Multi-view Rendering Approach for Cloud-based Gaming Services

Sung-Soo Kim, Kyoung-Ill Kim and Jongho Won
Electronics and Telecommunications Research Institute (ETRI)
Daejeon, South Korea
{*sungsoo, kki, jhwon*}*@etri.re.kr*

***Abstract*—In order to render hundreds or thousands of views for multi-user games on a cloud-based gaming at interactive rates, we need a solution which is both scalable and efficient. We present a new cloud-based gaming service system which supports multiple viewpoint rendering for visualizing a 3D game scene dataset at the same time for the multi-user games. Our multi-view rendering algorithm maps well to the current graphics processing units (GPUs) and we have evaluated its performance on two different GPUs with different rendering resolution. The experimental results demonstrate the multi-view rendering method can be successfully applied to the multi-user games.**

***Keywords*-gaming on demand, multi-view rendering, video encoding, video streaming, cloud computing**

I. INTRODUCTION

Cloud computing is a general term for complete services delivered over the networks using self-service end user portals and flexible business model that incorporates hardware, software, and services into a single revenue stream. 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 are *cloud-based gaming service systems*, also called *Gaming on Demand* (GoD) [1]. They are networked media platforms that offer web-based services allowing game play on smaller end-devices like low-cost PCs or set top boxes without requiring the games to be installed locally.

The ultimate goal of the cloud-based gaming services is to provide pervasive game access on devices such as set top boxes and mobile devices that typically do not have a full set of technical requirements to run high-quality video games. This goal can be achieved by processing the game execution logic such as rendering, audio, artificial intelligence (AI) and physics, on at remote high-end servers being streamed as an interactive video over a network to be played on lightweight clients. Recent work in this area has been focused on video encoding and streaming techniques to reduce the latency in games. Most of the earlier systems were serial in nature and designed for a single core or processor in terms of 3D rendering.

The recent trend in computer architecture has been toward developing parallel commodity processors, including multi-core CPUs and many-core GPUs. 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 [2] and parallel rendering algorithms [3][4] have been proposed for commodity parallel processors.

In this paper, we propose a new cloud-based gaming service system to support the multi-view rendering based on multi-threaded game engine [2] for the multi-user games. To provide convincing streaming-based GoD services, we describe the key requirements of cloud-based service systems as follows:

- *User responsiveness*: Latency is defined as the time between a player's action and the time the actual resulting game output on the player's screen. Since computer games are highly interactive, extremely low latency has to be achieved. The interaction delay in the games should be kept below 80ms in order to guarantee suitable user responsiveness [5].
- *High-quality video*: In order to provide a high-quality video (above 720p) interactively, data shall be reduced as much as possible but keeping quality. However, video encoding is computationally quite demanding.
- *Quality of services*: In the case of network congestion, the network problems like increased latency, jitter, and packet losses distribute evenly on all competing traffic. However, the quality can be enhanced using quality of service (QoS) technologies to giver higher priority to game traffic in the network bottlenecks [6].
- *Operating costs*: Since the servers of cloud-based gaming service system have high-performance CPUs and GPUs, the operating costs for the servers are quite expensive. So, it is necessary to develop the optimization technologies to minimize power consumption and network bandwidth [7].

Our contributions: We present a novel system architecture for the cloud-based 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. Our algorithm can easily handle insertion and removal of viewpoints and can also take advantage of scalable and parallel processing using multi-core CPUs. In addition, our approach give the benefits in terms of *arbitrary focal positions* for viewpoints and better rendering quality over prior parallel multi-view rendering methods [8].

The rest of the paper is organized as follows. We briefly

 ISBN: 978-1-61208-148-9

survey previous work on Gaming on Demand (GoD), parallel rendering and video encoding for the cloud-based gaming services in Section II. Section III describes the proposed the system architecture and the core systems in our system. We explain implementation details of our multi-view rendering algorithm and describe the performance result in Section IV. In Section V, we compare our system and algorithm with prior GPU-based algorithms and highlight some of the benefits. Finally, we discuss future work and conclude in Section VI.

II. Related work

In this section, we give a brief overview of related work on cloud-based gaming technology and parallel rendering algorithms. We also highlight many technical characteristics of cloud-based gaming services, parallel rendering and video encoding.

Gaming on Demand (GoD): There are a number of commercial Gaming on Demand systems that have been presented to the market [6]. OnLive is a gaming-on-demand entertainment platform, announced at the Game Developers Conference (GDC) in 2009 [7]. Gaikai launched GoD beta service; *Gaikai beta*, based on a cloud-based gaming technology that allows users to play major PC and console games [9]. The clients of their service can display audio/video (AV) game streams, which were streamed from the cloud, by using previously installed plug-ins such as Java or Adobe Flash on client devices. Even though both are cloud base gaming, OnLive and Gaikai have different goals in mind. OnLive sells full games, provides demos, brag clips, and being able to watch other players play games (Arena) while Gaikai advertises games via a webpage as demos [7].

Visual effects rendering based on *global illumination* that commonly requires extensive hardware and processing time. However, the OTOY can provide visual effects rendering in real-time using the cloud; *Fusion Render Cloud* (FRC), through the power of *server side rendering* [10]. However, there is very little detailed technical information publicly available about these commercial systems.

The *Games@Large* (G@L) framework enables commercial video game streaming from a local server to remote end devices in local area networks (LANs) [11]. This system and streaming protocols are developed and adapted for highly interactive video games [1] [12].

There are two major approaches for the game streaming. One is *3D graphics streaming* approach which exploited for streaming the game's output is to directly transmit the graphics commands to the client device and render the image on the client device [3]. The other approach is *video streaming* that the server renders the game graphics scene, the framebuffer is captured, eventually downsampled to match the target resolution, and the current image is encoded using standard video codes such as MPEG-2, MPEG-4 and H.264 [13][14]. The video streaming is intended for thin-client devices lacking hardware accelerated rendering capabilities [15]. In our research, we exploit a video streaming method since our system should support the thin-client devices.

Parallel rendering: Much of the recent work in the area of parallel rendering has focused on using networked clusters of commodity PCs. Such systems can generally drive a tiled display using a commodity local network as well. There are three major approaches according to a sorting classification of parallel rendering such as *sort-first*, *sort-middle* and *sort-last rendering* [16].

Also, there have been various research efforts to multi-view rendering and scalable rendering [4]. However, those methods cannot be directly employed for multi-view rendering for multi-user games since those methods usually focus on multipipe display systems, workstations with multiple monitors, walls build out of multiple screens or projectors as well as immersive environments.

A parallel multi-view rendering architecture in a cluster of GPUs has been proposed in [8]. 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. This is not suitable for common mutli-user game applications. Moreover, it is difficult to apply this method to a *high visual quality* (photo-realistic) games since they used a simple phong shader for lighting and shading.

In this paper, we exploit a parallel game engine for improving the multi-view rendering performance as well as the visual realism in the games as shown in Fig. 1.

Video encoding: Many techniques have been proposed to accelerate the performance of video encoding algorithms. In H.264/AVC encoders, *macroblock partitioning* and *motion vector calculation* are computationally very demanding. An acceleration based on render context information has been developed, which allows the direct calculation of motion vectors, similar to [13]. The parallel model of the encoder using multiprocessor platforms has been introduced in order to improve the encoding performance in [17].

OnLive introduced *interactive video compression* method designed for video games. In order to achieve high performance encoding, they developed two dedicated compression hardware for video encoding; *optimized compressor* based on *human perception* and *live compressor* similar to conventional compressor [7].

Current GPUs are regards as high-throughput processors, which have a theoretical peak performance of a few Tera-Flops. In order to accelerate the performance of the motion estimation, fast motion estimation implementation using CUDA on GPU has been proposed in [18]. Recently, OTOY introduced new video encoding method, so-called *ORBX*. ORBX has been designed from the ground up to take advantage of OpenCL based GPU servers (FRC). ORBX encodes video entirely on the GPU, with more than 30-100x the scaling of H.264 encoding solutions requiring either a CPU or specialized encoding ASIC [10]. Unfortunately, the technical information of the ORBX encoding method is not publicly available.

 ISBN: 978-1-61208-148-9

Fig. 1. The result of our multi-view rendering algorithm (eight views with 640x480 resolution for each view.)

III. System Architecture

In this section, we describe the proposed system architecture for the cloud-based gaming services. Our architecture consists of three major systems such as distributed service platform (DSP), distributed rendering system (DRS) and encoding/QoS/streaming system (EQS) as shown in Fig. 2.

A. System Overview

The Distributed Service Platform (DSP) is responsible for launching the game processes on the game execution nodes or rendering job on the Distributed Rendering System (DRS) after client-side invocation, monitoring its performance, allocating computing resources and managing user information. And, the DSP is responsible for processing 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 Encoding/QoS/Streaming System (EQS).

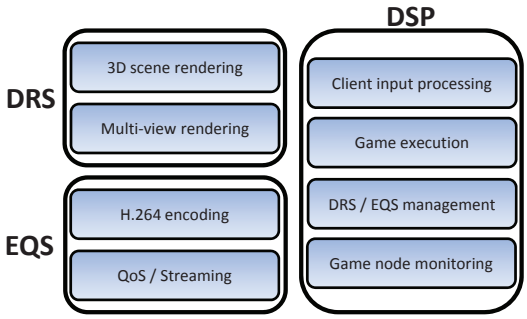

Fig. 2. Our system architecture: *DSP-Distributed Service Platform, DRS-Distributed Rendering System, EQS-Encoding, QoS and Streaming System*

The DRS is responsible for rendering a 3D scene and *multi-view rendering* for multi-user games. To improve 3D rendering performance in games, we utilize the multi-threaded game engine [2] that is designed to scale to as many processors as are available within a platform.

The EQS is responsible for audio/video encoding and streaming the interactive game content to the clients. In order to implement the visual capturing for the games, we utilize the DirectShow SDK. And 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 [19][20].

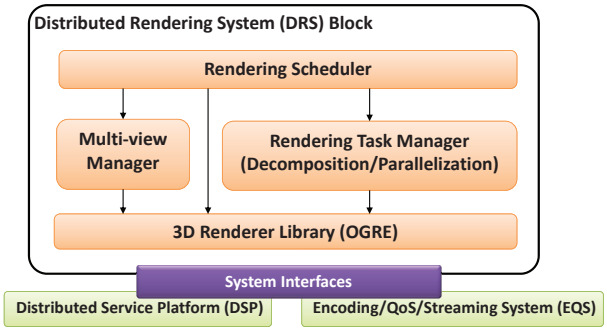

Fig. 3. The DRS block architecture

B. Distributed Rendering System

The Distributed Rendering System (DRS) consists of four major block components such as *rendering scheduler*, *multi-view manager*, *rendering task manager* and *renderer library* as shown in Fig. 3. 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 improvements for the game applications is the use of per-thread task queues. This eliminates the synchronization checkpoint when one shared task queue is used.

 ISBN: 978-1-61208-148-9

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) [21], which is highly optimized scheduler for Windows, Mac OS X, and Linux. 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) [22], which performs a 3D scene graph management and rendering. In order to provide the cloud-based gaming service, the DRS should have common system interfaces to the DSP and the EQS.

C. Multi-view Rendering

In the case of multi-user games, multiple viewpoints are needed if we want to support several users visualizing a given 3D scene at the same time. However, rendering multiple views using the standard graphics pipeline is a challenging problem.

In order to provide the interactive multi-view rendering results for the cloud-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.

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 $\mathcal{CP}_i$ according to the requests of the clients.

Algorithm 1 Viewpoint addition algorithm.

```
1: procedure ADDVIEW(U_i, CP_i)
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);
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 is shown in **Algorithm 1**. Another function for multi-view rendering is deletion function for viewpoints in the multi-user games. This function can be easily implemented similar to the viewpoint addition algorithm.

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. Before the EQS performs the H.264 encoding, a color space conversion from RGB to YUV (convertRGB2YUV function in **Algorithm 2**) takes place on the captured frames F_i. We utilized the 4:2:0 method for the YUV sampling to achieve the reduction of storage data. We also capture and encode the audio data for the games to transmit the interactive game content to the clients. In our work, HE-AACv2 is utilized for audio streaming. And then, the EQS transmits the encoded AV content for the game to the client via RTP/RTCP [20]. The details of our video encoding and streaming algorithm for interactive gaming content is shown in **Algorithm 2**.

Algorithm 2 Video encoding and streaming algorithm.

```
1: procedure ENCODINGANDSTREAMING(U_i)
2:     YUVImage Y_i;
3:     FrameCapture F_i;
4:     FrameNumber f;
5:     EncodedAudio EA_i;
6:     EncodedVideo EV_i;
7:     F_i ← captureRenderedFrameBuffer(U_i, f);
8:     while F_i ≠ NULL do
9:         Y_i ← convertRGB2YUV(F_i);
10:        EV_i ← encodeFrame(Y_i);
11:        EA_i ← captureAndencodeAudio(U_i, f);
12:        ES_i ← transmitAVstream(EA_i, EV_i, f);
13:        F_i ← captureRenderedFrameBuffer(U_i,f);
14:    end while
15: end procedure
```

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. Implementation and Performance

In this section, we describe the implementation of our system and highlight the performance of our multi-view algorithm.

Implementation: We have implemented our multi-view algorithm on two different commodity GPUs: a NVIDIA GTX 480, a NVIDIA Quadro 4000. In order to test the performance

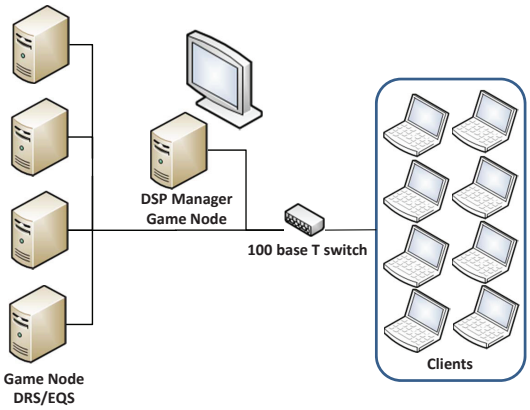

Fig. 4. Performance test setup: *This figure shows the system configuration; five workstations for servers, eight laptops as thin-clients, for the performance testing.*

of our system, we used five workstations (Intel Core i7, 8G RAM) which were connected to a 100 Mbps switch via a wired Ethernet connection. Also, the eight laptops as thin-clients (Intel 2GHz, 1G RAM) were connected to the 100Mbit LAN. These laptops are capable of H.264 decoding and displaying the game videostream as shown in Fig. 4.

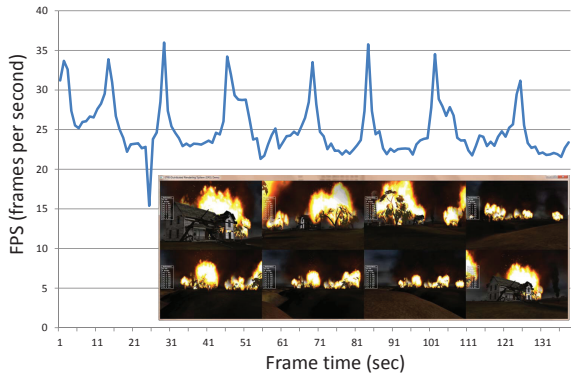

Fig. 5. Performance of multi-view rendering: *This figure shows the frames per second (fps); 25.4 on average, for multi-view rendering (640x480 resolution for each view) on a NVIDIA Quadro 4000.*

Performance: First we evaluate the performance of multi-view rendering on a PC running Windows 7 operating system with Intel Core i7 2.93GHz CPU, 8GB memory and a NVIDIA Quadro 4000. We used OGRE library based on DirectX as a graphics API and Microsoft HLSL for a shading language.

The frames for 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. Our system rendered 8 views at 25.4 fps on average with one GPU. We measured the FPS every second at the DRS for rendering at 640x480 resolution for each view. Fig. 5 shows the performance result of multi-view rendering.

We utilized the parallel game engine; *Intel Smoke* [2], and we adopted our multi-view rendering algorithm. Then we ran it on a 8 core system with a NVIDIA GTX 480 to measure the scalability of our rendering system as shown in Fig. 6.

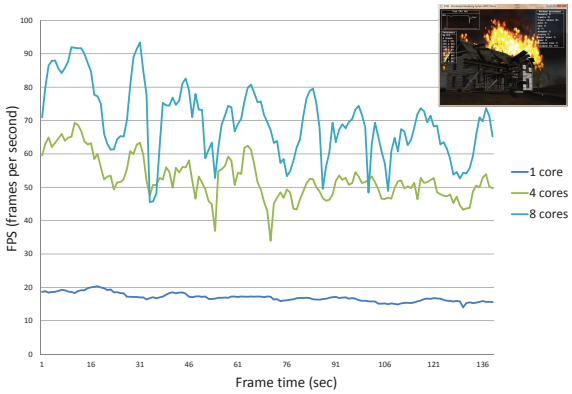

Fig. 6. Scaling performance of the DRS according to the number of CPU cores: *This figure shows the scalability of our parallel (multi-threaded) rendering (1600x1200 resolution) on a Intel i7 8-core (quad-core with hyperthreading) with a NVIDIA GTX 480. Average FPS - 1 core: 16.9, 4 cores: 52.6, 8 cores: 69.7*

In terms of the performance of our encoding system, our system can encode in 25.6ms on average for eight views (interactive gaming videos) with 640x480 resolution in parallel and 24.9ms for a 1600x1200 video. **Table I** shows the supported technical features of AV encoding in the proposed system.

TABLE I
THE SUPPORTED FEATURES OF AUDIO/VIDEO ENCODING IN OUR SYSTEM.

Supported features	Audio	Video
Codec	HE-AACv2	MPEG-4, H.264
Resolution	-	320x240 - 1600x1200
Bitrate	16Kbps - 64Kbps	384Kbps - 5Mbps
Frame Rate	-	5 - 30fps
Sampling Rate	22.05KHz - 48KHz	-
Channel	Mono, Stereo	-

V. Analysis

In this section, we evaluate the performance of our system in terms of rendering and encoding functionalities and highlight some of the benefits.

Analysis: Our rendering system provides good performance scaling of multi-core CPUs for multi-view rendering. And the multi-view rendering algorithm maps well to the current GPUs and we have evaluated its performance on two different GPUs with different rendering resolution. Furthermore, it is relatively simple to combine the video encoding methods and optimizations in the streaming-based gaming service framework. This makes it possible to develop a more flexible

 ISBN: 978-1-61208-148-9

GPU-based framework for the video encoding methods like H.264/AVC or ORBX which is GPU-based encoding schemes.

Limitations: 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.

VI. Conclusion and Future Work

In this paper, we have presented a system architecture for the cloud-based gaming service and multi-view rendering. Our rendering system greatly improves utilization of hardware resources present in the system, allowing to utilize both multi-core CPUs and a GPU simultaneously.

We found that the proposed system provide the multi-view rendering for different focal positions for each viewpoint with high visual quality. Moreover, our approach is flexible and maps well to current GPUs in terms of shared resources such as textures and shaders for rendering. 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 cloud-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 Fig. 1 is courtesy of the Intel Corporation. The authors appreciate valuable comments for the development of the multi-view rendering algorithm from Dr. Choong-Gyu Lim. This work was supported in part by the SW computing R&D program of MKE/KEIT [10035184], *Game Service Technology Based on Realtime Streaming*.

References

[1] 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

[2] J. Andrews. (2009, June) 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/

[3] G. Humphreys, M. Houston, R. Ng, R. Frank, S. Ahern, P. D. Kirchner, and J. T. Klosowski, "Chromium: a stream-processing framework for interactive rendering on clusters," *ACM Trans. Graph.*, vol. 21, pp. 693–702, July 2002. [Online]. Available: http://doi.acm.org/10.1145/566654.566639

[4] 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

[5] M. Claypool and K. Claypool, "Latency and player actions in online games," *Commun. ACM*, vol. 49, pp. 40–45, November 2006. [Online]. Available: http://doi.acm.org/10.1145/1167838.1167860

[6] A. Jurgelionis, F. Bellotti, A. D. Gloria, J.-P. Laulajainen, P. Fechteler, P. Eisert, and H. David, "Testing cross-platform streaming of video games over wired and wireless lans," in *Proceedings of the 2010 IEEE 24th International Conference on Advanced Information Networking and Applications Workshops*, ser. WAINA '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 1053–1058. [Online]. Available: http://dx.doi.org/10.1109/WAINA.2010.186

[7] S. Perlman. (2009, Dec.) The process of invention: Onlive video game service. [Online]. Available: http://www.youtube.com/watch?v=2FtJzct8UK0

[8] 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

[9] D. Perry. (2010, Nov.) What is gaikai? [Online]. Available: http://http://www.gaikai.com/

[10] J. Urbach. (2009, Dec.) Otoy technology. [Online]. Available: http://www.otoy.com/

[11] F. B. A. J. Y. Tzruya, A. Shani, "Games@large - a new platform for ubiquitous gaming and multimedia," in *Proceedings of the Broadband Europe Conference*, ser. BBEurope '06, 2006, pp. 11–14.

[12] A. S. Y. T. A. L. P. E. Itay Nave, Haggai David and P. Fechteler, "Games@large graphics streaming architecture," in *Proceedings of the 12th Annual IEEE International Symposium on Consumer Electronics*, ser. ISCE '08, 2008, pp. 1–4.

[13] L. Cheng, A. Bhushan, R. Pajarola, and M. E. Zarki, "Realtime 3d graphics streaming using mpeg-4," in *In Proc. IEEE/ACM Wksp. on Broadband Wireless Services and Appl*, ser. IEEE/ACM BroadWise 04, 2004, pp. 1–16.

[14] 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

[15] D. De Winter, P. Simoens, L. Deboosere, F. De Turck, J. Moreau, B. Dhoedt, and P. Demeester, "A hybrid thin-client protocol for multimedia streaming and interactive gaming applications," in *Proceedings of the 2006 international workshop on Network and operating systems support for digital audio and video*, ser. NOSSDAV '06. New York, NY, USA: ACM, 2006, pp. 15:1–15:6. [Online]. Available: http://doi.acm.org/10.1145/1378191.1378210

[16] S. Molnar, M. Cox, D. Ellsworth, and H. Fuchs, "A sorting classification of parallel rendering," in *ACM SIGGRAPH ASIA 2008 courses*, ser. SIGGRAPH Asia '08. New York, NY, USA: ACM, 2008, pp. 35:1–35:11. [Online]. Available: http://doi.acm.org/10.1145/1508044.1508079

[17] H. K. Zrida, A. Jemai, A. C. Ammari, and M. Abid, "High level h.264/avc video encoder parallelization for multiprocessor implementation," in *Proceedings of the Conference on Design, Automation and Test in Europe*, ser. DATE '09. 3001 Leuven, Belgium, Belgium: European Design and Automation Association, 2009, pp. 940–945. [Online]. Available: http://portal.acm.org/citation.cfm?id=1874620.1874851

[18] A. Colic, H. Kalva, and B. Furht, "Exploring nvidia-cuda for video coding," in *Proceedings of the first annual ACM SIGMM conference on Multimedia systems*, ser. MMSys '10. New York, NY, USA: ACM, 2010, pp. 13–22. [Online]. Available: http://doi.acm.org/10.1145/1730836.1730839

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

[20] R. 3984, *RTP Payload Format for H.264 Video*.

[21] Intel. Intel threading building blocks. [Online]. Available: http://www.threadingbuildingblocks.org/

[22] OGRE. Object-oriented graphics rendering engine. [Online]. Available: http://www.ogre3d.org

 ISBN: 978-1-61208-148-9