Multimedia. Springer, 1105–1111.
[29] Mark Claypool and Kajal Claypool. 2006. Latency and player actions in online games. *Commun. ACM* 49, 11 (2006), 40–45.
[30] Mark Claypool and Kajal Claypool. 2010. Latency can kill: precision and deadline in online games. In Proceedings of the first annual ACM SIGMM conference on Multimedia systems. 215–222.
[31] Tongyu Dai, Xinggong Zhang, and Zongming Guo. 2018. Learning-based congestion control for internet video communication over wireless networks. In 2018 IEEE International Symposium on Circuits and Systems (ISCAS). IEEE, 1–5.
[32] Andrea Di Domenico, Gianluca Perna, Martino Trevisan, Luca Vassio, and Danilo Giordano. 2021. A network analysis on cloud gaming: Stadia, GeForce Now and PSNow. (2021). arXiv:cs.NI/2012.06774
[33] Chao Dong, Chen Change Loy, Kaiming He, and Xiaoou Tang. 2015. Image super-resolution using deep convolutional networks. IEEE transactions on pattern analysis and machine intelligence 38, 2 (2015), 295–307.
[34] Christoph Fehn. 2004. Depth-image-based rendering (DIBR), compression, and transmission for a new approach on 3D-TV. In Stereoscopic Displays and Virtual Reality Systems XI, Vol. 5291. International Society for Optics and Photonics, 93–104.
[35] Chun-Ying Huang, Kuan-Ta Chen, De-Yu Chen, Hwai-Jung Hsu, and Cheng-Hsin Hsu. 2014. GamingAnywhere: The first open source cloud gaming system. ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM) 10, 1s (2014), 1–25.
[36] Po-Han Huang, Kevin Matzen, Johannes Kopf, Narendra Ahuja, and Jia-Bin Huang. 2018. Deepmvs: Learning multi-view stereopsis. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2821–2830.
[37] Vincent Jantet, Christine Guillemot, and Luce Morin. 2011. Object-based layered depth images for improved virtual view synthesis in rate-constrained context. In 2011 18th IEEE International Conference on Image Processing. IEEE, 125–128.
[38] Jian Jin, Anhong Wang, Yao Zhao, Chunyu Lin, and Bing Zeng. 2016. Region-aware 3-D warping for DIBR. IEEE Transactions on Multimedia 18, 6 (2016), 953–966.
[39] Philipp Krähenbühl. 2018. Free supervision from video games. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2955–2964.
[40] Zeqi Lai, Y. Charlie Hu, Yong Cui, Linhui Sun, Ningwei Dai, and Hung-Sheng Lee. 2020. Furion: Engineering High-Quality Immersive Virtual Reality on Today's Mobile Devices. IEEE Transactions on Mobile Computing 19, 7 (2020), 1586–1602. https://doi.org/10.1109/TMC.2019.2913364
[41] Ji-Tang Lee, De-Nian Yang, and Wanjiun Liao. 2016. Efficient caching for multi-view 3D videos. In 2016 IEEE Global Communications Conference (GLOBECOM). IEEE, 1–7.
[42] Kyungmin Lee, David Chu, Eduardo Cuervo, Johannes Kopf, Yury Degtyarev, Sergey Grizan, Alec Wolman, and Jason Flinn. 2015. Outatime: Using Speculation to Enable Low-Latency Continuous Interaction for Mobile Cloud Gaming. In Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys '15). Association for Computing Machinery, New York, NY, USA, 151–165. https://doi.org/10.1145/2742647.2742656
[43] Shuai Li, Kaixin Wang, Yanbo Gao, Xun Cai, and Mao Ye. 2022. Geometric warping error aware CNN for DIBR oriented view synthesis. In Proceedings of the 30th ACM International Conference on Multimedia. 1512–1521.
[44] Yuhua Lin and Haiying Shen. 2017. CloudFog: Leveraging Fog to Extend Cloud Gaming for Thin-Client MMOG with High Quality of Service. IEEE Transactions on Parallel and Distributed Systems 28, 2 (2017), 431–445. https://doi.org/10.1109/TPDS.2016.2563428
[45] Liying Lu, Wenbo Li, Xin Tao, Jiangbo Lu, and Jiaya Jia. 2021. Masa-sr: Matching acceleration and spatial adaptation for reference-based image super-resolution. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 6368–6377.
[46] William R Mark. 1999. Post-rendering 3D image warping: visibility, reconstruction, and performance for depth-image warping. Ph.D. Dissertation. Citeseer.
[47] Leonard McMillan. 1997. An image-based approach to three-dimensional computer graphics. Ph.D. Dissertation. Citeseer.
[48] Voicu Popescu, John Eyles, Anselmo Lastra, Joshua Steinhurst, Nick England, and Lars Nyland. 2000. The WarpEngine: An architecture for the post-polygonal age. In Proceedings of the 27th annual conference on Computer graphics and interactive techniques. 433–442.
[49] Xiaojuan Qi, Zhengzhe Liu, Qifeng Chen, and Jiaya Jia. 2019. 3D motion decomposition for RGBD future dynamic scene synthesis. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 7673–7682.
[50] Nasim Rahaman, Aristide Baratin, Devansh Arpit, Felix Draxler, Min Lin, Fred Hamprecht, Yoshua Bengio, and Aaron Courville. 2019. On the spectral bias of neural networks. In International Conference on Machine Learning. PMLR, 5301–5310.
[51] Konstantinos Rematas, Ira Kemelmacher-Shlizerman, Brian Curless, and Steve Seitz. 2018. Soccer on your tabletop. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 4738–4747.
[52] Stephan R Richter, Vibhav Vineet, Stefan Roth, and Vladlen Koltun. 2016. Playing for data: Ground truth from computer games. In European conference on computer vision. Springer, 102–118.
[53] Saeed Shafiee Sabet, Steven Schmidt, Saman Zadtootaghaj, Babak Naderi, Carsten Griwodz, and Sebastian Möller. 2020. A latency compensation technique based on game characteristics to mitigate the influence of delay on cloud gaming quality of experience. In Proceedings of the 11th ACM Multimedia Systems Conference. 15–25.
[54] Michael Schaffner, Pierre Greisen, Simon Heinzle, Frank K Gürkaynak, Hubert Kaeslin, and Aljoscha Smolic. 2013. MADmax: A 1080p stereo-to-multiview rendering ASIC in 65 nm CMOS based on image domain warping. In 2013 Proceedings of the ESSCIRC (ESSCIRC). IEEE, 61–64.
[55] Shu Shi, Cheng-Hsin Hsu, Klara Nahrstedt, and Roy Campbell. 2011. Using graphics rendering contexts to enhance the real-time video coding for mobile cloud gaming. In Proceedings of the 19th ACM international conference on Multimedia. 103–112.
[56] Shu Shi, Won J Jeon, Klara Nahrstedt, and Roy H Campbell. 2009. Real-time remote rendering of 3D video for mobile devices. In Proceedings of the 17th ACM international conference on Multimedia. 391–400.
[57] Shu Shi, Mahsa Kamali, Klara Nahrstedt, John C Hart, and Roy H Campbell. 2010. A high-quality low-delay remote rendering system for 3D video. In Proceedings of the 18th ACM international conference on Multimedia. 601–610.
[58] Shu Shi, Klara Nahrstedt, and Roy Campbell. 2012. A real-time remote rendering system for interactive mobile graphics. ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM) 8, 3s (2012), 1–20.
[59] Jiahao Su, Wonmin Byeon, Jean Kossaifi, Furong Huang, Jan Kautz, and Anima Anandkumar. 2020. Convolutional tensor-train lstm for spatio-temporal learning. Advances in Neural Information Processing Systems 33 (2020), 13714–13726.

[60] Mirko Suznjevic, Ivan Slivar, and Lea Skorin-Kapov. 2016. Analysis and QoE evaluation of cloud gaming service adaptation under different network conditions: The case of NVIDIA GeForce NOW. In 2016 Eighth International Conference on Quality of Multimedia Experience (QoMEX). 1–6. https://doi.org/10.1109/QoMEX.2016.7498968
[61] Dong Tian, Po-Lin Lai, Patrick Lopez, and Cristina Gomila. 2009. View synthesis techniques for 3D video. In Applications of Digital Image Processing XXXII, Vol. 7443. SPIE, 233–243.
[62] Shishun Tian, Lu Zhang, Luce Morin, and Olivier Déforges. 2019. A Benchmark of DIBR Synthesized View Quality Assessment Metrics on a New Database for Immersive Media Applications. IEEE Transactions on Multimedia 21, 5 (2019), 1235–1247. https://doi.org/10.1109/TMM.2018.2875307
[63] Yunbo Wang, Zhifeng Gao, Mingsheng Long, Jianmin Wang, and S Yu Philip. 2018. Predrnn++: Towards a resolution of the deep-in-time dilemma in spatiotemporal predictive learning. In International Conference on Machine Learning. PMLR, 5123–5132.
[64] Yunbo Wang, Haixu Wu, Jianjin Zhang, Zhifeng Gao, Jianmin Wang, Philip Yu, and Mingsheng Long. 2022. Predrnn: A recurrent neural network for spatiotemporal predictive learning. IEEE Transactions on Pattern Analysis and Machine Intelligence (2022).
[65] Yunbo Wang, Jianjin Zhang, Hongyu Zhu, Mingsheng Long, Jianmin Wang, and Philip S Yu. 2019. Memory in memory: A predictive neural network for learning higher-order non-stationarity from spatiotemporal dynamics. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. 9154–9162.
[66] Lingfeng Xu, Xun Guo, Yan Lu, Shipeng Li, Oscar C. Au, and Lu Fang. 2014. A low latency cloud gaming system using edge preserved image homography. In 2014 IEEE International Conference on Multimedia and Expo (ICME). 1–6. https://doi.org/10.1109/ICME.2014.6890279
[67] Roy D Yates, Mehrnaz Tavan, Yi Hu, and Dipankar Raychaudhuri. 2017. Timely cloud gaming. In IEEE INFOCOM 2017-IEEE Conference on Computer Communications. IEEE, 1–9.
[68] Mengsi Yeh, Chih-Hang Wang, Jitang Lee, De-Nian Yang, and Wanjiun Liao. 2020. Mobile proxy caching for multi-view 3D videos with adaptive view selection. IEEE Transactions on Mobile Computing (2020).
[69] Wucherl Yoo, Shu Shi, Won J Jeon, Klara Nahrstedt, and Roy H Campbell. 2010. Real-time parallel remote rendering for mobile devices using graphics processing units. In 2010 IEEE International Conference on Multimedia and Expo. IEEE, 902–907.
[70] Yizhong Zhang, Jiaolong Yang, Zhen Liu, Ruicheng Wang, Guojun Chen, Xin Tong, and Baining Guo. 2022. Virtualcube: An immersive 3d video communication system. IEEE Transactions on Visualization and Computer Graphics 28, 5 (2022), 2146–2156.
[71] Zhengming Zhang, Yaoqing Yang, Meng Hua, Chunguo Li, Yongming Huang, and Luxi Yang. 2019. Proactive caching for vehicular multi-view 3D video streaming via deep reinforcement learning. IEEE Transactions on Wireless Communications 18, 5 (2019), 2693–2706.
[72] Nikolaos Zioulis, Antonis Karakottas, Dimitrios Zarpalas, Federico Alvarez, and Petros Daras. 2019. Spherical view synthesis for self-supervised 360 depth estimation. In 2019 International Conference on 3D Vision (3DV). IEEE, 690–699.

A APPENDIX

Appendices are supporting material that has not been peer-reviewed.

A.1 How does ZGaming obtain data through Hooking?

As shown in Figure 19, most 3D video games render images by directly or indirectly calling 3D Graphics APIs such as OpenGL [11], DirectX [6], etc. Specifically, some games directly call the libraries of these APIs to communicate with the hardware to generate images, while other games use game engines to render images. Although there are a variety of game engines available (Unity [16], UE [15], etc.), they all still rely on the same set of 3D graphics APIs.

ZGaming obtains the required data by hooking these Graphics APIs. Specifically, apart from RGB, ZGaming needs the depth map, the mask map, and the viewpoint information. Fortunately, these data can be directly obtained or inferred from the parameters passed in when games call these APIs. For example, the rendering pipeline of Graphics APIs maintains a Z-buffer to handle occlusion relationships of 3D objects, which stores pixel-wise depth values. Therefore, ZGaming can obtain the depth map from the Z-buffer.

To hook these parameters, we can wrap the DLLs of these APIs by injecting our own codes while maintaining the origin functions. Before launching the games, we need to replace the original DLLs with modified fake ones, so the injected codes hook the parameters when the games run. To make ZGaming work for more games, we need to wrap more mainstream Graphics APIs. In fact, there are some open-source tools that have wrapped several mainstream APIs, such as Renderdoc [13] and GameHook [2]. In the vision community, some works adopt these tools to hook the data from commercial games to build their synthesis datasets [36, 39, 51, 52].

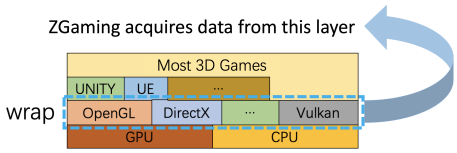


Figure 19: ZGaming acquires data by Graphics API hooking.

A.2 Real-time hole filling by Q3B cache

To predict an image, the client first uses the most recently received frame to predict an image under the current viewpoint through DIBR. The predicted image often exhibits many holes, and the Q3B cache is utilized to provide materials for hole filling. Specifically, we propose a progressive hole-filling process. First, we find an adjacent pixel of the uncovered area and get the corresponding 3D location (which can be computed by the current viewpoint and the depth information) as the target, search this location in the Q3B cache to find the closest block. Usually, this block can cover at least part of the uncovered area. For the remaining uncovered area, we pick another adjacent pixel and search for a new block, continue the process until there is no uncovered area or the uncovered area cannot be filled by history blocks further. For the latter case, we apply the nearest-neighbor interpolation to fill the remaining holes.

Due to the luminance differences between the history block and the current frame, color discrepancies may exist between the filled hole and its edges. Therefore, we design an adaptive luminance algorithm. Firstly, when a block is mapped onto the image, there is often a portion of the block that overlaps with the hole and another portion that overlaps with the valid pixels. We calculate the average luminance difference within the overlapping region between the block and the pixels. This average luminance difference is then added to the overlapping region between the block and the hole. This ensures that the filled hole's luminance is closer to the luminance of the surrounding pixels.

A.3 Details of the Q3B caching strategy

Among all the variables in Equation 1, T_{now} and t_b can be got by clock reading. For $PSNR_{b,i}$, when the client uses block b to do image prediction, there is no ground truth of the current frame on the client side, thus cannot compute the PSNR value immediately. However, since the cloud server will eventually stream each frame to the client, the ground truth of the current frame is usually received after a short period of time (usually after several frames). At that time, the client can compare the predicted frame and the ground truth frame, then compute the PSNR value. Similarly, the client can also predict the current frame without any history blocks to compute $PSNR'_{b,i}$.