Improving the Quality and Efficiency of 3D Immersive Interactive Cloud Based Services by Providing an Adaptive Application Framework for Better Service Provisioning

DE-YU CHEN, University of California, Irvine
MAGDA EL-ZARKI, University of California, Irvine

Cloudifying 3D interactive multimedia applications is a promising way to provide flexible and cost efficient online high bandwidth immersive services to a large population of end users. One main reason cloud systems are popular among users is the fact that it relaxes the hardware requirements for high-end interactive visual applications. As most of the computational tasks are done on cloud servers, users no longer need to upgrade their hardware frequently to keep up with the ever increasing high end computing requirements of the latest applications. Moreover, cloud systems make it easier for a user to enjoy applications on different platforms, including mobile devices that are usually not powerful enough to run high-end, memory intensive services. In short, *cloudifying* high end immersive applications has advantages in cost efficiency and flexibility both for the end users and the service providers. There are two main drawbacks to *cloudifying* 3D interactive multimedia services: 1) latency and 2) high bandwidth utilization. We address these two problems in this paper.

1. INTRODUCTION

We believe there is a solution to *cloudifying* interactive immersive 3D applications, one that suits the providers, the developers and the users. In this paper, we propose a flexible way to provide cloud services to end users with different resources and quality constraints. By utilizing collaborative rendering and progressive meshes techniques, the proposed system is well suited to address different combinations of resource usages including local computing power, remote computing power and network bandwidth. Furthermore, thanks to the nature of progressive meshes, the system can switch between different usage cases dynamically and smoothly as the available resources change over time.

Some recent trends in the IT industry have changed the environment that once favored cloud usage. The slowing down of Moore's law makes upgrading hardware less urgent. According to the latest news, the PC upgrade cycle has slowed down to every five to six years [pc 2016]. At the same time, mobile devices are becoming more and more powerful. In addition, data transfer cost is still always a major concern for data intensive services such as video streaming, etc.. Amazon EC2 now charges $0.05 to $0.09 for every gigabyte of data transferred over the Internet [ec2 2016]. The price may not seem extremely high, but it adds up to the overall cost to the end users, given what they already need to pay for basic Internet access. Aside from the per unit cost for data transfer, the excessive network traffic can cause other problems such as packet loss and high latency, which impacts the users' quality of experience. All of these factors combined are making *cloudification* of interactive and rich 3D content applications less attractive.

However, given the wide diversity in user end devices, choosing the solution that best fits the needs of all end users is becoming a difficult problem for developers that want to offer a one size fits all solution. One cannot deny the long term benefit that cloud services can provide the industry and their clients - flexibility and economy of scale. If we can offer a solution that makes *cloudification* of new services, especially 3D immersive multimedia, feasible (i.e., high quality), at low cost to the infrastructure, than we have achieved that goal.

2. RELATED WORK

Most of the existing cloud based interactive 3D systems, that most often use gaming (due to its demanding service requirements) as their target or example application, use one of the following three methods: video streaming, graphics streaming, and collaborative rendering.

Video streaming is the most common way to provide online 3D immersive services. In this type of system, the server is in charge of executing both the application logic and the rendering. The rendered images are encoded and streamed to the client as a video service. Most of the commercialized services adopt this mechanism, including GameNow [gam 2016] and Gaikai [gai 2016]. Aside from proprietary systems, GamingAnywhere [Huang et al. 2014] is an open source example of a video streaming based cloud immersive system. The thin-client nature in this type of systems makes it an ideal solution to bring high-end immersive experiences to devices of all kinds. The main disadvantage is that video streaming consumes substantial bandwidth even with efficient encoding [Claypool et al. 2012].

In ***graphics streaming***, the server only updates the environment state. All API calls to graphics libraries such as OpenGL and DirectX are intercepted, encoded and streamed to the client. The rendering takes place on the client by executing the received graphics commands. Due to the complexity of the system, graphics streaming is not as popular as video streaming. Games@Large [Nave et al. 2008] is one of the very few systems that supports graphics streaming. Without efficient compression, the bandwidth required for streaming graphics commands can be higher than video streaming, and it fluctuates more dramatically. However, since the rendering is done locally on the client, it does not have the issue of lossy encoding and thus has less distortion in the visual quality than the video streaming method that is based on lossy compression.

Collaborative rendering requires even more resources on the client device. In this type of system, the server and the client both maintain a synchronized state space. The client renders a low quality version of the application locally, and the server renders it in both high and low quality, computes the difference, and sends it as a patch to the client for generating better visual quality. Since the difference is usually small and thus can be encoded efficiently, it does in general provide better visual quality with lower bandwidth costs, but with the price of extra computation cost on both the server and the client. Cuervo et al. [2015] proposed Kahawai, a system specifically for games that supports collaborative rendering.

2.1. Adaptive Cloud Based Interactive 3D Systems

The quality of service of all immersive and interactive cloud services depends heavily on network conditions. Various studies have addressed the issue of the dynamic and unstable nature of the Internet. Below we summarize some of those works that attempt to address the requirements of the applications, on different types of end user platforms, at the same time aiming for *best* performance in the face of resource constraints that are very unpredictable and that can severely impact the quality.

Hong et al. [2015] propose dynamic codec reconfiguration techniques to enable frame rate and bit rate adaptation in video streaming based systems. Shirmohammadi [2013] suggests that by omitting some less important objects, the complexity of the scene can be decreased, which not only lowers the bandwidth usage but also reduces the encoding time. However, it is also shown that the importance of the objects is context dependent. Therefore, the adaptation cannot be accomplished without additional efforts of the application developer associated with object labeling. In [Liu et al. 2015],

Liu et al. dynamically allocate a bit rate to macroblocks within each frame based on their importance calculated from rendering information.

Cai et al. [2015] propose a more generalized adaptation framework which considers both network and computing resources. By decomposing the environment into a set of inter-dependent components, each component can be assigned dynamically to either the server or the client for execution in order to meet different performance requirements. Although this framework is highly flexible, it is unlikely to be widely accepted by developers because it requires them to develop the 3D applications, such as games, in a whole new way. Besides, the computing and communication costs will most likely be dynamic and thus hard to estimate accurately.

2.2. Other Similar and/or Hybrid Approaches

Ahmadi et al. [2014] proposed a context based visual attention model called Game Attention Model (GAM). GAM considers two different forms of human attention: 1) Bottom-up stimulus driven attention, which is related to intensity, color, texture, etc. A saliency map is computed to characterize each feature. 2) Top down goal driven attention, which is related to the game context or goals. For example, users usually put more attention on objects that they can interact with, and less attention on the surrounding environment. Importance factors may vary in different applications, and they need to be decided by an application expert independently. The importance of each macro-block in a frame is determined using both the saliency map and the importance factors, which is then used to help the bit-rate allocation in the encoding process to improve the perceived quality with a certain bandwidth requirement. Their work is focused more on computer vision techniques while the approach we propose uses more graphics information. Furthermore, the fact that the importance factors need to be decided by a human for every application makes GAM less likely to be a good solution in a general purpose cloud immersive interactive system that aims to support all kinds of 3D applications, including games.

Wang and Dey [2010] propose a rendering adaptation technique for mobile 3D applications such as cloud gaming. The work adjusts the communication and computation cost to satisfy given constraints by changing a set of rendering parameters, including color depth, view distance, texture detail, environmental detail, and frame rate. They consider more rendering parameters then we do in our proposed work. However, the approach is not as generic as ours since not all of these parameters are used in all 3D applications. Besides, an off-line pre-processing step is required to characterize how these parameters affect the communication and computation cost. Another major difference with our proposed approach is the actual outcome perceived by the users. In their work, changing the rendering parameters does not only affect the communication and computation cost, but also changes significantly how the 3D scene looks, which may not be acceptable or even possible for some applications. In our approach we aim to keep the final visible result intact so that the users have a more consistent experience, irrespective of the resource constraints.

Nan et al. [2014] propose a cloud framework specifically for games where the distortion perceived by the players is minimized by maintaining and synchronizing graphics buffers on both the server and the client. At the beginning of the game, the client buffer is empty and the system works essentially as a pure video streaming system. In addition to streaming the compressed frames, the server also sends graphics data to the client progressively throughout the session. The graphics data is used to render reference frames on both the server and the client. The bandwidth consumption for streaming the frames can be reduced by choosing the reference frame with lowest residual error. The paper formulates and solves an optimization problem where available bandwidth is allocated between the compressed frame and the graphics data to

minimize visual distortion. Although they also use progressive meshes in their framework, the approach and optimization goal are different from ours. In their system, the graphics data is sent by the server at run time thus the client does not store any data locally. Nor does the client execute any game logic locally. It basically assumes a lightweight client, whereas we allow the end device to determine how ”lightweight” it wants to be. We claim that by using some resources on the client, our system will be able achieve several important advantages. Storing graphics data locally not only reduces the bandwidth usage but also improves the visual quality especially during scene changes since the application does not need to wait for the graphics data to arrive in the buffer. Executing the application locally requires more computing power on the client, but it allows our method to reduce or even eliminate interactive latency.

Chuah and Cheung [2014] propose a layered coding technique to reduce the bandwidth requirement for mobile cloud services such as gaming. The client renders the “base layer” of the scene by modifying several rendering parameters, including the number of polygons, illumination and texture mapping. The server compresses and sends the residue, or “enhancement layer”, to the client for it to improve the final image quality. Their idea is very similar to ours. However, their work does not tackle the practical issue of generating the 3D content for the base layer in real-time. We propose to use progressive meshes to support dynamic generation of graphics data with different levels of detail. Therefore, our system does not only take into consideration the variety in capabilities of clients' devices, but also has the ability to adapt to different use cases ever-changing network conditions.

Hemmati et al. [2013] adapt the number of objects rendered on the server to reduce the bandwidth requirement. The proposed system monitors the user's actions and prioritizes all of the objects based on their importance given the user's current activity. Depending on the client device's capability, the server renders only a number of the most important objects and streams the resulting frames as a video to the client. Similar to [Chuah and Cheung 2014], the importance of the objects is decided by the application designers who understand the context of the applications. Thus, it shares the same disadvantages. The quality of the solution depends largely on the designers' ability to identify important objects. It therefore makes it very hard to apply to all existing applications that suffer from the same issues. In addition, simplification by removing unimportant objects is not always possible. For example, when multiple objects are interacting in the scene, how will you decide which ones to remove? This approach makes it very hard to fine tune the system to meet all different performance constraints.

Shi et al. [2011] make use of depth information and 3D image warping technique to improve the bitrate-distortion ratio. The main idea of reducing the bit rate by utilizing low entropy residue images is similar to many other works including our approach. However, the residue is calculated by comparing the current frame with a previous frame, instead of comparing frames with different levels of detail. In order to minimize the entropy of the residue image, each frame is transmitted with the viewpoint information so the frame can be warped to another viewpoint to match a new frame. While making use of the already transmitted data to avoid the need of sending too much additional data is a great idea, but there are several fundamental problems. Firstly, the warped image will miss some information due to exposure and sampling. The authors try to solve the problem with double warping, where a previous frame and a predicted future frame are used together to generate a warped frame with the current viewpoint. This method requires a good prediction of change in user viewpoint. Otherwise, it can result in using even more bandwidth than simply streaming 3D graphics data. A more important problem is related to the moving objects. If there are a lot of moving objects in the game scene, the warped image will differ with the real image significantly

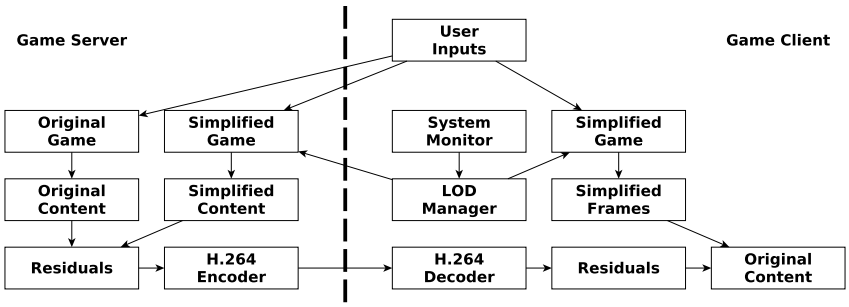

Fig. 1. System Model

since image warping technique does not consider the translation of all other objects. Due to these problems, this approach is mostly suited for applications such as remote virtual environment walkthroughs, where most objects are fixed and the movement of the viewpoint is fairly predictable – the path that a user can take through the space is known.

3. PROPOSED FRAMEWORK

3.1. System Overview

The main goal of our framework is to reduce the network bandwidth requirement and the end to end latency between the server and the client without increasing the overall workload of the system. The basic idea is to utilize the computing capacity of the client machine to generate close-to-original rendering results, thus only the residual frames are required to be transferred from the server for the client to restore the original visual quality. The system is designed to 1) support a wide variety of client machines that have different rendering capability and 2) be flexible, i.e., adjust dynamically to different hardware loads and network conditions.

Figure 1 above illustrates an overview of the system architecture. A system monitor runs on the client to keep track of the current GPU load and bandwidth usage. The Level of Detail (LOD) manager uses the collected information to decide on a desirable LOD that the environment should display and dynamically change the 3D models used in the application to meet that requirement. The server runs two application instances, one with the LOD decided by the LOD manager, the other with the original or highest possible LOD. A residual frame is generated by comparing the rendering results of the two versions for each frame. The server encodes the residual frames using an H.264 encoder and sends the compressed data to the client. The decoded residuals are added to the corresponding frames rendered by the client to recover the original quality.

To get the best bandwidth savings, the system requires that all environment instances running on both machines be perfectly synchronized. In other words, the states of each instance need to be consistent when rendering each frame. For now, we use a simple lockstep synchronization to make sure all environment instances read the same sequence of user inputs and execute the same set of state updating commands between any pair of consecutive frames. This means the proposed solution suffers a network latency of a round-trip delay. Latency is a very important issue, in particular for some classes of interactive applications. The initial emphasis in our work is on bandwidth savings for cloud services, and the solution we present below works fine for applications that do not have stringent latency requirements. Latency will be addressed in the proposed research section, in conjunction with bandwidth savings.

In our experimental system, both the server and the client have the original 3D models, including the mesh information, textures, etc., stored locally before the application

starts, so there is no bandwidth consumption for transferring the data. When a new base LOD is agreed upon, both sides run the mesh simplification algorithm and generate the new models independently. The base LOD of a model is chosen based upon the availability of resources. Lower detail requires more bandwidth with lower computation costs at the client. Higher detail models will translate to lower bandwidth requirements but higher workload at the client. In the next section we discuss these tradeoffs.

An alternative approach for client devices that lack storage capacity is to transmit the base model information in real time. When a new LOD is chosen by both sides, it asks the server to generate and transfer the new models. While saving storage space, this does require extra network bandwidth for the data transmission and an extra delay will be incurred before the new models are ready to be used on the client. These costs are not considered in our initial investigations of the viability of our approach since we believe the change in LOD of the base model should not happen too frequently and some possible approaches will be addressed in our proposed research section.

3.2. Bandwidth Reduction

3.2.1. Collaborative Rendering. In our proposed approach, we make use of the graphics processing capability of the client to reduce the network bandwidth requirement by means of collaborative rendering. In our collaborative rendering scheme, the client renders each frame with a lower LOD and uses the difference image calculated by the server to reconstruct the original high quality frame. By using delta encoding, the deltas, which are the difference images in our case, can be better compressed if the lower quality frames are close enough in detail to the original.

To calculate the difference image for each frame, we adopt the solution proposed by Levoy [1995]. The server first renders both the high and low quality frames, which are represented as 24-bit RGB pixels. The difference image is calculated pixel-by-pixel using the formula $D = 127 + (H - L)/2$, where (D, H, L) are the pixels in the difference image, high quality and low quality frames respectively. As one might notice, this method has the effect of losing one bit of information, but our experimental results show that the outcome is good. The difference images are then encoded and sent to the client. After decoding, the client adds the difference image to the low quality frame rendered locally to reconstruct the original, high quality frame. Since H.264 is lossy, the pixel values of the reconstructed frame may not be in the range 0-255. We ensure the values are valid by applying the formula $H = Max(Min(2 * (D - 127) + L, 255), 0)$ to each pixel.

There are several challenges when using the collaborative rendering mechanism. The first issue is related to the interaction delay. Assume the application starts with an initial state S_0. In each loop, it reads a set of user inputs I_i to update the environment state from S_{i-1} to S_i. To generate correct difference images, S_i has to be consistent on the server and the client for all i. As a result, the server needs to wait for the client to transmit I_i before rendering the high and low quality frames. On the other hand, the client has to wait for the difference image to reconstruct the high quality frame. The need for synchronization implies that the delay of the system is at least one full network round-trip time. Compared to traditional video streaming based cloud systems where the rendering is done completely on the server, our method does not incur any extra delay in network communication. However, the processing delay is expected to be longer due to the additional computation required at the client that consists of rendering, frame comparison and reconstruction. The overall added cost in delay will be discussed in Section 3.2.3.2.

The second issue is related to network bandwidth utilization. There are two key targets to achieving better bandwidth savings. The first is to generate difference images

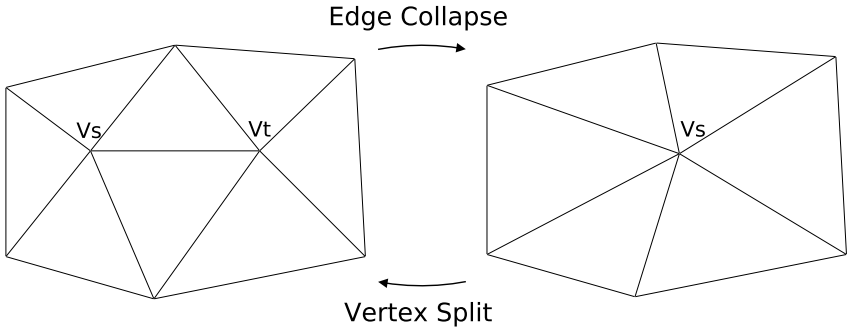

Fig. 2. Mesh Collapse and Mesh Split

that are more compressible. It is obvious that the more similar the low quality frame is to the high quality frame, the easier it is to compress the difference image. In our system, we dynamically adjust the LODs of the environment models according to the client's computing capacity by using progressive mesh, which will be discussed in Section 3.2.2. The second is the compression method. We choose to encode the difference images as H.264 video since the frames share similar spatial-temporal correlation. Regardless of the fact that the video coding standard H.264 is not designed specifically for this kind of application, our experimental results show that we can achieve decent compression ratios. Another advantage of using H.264 is that it is a popular standard and is widely supported on devices.

3.2.2. Dynamic Mesh Simplification and Restoration. As explained in the previous section, the bandwidth savings of the system depends on the graphics computing capability of the client. At one extreme, using our proposed system, a high-end PC that can run the application locally, there is no need for any network communication with the server and the full detailed model mode will be used. At the other extreme, for a client that is not powerful enough to do any graphics computation, the proposed approach will work as a traditional video streaming-based system. For all devices in between, who's computing capability lies between these two extreme cases, the system will perform in collaborative mode. In addition, often the available computing capacity and bandwidth resources for a client will vary over time depending on what other applications are running on the client device. There may also be instances where the developer will not release the application as a standalone and only allow users to use it on the provided cloud servers or the other extreme, not offer it as a cloud service. Our proposed method can operate in any of those modes.

The flexibility of our approach hinges on using progressive meshes proposed by Hoppe [1996] to enable dynamic change of the Level of Detail (LOD) of the base quality models. The transformation is accomplished by two operations: edge collapse and vertex split. Figure 2 illustrates an example of both operations. As shown in the figure, an edge collapse unifies v_s and v_t into a single vertex v_s. The edge connecting the two vertices and the faces associated with it are removed. A vertex split is the direct opposite to edge collapse. It splits a vertex into two vertices and creates a new edge between them, resulting in 1 or 2 additional triangles.

To construct the progressive mesh representation for a given mesh, we need to decide on a sequence of edge collapses that simplifies the original mesh until there is no edge to collapse or the number of edges is lower than a certain threshold. For each edge collapse, the attributes of the affected vertices and faces, e.g., the position of the new vertex, are recorded. These records are used in future vertex splits to transform

the mesh from a simpler form to a more complex one. It is shown by Hoppe in [Hoppe 1996] that the progressive mesh can be space-efficient with carefully designed encoding. However, since speed is more important than space in our application, we did not make any attempt to reduce the space usage. Many mesh simplification algorithms with different trade-offs between speed and accuracy can be used to construct the progressive mesh. In some applications, the progressive meshes can be constructed during the scene loading phase, in such cases, accuracy is considered more important than time complexity.

In our system, the LOD manager constructs and maintains a progressive mesh for each model loaded into the system. It dynamically decides on an ideal LOD based on the information acquired by the system monitor on available resources, then transforms the meshes using a sequence of edge collapses or vertex splits until they match the target LOD. The higher the LOD used for the locally rendered models, the more accurate it is in approximating the original ones, which results in less network bandwidth for transmitting the difference. However, rendering higher quality models takes more graphics computing power. Therefore, the LOD needs to be carefully selected to achieve acceptable computational cost, with highest bandwidth savings, while avoiding undesirable effects such as frame drops resulting in additional delay.

An alternative is to use a set of pre-built models with different LODs, as proposed by Cuervo et al in [Cuervo et al. 2015]. Although constructing progressive meshes requires some time and effort, it comes with several important advantages. First of all, progressive meshes provide a continuous sequence of models with different LODs. It makes the system more flexible and can be dynamically fine-tuned to adapt to different combinations of hardware capabilities, usage cases, and network environments. Second, since the progressive meshes are generated automatically as long as the original meshes are accessible, our solution is not limited to applications that have configurable LOD. Third, since the progressive meshes can be generated dynamically, we do not need to store multiple versions of a single model. This is especially important for mobile devices where the storage capacity is extremely limited. One last advantage of progressive meshes is that they naturally support progressive transmission. Although we currently assume the client can access the original models and construct the progressive meshes locally, it is possible to eliminate this storage and computation overhead by transmitting the simplified models on demand from the server. Progressive transmission will turn out to be useful in such situations.

3.2.3. Performance. To measure the performance of our approach, we developed an OpenGL-based application where a player can freely walk around a virtual city. The 3D model of the city contains about 160,000 faces and 36 texture files. Currently, we have not implemented a texture mapping algorithm for the progressive meshes, instead, we generate a non-textured model which replaces the texture content with the average color of the texture for every face. Both the textured and non-textured models are tested and analyzed to investigate the effect of our scheme. We defined the base LOD as the number of edges in the base mesh divided by the number of edges in the original mesh. Figure 3 shows the base model in different LOD configurations. A 600-frame movement sequence is recorded and replayed with base LOD set to 0, 10, 20, ... 100, while relevant information is saved for analysis. In the case of base LOD of 0, we stream the original frames instead of difference images, basically, the approach reverts to traditional video streaming. At the other extreme, for a value of 100, the approach reverts to operation as a standalone device.

The environment is setup on a server that runs on a 64-bit Windows 7 PC. The client is run on a Lenovo X1 Carbon laptop running the Ubuntu 16.04 operating system. Both machines are connected to the same local area network. As for the H.264 configuration,

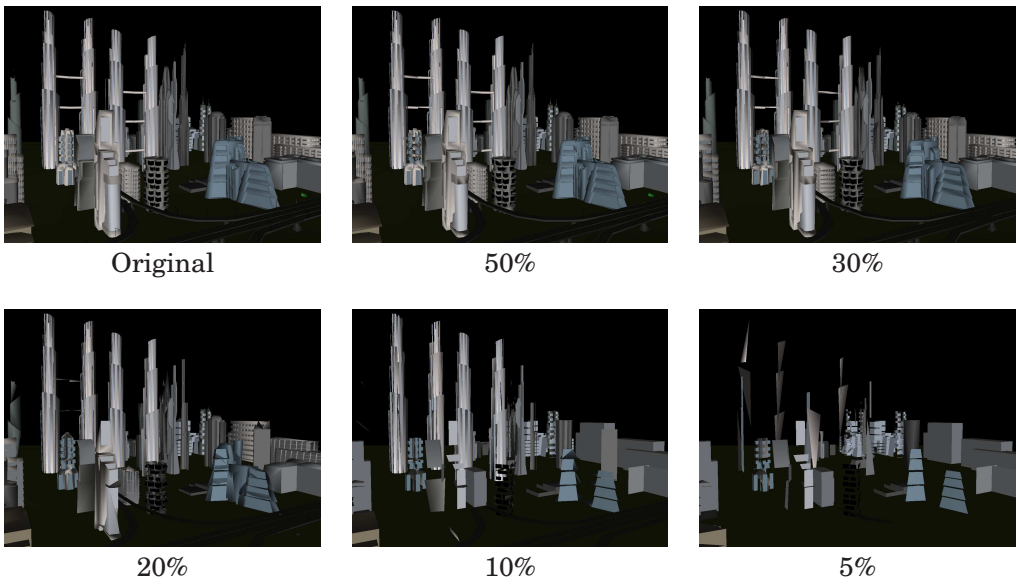

Fig. 3. Rendering Results with Different Levels of Detail

we use a frame rate of 30FPS, group of picture (GOP) of 10, and limit the output bit rate to 8Mbps. The prototype application is rendered and encoded at 1280x960pixel resolution.

Our measurement is focused on three performance metrics: network bandwidth, rendering delay, and visual quality. For network bandwidth, we are only interested in the encoded difference images sent from the server to the client, since the control messages sent from the client to the server are so small and can be ignored. The rendering delay is the time the client uses to render the base model in each frame. We use it as an indicator of the graphics computing workload on the client hardware. For visual quality, we calculate the PSNR from the original image rendered by the server and the reconstructed image generated by the client for each frame. Since we are using a lossy compression algorithm (H.264) to encode the image differences, there is a trade-off between compression ratio and image quality. We need to be aware of how much we sacrifice for the bandwidth savings. In the remainder of this section, we will discuss how the base LOD affects these three performance metrics.

3.2.3.1. Network Bandwidth. Figure 4 shows how base LOD affects the bandwidth usage. The average size of the encoded difference images is very small when the base LOD is higher than 70%, then it starts to grow faster as the LOD drops. When the base LOD is lower than 50%, the bandwidth savings become insignificant.

With closer examination, we found that when the model is simplified to a certain level, some buildings disappear completely and that makes the objects behind them visible, which may increase the size of the encoded difference image dramatically. A potential solution is to use a view-dependent progressive mesh that tries to keep faces closer to the viewpoint intact. However, view-dependent progressive meshes need to be updated whenever the user changes the viewpoint, which happens constantly in most of these applications. Finding a way to reduce the bandwidth usage in low base LOD with acceptable extra effort is an important issue that we will address in future research described in Section 4.

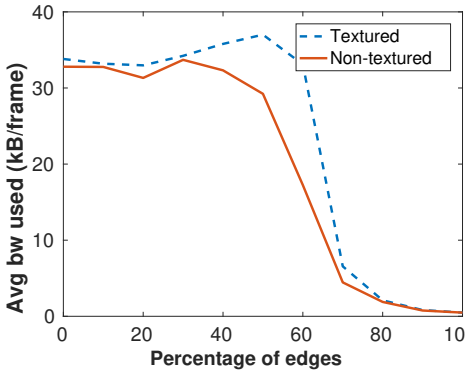

Fig. 4. Network Bandwidth Comparison

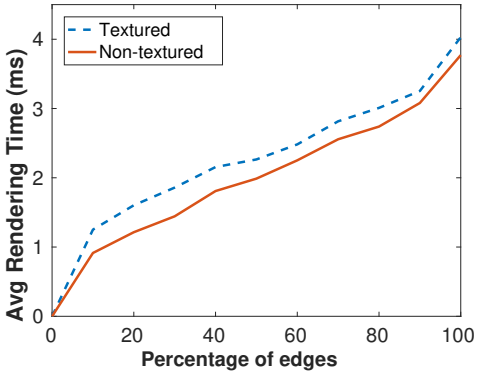

Fig. 5. Rendering Delay Comparison

Another phenomenon that requires explanation is the high peak in bandwidth for Textured Model when the base LOD is 50%. The main reason for this effect is the lack of a texture mapping algorithm in the current progressive mesh implementation. When the mesh is simplified, several faces may be merged into a larger one. To more accurately approximate the original mesh, the texture mapping of the remaining faces should be updated accordingly. Otherwise, the texture will be stretched and thereby introduce a significant difference in the two frames. When the base LOD is decreased to a fairly low value, the faces that introduce large differences may disappear altogether, which will actually reduce the complexity in the difference images. This effect may explain the irregular trend of bandwidth usage when the base LOD is between 0% and 50%.

3.2.3.2. Rendering Delay. The advantage of rendering lower LOD models locally on the client is to lower the graphics computational workload. We use the time it takes to render a frame to quantify the graphics computing power used. As most applications are configured to run using a specific frame rate, the shorter time it takes to render a single frame, the better it will be for a lower powered machine to achieve the required frame rate. Figure 5 illustrates a linear relationship between the rendering time per frame and the base LOD, which proves that using progressive meshes allows us to flexibly adjust the workload requirement on the available graphics computing power.

3.2.3.3. Visual Quality. The visual quality can be an issue for our system for two reasons. First, we use H.264 to encode the difference images, which is not the typical data the standard was designed to encode. Second, as was mentioned in Section 3.2.1, the difference images lose one bit of information for every pixel before encoding, possibly impacting quality.

In spite of these issues, our experimental results show that the system is capable of providing acceptable visual quality. As shown in Figure 6, all collaborative rendering cases generate better visual quality than traditional video streaming in terms of average PSNR. The system achieves a PSNR of about 87.5 when the LOD is 90%, regardless whether textured or non-textured models are used. The PSNR value drops gradually to about 42.5 and 32.5, when non-textured models and textured models are used, respectively. In terms of average SSIM, the measurement is near 1, which means least distortion, with higher base LODs. The performance starts to degrade rapidly as the base LOD decreases below 30%. When the non-textured model is used, the lowest average SSIM is about 0.98, which is still a satisfying quality. The performance is worse when we use the textured model, where the average SSIM goes down to as low as

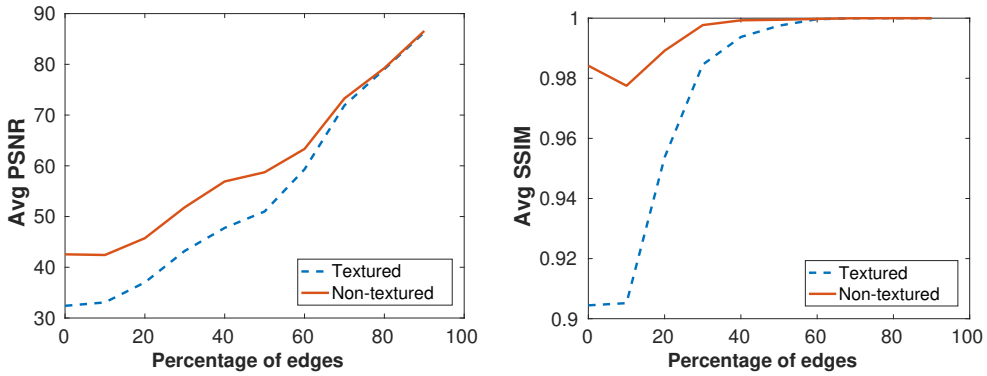

Fig. 6. Visual Quality Comparison

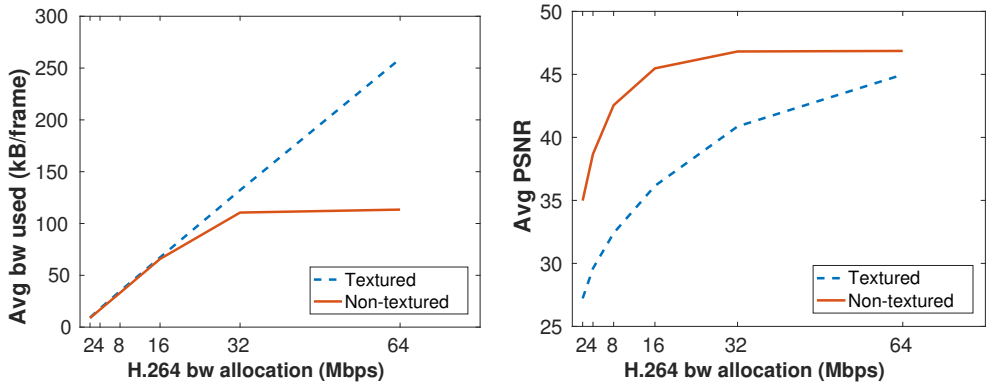

Fig. 7. Effect of H.264 BW constraints on video streaming a) bitrate/frame, b) quality

0.9. However, the relative performance is actually better in comparison to traditional video streaming, since the more complex frames generated by the textured model are hard to be encoded efficiently even for traditional video streaming. As shown in Figure 7, it takes at least 32Mbps for traditional video streaming to achieve a PSNR of 40 or higher. That is a very high bandwidth cost.

3.3. Reducing Interactive Latency

Interactive latency is one of the most important issue in interactive cloud services. It is shown in user studies that the maximum tolerable delay for online interaction is around 100ms to 150ms [Beigbeder et al. 2004; Dick et al. 2005; Quax et al. 2004]. High interactive latency not only damages users subjective impression of the service, but can also affect their objective performance. It is shown in [Lee et al. 2015] that it is possible to trade-off other resources to reduce latency. However, different types of applications may have different latency requirements. In general, slower-paced environments (chatting, walk through), usually have higher tolerable delay than faster-paced ones (action, drive through). Therefore, we want to design a mechanism where the trade-off conditions can be modified more flexibly.

In a naive lock-step system, the client always waits until it gets the residue for a certain frame from the server before it displays the patched frame to the user, as is shown in Figure 8. The interactive latency in this implementation depends on the network round-trip time between the server and the client, which is unpredictable

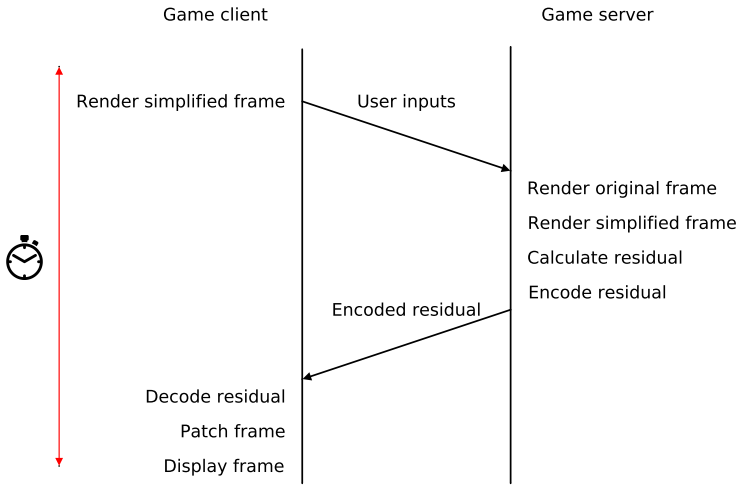

Fig. 8. Lock Step Approach

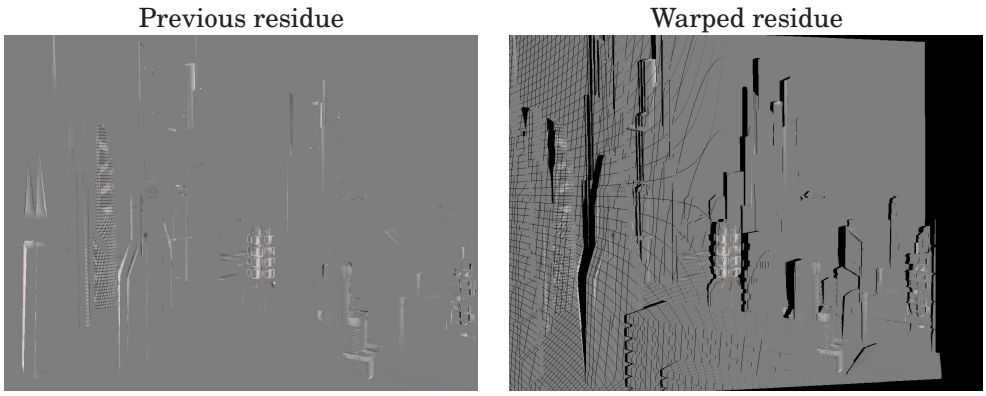

Fig. 9. 3D image warping

and usually not acceptable for highly interactive applications. Since the client already renders a lower LOD version of the frame locally, it is possible to reduce the interactive latency by sacrificing some visual quality.

3.3.1. 3D image warping. The main idea is to make use of previous residual frames the client has already received. In such a system, the client sets a fixed delay period for each frame to be displayed. At the time a frame is due to be displayed, the client first checks if it has already received the residue of this frame. In case where the residue is ready on the client, it patches the frame and displays it as usual and there will be no extra degradation to the image quality. Otherwise, the client uses the latest residue it has received and buffered for patching the current frame. Although the viewpoint of the latest residue is expected to be very close to the current viewpoint, they are usually not exactly the same. In order to make the previous residue perfectly match up with the current frame, we use a ***3D image warping technique*** [McMillan Jr 1997]. In Figure 9 we show an example of how it works. As is shown, the information of some pixels in the warped image will be missing due to the occlusion effect or insufficient sampling. As one can imagine, the closer the viewpoints of the two frames are, the

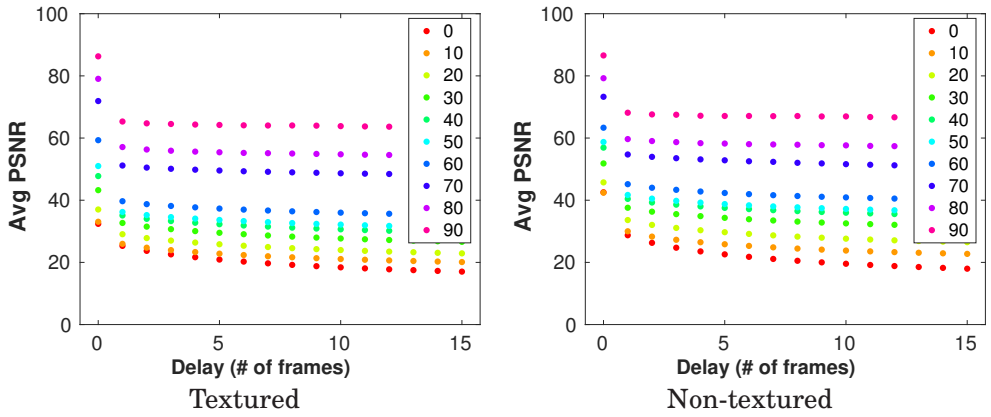

Fig. 10. PSNR vs Different Delay

less information will be missing (left image further than right). Thus, there is a trade-off between latency and visual quality, if the client can wait a longer period (lower interactivity), it might receive a residue that matches up with the current frame better and less information will be missing.

Except higher distortion, the reduced delay also comes with higher bandwidth requirement. For 3D image warping to work, we need extra data about the viewpoint and depth information for each frame. Both can be retrieved by the client locally. But since the client runs the game with lower level-of-detail objects, the depth information it gets is inaccurate. A solution is to calculate the residue for the depth information on the server and send it to the client. This will approximately double the bandwidth usage. In fact, as most of the existing video encoders do not support this type of information, we need to come up with a good compression method or it may end up consuming more bandwidth than the residual image itself. The compression of the depth information is left for future work.

As is mentioned earlier, information will be missing in the warped images. To solve the problem, many hole filling algorithms have been proposed. However, there are a few things we need to consider when choosing or designing the hole filling algorithm. First, since the image warping and the hole filling have to be done on the client side, they will make even more workload on the client device that is supposed to have limited computing capability. Second, the missing information is just the residue. It means that we can possibly get a reasonably accurate value even without recovering the residue. It is yet another trade-off problem between time complexity and accuracy that needs to be investigated. Our current prototype system uses a simple algorithm that fills the value of a missing pixel with the mean of its closest neighbor.

3.3.2. Moving objects. Moving objects is another important issue for reducing interactive latency. 3D Image warping technique considers only the change of the viewpoint but not the movement of other objects. We plan to implement a two layer rendering approach. The first layer includes all stationary objects, which is patched by the residual image. The second layer includes the moving objects and some other on-screen information such as menu, etc. The second layer is rendered locally on top of the first layer, and is not affected by the residual image. As a result, moving objects need to have higher LOD. The issue of supporting different LODs for different objects will be the focus of future work when we take into consideration both user movement and moving objects.

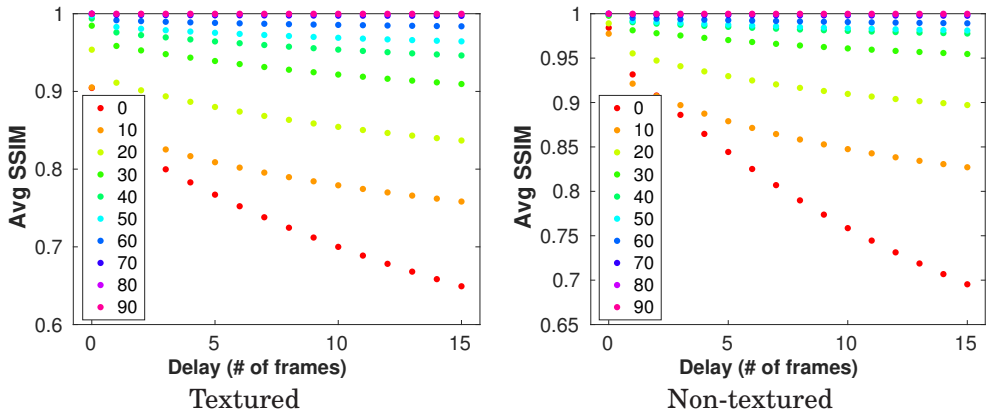

Fig. 11. SSIM vs Different Delay

3.3.3. Performance. To understand the performance of our latency reduction approach, we conduct an offline experiment that changes the interactive latency, in terms of number of frames, and measures the visual quality of the output video. For example, when the delay is set to N, the system always uses the residual frame it received N frames earlier to patch the current frame for display. We assume that the original depth information for each residual frame is available on the client. All of the other settings are kept the same as our previous experiment. The results of the PSNR and SSIM measurement are shown in Figure 10 and Figure 11, respectively.

When the delay is 0, the residue of the current frame is available on the client and can be used for patching directly without being warped, thus there will be no distortion due to the 3D image warping procedure. When the delay is larger than 0, image warping is required and a significant performance drop can be seen in the figures, regardless textured or non-textured models are used. As the delay increases, the difference in the viewpoints of the current frame and the residual frame becomes larger. As a result, the image warping process causes more information loss and the image quality decreases gradually.

When comparing the results of different LODs, we found that the delay affects the image quality more significantly when lower quality models are used. We believe the discrepancy can be explained by the amount of information within the residual frames. When higher LOD models are available, the locally rendered frames are very close to the original ones, and very little meaningful information is provided by the residual frames. In such a case, the benefit can be easily diminished by the distortion caused by the image warping. On the other hand, if lower LOD models are used, the client depends largely on the residual frames to display the image with reasonable quality. Thus, even distorted residual frames can provide large amounts of useful information that helps improve the image quality. Nevertheless, using the warped residue always has positive impact on the image quality in all of our experiment cases. Whether the improvement on image quality is worth the extra computation and communication costs requires further study.

4. CONCLUSION AND FUTURE WORK

In this paper we proposed an adaptive streaming framework for 3D interactive immersive cloud services such as gaming and virtual environments. By utilizing techniques such as collaborative rendering and progressive mesh, the proposed system dynamically adjusts its resource usage to fit different hardware constraints and performance

requirements. The proposed approach presents a flexible solution for providing gaming services to devices with different capabilities and having different network resources. Our experimental results show that rendering time on the client is almost linear to the number of edges in the base model, and that in most cases the system provides better video quality than pure video streaming. However, the bandwidth usage can be higher than pure video streaming when less than 50% of the edges are used in the base model, which may be caused by inaccurate texture mapping when the model is simplified. We plan to implement a better texture mapping algorithm for the progressive meshes in order to lower the entropy in the residue and decrease the bandwidth usage for lower LOD models.

In order to tackle the interactive latency issue, we utilized 3D image warping techniques whereby previously received information is used to improve the image quality of the current frame. With this technique, the delay can be flexibly controlled to meet any latency requirement. However, reduction in the interactive latency will come at a cost of image quality. The experimental results show that it is possible to eliminate interactive latency of up to 15 frames, or 500 milliseconds, and still provide an acceptable image quality. However, extra information, such as accurate depth information, needs to be sent from the server for 3D image warping to work adding to the bandwidth usage. How to efficiently compress the depth information we have not yet addressed and plan to tackle that in future work.

View-dependent progressive meshes is another of our future foci. For the current prototype, the LOD of all environment objects are kept synchronized, which is calculated as a function of the available resources on the client device. But as different objects may have different ***importance***, it would be better if we can set the LOD of each object separately depending on its importance. There are two types of importance we need to consider. One is related to the application context. The other type of importance is related to the geological locations of the objects or *view dependent LOD* [Hoppe 1997; 1998; Levenberg 2002].

In our current version of the system, the residual images are encoded to H.264 video format. Although residual images may have similar temporal and spatial correlations to those of regular video stream, there are subtle differences between these two types of data: (1) the value distribution is less dispersed in the residual images; (2) the pixel values hold less importance in residual images, especially when higher LOD content is rendered on the client. Thus, existing encoders that are designed for regular images may not be able to reach optimal compression ratio for residual images. An ultimate solution is to design a specialized codec, or modify some parts, such as the quantization mechanism or the entropy coding, of an existing codec, to make it more efficient in compressing residual images. No matter which solution we take, the decrease of bitrate will come with an increase in the image distortion. Thus, a user study or a perceptual video quality assessment method [Wang et al. 2002; Moorthy and Bovik 2011; Wang and Li 2011] is required for measuring the quality-bitrate ratio.

The use of collaborative rendering and progressive mesh means that our approach has to be intrusive. We therefore believe that the best way to make the proposed framework available is by integrating it with an existing 3D environment engine. While the involvement of the application developers is unavoidable, one of the main design goals is to make the tool as easy to use as possible so developers can easily cloudify their existing applications. With the help of progressive meshes, generating different LOD representations of an object is done automatically. However, as is mentioned in previous sections, there are several attributes that can be associated with an object, such as: (1) whether it is a moving or stationary object; (2) the minimum acceptable LOD for the object based on its importance. These attributes will be considered in the LOD management component of our framework. To further help the system improve in quality,

we plan to build an interface for the developers to specify requirements on how their system should work. The available requirements will include desirable frame rate, maximum acceptable latency, available bandwidth to LOD mapping, etc.

REFERENCES

2016. Amazon EC2 Pricing. https://aws.amazon.com/ec2/pricing/. (July 2016).

2016. Gaikai official website. http://www.gaikai.com/. (July 2016).

2016. GameNow official website. http://www.ugamenow.com/. (July 2016).

2016. The PC upgrade cycle slows to every five to six years, Intel's CEO says. http://www.pcworld.com/article/3078010/hardware/the-pc-upgrade-cycle-slows-to-every-five-to-six-years-intels-ceo-says.html. (July 2016).

Hamed Ahmadi, Saman Zad Tootaghaj, Mahmoud Reza Hashemi, and Shervin Shirmohammadi. 2014. A game attention model for efficient bit rate allocation in cloud gaming. *Multimedia Systems* 20, 5 (2014), 485–501. DOI:http://dx.doi.org/10.1007/s00530-014-0381-1

Tom Beigbeder, Rory Coughlan, Corey Lusher, John Plunkett, Emmanuel Agu, and Mark Claypool. 2004. The Effects of Loss and Latency on User Performance in Unreal Tournament 2003®. In *Proceedings of 3rd ACM SIGCOMM Workshop on Network and System Support for Games (NetGames '04)*. ACM, New York, NY, USA, 144–151. DOI:http://dx.doi.org/10.1145/1016540.1016556

W. Cai, H. C. B. Chan, X. Wang, and V. C. M. Leung. 2015. Cognitive Resource Optimization for the Decomposed Cloud Gaming Platform. *IEEE Transactions on Circuits and Systems for Video Technology* 25, 12 (Dec 2015), 2038–2051. DOI:http://dx.doi.org/10.1109/TCSVT.2015.2450171

Seong-Ping Chuah and Ngai-Man Cheung. 2014. Layered Coding for Mobile Cloud Gaming. In *Proceedings of International Workshop on Massively Multiuser Virtual Environments (MMVE '14)*. ACM, New York, NY, USA, Article 4, 6 pages. DOI:http://dx.doi.org/10.1145/2577387.2577395

M. Claypool, D. Finkel, A. Grant, and M. Solano. 2012. Thin to win? Network performance analysis of the OnLive thin client game system. In *Network and Systems Support for Games (NetGames), 2012 11th Annual Workshop on*. 1–6. DOI:http://dx.doi.org/10.1109/NetGames.2012.6404013

Eduardo Cuervo, Alec Wolman, Landon P. Cox, Kiron Lebeck, Ali Razeen, Stefan Saroiu, and Madanlal Musuvathi. 2015. Kahawai: High-Quality Mobile Gaming Using GPU Offload. In *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys '15)*. ACM, New York, NY, USA, 121–135. DOI:http://dx.doi.org/10.1145/2742647.2742657

Matthias Dick, Oliver Wellnitz, and Lars Wolf. 2005. Analysis of Factors Affecting Players' Performance and Perception in Multiplayer Games. In *Proceedings of 4th ACM SIGCOMM Workshop on Network and System Support for Games (NetGames '05)*. ACM, New York, NY, USA, 1–7. DOI:http://dx.doi.org/10.1145/1103599.1103624

Mahdi Hemmati, Abbas Javadtalab, Ali Asghar Nazari Shirehjini, Shervin Shirmohammadi, and Tarik Arici. 2013. Game As Video: Bit Rate Reduction Through Adaptive Object Encoding. In *Proceeding of the 23rd ACM Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV '13)*. ACM, New York, NY, USA, 7–12. DOI:http://dx.doi.org/10.1145/2460782.2460784

H. J. Hong, C. F. Hsu, T. H. Tsai, C. Y. Huang, K. T. Chen, and C. H. Hsu. 2015. Enabling Adaptive Cloud Gaming in an Open-Source Cloud Gaming Platform. *IEEE Transactions on Circuits and Systems for Video Technology* 25, 12 (Dec 2015), 2078–2091. DOI:http://dx.doi.org/10.1109/TCSVT.2015.2450173

Hugues Hoppe. 1996. Progressive Meshes. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '96)*. ACM, New York, NY, USA, 99–108. DOI:http://dx.doi.org/10.1145/237170.237216

Hugues Hoppe. 1997. View-dependent Refinement of Progressive Meshes. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '97)*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 189–198. DOI:http://dx.doi.org/10.1145/258734.258843

H. Hoppe. 1998. Smooth view-dependent level-of-detail control and its application to terrain rendering. In *Visualization '98. Proceedings*. 35–42. DOI:http://dx.doi.org/10.1109/VISUAL.1998.745282

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 Trans. Multimedia Comput. Commun. Appl.* 10, 1s, Article 10 (Jan. 2014), 25 pages. DOI:http://dx.doi.org/10.1145/2537855

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)*. ACM, New York, NY, USA, 151–165. DOI:http://dx.doi.org/10.1145/2742647.2742656

J. Levenberg. 2002. Fast view-dependent level-of-detail rendering using cached geometry. In *Visualization, 2002. VIS 2002. IEEE*. 259–265. DOI:http://dx.doi.org/10.1109/VISUAL.2002.1183783

Marc Levoy. 1995. Polygon-assisted JPEG and MPEG Compression of Synthetic Images. In *Proceedings of the 22Nd Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '95)*. ACM, New York, NY, USA, 21–28. DOI:http://dx.doi.org/10.1145/218380.218392

Y. Liu, S. Dey, and Y. Lu. 2015. Enhancing Video Encoding for Cloud Gaming Using Rendering Information. *IEEE Transactions on Circuits and Systems for Video Technology* 25, 12 (Dec 2015), 1960–1974. DOI:http://dx.doi.org/10.1109/TCSVT.2015.2450175

Leonard McMillan Jr. 1997. *An image-based approach to three-dimensional computer graphics*. Ph.D. Dissertation. University of North Carolina at Chapel Hill, Department of Computer Science.

A. K. Moorthy and A. C. Bovik. 2011. Blind Image Quality Assessment: From Natural Scene Statistics to Perceptual Quality. *IEEE Transactions on Image Processing* 20, 12 (Dec 2011), 3350–3364. DOI:http://dx.doi.org/10.1109/TIP.2011.2147325

X. Nan, X. Guo, Y. Lu, Y. He, L. Guan, S. Li, and B. Guo. 2014. A novel cloud gaming framework using joint video and graphics streaming. In *2014 IEEE International Conference on Multimedia and Expo (ICME)*. 1–6. DOI:http://dx.doi.org/10.1109/ICME.2014.6890204

I. Nave, H. David, A. Shani, Y. Tzruya, A. Laikari, P. Eisert, and P. Fechteler. 2008. Games@Large Graphics Streaming Architecture. In *2008 IEEE International Symposium on Consumer Electronics*. 1–4. DOI:http://dx.doi.org/10.1109/ISCE.2008.4559473

Peter Quax, Patrick Monsieurs, Wim Lamotte, Danny De Vleeschauwer, and Natalie Degrande. 2004. Objective and Subjective Evaluation of the Influence of Small Amounts of Delay and Jitter on a Recent First Person Shooter Game. In *Proceedings of 3rd ACM SIGCOMM Workshop on Network and System Support for Games (NetGames '04)*. ACM, New York, NY, USA, 152–156. DOI:http://dx.doi.org/10.1145/1016540.1016557

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 (MM '11)*. ACM, New York, NY, USA, 103–112. DOI:http://dx.doi.org/10.1145/2072298.2072313

Shervin Shirmohammadi. 2013. Adaptive streaming in mobile cloud gaming. *IEEE COMSOC Multimedia Communications Technical Committee E-Letter* (2013), 20–23.

S. Wang and S. Dey. 2010. Rendering Adaptation to Address Communication and Computation Constraints in Cloud Mobile Gaming. In *Global Telecommunications Conference (GLOBECOM 2010), 2010 IEEE*. 1–6. DOI:http://dx.doi.org/10.1109/GLOCOM.2010.5684144

Z. Wang and Q. Li. 2011. Information Content Weighting for Perceptual Image Quality Assessment. *IEEE Transactions on Image Processing* 20, 5 (May 2011), 1185–1198. DOI:http://dx.doi.org/10.1109/TIP.2010.2092435

Zhou Wang, H. R. Sheikh, and A. C. Bovik. 2002. No-reference perceptual quality assessment of JPEG compressed images. In *Image Processing. 2002. Proceedings. 2002 International Conference on*, Vol. 1. I–477–I–480 vol.1. DOI:http://dx.doi.org/10.1109/ICIP.2002.1038064