A Hybrid Game Contents Streaming Method: Improving Graphic Quality Delivered on Cloud Gaming

Kar-Long Chan, Kohei Ichikawa, Yasuhiro Watashiba, Uthayopas Putchong, Hajimu Iida

A Hybrid Game Contents Streaming Method: Improving Graphic Quality Delivered on Cloud Gaming

Kar-Long Chan, Kohei Ichikawa, Yasuhiro Watashiba, Uthayopas Putchong, and Hajimu Iida

Nara Institute of Science and Technology, Ikoma, Japan
Kasetsart University, Bangkok Thailand
{kar_long.chan.jr2,ichikawa,watashiba}@is.naist.jp
pu@ku.ac.th, iida@itc.naist.jp
http://sdlab.naist.jp/

Abstract. The emerging Cloud Gaming Service provides highly accessible video gaming experience. However, in term of the gaming quality, Cloud Gaming is not competitive to rival with traditional gaming because of network constraints. Especially, 3D game contents streamed as encoded video sequence is suitable in a network environment, but the resulted lower graphic quality may not meet client's demand. Therefore, we propose a Hybrid-Streaming System that aims at improving graphic quality delivered on Cloud Gaming. By utilizing available rendering power from both Cloud Server and client's PC, the system distributes rendering operations to both sides to achieve the desired improvement. Quantitative results show the improvement of graphic quality from the proposed method, as well as reducing server's workload and attaining acceptable network bandwidth consumption.

Keywords: Cloud Gaming, Hybrid-Streaming, Graphic Quality

1 Introduction

The emerging Cloud Gaming Service envisions an intriguing future of providing million clients with novel and highly accessible gaming experiences. By leveraging reliable, elastic and high-performance computing resources, Cloud Gaming shifts the intensive workloads of game processing from client's device to powerful Cloud Server. In its simplest form, which is shown in Figure 1, the actual interactive gaming application is stored at the Cloud Server and gets executed once requested. The rendered game scenes are then streamed back to the client's device as encoded video sequence over a network. At the client's side, the control events input from devices such as mouse, keyboard, joystick are recorded and sent back to the Cloud Server for the manipulation of game logics.

The on demand feature of Cloud Gaming benefits both clients and game developers by easing the possible incompatibility issues between gaming software and hardware environment. Therefore, it has been not only an active topic

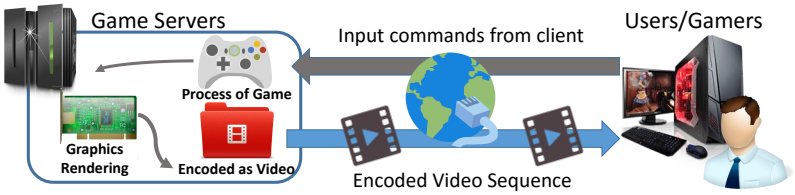


Fig. 1. Brief Structure of Cloud Gaming

both in industries[6] and research fields[7] in recent, but also a new area which possesses tremendous market value[5].

However, a challenging objective of developing a sustainable Cloud Gaming service is to maintain and improve client's Quality of Experience (QoE), as network constraints play critical roles in affecting the system performance[3, 4]. In general, interaction delay and graphic quality are two significant criteria that determine client's QoE. In Cloud Gaming, rigid real-time responsiveness is demanded in order to achieve good enough QoE[2, 7], so currently most related researches focus on alleviating interaction delay[9, 1]. On the other hand, findings from a conducted subject test show that clients are sensitive to changes in graphic quality and smoothness during gameplay, which implies that graphic quality notably affects QoE of Cloud Gaming as well[8]. Furthermore, alongside the advancement of in-game visual effects and high resolution display, client's demands for gaming with more realistic graphics on their devices are uprising. However, in term of graphic quality, there is an obvious gap between traditional local rendering and Cloud Gaming's streaming encoded video, which the graphic quality is degraded from the original.

With this consideration, this study aims to enhance graphic quality delivered on existing Cloud Gaming system. Especially we address the use case of PC, which is not necessary to be a significant powerhouse, but rendering-capable to a certain extent. Based on the two existing streaming methods which will be introduced in later section, our approach is to allocate available rendering power at client's side to achieve the improvement of graphic quality. Furthermore, the distribution of rendering tasks also mitigates server's workload as well.

2 Existing Techniques

In general, two major streaming methods are used in Cloud Gaming: the Image-based Streaming and the Instruction-based Streaming. These two methods differ from each other in how the game contents are delivered from server to client.

2.1 Image-based Streaming

In the Cloud Gaming environment adopting the Image-based Streaming, the game logics, which drive the progression of gameplay, are manipulated according to the client's inputs received by the server CPU. Afterwards, graphics are

rendered through a dedicated Graphic Processing Unit. The rendered game contents are then encoded into video and streamed back to the client's device. Up on receiving the encoded video, the client's device decodes the contents and finally shows the corresponding frame on the display.

One advantage of Image-based Streaming is that the encoded video, in term of bandwidth consumption, is suitable to be streamed in a general network environment. Normally game contents are streamed as 720p video, while the quality can be raised to 1080p in a faster network environment. Another advantage is that since decoding can be processed by using low-cost decoder chips which are massively embedded in client's device, this approach is ideal for running on resource-constrained devices. Therefore, given the wide availability of this method, most commercial Cloud Gaming service providers apply Image-based Streaming to deliver game contents.

On the other hand, GamingAnyWhere is the first and the only available open source Cloud Gaming platform[7], which is based on the structure of Image-based Streaming. It is designed to be highly extensible and customizable that allows user to extend the capabilities of the platform with ease. In this research, Cloud Gaming, in general, refers to the system utilizing Image-based Streaming.

2.2 Instruction-based Streaming

As for the Instruction-based Streaming, after game logics are processed at the server, every API call to perform the corresponding rendering task is intercepted. The intercepted API functions, or referred as graphics command, are compressed and sent to the client's device. Together with the graphics command, related 3D data such as geometry mesh and texture are streamed to the client's device as well. Soon after the arrival of the data, game rendering according to the received graphic commands is processed on-site at the client's device.

The biggest advantage of Instruction-based Streaming over Image-based Streaming is the preserved original graphic quality, as the actual rendering is operated at client's device. Furthermore, without the heavy burden of 3D graphics processing, Cloud Server is more effective at simultaneously handling more clients' requests. However, it requires client's device to be not only compatible with the delivered 3D graphics command, but also powerful enough to process high quality rendering, thus indicating less availability than the Image-based Streaming.

3 Hybrid-Streaming System

The goal of this research is to enhance graphic quality based on the structure of Image-based Streaming. As such, our proposed solution is to integrate the mechanism of Instruction-based Streaming to achieve the desired improvement.

This Hybrid-Streaming System, which stands for adopting both the streaming methods, distributes partial game contents to be streamed as graphic commands and rendered locally at client's PC. Simultaneously, it maintains partial rendering tasks at Cloud Server and the corresponding contents are streamed as

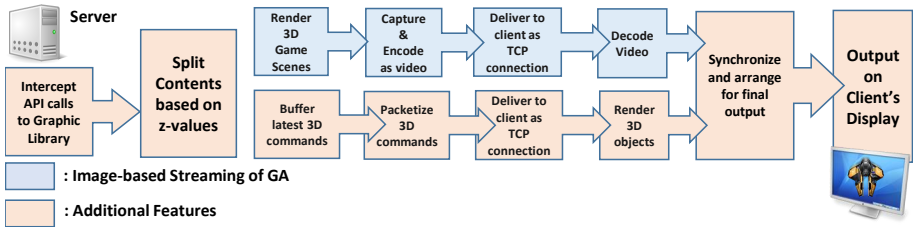


Fig. 2. Data Flow of Proposed Hybrid-Streaming

a video sequence. The improvement of graphic quality is mainly contributed by the portion of game contents rendered at client's PC. Furthermore, by offloading some rendering tasks to client's device, server's workload is mitigated as well.

3.1 System Structure Overview

Figure 2 presents the overall structure of the Hybrid-Streaming System. The blue boxes indicate the original data flow of Image-based Streaming which our Hybrid-Streaming System is based on, while the orange boxes refer to the additional features for achieving the whole system.

Within the system, how the final representation of game contents is correspondingly composed from the products of two different streams is an important objective. It largely depends on the way of splitting game contents at the server. By comparing the depth value, all the objects are separated into two groups, the upper layer which contains shallower objects (closer to the client's view), and the lower layer which contains deeper objects (further away from the client's view). Considering that the contents delivered through Image-based Streaming result in video frame without any depth factor, the game contents represented in this form should be treated as the background. For this purpose, objects belonging to the lower layer are streamed as a video sequence, which undergoes the original process of Image-based Streaming. On the other hands, objects belonging to the upper layer are streamed as graphics command. Therefore, as soon as the rendering operation is completed at the client's PC, the products can be overlaid on top of the background filled by the contents from Image-based Streaming. The graphical representation of the final product is indicated by Figure 3.

In the following subsections, the data flow of Image-based Streaming is explained by exploring GamingAnyWhere (GA), which is the open source Cloud Gaming platform that our work is based on. Furthermore, the additional features for achieving the Hybrid-Streaming System will be introduced as well.

3.2 Structure of GamingAnyWhere's Image-based Streaming

In GA, the data flow refers to streaming audio and video frames from the server to the client. Here we focus on the graphics data manipulation throughout the data flow, which is indicated as the blue boxes in Figure 2. After the game source

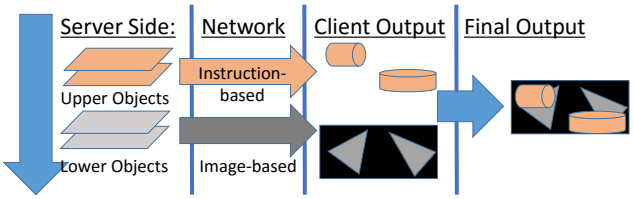


Fig. 3. Graphical representation of constructing the final output from two streaming

is processed and rendered at the server, a designated video capturer implemented in GA is triggered to capture the contents in a polling manner. The captured data is then buffered and encoded by a customizable encoder module. Afterwards, the encoded video frame is delivered to the client through the network. At GA's client module, only the packets representing the most currently encoded video frame is buffered at the decoder. Once the video is decoded, the buffer is cleared for the next frame. On the other hand, audio streams and control inputs from client's device are the other two main streams that affect the overall gaming experience, but they are not the focus of this research. Compared with graphics data, these two streams consume significantly less network bandwidth in the GA structure.

3.3 Additional Features

For achieving the Hybrid-Streaming System as shown in Figure 2, features including splitting the game contents, Instruction-based Streaming, synchronizing the data from two streams and arranging the final output are mandatory.

Splitting the Contents In order to distinguish if an object belongs to the upper layer or the lower layer during the game processing, it is necessary to dynamically intercept API calls to graphic library and looks for the most updated z-value of each object's center point. A threshold of z-value is defined for classifying which layer the object belongs to, and it can be defined by the developer to handle different in-game scenarios.

Contents' Processing Prior to Streaming After the lower layer object is rendered at the server, it undergoes the normal process of Image-based Streaming of GamingAnyWhere, as mentioned in last subsection.

As for the Instruction-based Streaming, the intercepted OpenGL instructions that correspond to render the upper-layer objects are overridden for not sending to the server's GPU, but instead packetized for streaming to the client's PC. Considering that usually the same sequence of graphic commands are routinely called for performing a particular task, we encode the sequence as a fixed code which indicates the corresponding task. As such, rather than delivering every native OpenGL instructions individually to client, streaming a light-weight code with other necessary parameters can maintain smaller overhead.

Furthermore, regarding the other graphic data such as objects vertices and textures that potentially incur heavy network load, a copy of such data is streamed in advance and saved at the client's PC prior to actual gaming. Therefore, during the actual gameplay, only the graphics command set, which include the code and parameters, is streamed to the client's device and hence, reducing the overhead as well. The streaming through the network is based on a normal TCP connection, which is separated from the one of Image-based Streaming.

Data Synchronization Synchronizing the data of two streaming is another critical design objective. In our Hybrid-Streaming System, the latest graphics command which is for rendering the upper layer objects is buffered and updated in every game processing cycle. Whenever GA's source capturing is triggered, the most currently buffered graphics command together with the related parameters is packetized and streamed to the client. After the graphics command is sent to the client, the buffer is cleared and continues for the next iteration of API interception. Since the contents of the two streams undergo separated TCP connection, a precise global time stamp is appended to the respective headers during the packetization. The global timestamp is for the client module to recognize the same-frame contents from the two streams.

Final Arrangement at Client's Device After the video is decoded from Image-based Streaming, an additional step is to assign a suitable depth value to the video frame for maintaining the corresponding contents to be always at the back of the locally rendered objects.

For the on-site rendering, a parallel running thread is responsible for receiving inputs from Instruction-based Streaming and decoding the contents afterwards. Based on the decoded code which represents a particular rendering task, the corresponding sequence of graphic commands is called and assigned with the received parameters. As such, local rendering at the client's PC can be performed.

Finally, the embedded global timestamp is checked for the confirmation of the same-frame contents before updating the next game frames. Once the confirmation is made, the game frame, which is composed of the locally rendered objects' overlaying on top of the video frame, is updated on the client's display.

4 Implementation

Based on the Image-based Streaming structure of GA, we implemented a prototype to simulate the output of our system, which allows us to achieve the current main goal of evaluating the graphics quality. At this stage, graphic commands together with related parameters are streamed directly from the source of demo application. At the client side, the corresponding 3D objects are then rendered based on the received instructions together with necessary texture data which is saved at client's device in advance.

4.1 Instruction-based Streaming

The current Instruction-based Streaming sends 3D graphics command directly from our created demo application which is OpenGL-based. At the client module, a separated thread for requesting graphic commands was developed. This thread, which operates in parallel with the original GA's thread for decoding video sequence, periodically sends requests to server for the latest graphic commands. Furthermore, a lock is used to synchronize the contents from the two streams.

As for the construction of 3D graphics command, a sequence of OpenGL instructions which are responsible for a particular function is represented as an ID. Functions here refer to changing the object's location, rotation, scale, environment lighting effects or camera positions. For the corresponding rendering operation to be performed properly at the client side, the data that should be packetized includes Object ID which indicates the object to be rendered, the Function ID which refers to the function to be performed, and the set of related parameters. In addition, multiple functions can be aggregated into a single packet by simply putting the function set, which includes the parameters, one after another. The current packet layout is designed only for the minimum required data in this preliminary implementation, as it can be adjusted for adopting more patterns in favor of more complex gaming application.

4.2 Arrangement at the Client Module

After decoding the received video frame, the contents of lower layer are mapped as a texture on an OpenGL-based rectangular-shaped polygon, which represents the background objects of our final output. By setting a suitable depth value for this polygon object, the background contents are always further away than any locally rendered 3D objects.

As for the contents of upper layer, the client module decodes the corresponding packetized graphic command sets. Afterwards, OpenGL functions are called accordingly to render the objects into a buffer, which also contains the polygon object representing the background contents.

Finally, once the buffered products are requested for updating the next frame on the display, the locally rendered 3D objects accordingly overlay on top of the background contents.

5 Measurement

As for our current primary goal, we compared the graphic quality of the prototype Hybrid-Streaming System to original Image-based Streaming system of GamingAnyWhere. Besides, we also preliminarily evaluated graphic card usage and network load on both systems. Considering that using existing game software as test case is sophisticated for the prototype stage of the system, we instead created a simple interactive demo application for evaluation.

Fig. 4. Demo Application

5.1 Demo application for current evaluation

Figure 4 shows the OpenGL-based demo application, in which three jet-fighters are presented. The main background, which includes the scene of mountain landscape, is the base element that is always classified as a lower layer object, which represents the background contents at the client side. Therefore, it is always delivered to the client by Image-based Streaming.

In addition, the location of all the jet-fighter objects can be explicitly defined in the demo, so we can also conveniently classify if the object belongs to the lower layer or the upper layer. For example, in Figure 4, the two closer jet-fighters can be classified as upper layer objects, while the further one together with the landscape background is streamed as encoded video. Moreover, it is convenient to add more 3D objects in the scene, which enables us to perform testing under more complicated scenarios.

5.2 System Environment

As for the environment setup, the server machine is equipped with an Intel Core i7 4770K 3.5GHz CPU, a Nvidia GeForce GTX780 graphic card and 16GB system memory, running on a 64-bit CentOS Linux system. On the other hand, client machines include a Laptop A (2.6GHz Intel Core i7 CPU, Nvidia GeForce GT750M with 16GB system memory) and four Laptop B (2.1GHz Intel Core i7, Intel HD Graphics 4000 with 8GB system memory). Each machine is installed with a pre-compiled GA modules. The Image-based Streaming of GA is set to stream the demo application at the resolution of 1280 x 720. The testing was conducted within a campus network environment.

5.3 Procedure of Measuring Graphic Quality

Referring to previous researches related to Cloud Gaming, Peak Signal-to-Noise Ratio (PSNR) and Structural Similarity (SSIM) are measured to evaluate the graphic quality. PSNR refers to the ratio between the signal and the noise introduced by compression, with a typical range between 30 and 50 dB. On the other hand, SSIM is an index for determining the similarity between two images, which the value is ranged from -1 to 1. Both metrics are same in the way that the higher the value is, the better the graphics quality it indicates.

In order to take the measurement of PSNR and SSIM, a short video is captured for the output at the client's PC respectively when the contents are delivered as either Image-based Streaming or Hybrid-Streaming. As for the reference source, we capture a short video of the demo natively running on our Linux server as well. After acquiring a video source for each mode, we extracted every frame from each video source. Referencing to the frame of the original quality, a chosen frame each from Image-based Streaming and Hybrid-Streaming System is used for calculating PSNR and SSIM respectively. We took the measurement of PSNR and SSIM under three scenarios.

5.4 Measuring GPU usage and Network Traffic Load

The Hybrid Streaming System is also expected to reduce server workload as well, since partial rendering tasks are offloaded to the client's PC. Therefore, we measure the respective GPU usages of the server and all the clients' devices by utilizing specified tools including nvidia-smi[1] and Intel-gpu-tools[2]

In addition, developing a Cloud Gaming system that is viable in common broadband network environment is another critical design objective. Since the implemented Instruction-based Streaming in our proposal can introduce additional overhead, it is necessary to measure the traffic load for evaluating the availability of the system in a broadband network. For this reason, IpTraf[3], which is a network monitoring software, is used to investigate the network bandwidth consumption during the streaming of game contents.

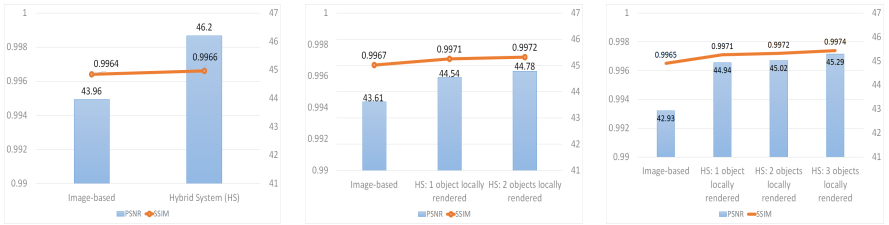
Both measurements were taken under the scenario of presenting 30 jet-fighters in the scene. At the Cloud Server, five instances of the demo application were executed and streamed to five clients' devices respectively. Furthermore, each client's device were distributed equal number of jet-fighters to be rendered locally, where the measured cases include 0 jet-fighter (Image-based Streaming), 6 jet-fighters, 12 jet-fighters, 18 jet-fighters, 24 jet-fighters and 30 jet-fighters.

6 Results

6.1 Results of PSNR and SSIM

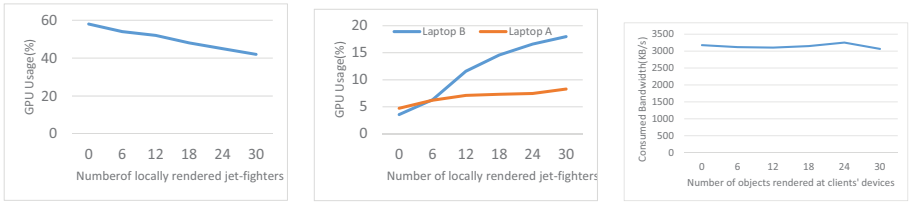
Figure 5(a) shows the PSNR and SSIM of the test case with one jet-fighter displaying in the demo application. The proposed Hybrid-Streaming System, referred as HS in the table, in which the jet-fighter object is locally rendered at the client's PC, achieves 46.20dB compared to 43.96dB from GA's Image-based Streaming. As for the SSIM, the Hybrid-Streaming System also achieves better results than Image-based Streaming, but the difference is not very significant. It can be due to the reason that SSIM is sensitive in different aspect from PSNR.

[1] nvidia-smi: https://developer.nvidia.com/nvidia-system-management-interface/
[2] Intel-gpu-tools: https://01.org/linuxgraphics
[3] IpTraf: http://iptraf.seul.org/



(a) With 1 jet-fighter (b) With 2 jet-fighters (c) With 3 jet-fighters

Fig. 5. Evaluation Results of Graphic Quality



(a) GPU usage at server (b) GPU usages at clients' devices (c) Consumed network bandwidth

Fig. 6. Evaluation Results of GPU usage and Consumed Bandwidth

As for the other two testing scenarios, the proposed Hybrid-Streaming System achieves overall better results than Image-based Streaming in terms of both PSNR and SSIM. Furthermore, while comparing only the scenarios of Hybrid-Streaming System, better graphic quality is indicated while streaming more jet-fighter objects as 3D graphics command. This is reasonable because local rendering at the client's PC preserves the original quality of the 3D object.

6.2 GPU Workload at Server and Client

As shown in Figure 6(a), the measured GPU usages at the server and clients' PCs demonstrate expected behaviors. When more objects are distributed to clients' PC, GPU usage at the server, which is mainly leveraged by object's renderings and video encoding, decreases gradually. Simultaneously, as more objects are assigned to be rendered locally, GPU usage of each client's device, which mainly includes video decoding and objects' renderings, increases. Furthermore, Figure 6(b) shows the respective GPU usages of Laptop A and Laptop B (Average value of 4 laptops). Since Laptop A is equipped with a more powerful GPU, the utilization is lower than the less capable built-in GPU of Laptop B.

6.3 Network Bandwidth Consumption

As for the results of measured network bandwidth, data presented in Table 6(c) do not indicate an increasing trend. As such, the overhead incurred by the

additional Instruction-based Streaming mechanism is relatively trivial. It is considered that the bandwidth is mostly consumed by delivering the contents as encoded video. We expect that more sophisticated graphics command in formal gaming application will likely lead to larger overhead. Even so, as Hybrid-Streaming System saves related graphical data such as textures and geometry meshes at clients' devices prior to actual gaming, low-bit overhead should be maintained by streaming only the light-weight graphics command set.

7 Discussion

7.1 Determination of the z-value threshold

The depth threshold for determining if an object belongs to the upper layer or the lower layer is currently specified by game developers. However, in the case when an object may cross the depth boundary, the same object is required to be rendered both at the Cloud Server and the client's PC in order to align correctly. Hence, it may lead to redundant rendering. As a solution, we consider to use a different set of zNear and zFar at the server side to clip out the corresponding contents, thus reducing the redundancy. Furthermore, we plan to develop an approach to automatically decide the threshold according to the monitoring of GPU usages at client's PC and server. As such, the system should preserve as much the graphic fidelity without overloading the client's PC.

7.2 Compatibility with other genres of games

The developed demo proves to be a perfect situation for evaluating the Hybrid-Streaming system. However, further in-depth testings with different genres of gaming applications are necessary. Especially we need to address if the difference in term of the graphic quality is significant. We consider that our system should benefit game genres such as 2D scrolling, fighting and FPS since these games usually have distinguishing background and foreground objects.

7.3 Significance of the Improvement

The current quantitative evaluation based on the demo shows the improvement of graphic quality, but the difference is not significant. Therefore, it is not yet valid to be conclusive about our work. We expect that with more complicated visual effects, our system should demonstrate more obvious advantages. Furthermore, qualitative evaluation is also planned to be conducted so the actual improvement can be evaluated more comprehensively from player's perspective.

8 Conclusion and Future Works

In this paper, we have presented a Hybrid-Streaming System for the purpose of enhancing graphic quality delivered on Cloud Gaming. Based on the more

prevalent structure of Image-based Streaming, we integrated the mechanism of Instruction-based Streaming to construct the proposed system. As such, by distributing rendering tasks and utilizing graphics processing resource from both client's device and server, enhanced graphic quality of Cloud Gaming can be delivered. Quantitive results show that the implemented prototype of Hybrid-Streaming System achieves better graphics quality than GamingAnyWhere's Image-based Streaming, while maintaining insignificant overhead during network streaming. Furthermore, the Hybrid-Streaming System also alleviates server's graphics processing workload by offloading rendering tasks to client's device.

Currently, the Hybrid-Streaming System is still at the prototype stage, as more thorough investigations are needed to evaluate the system as a prospective Cloud Gaming platform. As future works, we plan to qualitatively evaluate the QoE of the system, as well as applying actual gaming softwares to conduct further testing. Moreover, we will continue to improve the applicability of the Hybrid-Streaming System by developing a more comprehensive framework.

References

1. W. Cai, C. Zhou, V. Leung, and M. Chen. A cognitive platform for mobile cloud gaming. In *2013 IEEE 5th International Conference on Cloud Computing Technology and Science (CloudCom)*, volume 1, pages 72–79. IEEE, 2013.
2. K.-T. Chen, Y.-C. Chang, P.-H. Tseng, C.-Y. Huang, and C.-L. Lei. Measuring the latency of cloud gaming systems. In *Proceedings of the 19th ACM international conference on Multimedia*, pages 1269–1272. ACM, 2011.
3. K.-T. Chen, C.-Y. Huang, and C.-H. Hsu. Cloud gaming onward: research opportunities and outlook. In *2014 IEEE International Conference on Multimedia and Expo Workshops (ICMEW)*, pages 1–4. IEEE, 2014.
4. V. Clincy and B. Wilgor. Subjective evaluation of latency and packet loss in a cloud-based game. In *2013 Tenth International Conference on Information Technology: New Generations (ITNG)*, pages 473–476. IEEE, 2013.
5. Cloud Gaming Report 2012. Distribution and monetization strategies to increase revenues from cloud gaming. `http://www.cgconfusa.com/report/documents/Content-5minCloudGamingReportHighlights.pdf`, 2012.
6. T. Geron. Sony to acquire cloud gaming startup gaikai for $380 million. `http://www.forbes.com/sites/tomiogeron/2012/07/02/sony-to-acquire-cloud-gaming-startup-gaikai-for-380-million/`, 2012.
7. C.-Y. Huang, C.-H. Hsu, Y.-C. Chang, and K.-T. Chen. Gaminganywhere: An open cloud gaming system. In *Proceedings of the 4th ACM multimedia systems conference*, pages 36–47. ACM, 2013.
8. M. Jarschel, D. Schlosser, S. Scheuring, and T. Hoßfeld. An evaluation of qoe in cloud gaming based on subjective tests. In *2011 Fifth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS)*, pages 330–335. IEEE, 2011.
9. L. Xu, X. Guo, Y. Lu, S. Li, O. C. Au, and L. Fang. A low latency cloud gaming system using edge preserved image homography. In *2014 IEEE International Conference on Multimedia and Expo (ICME)*, pages 1–6. IEEE, 2014.