This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE Globecom 2010 proceedings.

Rendering Adaptation to Address Communication and Computation Constraints in Cloud Mobile Gaming

Shaoxuan Wang, Sujit Dey
Mobile System Design Lab, Dept. of Electrical and Computer Engineering
University of California, San Diego
{shaoxuan, dey}@ece.ucsd.edu

***Abstract*—A new Cloud Mobile Gaming (CMG) approach, where the responsibility of executing the gaming engines, including the most compute intensive tasks of graphic rendering, is put on cloud servers instead of the mobile devices, has the potential for enabling mobile users to play the same rich Internet games available to PC users. However, the mobile gaming user experience may be limited by the communication constraint imposed by available wireless networks and computation constraint imposed by the cost and availability of cloud servers. In this paper, we propose a rendering adaptation technique which can adapt the game rendering parameters to satisfy CMG communication and computation constraints, such that the overall mobile gaming user experience is maximized. Experiments conducted on a commercial UMTS network demonstrate that the proposed rendering adaptation techniques can make the CMG approach feasible: ensuring protection against wireless network conditions, and ensuring server computation scalability, thereby ensuring acceptable mobile gaming user experience.**

I. INTRODUCTION

With the evolution of smart mobile devices including smart phones, smartbooks, and netbooks, and their capabilities and rapid adoption for Internet access, there is a growing desire to enable rich Internet games on these mobile devices. However, despite the progress in their capabilities, it is challenging to play Internet PC games even on highly capable mobile devices. For example, we experimented playing the game World of Warcraft with high graphic settings on one of the latest netbooks which has processor and memory capabilities of 1.66GHz and 1GB respectively (much higher than smart phones). The netbook could render an average only five frames per second, leading to a bad user experience, as the typical desired rendering frame rate is above 30. The gap between the growing requirements of Internet video games and the capabilities of mobile devices may not come close any time soon, at least for the vast majority of small footprint, battery constrained mobile devices.

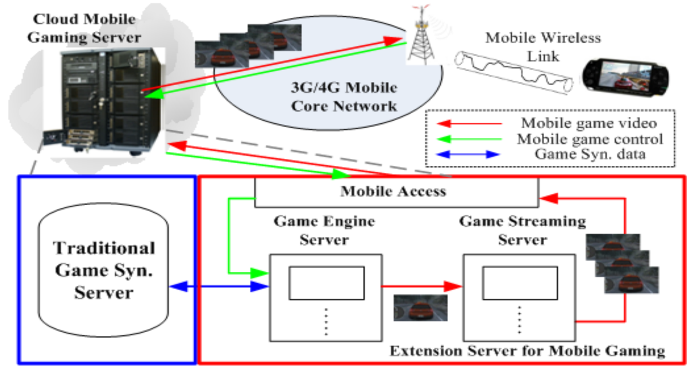

Figure 1. Overview of CMG architecture and data/control flow

Thereby it is promising to investigate an alternative approach Cloud Mobile Gaming (CMG), where cloud servers are responsible for executing the appropriate gaming engines, and streaming the resulting game video to the client devices. Figure 1 presents the overall architecture of CMG. It extends the traditional game synchronization server with two additional components: game engine servers and game streaming servers. The game engine server synchronizes the client's game data with the game synchronization server, and processes the game logic and data to render the raw game video. The generated raw game video is encoded by the game streaming server, and finally sent to the mobile client. On the other hand, the mobile user's inputs are delivered to the cloud mobile gaming server and accepted by the game engine server directly.

Though the CMG approach can address the hardware constraint of mobile devices, we still have to cope with two challenges which are vital for the success of CMG approach: 1) communication constraint in terms of limited and fluctuating mobile network bandwidth; 2) computation constraint reflected by the available server computing resource for each client, considering cloud server hosts numerous clients at the same time. Communication bandwidth constraint may cause unexpected delay and packet loss, leading to an increase in gaming response time besides adverse impact on the video quality. Computation constraint may lead to a slow rendering frame rate, thereby unacceptable gaming experience. Therefore, we need properly address above two challenges to make CMG approach feasible.

In [7], we developed Game Mean Opinion Score (GMOS) as a new metric for Mobile Gaming User Experience (MGUE) in the Cloud Mobile Gaming approach, and developed techniques to quantitatively measure the GMOS in a real-time gaming session. In [8], we have proposed a set of optimization techniques, including game video adaptation, to address the challenges of a wireless network in meeting acceptable response time of a gaming session. In this paper, we address the bandwidth constraint of wireless networks and the computation constraint of CMG servers by developing a scalable, adaptive game rendering approach. Our proposed rendering adaptation technique adapts multiple game rendering parameters to simultaneously satisfy CMG communication and computation constraints, such that the overall Mobile Gaming User Experience is maximized.

In section II, we briefly introduce the 3D rendering pipeline and reveal two opportunities for rendering adaptation for CMG. In section III, we identify several rendering parameters which can effectively scale rendering complexity, and thereby the communication and computation requirements associated with game rendering. We characterize the communication and computation costs of the identified rendering parameters, so they can be effectively used for adaptive rendering. In section IV, we propose our rendering adaptation technique, which includes an off-line step of deriving optimal rendering settings for different adaptation levels corresponding to different communication and computation costs, and a run-time adaptation scheme which can select the optimal adaptation level in response to the constantly changing communication and computation constraints during a CMG session. In section V, we demonstrate the performance and benefits of our rendering adaptation technique in a commercially available cellular UMTS network. Finally, we summarize our findings in section VI.

There have been several approaches [1][2][3] that have been developed to model the computation cost of graphic rendering parameters. However they do not study the impacts of rendering parameters on communication cost, which is one of vital subjects in this paper. There have been efforts [4][5][6] to address the costs associated with rendering on clients in client-server architectures. These techniques attempt to adapt the complexity of the rendering objects, depending on the computational capability of the clients. The lower the complexity of the objects, the less the rendering cost at the client, and also less the communication bandwidth needed to transmit the objects to the client. However, in the CMG approach, where rendering is performed by CMG servers and not clients, the computation cost can be affected by adapting the rendering tasks themselves, as proposed in our approach (Section III), with much more impact than just by adapting the graphics objects. Moreover, the communication cost in the CMG approach is determined by the game video that needs to be streamed from the CMG servers to the clients, and not the size of the rendering objects as in the traditional client-server architectures. Hence, the above approaches [4][5][6] are sub-optimal to address CMG computation cost, and not applicable to address the CMG communication cost.

II. Background and Opportunities for Rendering Adaptation

Before presenting our rendering adaptation technique, in this section, we first introduce the graphic rendering pipeline as a basic background, followed by revealing two opportunities for developing rendering adaptation technique to address communication and computation constraints.

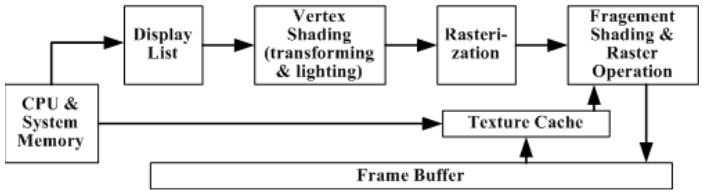
Display List
Vertex Shading (transforming & lighting)
Rasteri-zation
Fragement Shading & Raster Operation
CPU & System Memory
Texture Cache
Frame Buffer

Figure 2. Primary stages of graphic rendering pipeline

Figure 2 shows the primary stages of graphics pipeline in modern GPU. All the graphic data for one frame is first cached in a display list. When a display list is executed, the data is sent from the display list as if it were sent by the application. All the graphic data (vertices, lines, and polygons) are eventually converted to vertices or treated as vertices once the display list starts. Vertex shading (transform and lighting) is then performed on each vertex, followed by rasterization to fragments. Finally the fragments are subjected to a series of fragment shading and raster operations, after which the final pixel values are drawn into the frame buffer. To best present a geometric object, graphic rendering usually applies texture images onto the object, so as to make it look more realistic. The texture images are usually pre-calculated and stored in system memory, and loaded into GPU texture cache when needed.

We next describe the communication and computation costs associated with the CMG approach : 1) **Communication Cost (CommC)** of CMG is the bit rate of compressed game video delivered over the wireless network. However, while the video bit rate is determined by the video compression rate used, it is also affected by the video content complexity. Therefore, it is possible to reduce CommC by reducing the content complexity of game video; 2) **Computation Cost (CompC)** of CMG is mainly consumed by graphic rendering, which can be reflected by the GPU utilization by the game engine. When the game engine is the only application using the GPU resource, CompC is equivalent to the product of rendering Frame Rate (FR) and Frame Time (FT), the latter being the time taken to render a frame. Most of the stages in figure 2 are processed separately by their own special-purpose processors in GPU. The FT is limited by the bottleneck processor. To reduce CompC, we need to alleviate the computing load on the bottleneck processor.

Having defined CommC and CompC, the objective of rendering adaptation in this paper is to properly adjust CommC and CompC of CMG application to satisfy communication bandwidth and computation constraints. Our rendering adaptation mainly relies on two potential opportunities. The first idea is to reduce the number of objects in the display list, as not all objects in the display list created by game engine are necessary for playing the game. For example, in the Massively Multiplayer Online Role-Playing Game (MMORPG), a player mainly manipulates one object, his avatar, in the gaming virtual world. Many other unimportant objects (eg. flowers, small animals, and rocks) or far away avatars will not affect user playing the game. Removing some of these unimportant objects in display list will not only release the load on graphic computation but also reduce the video content complexity, and thereby CommC and CompC. The second opportunity of adaptive rendering is related to the complexity of rendering operations. In the rendering pipeline, many operations are applied to improve the graphic reality. The complexities of these rendering operations directly affect CompC. More importantly, some of the operations also have significant impact on content complexity, and thereby CommC, such as texture detail and environment detail. If we can scale these operations, we will be able to scale CommC and CompC as needed.

III. Identification and Characterization of Adaptive Rendering Parameters

The first step in enabling adaptive 3D rendering in CMG approach is to identify the adaptive rendering parameters. Besides, we have to explore how these parameters affect CommC and CompC. In this section, we start by the discussion and identification of adaptive rendering parameters for most modern 3D games. Then using a popular online MMORPG game PlaneShift [9] as an example, we conduct experiments to characterize how these parameters affect CommC and CompC.

A. Identification of Adaptive Rendering Parameters

As discussed in section II, reducing the number of objects in the display list or reducing the complexity of rendering operations could lead to the decreases in CommC and CompC. Based on this concept, we identify five parameters which we believe are suitable for adaptive rendering in most 3D games.

1) Realistic effect: Realistic effect basically includes four parameters: color depth, anti-aliasing, texture filtering, and lighting mode. The first parameters, color depth refers to the amount of video memory that is required for each screen pixel. Anti-aliasing and texture filtering are employed in the "fragement shading" stage as shown in figure 2. Anti-aliasing is used to reduce stair-step patterns on the edges of polygons, while texture filtering is used to determine the texture color for a texture mapped pixel. The lighting mode will decide the lighting methodology for the "vextex shading" stage in figure 2. Common lighting models for games include lightmap and vertex lighting. Vertex lighting gives a fixed brightness for each corner of a polygon, while the lightmap model adds an extra texture on top of each polygon which gives the appearance of variation of light and dark levels across the polygon. Each of above four parameters only affects one stage of graphic pipeline. Varying any one of them will only have an impact on one special-purpose processor, which may not reduce the load on bottleneck processor. Thus when we reduce/increase the realistic effect, we vary all four parameters to have a reduced/increased CompC.

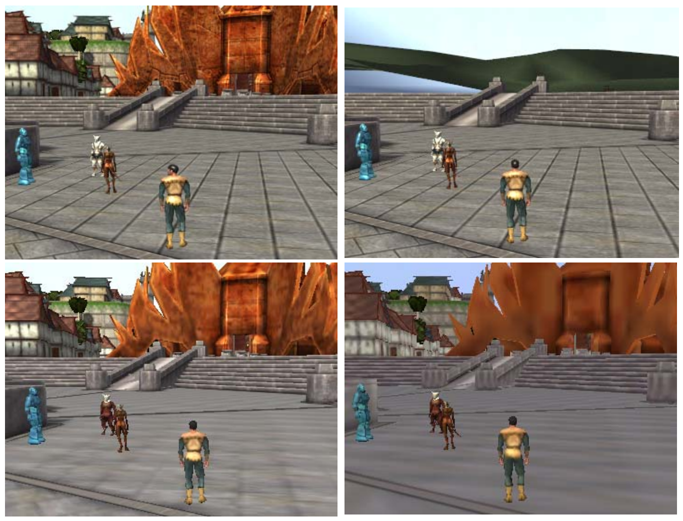

Figure 3. Screenshots of game "PlaneShift" in different settings of view distance and texture detail (LOD): (a) top-left, 300m view and high LOD; (b) top-right, 60m view and high LOD; (c) bottom-left, 300m view and medium LOD; (d) bottom-right, 300m view and low LOD;

2) View distance: This parameter determines which objects in the camera view will be included in the resulting frame, and thereby should be sent to the display list for graphic rendering. Figure 3(a)(b) compare the visual effects in two different view distance settings (300m and 60m) in the game PlaneShift. Though shorter view distance has impairments on user perceived gaming quality, the game will be still playable if we properly control the view distance above a certain acceptable threshold. Since the view distance affects the number of objects to be rendered, it has the impact on CompC as well as the CommC.

3) Texture detail: This is also known as Level Of Detail (LOD). It refers to how large and how many textures are used to present objects. The lower texture detail level, the lower resolution the textures have. As shown in figure 3(a)(c)(d), the surfaces of objects get blurred as we decrease the texture detail. It is also important to be aware of that reducing texture detail has a less impact on important objects (avatars and monsters) than unimportant objects (ground, plants, and buildings), because the important objects in game engines have much more textures than unimportant objects. Thereby we could leverage this information to properly downgrade the texture detail level for less communication cost, while mainatining the good visual quality of important objects.

4) Environment detail: Many objects and effects (grass, flowers, and weather) are applied in modern games, especially the RPG games, to make the vitual world look more realistic. However they will not affect the real experience of users playing the game. Therefore, we could elliminate some of these objects or effects to reduce CommC and CompC if needed.

5) Rendering Frame Rate: The total computation cost of GPU is the product of rendering cost for a frame and frame rate. While the above parameters are mainly focusing on adapting the computation cost for one frame, the rendering frame rate is a vital parameter in adapting the total computation cost. In addition, if we adapt the rendering frame rate together with the encoding frame rate, the resulting video bit rate will also change consequentially. However, changing frame rate can significantly degrade user experience by introducing jerkiness and a perceived increase in reponse time. Hence we will not use this parameter in our rendering adaptation technique.

B. *Characterization of Communication and Computation Costs*

Having introduced the adaptive rendering parameters, in this subsection, we use a popular online 3D game PlaneShift as an example to characterize how the adaptive rendering parameters affect the Communication Cost and Computation Cost.

TABLE I. RENDERING PARAMETERS AND EXPERIMENT SETTINGS

Parameters		Experiment Values		
Realistic Effect		H(High)	M(Medium)	L(Low)
	color depth	32	32	16
	multi-sample (factor)	8	2	0
	texture-filter (factor)	16	4	0
	lighting mode	Vertex light	Lightmap	Disable
Texture Detail (down sample)		0, 2, 4		
View Distance (meter)		300, 100, 60, 40, 20		
Enabling Grass		Y(Yes), N(No)		

PlaneShift is a cross-platform open source MMORPG game. Its 3D graphic engine is developed based on Crystal Space 3D engine. Table I shows the four rendering parameters (corresponding to the four adaptation parameters introduced in the previous sub-section), and their settings we will use. The parameter "Realistic Effect" consists of four factors: color depth, multi-sample (technique that uses additional samples to anti-alias graphic primitives), texture-filter, and lighting mode. Instead of using the numerous values possible for the Realistic Effect factors, we choose to use three levels, with the corresponding values shown in Table I. For the parameter "Texture Detail", we use texture down sample rates {0, 2, 4}. The higher down sample rate, the lower resolution textures have. Similarly, we use five choices for the parameter "View Distance", and two options for the effect of the Environment Detail parameter "Enabling Grass". Note that the lowest setting selected for any of the parameters is the minimal user acceptable threshold, such that the gaming quality using the settings in Table I will always be above the acceptable level.

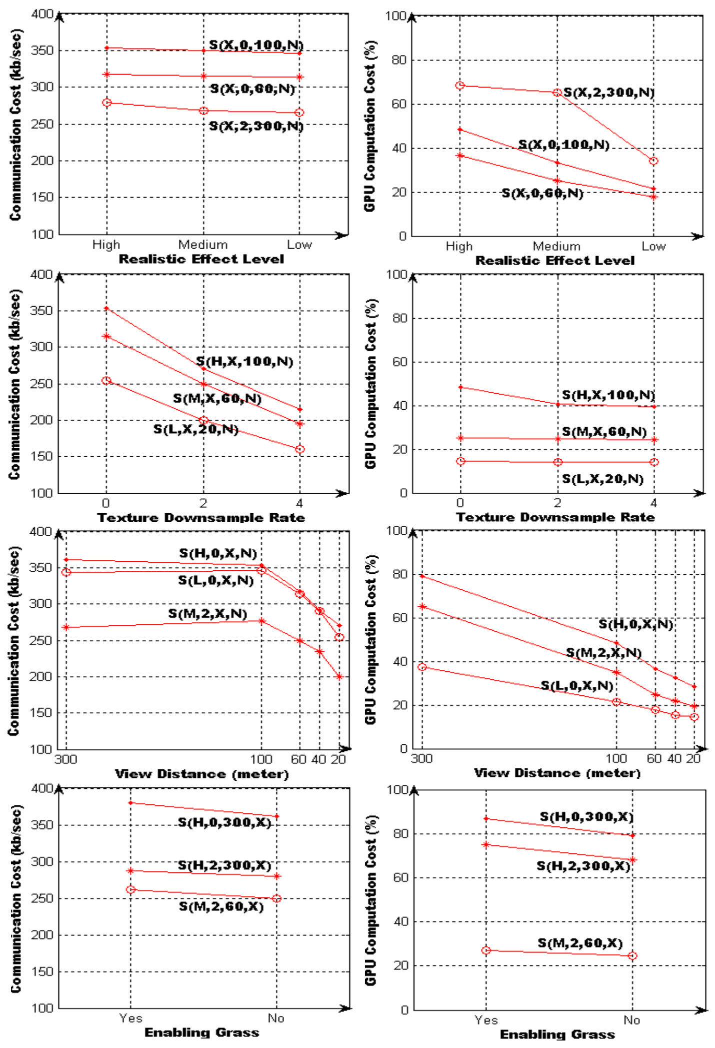

Figure 4. CommC and CompC for different rendering parameters

Since we have four adaptive rendering parameters, we use a 4-tuple S to denote a rendering setting. The elements of S indicate the value of the four adaptive rendering parameters respectively used in a rendering setting. Then we conduct experiments to measure the CommC and CompC for every possible rendering setting S using the settings of parameters in Table I. The experiments are conducted on a desktop server which integrates a NVIDIA Geforce 8300 graphic card. For each setting S, we control the game avatar roaming in the gaming world about 30 seconds along the same route. The Quantization Parameter (QP) of video encoder x264 is 28, while the encoding frame rate is 15fps and GOP size is 60. We measure compressed video bit rate (CommC) and GPU utilization (CompC). Figure 4 shows some representative data points from the experimental results. Each plot represents a rendering setting where one of the rendering parameters is varied (marked by "X" in the associated setting) while keeping the other parameters to fixed values shown in the setting tuple. Note figure 4 does not show results of all possible settings, and the results of CompC will be different for different hardware platforms. From figure 4, we have the following observations:

1) Realistic Effect has a great impact on CompC. But it almost does not affect CommC, because it has little impact on video content complexity.

2) Texture Detail significantly affects CommC, as the highest video bit rate is about 1.6 times of the lowest video bit rate when we vary the texture down sample rates. However, texture detail almost does not affect the CompC. This is because the reduced textures in different levels for an object are pre-calculated and saved in memory, so that the graphic pipeline can load the textures quickly without any additional computation.

3) View Distance will significantly affect both CommC and CompC. While its impact on CompC is almost linear, imapct on CommC becomes clear only below a certain point (100 meters).

4) The impact of *Enabling Grass* on CommC and CompC are limited (up to 9%), mainly due to the simple design of this effect in PlaneShift.

Note that the above characterizing measurement is an off-line pre-processing step, resulting in a table of cost models for each rendering setting, which is subsequently used by the rendering adaptation scheme described in the next section.

IV. Rendering Adaptation Scheme: Optimal Adaptive Rendering Settings and Level-Selection Algorithm

In this section, we present a 3D rendering adaptation scheme for Cloud Mobile Gaming, which includes: 1) an off-line step of identifying the optimal rendering settings for different adaptation levels, where each adaptation level represents a certain communication and computation cost; 2) a run-time level-selection algorithm, that automatically adapts the adaptation levels, and thereby the rendering settings, such that the rendering cost will satisfy the communication and computation constraints imposed by the fluctuating network bandwidth and server available capacity. We start by describing the way we select the optimal adaptive rendering settings. Later in this section, we present the level-selection algorithm.

A. Optimal Adaptive Rendering Settings

In our rendering adaptation technique, we adapt application cost by adjusting the adaptation level. The higher adaptation level, the higher CommC and CompC will be. Each adaptation level has a corresponding rendering setting as mentioned in section III. Since there are too many possible rendering settings and different rendering parameters have different impacts on CommC and CompC, it is important to identify the optimal rendering setting for each adaptation level.

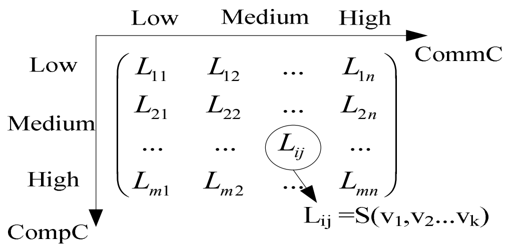
Low
Medium
High
CommC
Low
Medium
High
CompC
L_{11} L_{12} … L_{1n}
L_{21} L_{22} … L_{2n}
… … L_{ij} …
L_{m1} L_{m2} … L_{mn}
$L_{ij} = S(v_1, v_2 \ldots v_k)$

Figure 5. Adaptatation levels and matrix L

As shown in figure 5, the computation cost and communication cost can be divided into m levels and n levels respectively. The higher level denotes the higher resource cost. We use an $m \times n$ matrix L to represent all the adaptation levels. For example, adaptation level L_{ij} has the computation cost at level i and communication cost at level j. In each adaptation level, there is a k-dimensional (k is the number of rendering parameters) rendering setting S. The elements of S indicate the values of the rendering parameters. All the adaptation levels in L should be able to provide acceptable gaming quality to the user. The more resource cost, the better gaming quality will be.

Having defined adaptation levels and adaptation matrix L, next we present how to select the optimal rendering settings in matrix L for different adaptation levels. We first identify the highest setting L_{mn} and lowest setting L_{11}. L_{mn} provides the best gaming quality but costs the most resource, while L_{11} has minimal CommC and CompC but provides the minimal acceptable quality below which user cannot tolerate [7]. From the characterizing experiments described in section II-B, we could know the CommC and CompC of highest setting L_{mn} and lowest setting L_{11}. We evenly divide the CommC and CompC ranges between the desired numbers of levels, considering that the effect of adaptation will be obvious if the cost differences between adaptation levels are significantly distinct. Knowing the CommC and CompC for each level, we then select all other optimal rendering settings, according to the following two requirements: 1) CommC and CompC of the optimal setting for a level should meet the desired CommC and CompC for the level; 2) of all the settings that meet the above requirement, the optimal setting should provide the best gaming quality.

We still use game PlaneShift as a demonstrative example to illustrate how we select the optimal rendering settings for this game. We design a 4x4 adaptation matrix for PlaneShift. The setting of L_{11} for PlaneShift is S(L,4,20,N) using the lowest values in Table I, while the setting of L_{44} is S(H,0,300,Y) using the highest values in Table I. From the experiment results presented in figure 4, we know that CommC and CompC of S(L,4,20,N) are 159.74kb and 14% (see Texture Downsample graph), while they are 380kb and 86.9% for S(H,0,300,Y) (see Enabling Grass graph). Hence the CommC of the best setting (level 4) is 2.38 times of the lowest setting (level 1), while the CompC of the best setting (level 4) is 6.19 times of the lowest setting (level 1). With these maximum ranges, we define CommC of level 2 and level 3 as 1.46 times and 1.92 times of level 1 respectively, while they are 2.73 times and 4.46 times for CompC. Knowing the CommC and CompC for different levels, we could finalize all the optimal rendering settings, resulting in an adaptation matrix as shown in figure 6.

	1X	1.46X	1.92X	2.38X	CommC
1X	S(L,4,20,N)	S(L,4,100,Y)	S(L,0,60,N)	S(L,0,100,Y)	
2.73X	S(H,4,20,Y)	S(L,4,300,Y)	S(H,0,40,Y)	S(M,0,100,Y)	
4.46X	S(M,4,300,Y)	S(M,2,300,N)	S(H,0,60,Y)	S(H,0,100,N)	
6.19X	N/A	S(H,4,300,Y)	S(H,2,300,Y)	S(H,0,300,Y)	
CompC					

Figure 6. Adaptation levels and matrix for game PlaneShift

B. *Level-selection Algorithm*

Having identified the optimal rendering settings for different adaptation levels, we next develop a level-selection algorithm, which can decide when and how to switch the rendering settings during a gaming session, in response to the current network conditions and server utilization, so as to satisfy the network and server computation constraints.

The level-selection algorithm should first be able to realize when the network is constrained or the server is over utilized. We use both network delay and packet loss as indicators to detect constrained network. In the best network conditions (not overloaded), packet loss rate is 0, but there is a certain minimum round-trip delay, denoted by *MinDelay*, due to the time needed to transmit a packet through the core network and the RF link. We empirically use 1.2 times of *MinDelay* as the threshold for round-trip delay together with the packet loss to estimate the constrained network. To detect server over-utilization, we empirically use 90% as the threshold of GPU utilization. If GPU utilization is above this threshold we will identify the GPU is getting over utilized. When either a constrained network or an over utilized server is detected, the level selection algorithm will select a lower adaptation level (with lower cost). When the network condition and/or server utilization improves, the algorithm will select a higher level (with higher cost, and thereby higher user experience). However, to avoid oscillations between adaptation levels, we also have set conditions for increasing the adaptation level. For communication cost level, we increase it if the network has stayed in the good condition (no packet loss, delay less than threshold) for 2 minutes. For computation cost level, we increase it when the server utilization is below 80%.

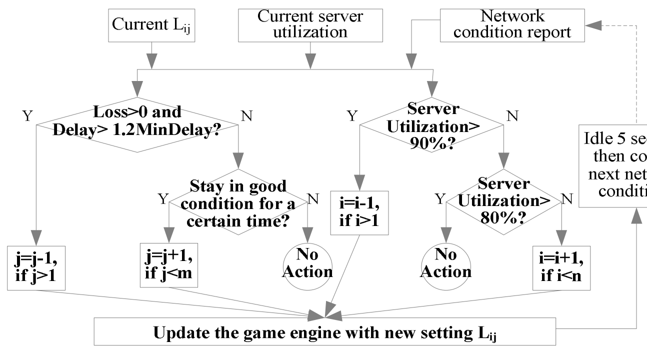
Current L_{ij}
Current server utilization
Network condition report
Loss>0 and Delay> 1.2MinDelay?
Y
N
Stay in good condition for a certain time?
Y
N
Server Utilization> 90%?
Y
N
Server Utilization> 80%?
Y
N
Idle 5 seconds then collect next network conditions
j=j-1, if j>1
j=j+1, if j<m
No Action
i=i-1, if i>1
No Action
i=i+1, if i<n
Update the game engine with new setting L_{ij}

Figure 7. Work flow of level-selection algorithm

Figure 7 shows the proposed level-selection algorithm. Depending on the network conditions (delay and packet loss), and server utilization, the level-selection algorithm decides to select a lower or higher adaptation level, or keep the level same as the current level L_{ij}. Its mechanism is as follows:

1) If there is packet loss and the network round-trip delay is greater than the threshold (1.2**MinDelay*), we lower the CommC level j by 1. On the other hand, if the network has been in good condition (no loss and delay less than the threshold) for sufficient time (we use 2 minutes empirically), we increase j to the next higher level, unless j is already at level n. For other situations, the algorithm keeps the same level.

2) If server utilization is over 90%, we decrease the computation cost level (index i) by 1. Otherwise, if server utilization is not over 80%, we increase i by 1 unless it is already at level m.

For the new adaptation level selected by the rate-selection algorithm above, the optimal rendering settings are selected from the adaptation matrix, and updated into the game engine.

V. EXPERIMENTAL VALIDATION

We have prototyped the rendering adaptation technique on a cloud mobile gaming platform shown in figure 1. The level-selection algorithm (section IV-B) is designed as a software script running on the CMG server, while the adaptation matrix (section IV-A) is pre-calculated based on off-line experiments. Server GPU utilization is calculated from GPU idle time. Network delay and packet loss conditions are measured by a network probing mechanism introduced in [7]. A new API in 3D game engine is implemented to expose the adaptive rendering parameters (Table I), such that they could be updated during a gaming session by the level-selection software. In this section, we report on experiments conducted to test the effectiveness of the proposed rendering adaptation technique to satisfy communication and computation constraints so as to maximize user perceived gaming quality.

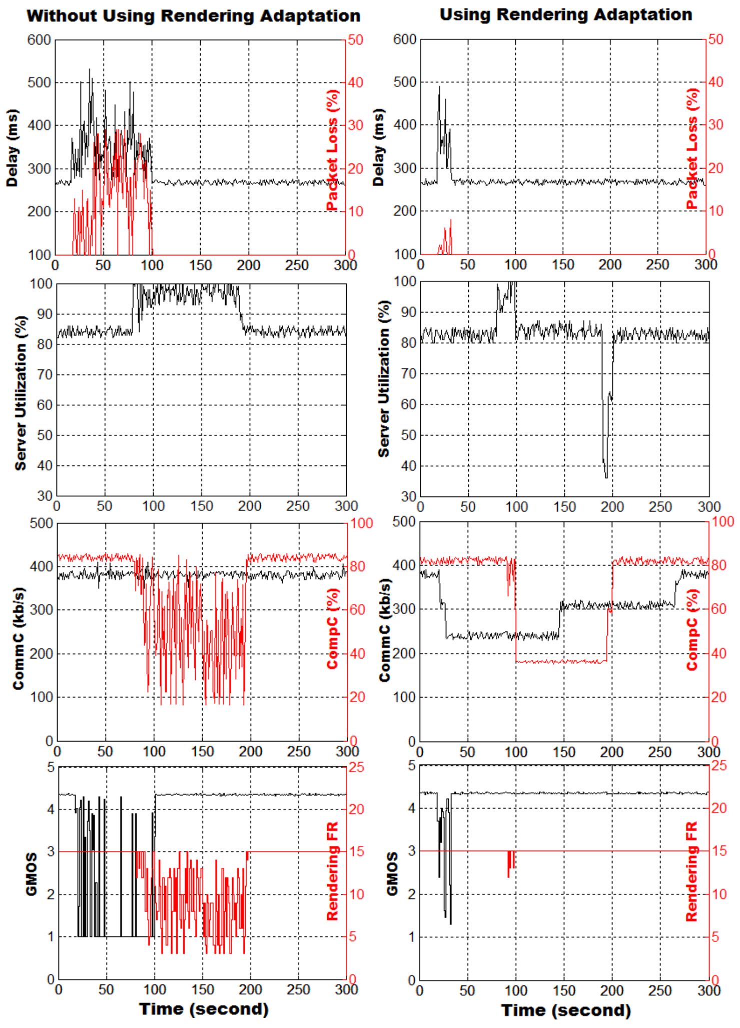

Figure 8. Experiment results for game PlaneShift in UMTS network: (a) left, without using rendering adaptation; (b) right, using rendering adaptation.

The experiments are conducted using a HSDPA enabled UMTS networks. The game server is set up in our lab in UCSD campus, while the game PlaneShift is played on a netbook. We initialize the netbook in outdoor locations with good signal strength during the midnight. Then we generate background traffic to affect the network bandwidth between game server and mobile client, and manually execute 3D rendering tasks to increase server GPU load. Figure 8 shows a representative sample of data collected from numerous gaming sessions of the PlaneShift. We use x264 as our video encoder. The video settings are: Quantization Parameter of 28, GOP size of 60, frame rate of 15, and VGA resolution. The left column (a) in Figure 8 presents results using the original CMG approach, while the right column (b) shows results when additionally using our rendering adaptation technique. In each column from top to bottom, we sequentially present the results of network round-trip delay and packet loss; server GPU utilization; resulting network communication cost (video bit rate) and graphic computation cost (GPU utilization by game) due to adaptive rendering; mobile gaming user experience mainly represented by the Game Mean Opinion Score (GMOS) [7]. Note that the rendering Frame Rate may be below 15 when server is over utilized. The lower rendering FR, the lower user experience will be. However GMOS model in [7] does not take into account such a low frame rate. Therefore, we also present the results of rendering frame rate as another indicator for user experience besides GMOS. We provide below the key observations from our experiments:

1) When network bandwidth is constrained due to the heavy background traffic load (during 20-100 seconds), the gaming session without using rendering adaptation technique experiences unacceptably high and fluctuating network delay and packet loss, leading to unacceptable user experience (GMOS). In contrast, in the gaming session using rendering adaptation technique, user experienced delay and packet loss are significantly reduced and stabilized under the same network conditions, and hence significantly improved GMOS, though there is a small period of unpleasant time where the adaptation algorithm is responding to the constrained conditions.

2) When GPU load of server increases (during 80-200 seconds), the original CMG gaming session will have a bad user experience, reflected by extremely low and unstable rendering frame rate produced. This is successfully addressed by the rendering adaptation technique. As shown in figure 8(b), the rendering frame time has almost not been affected in the whole gaming session where we use the adaptation technique.

VI. Conclusion

In this paper, we presented a rendering adaptation technique to address the constraints of wireless network communication bandwidth and server computation capacity in a Cloud Mobile Gaming (CMG) approach. Several rendering parameters are characterized and demonstrated having significant effects on communication cost (game video bit rate) as well as computation cost (GPU utilization). Optimal adaptive rendering settings for different adaptation levels and a level-selection algorithm are proposed to enable adaptive 3D rendering for cloud mobile games. The experiments conducted on commercial wireless networks demonstrate the proposed rendering adaptation technique can make the CMG approach feasible: ensuring protection against wireless network conditions, and ensuring server/computation scalability, thereby ensuring acceptable mobile gaming user experience.

References

[1] N. Tack, et al. , "3D Graphics Rendering Time Modeling and Control for Mobile Terminals," in *Proc. of Int. Conf. on 3D Web technology* , 2004.

[2] M. Wimmer and P. Wonka, "Rendering Time Estimation for Real-Time Rendering," in *Eurographics Symposium on Rendering, 2003.*

[3] W. Van Raemdonck, et al., "Scalable 3D Graphics Processing in Consumer Terminals", *IEEE International Conference on Multimedia and Expo*, 2002.

[4] B.-O. Schneider, and I. Martin, "An adaptive framework for 3D graphics over networks", *in Computers and Graphics*, 23, 867-874, 1999.

[5] N. Pham Ngoc,et al., "A QoS Framework for Interactive 3D Applications", *in Proc. of the 9th international conference on 3D Web technology*, 2004.

[6] F. Morán, et al., "Adaptive 3D Content for Multi-Platform On-Line Games," *IEEE International Conference on Cyberworlds*, 2007.

[7] S. Wang, S. Dey, "Modeling and Characterizing User Experience in a Cloud Server Based Mobile Gaming Approach," *in Proc. of IEEE Global Communications Conference (GLOBECOM'09),* 2009.

[8] S. Wang, S. Dey, "Addressing Response Time and Video Quality in Remote Server Based Internet Mobile Gaming," *in Proc. of IEEE Wireless Communication & Networking Conference (WCNC'10),* 2010.

[9] PlaneShift, http://www.planeshift.it/.