Multiscreen-based Gaming Services using Multi-view Rendering with Different Resolutions

Sung-Soo Kim and Chunglae Cho
Electronics and Telecommunications Research Institute (ETRI)
Daejeon, South Korea
{*sungsoo, clcho*}*@etri.re.kr*

***Abstract*—We present a novel multiscreen-based gaming service system which supports multiple-viewpoint rendering with different resolutions for visualizing a 3D game scene dataset at the same time. Our approach is based on multi-view rendering and reduces the computation to generating video streams for cloud-based gaming services. In addition, the performance speedup of our rendering system is achieved by utilizing both multicore CPUs and a GPU simultaneously without additional requirement for any special hardware. The experimental results demonstrate our multi-view rendering method can be successfully applied to the multiscreen services for the multiplayer games.**

Keywords-multiscreen services; gaming on demand; multi-view rendering; video encoding; video streaming.

I. INTRODUCTION

Consumers desire to access rich multimedia and realistic 3D game content via smartphone, PCs, netbooks, tablets anytime and anywhere. *Multiscreen services* have emerged as a consequence of this user requirement. Users can watch multimedia and game content from any source on any screen through the the multiscreen services. Also, the growth in connectivity and capacity of broadband networks have enabled new forms of *cloud computing*, where data and processing on a remote server is acted upon on a local computing device. This computing model allows a performance focus at a single location, the cloud server, and enables user mobility and pervasive access for the users.

One of the latest advancements in gaming technology that enables such a ubiquitous gaming is *cloud-based gaming services*, also called *Gaming on Demand* (GoD) [1]. Cloud gaming will liberate games from their limiting dependence on consoles, without sacrificing realism, speed, or any other aspect of the true gaming experience. This is a platform-as-a-service approach, analogous to video on demand, where players interact via streamed content generated on the game operator's server rather than players' local systems. There are a number of commercial GoD systems that have been presented to the market such as OnLive [2] and Gaikai [3]. We have idenified four key requirements of cloud-based service systems to provide convincing multiscreen-based gaming services [4]. They are *user responsiveness*, *high-quality video*, *quality of services* and *operating costs*.

Our contributions: We present a novel system architecture for the multiscreen gaming services, which utilizes parallel commodity processors, multi-core CPUs. We also present a novel multi-view rendering algorithm to efficiently support multi-user game on the server, which has a single GPU with multi-core CPUs. In addition, our approach gives the benefits in terms of *arbitrary focal positions* for viewpoints and better rendering quality over prior parallel multi-view rendering methods [5]. This is one of the important features for the multiplayer games in cloud-based gaming services.

The rest of the paper is organized as follows. Section II shows a brief overview of related work. Section III describes the proposed system architecture for cloud-based gaming services. Section IV shows the performance of the proposed method. Finally, Section V ends the paper with some concluding remarks and perspectives for future work.

II. RELATED WORK

In this section, we give a brief overview of related work on cloud-based gaming platforms and parallel rendering algorithms.

Cloud-based gaming platforms: There is a number of commercial cloud-based gaming platforms that have been presented to the market. The *Games@Large* framework enables commercial video game streaming from a local server to remote end devices in local area networks (LANs) [6]. This system and streaming protocols are developed and adapted for highly interactive video games. OnLive is a gaming-on-demand entertainment platform, which their service is available in USA, UK and Belgium [2]. The hardware used is a custom set up consisting of OnLive's proprietary video compression chip as well as standard CPU and GPU chips. Gaikai is developing and delivering a cloud technology platform to put games where they have never been before, including digital TVs, tablets, smartphones, Facebook, and embedded directly into websites. Recently, NVIDIA introduced the *GeForce GRID platform* for gaming-as-a-service providers [7]. The key technologies of this platform are NVIDIA GeForce GRID GPUs with dedicated ultra-low-latency streaming technology and cloud graphics software.

Parallel rendering: Recent work in this area has been focused on *video encoding* and *streaming* techniques to reduce the latency in games [8]. Most of the earlier systems were serial in nature and designed for a single core or processor in terms of 3D rendering. However, the recent trend in computer architecture has been toward developing parallel commodity processors, including multi-core CPUs and many-core GPUs.

 ISBN: 978-1-61208-229-5

It is expected that the number of cores would increase at the rate corresponding to Moore's Law. Based on these trends, many parallel game engines and parallel rendering algorithms have been proposed for commodity parallel processors [9]. Dual-core and quad-core CPU chips are currently available, with some motherboards supporting multiple such chips. Parallel computing is quickly becoming mainstream in the development of computation-intensive applications related to the realistic 3D rendering [10].

OTOY [11] provides technologies that move processor intensive experiences into the cloud; computer applications, operating system, video games, high-definition media content, film/video special effects graphics - fully interactive, in real time, through the power of *server side rendering*. They can deliver the high-quality media content via an interactive stream to any internet enabled device for multiscreen-based gaming services, including PC, iPhone, iPad or TV set top box.

A parallel multi-view rendering architecture in a cluster of GPUs has been proposed in [5]. This system have shown a theoretical analysis of speedup and scalability of the proposed multi-view rendering. However, the critical limitation of this method is that all the cameras are always looking to the center of arbitrary tile. Therefore, this method is not suitable for common mutli-user game applications. Moreover, it is difficult to apply this method to a *high visual quality* games since they used a simple phong shader for lighting and shading.

III. System Architecture

In this section, we describe the proposed system architecture for cloud-based gaming services. Our system consists of three major systems such as distributed service platform (DSP), distributed rendering system (DRS) and encoding, QoS and streaming system (EQS), as shown in Figure 1. The DSP is responsible for launching the game processes on the game execution nodes or rendering job on the DRS after client-side invocation, monitoring its performance, allocating computing resources and managing user information. And, the DSP handles user's game input via UDP from the client-side devices. In client-side, the user's game input is captured and transmitted via UDP by the user input capturing and transmission software on the client devices. Also, the DSP performs execution management of multiple games. In order to perform streaming the game A/V streams to the clients, the DSP requests capturing rendered frame buffer for video encoding and streaming to the EQS.

To improve 3D rendering performance of the DRS, we utilize the multi-threaded game engine [9] that is designed to scale to as many processors as are available within a platform. In order to provide the cloud-based gaming service, the DRS has common system interfaces to the DSP and the EQS. The EQS is responsible for audio/video encoding and streaming the interactive game content to the clients. We use the DirectShow SDK to implement the visual capturing of the games rendered from the DRS. We utilize the H.264 video coding standard for low-delay video encoding of the captured game content. Before the EQS performs the H.264 encoding, we perform a color space conversion from RGB to YUV on the captured frames. Finally, we exploit the Real Time Protocol (RTP) packetization to transmit the encoded video stream in real-time [12].

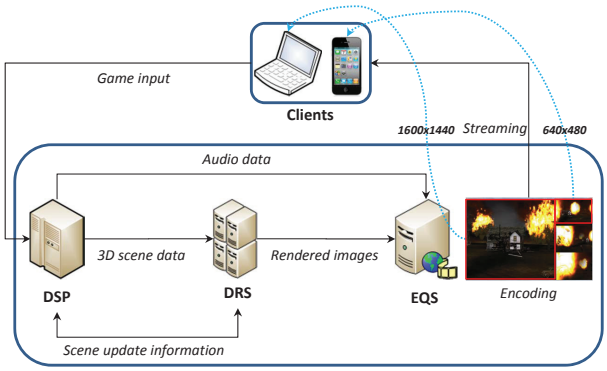

Fig. 1. Our system processing flow: *DSP-Distributed Service Platform, DRS-Distributed Rendering System, EQS-Encoding, QoS and Streaming System*

A. Multi-view Rendering with Different Resolutions

The DRS consists of four major block components such as *rendering scheduler*, *multi-view manager*, *rendering task manager* and *renderer library*. The rendering scheduler is responsible for rendering process monitoring, performance timer control, rendering statistics management and communicating other modules for external rendering requests in the DRS blocks. The key performance improvement for the game applications is the use of per-thread task queues. This eliminates the synchronization checkpoint when one shared task queue is used. Advanced task schedulers may use heuristics to determine which thread to steal from and which task to steal and this may help cache performance. In order to implement the rendering scheduler, we use the Intel Threading Building Blocks (TBB) [9], which is highly optimized scheduler. The multi-view manager is responsible for performing the management of user's viewpoints (such as insertion, deletion, update and search operations) for the shared spaces in the multi-user games. The rendering task manager module performs the rendering task decomposition and parallelization in order to improve the rendering performance. In our work, we use the Object-oriented Graphics Rendering Engine (OGRE) [9], which performs a 3D scene graph management and rendering. In the case of multiscreen-based gaming services, multiple viewpoints for different resolutions are needed if we want to support several users visualizing a given 3D scene at the same time. However, rendering multiple views with different resolutions using the standard graphics pipeline is a challenging problem. In order to provide the interactive multi-view rendering results for the multiscreen-based gaming service, we utilized the shared resources for the rendering such as scene graph, textures and shaders in a GPU as much as possible and keeping the quality of the rendering results. We exploit a video streaming approach to provide the game's encoded video with different resolutions at the same time

 ISBN: 978-1-61208-229-5

Fig. 2. The result of multi-view rendering with different resolutions (1: 1600x1440, 2,3,4: 640x480)

according to the requests of client devices such as PCs, laptops (e.g., 1600x1440 resolution) and smartphones (e.g., 640x480 resolution) as shown in Figure 2.

If R_i denotes a i-th rendered image in framebuffer of the DRS, then $\mathcal{S}_k$, which has i image sequences is defined as:

$$\mathcal{S}_k = \{R_1, R_2, ..., R_i\}$$

The $\mathcal{CP}_i$ denotes the i-th viewpoint parameters, which contains internal parameters such as focal length $f_l(f_x, f_y)$, center $c(c_x, c_y)$, aspect ratio a and external parameters such as position $p(c_x, c_y, c_z)$ and orientation $r(r_x, r_y, r_z)$. The DSP generates this $\mathcal{CP}_i$ and the resolution of the client screen, (x_r, y_r), according to the requests of the clients.

Algorithm 1 Viewpoint addition algorithm.

```
1: procedure ADDVIEW(U_i, CP_i, x_r, y_r)
2:     RenderWindow W;
3:     Camera C_i;
4:     Viewport V_i;
5:     RenderedFrameBuffer R_i;
6:     C_i ← createCamera(U_i, CP_i);
7:     V_i ← addViewport(C_i, x_r, y_r);
8:     R_i ← renderOneFrame(W, V_i, C_i);
9:     return R_i
10: end procedure
```

The DRS provides the function for adding the multiple viewpoints to support the multi-view rendering. First, the DSP receive the service requests from the clients. These requests include several user information, U_i, such as user identification, selected game, which they want to play and initial or previous viewpoints in the 3D game space. Then, the DSP sends these information to the DRS to request for multi-view rendering. According to this request, the DRS provides the function for adding viewpoints, $\mathcal{CP}_i$. To perform this function on the DRS, we create the cameara C_i and viewport V_i objects to attach the viewport to the render window W_i. After the viewport was successfully added to the render window, the DRS performs the rendering procedure to generate an image on the framebuffer in a GPU. The pipeline of our algorithm for multi-view rendering with different resolutions is shown in **Algorithm 1**.

If EA_i and EV_i denote a i-th encoded audio and video in interactive game content respectively, then $\mathcal{ES}_k$, which has i encoded audio/visual gaming sequences is defined as:

$$\mathcal{ES}_k = \{(EA_1, EV_1), (EA_2, EV_2), ..., (EA_i, EV_i)\}$$

Therefore, the EQS performs the streaming $\mathcal{ES}_k$ to the clients for the cloud-based gaming services. In order to address the game's audio/visual output capturing, we develop the capturing module on the EQS in C++ and DirectShow SDK. We also develop the H.264 encoder for achieving low-delay video coding. On the other hand, the client side devices for our system support the H.264 decoding functionality. Also, the client is responsible for capturing the commands of the input controller such as keyboard and mouse, and sending them to the DSP via UDP.

IV. Experimental Results

This section presents the performance results of multi-view rendering with different resolutions and video encoding performed using our method. We have evaluated the performance of multiview rendering on a PC running Windows 7 operating system with Intel Core i7 2.93GHz CPU, 8GB memory and a ATI Radeon HD 5770. We used OGRE library based on DirectX as a graphics API and Microsoft HLSL for a shading language. The frames per second (FPS) is the number of frames per second that have been rendered by the DRS. High FPS results with smooth movements in the 3D scene.

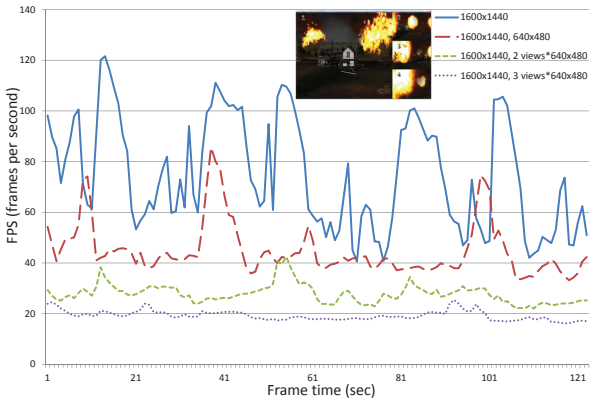

Fig. 3. The performance of multi-view rendering with different resolutions.

Our system rendered the single view (1600x1440) at 74.7 fps on average with one GPU. We measured the FPS at the DRS for multi-view rendering with 1600x1440 and 640x480 resolutions. In the case of multi-view rendering with a 1600x1440 view and two 640x480 views at the same time, we can get 27.8 fps on average with one GPU. Figure 3 shows the performance result of multi-view rendering according to the resolutions. In addition, in terms of scaling performance according to the number of CPU cores, our rendering performance using multi-core CPUs (8-core) achieves 4.2x speedup

 ISBN: 978-1-61208-229-5

over execution using single-core CPU. In order to analyze our video coding performance for the use of streaming game output to client devices, we have performed the experiments using H.264 codec. Our encoding system can encode in 25.6ms on average for eight views (interactive gaming videos) with 640x480 resolutions at 15 fps in parallel. Also, our encoding processing time is 23.1ms on average time per frame (1600x1440) as shown in Figure 4. The output of the first-person shooter game using the Unreal Development Kit (UDK) was captured and encoded.

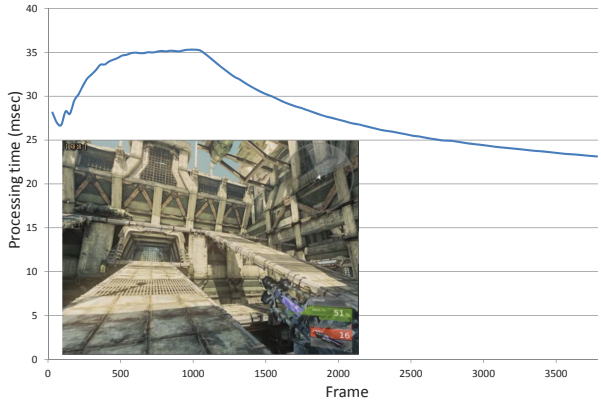

Fig. 4. The performance of video encoding for 1600x1440 at 30 fps.

Analysis: Our rendering system provides good performance scaling of multi-core CPUs for multi-view rendering with different resolutions and video encoding. And the multi-view rendering algorithm maps well to the current GPUs and we have evaluated its performance with different rendering resolutions. Compared to the prior parallel multi-view rendering method [5], our approach offers the advantage for multi-user games by supporting various viewpoints with *arbitrary focal positions*. Our algorithm can easily handle insertion and removal of viewpoints with different resolutions and can also take advantage of scalable and parallel processing using multi-core CPUs. Furthermore, it is relatively simple to combine the video encoding methods and optimizations in the cloud-based gaming platform. This makes it possible to develop a more flexible GPU-based framework for the video encoding methods like H.264/AVC.

Our approach has some limitations. First, we support the multi-view rendering for one multi-user game, since it is difficult to share the rendering resources in a GPU among different games. We believe that this can be resolved by using multi-GPUs. Secondly, our system performs directly rendering to the framebuffers on the server-side machines. However, in terms of efficient services in the cloud-based gaming, we should exploit the *off-screen rendering* approaches and *GPU virtualization* techniques [7].

V. Conclusion and Future Work

In this paper, we have presented the system architecture for the multiscreen-based gaming services and multi-view rendering with different resolutions. The performance speedup of our rendering system is achieved by utilizing both multicore CPUs and a GPU simultaneously without additional requirement for any special hardware. We found that the proposed system provide the multi-view rendering for different focal positions for each viewpoint with high visual quality. In addition, we demonstrate that the proposed rendering system could prove to be scalable in terms of parallel rendering. So, we believe that our rendering system will provide high-quality with good performance for the multiscreen-based gaming services.

There are many avenues for future work. It is possible to use new capabilities and optimizations to improve the performance of the video encoding especially H.264/AVC through the GPU-based implementation. Furthermore, we would like to develop algorithms for integrating the multi-view rendering with the video encoding in a GPU.

Acknowledgments

The game technology demo (Intel's smoke demo) in Figure 2 is courtesy of the Intel Corporation. This work was supported in part by the IT convergence R&D program of the Ministry of Knowledge Economy (MKE)/KEIT [10039202], *Development of SmartTV Device Collaborated Open Middleware and Remote User Interface Technology for N-Screen Service*.

References

[1] T. Karachristos, D. Apostolatos, and D. Metafas, "A real-time streaming games-on-demand system," in *Proceedings of the 3rd international conference on Digital Interactive Media in Entertainment and Arts*, ser. DIMEA '08. New York, NY, USA: ACM, 2008, pp. 51–56. [Online]. Available: http://doi.acm.org/10.1145/1413634.1413648

[2] S. Perlman. (July 2012) Onlive launches in belgium. [Online]. Available: http://blog.onlive.com/2012/07/30/onlive-launches-in-belgium/

[3] G. website. (2012) What is gaikai? [Online]. Available: http://www.gaikai.com/

[4] S.-S. Kim, K.-I. Kim, and J.-H. Won, "Multi-view rendering approach for cloud-based gaming services," in *The Third International Conference on Advances in Future Internet*, ser. AFIN 2011, 2011, pp. 102–107.

[5] W. Lages, C. Cordeiro, and D. Guedes, "A parallel multi-view rendering architecture," in *Proceedings of the 2008 XXI Brazilian Symposium on Computer Graphics and Image Processing*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 270–277. [Online]. Available: http://portal.acm.org/citation.cfm?id=1440461.1440894

[6] Y. T. A. S. F. Bellotti and A. Jurgelionis., "Games@large - a new platform for ubiquitous gaming and multimedia," in *Proceedings of the Broadband Europe Conference*, ser. BBEurope '06, 2006, pp. 11–14.

[7] P. Eisler. (2012, June) What to expect from geforce grid for cloud-based gaming. [Online]. Available: http://blogs.nvidia.com/2012/06/what-you-can-expect-from-geforce-grid/

[8] A. Jurgelionis, P. Fechteler, P. Eisert, F. Bellotti, H. David, J. P. Laulajainen, R. Carmichael, V. Poulopoulos, A. Laikari, P. Perälä, A. De Gloria, and C. Bouras, "Platform for distributed 3d gaming," *Int. J. Comput. Games Technol.*, vol. 2009, pp. 1:1–1:15, January 2009. [Online]. Available: http://dx.doi.org/10.1155/2009/231863

[9] J. Andrews. (June 2009) Designing the framework of a parallel game engine. [Online]. Available: http://software.intel.com/en-us/articles/designing-the-framework-of-a-parallel-game-engine/

[10] S. Eilemann, M. Makhinya, and R. Pajarola, "Equalizer: A scalable parallel rendering framework," *IEEE Transactions on Visualization and Computer Graphics*, vol. 15, pp. 436–452, May 2009. [Online]. Available: http://portal.acm.org/citation.cfm?id=1515609.1515684

[11] OTOY. (August 2012) Otoy website. [Online]. Available: http://www.otoy.com/

[12] R. 3550, *RTP: A Transport Protocol for Real-Time Applications*.

 ISBN: 978-1-61208-229-5