A Survey of Interactive Remote Rendering Systems

Article *in* ACM Computing Surveys · May 2015

DOI: 10.1145/2719921

CITATIONS
26

READS
1,369

2 authors, including:

Shu Shi
AT&T
25 PUBLICATIONS **286** CITATIONS

SEE PROFILE

A Survey of Interactive Remote Rendering Systems

Shu Shi, AT&T Labs Research
Cheng-Hsin Hsu, National Tsing-Hua University

Remote rendering means rendering 3D graphics on a computing device and displaying the results on another computing device connected through a network. The concept was originally developed for sharing computing resources remotely. It has been receiving increasing attention from researchers in both academia and industry in recent years due to the proliferation of cloud computing and mobile devices. In this article, we survey the interactive remote rendering systems proposed in the literature, analyze how to improve the state-of-the-art, and summarize the related technologies. The readers of this article will understand the history of remote rendering systems and obtain some inspirations of the future research directions in this area.

Categories and Subject Descriptors: H.5.1 [**Multimedia Information System**]: Video

General Terms: Design, Algorithms, Performance

Additional Key Words and Phrases: Cloud computing, cloud games, cloud rendering, distributed rendering, video streaming, QoS, QoE

1. INTRODUCTION

1.1. What is Remote Rendering?

A remote rendering system runs rendering applications on one computing device, which is referred to as rendering server or server, and displays the rendering results on another network-connected computing device, which is referred to as client. An interactive remote rendering system is capable of accepting user controls through the input devices on a client to interact with the rendering applications running on a server (Figure 1). In this survey, we focus on the interactive remote rendering systems that are designed specifically for 3D graphics. Unless otherwise noted, the short term "remote rendering" is used herein to indicate "interactive remote rendering for 3D graphics".

The concept of remote rendering appeared early when PCs were not powerful enough to process 3D graphics rendering. Initial research in this area investigated the potential of sharing a dedicated graphics workstation over networks to provide remote rendering services [Ohazama 1999]. In recent years, the wide deployment of high-speed wireless networks and the rising prosperity of mobile cloud computing have brought more research interests into remote rendering. Beermann and Humphreys [2003] predicted that graphics rendering would become a remote service and this prediction came true with the emergence of cloud gaming [Ross 2009].

Compared to the conventional approach that performs rendering locally, remote rendering has several advantages. First, it can provide rich rendering experiences to "thin" clients (e.g., mobile devices) with limited computing resources (i.e., GPU, mem-

Author's addresses: S. Shu, AT&T Labs Research, 1 AT&T Way, Bedminster, NJ 07921, USA; email: shushi@research.att.com. C. Hsu, Department of Computer Science, National Tsing Hua University, No. 101 Sec. 2 Kuang-Fu Road Hsin-Chu City, Taiwan; email: chsu@cs.nthu.edu.tw.

DOI 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

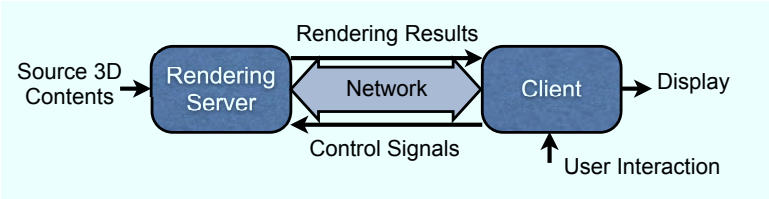


Fig. 1. Basic remote rendering framework

ory, power, etc.). Second, the computing resources on a rendering server can be efficiently shared by multiple clients. Third, remote rendering is a simple but effective cross-platform solution. After developing the client programs for different platforms, all applications need only be developed for the server and the same rendering experience will be achieved on all client platforms. Last but not the least, a remote rendering system can be designed to prevent the source contents from leaking to malicious users by streaming only rendering results to clients. In cloud gaming, for example, the currently deployed remote rendering solutions can prevent pirating concerns because game contents are never sent to gamers.

1.2. Key Challenges

Different remote rendering systems have been designed to target various problems and challenges. In this survey, we give a broad overview of the different types of remote rendering designs that have been proposed over the years. Our discussion focus on two fundamental problems: how to reduce interaction latency and how to transmit data from server to client effectively and efficiently.

We define "interaction latency" as the time from the generation of a user interaction request until the appearance of the first updated frame on the client screen. As an intrinsic problem of all network distributed systems, latency plays an important role in evaluating system performance and QoE (Quality of Experience) [Wang and Dey 2009]. Long interaction latency can significantly impair user experience. For example, 100 ms is the largest tolerable latency for an intensive first person shooting game [Beigbeder et al. 2004]. Therefore, the minimization of interaction latency is a major concern and we will review all related latency reduction techniques that can be applied to remote rendering.

Data transmission is always the core module of any remote rendering system. In most cases, a data transmission module design should consider all sorts of constraints, such as limited bandwidth, various channel conditions, and real-time requirements. More importantly, the success of data streaming has a direct impact on the overall system performance. In this article, we mainly discuss three aspects of data transmission: compression, streaming protocols, and QoS/QoE evaluation.

1.3. Related Work

Thin Clients and Remote Sharing. Thin client systems (e.g., SLIM [Schmidt et al. 1999], THiNC [Baratto et al. 2005], etc.) and remote desktop sharing systems (e.g., VNC [Richardson et al. 1998], RDP [Cumberland et al. 1999], etc.) allow users to access applications remotely and share computing resources. Users on the client side can interact with the applications running on the server side. Nevertheless, there are key differences between those systems and the remote rendering systems surveyed in this article.

First, most thin client and remote sharing systems appeared before 3D graphics rendering became popular. Early systems were only designed to support sharing desktop

elements and rendering 2D graphics. Only the recent approaches started to add support for 3D graphics (e.g., THiNC [Baratto et al. 2005], TurboVNC [Commander 2007], etc.). Second, the main research goal of sharing 2D applications is to design protocols that can efficiently update regional changes on the screen. This is because 2D rendering is considered lightweight, such that it can either happen on a server or a client. However, this assumption does not apply to 3D graphics due to the significantly increased rendering complexity. Last, 3D applications like video games usually refresh the whole screen at a much higher rate. Such applications also require different compression and streaming methods that are different from those used in conventional 2D thin client systems.

In this article, we will not include further detail on any specific thin client or remote sharing design (the systems that support 3D graphics will be briefly introduced in Section 2 for comparison with other remote rendering approaches). Therefore, readers are referred to other surveys [Yang et al. 2002; Lai and Nieh 2006] for information on those systems.

Distributed Graphics and Cluster Rendering. Another related work is distributed graphics. When the rendering computation is too complex for one server to execute, or the display system is a wall of multiple screens [Staadt et al. 2003], such as CAVE [DeFanti et al. 2011] and SAGE [Renambot et al. 2004], the rendering workload is divided and distributed to multiple servers for parallel executions and the final rendering images of these servers are stitched together. Such systems share similar network distributed architecture to remote rendering systems but solve a totally different research problem. Distributed rendering mainly focuses on how to divide computation for parallel rendering in a cluster, while remote rendering focuses on how server and client interact with each other. Examples of distributed rendering efforts include WireGL [Humphreys et al. 2001], Chromium [Humphreys et al. 2002], OpenGL Multipipe SDK [Bhaniramka et al. 2005], ParaView [Cedilnik et al. 2006], and Equalizer [Eilemann et al. 2009]. Although we are not going to cover distributed graphics or cluster rendering in more depth in this survey, we believe that the future remote rendering systems will adopt distributed graphics to improve the rendering performance for rendering-intensive applications.

1.4. Overview of this Survey

In Section 2, we survey some background information on rendering systems and review the remote rendering designs proposed for various applications. In Section 3, we use a specific example of the state-of-the-art cloud gaming systems to explain in detail the challenges of remote rendering. Next, we survey different techniques that can address two key challenges in Section 4 and Section 5, respectively. Section 6 shares our vision of the future research directions in this area, and Section 7 concludes the survey.

2. REMOTE RENDERING SYSTEMS IN THE LITERATURE

In this section, before we jump into the details of how remote rendering systems are built, we first review the rendering framework of X Window systems and how to modify X to a general-purpose remote rendering system. Next, we list the specialized remote rendering systems that have been designed for various applications. Last, we summarize with a novel classification method based on the type of data transmitted between server and client.

2.1. Rendering Basics

We take Unix/Linux as example to review some basic knowledge of rendering. The rendering system of Unix/Linux is managed by X [Scheifler and Gettys 1986]. Figure

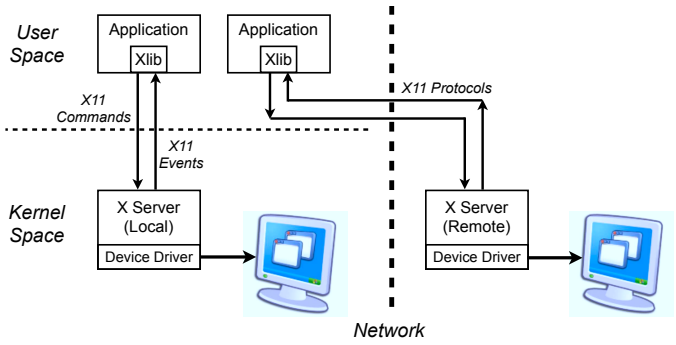


Fig. 2. X Window server-client model

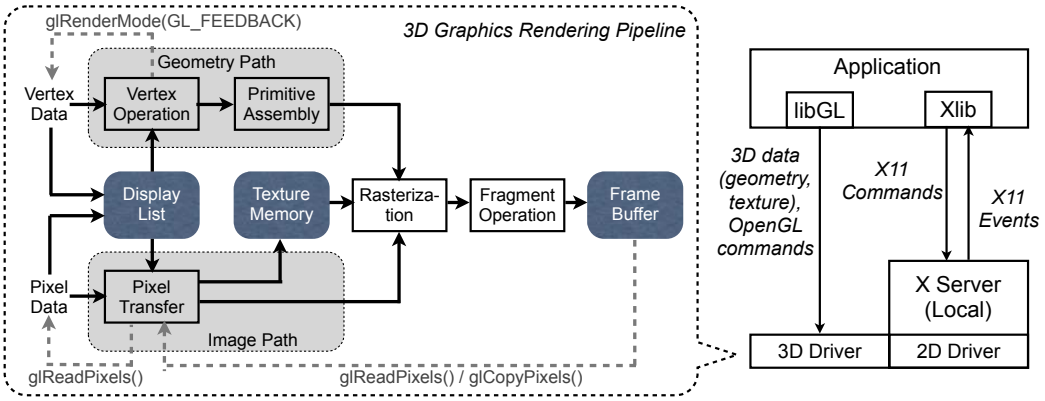


Fig. 3. OpenGL and X

2 shows an illustration of this server-client based window system. Each X application runs as a client while the operating system hosts an X server that directly connects to the hardware driver. An X application (X client) sends low level XLib operation commands to an X server using X11 protocols and the X server executes the rendering operations and displays the final image. X is network-transparent, meaning that an X application can connect to a remote X server by sending XLib operations over networks.

X was originally designed for 2D graphics. The evolution of 3D graphics rendering and OpenGL [Woo et al. 1999] makes the system more complex. The X server can directly interpret and execute XLib operations for 2D graphics, but not for 3D. The 3D rendering operations comprise the commands for data description (including both geometry and texture) and 3D drawing. The rendering process requires the support of a 3D graphics driver (Figure 3), which can consist of either graphics hardware or software libraries. The number and size of 3D operations for rendering one frame depend on the complexity of 3D data. For a complex scene with millions of polygons, rendering even one single frame requires passing a huge amount of data to the 3D graphics driver. In order to reduce the overhead of passing 3D rendering operations, X allows OpenGL programs to bypass the X server and send all 3D operations, including both drawing and data commands, to the OpenGL driver directly.

2.2. General Purpose Solutions

Apparently, the network-transparent nature of X makes it a good platform to implement remote rendering. GLX was designed as the OpenGL extension for X[1]. It comprises an API that provides the OpenGL functions to X applications, an extension of the X protocol that allows the X client to send 3D rendering operations to the X server, and an extension to the X server that can pass 3D rendering operations to the underlying OpenGL driver. GLX allows 3D graphics rendering applications to display on a remote X server like all other 2D applications (Figure 4(a)).

The problem with this command streaming approach is that 3D rendering is actually performed on the client side. All geometry models, texture maps, and operation commands are streamed to the client before rendering can begin. For applications that perform complex graphics rendering, streaming all graphical data to the client may require excessive network bandwidth.

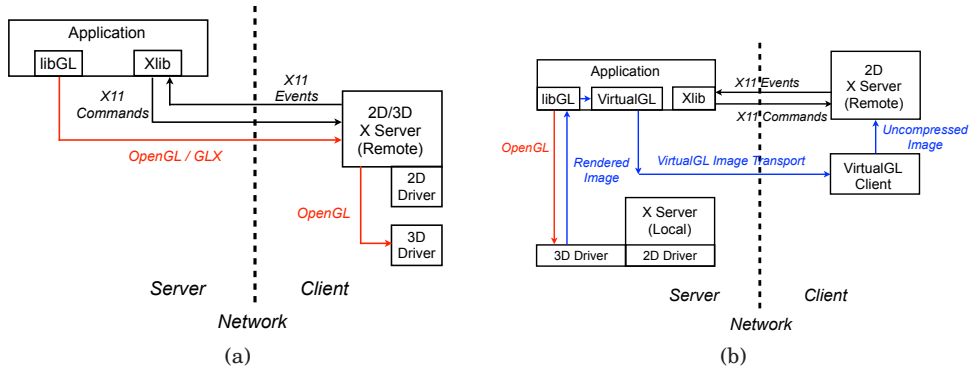


Fig. 4. Two general-purpose remote rendering solutions based on the X system: (a) GLX; (b) VirtualGL [Commander 2007]

A different approach was introduced by Stegmaier et al. [2002; 2003]. Only 2D operations are sent to the X server (on the client side) while 3D operations are rendered on the server side. The 3D rendering image is captured and streamed to the client, where it is merged with 2D rendering results. This image streaming approach led to an open source project: VirtualGL [Commander 2007]. Figure 4(b) shows an illustration of the VirtualGL framework.

The bandwidth usage for the image streaming approach is predictable and bounded. This is because the streaming bandwidth is proportional to the display resolution and refresh rate. In the worst case when very complex 3D scenes are rendered, the image streaming approach may require much less network bandwidth than the command streaming approach mentioned above. Moreover, using the image streaming approach requires no special graphical hardware on the client side.

Both GLX and VirtualGL are two examples of general-purpose remote rendering solutions. They can support any OpenGL applications without extra efforts from developers to modify the source code for remote rendering. There have also been several proprietary solutions belonging to this category, including SGI's OpenGL Vizserver [Ohazama 1999], HP's Remote Graphics Software (RGS) [HP 2006], and Mercury International Technology's ThinAnywhere [Technology 2007].

[1]WGL [Paik] and CGL [Apple 2009] are similar interfaces provided for Windows and Mac OS X, respectively.

The modern thin-client systems that support 3D graphics can also be considered to be general-purpose solutions. Both command streaming and image streaming approaches have been applied in those systems. For example, THiNC [Baratto et al. 2005] took a command streaming approach. The server creates a pseudo video card driver to collect all low-level drawing operations, including both OpenGL and 2D drawing commands. On the other hand, TurboVNC [Commander 2007] was an image streaming approach. The server maintains a software frame buffer to save all graphical outputs and sends the updates to the frame buffer to the client with an open protocol, known as Remote Frame Buffer (RFB).

Although the general-purpose solution has the advantage of being compatible with any 3D application, it also loses the flexibility of customized optimization. Most remote rendering systems in the literature have not taken the general-purpose approach, but instead were designed and optimized for specific applications.

2.3. Specialized Solutions

We classify all specialized remote rendering solutions based on two dimensions (Figure 5). The 3D model data are either dynamic or static, depending on whether they are constantly updated during executions. The user interactions are either restricted or unrestricted, depending on whether users are totally free to change the camera position, orientation, and other actions. Remote rendering solutions in different categories impose different design considerations. Detailed descriptions of these four categories are given below.

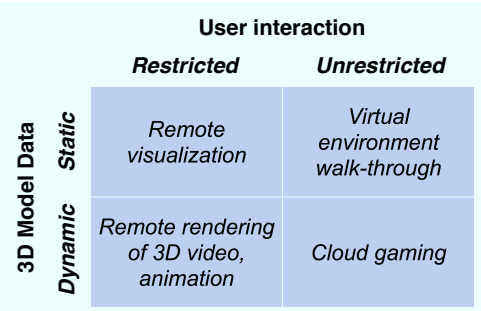


Fig. 5. Four categories of specialized remote rendering systems and sample applications

Static Model & Restricted Interaction. The systems belonging to this category render static 3D models and support restricted user interactions. "Restricted user interaction" means a user can only interact with the 3D data in a few limited ways. For example, a user can view a 3D model only from a few pre-defined viewpoints or change the viewpoint within a limited range. Most remote visualization systems belong to this category. Visualization systems are widely demanded to present various types of data, such as medical images, scientific data, industry designs, and artworks. Many visualization systems perform rendering remotely due to the complexity of the source data or the limitations of presentation devices.

Bethel [2000] built a visualization system for large-volume scientific data visualization. Since the volume date is too large to be transmitted over a network, it is stored on a server and only the rendering results are transmitted to client devices. Ma and Camp [2000] addressed the issues of rendering time-varying volume data in parallel on multi-core servers. Engel et al. [2000] developed a system to visualize large-scale medical volume data sets by converting the volume data to 2D slices. Prohaska et

al. [2004] designed an interactive Micro-CT scan exploration system to retrieve only sub-volume data from the storage server and render a much smaller data set locally. Ma and colleagues [Ma 2010; Tikhonova et al. 2010] adopted explorable images in remote visualization systems to allow users to manipulate rendering properties (e.g., color and opacity) on low-end hardware, such as mobile devices. The Deep Computing Visualization (DCV) system [Snell and Willard 2007] from IBM and the Chromium Render Server [Paul et al. 2008] are two examples of using clusters to render large and complex data in parallel.

Remote visualization has also been applied to provide rich rendering experiences on less powerful devices. Diepstraten et al. [2004] suggested generating a simplified model with only line primitives for clients that have no 3D graphics hardware, but can run 2D rasterization operations. Duguet and Drettakis [2004] converted 3D models to downscaled point clouds for clients with small display screens. Levoy [1995] proposed a hybrid solution to generate a simplified low-quality polygon representation as well as a difference image, avoiding loss of rendering quality on the client side. Mann and Cohen-Or [1997] designed a similar system that streams both polygons and pixels, but focused on selecting the most important pixels to save bandwidth.

Web-based remote visualization addresses a special type of "thin" client. It enables any client to access the rendering services through a standard web browser. The limitations of such systems include the overhead of standard web communications and the unknown computational capability available on the client. The Reality Server [Nvidia 2009] from Nvidia has was released to provide photorealistic rendering services through the web. Yoon and Neumann [2000] designed a web-based system using IBR (Image-Based Rendering) techniques.

Mobile-based remote visualization addresses another type of "thin" client: mobile devices (e.g., smartphones and tablets). [Woodward et al. 2002] and [D'Amora and Bernardini 2003] were two early attempts to use a PDA as a CAD (Computer-Aided Design) viewer while the actual CAD application ran on a remote server. Chang and Ger [2002] proposed building LDIs (Layered Depth Images) [Shade et al. 1998] on the server. The mobile client runs IBR algorithms to synthesize the display image. Such studies may seem "outdated" now because mobile chips have been drastically improved over the last decade. Current mobile devices are equipped with relatively powerful multi-core GPU chips for complex 3D graphics rendering. However, the limitations on mobile devices still exist for computationally-intensive applications such as video games, which we will discuss in more depth in the following sections.

Remote visualization systems can serve other purposes. Koller et al. [2004] developed a remote rendering system to prevent the valuable 3D model data scanned from famous artworks from leaking to malicious users. This system took an image streaming approach by sending only the rendering results to the user, effectively protecting the 3D source data. [Mark 1999] and [Smit et al. 2009] are two examples of how remote rendering can help boost the rendering rate when the server is unable to render fast enough. In their designs, the server only renders the key frames and the client runs the IBR algorithm to interpolate new frames between key frames. As a result, the perceived frame rate on the client side can be doubled or tripled compared to the actual rendering frame rate on the server side.

Dynamic Model & Restricted Interaction. For some applications like 3D video or animation, the 3D data is dynamically generated or updated and the scene is constantly changing. Rendering such applications requires that every frame be processed in real time before the arrival of the next frame. No offline preparation or time-consuming server computation (e.g., running time-consuming polygon simplification algorithms

or creating LDIs for the model) can be applied to this category of remote rendering systems.

Shi et al. [2008] proposed rendering depth images for 3D tele-immersive video (point cloud based), and exploited IBR algorithms to reduce the viewpoint change latency [Shi et al. 2009; Shi et al. 2010]. Lamberti et al. [2003] introduced an image-based remote rendering framework for mobile clients and proposed a more advanced approach in [Lamberti and Sanna 2007]. Moreover, Humphreys et al. [2002] developed an improved video streaming system that employed rendering clusters for not only 3D rendering but also real-time video coding.

Static Model & Unrestricted Interaction. "Unrestricted interaction" means the user can freely change the viewpoint in the 3D space and even perform other actions that can change the 3D data. The representative application of this category is VE (Virtual Environment) navigation. VE navigation and walk-through systems have been commonly adopted in online games and VR (Virtual Reality) systems [Singhal and Zyda 1999]. The major challenge of VE navigation is that the graphic model of the whole VE is considered too large to be transmitted fast enough for real-time interactive rendering. There are two main directions of VE research. One branch focuses on the peer-to-peer communications between multiple users to synchronously render the virtual world [Funkhouser 1995] and the other studies how to efficiently stream the virtual world from the central server to end users [Cohen-Or et al. 2003]. Here we survey the systems of the second branch that fall into the scope of remote rendering.

Schmalstieg [1997] proposed using a remote rendering pipeline to manage the geometry and bandwidth in a distributed VE. The pipeline uses the server as a graphic model database and transmits only the viewable models based on the user's viewpoint. The server database organizes all 3D graphic models as different LODs (levels of detail) and stores them in Oct-Tree structures. For different zones of AOIs (areas of interest), different LODs are transmitted. [Teler and Lischinski 2001] was a similar approach to streaming partial scenes in 3D format based on visibility, but it also took into consideration network condition and user motion pattern. However, it is not necessary to transmit all original 3D models to client because only the object surfaces are visible in such walk-through applications. Lluch et al. [2005] constructed multi-resolution meshes for remote rendering. Only the surface plus depth, also called 2.5D mesh, is created. Different techniques to build and store multi-resolution meshes are surveyed in [De Floriani and Magillo 2002]. A real-time system to extract a 2.5D mesh for remote rendering was proposed in [Li et al. 2011].

There are also VE systems taking the image streaming approach. The QuickTime VR system [Chen 1995] creates a 360-degree panorama image of the world based on the user's location. This environment map allows the client to pan around without any special graphical hardware or further updates from the server. Boukerche and Pazzi [2006] followed the same direction and developed a mobile VE navigation system based on panorama rendering. Bao and Gourlay [2004] designed a VE walk-through system using a different image-based approach. In their system, the server renders the virtual scene and sends the result image with the depth map to the client. The client can display the received images or run IBR algorithms to synthesize new images when the rendering viewpoint is changed. Noimark and Cohen-Or [2003] introduced an MPEG-4 streaming-based VE walk-through system. The VE scenes are constantly rendered on the server at a preset frame rate and encoded in real time using the MPEG-4 standard. This approach does not need to analyze the source 3D models to determine LOD, but requires constant network bandwidth between server and client for video streaming. [Lamberti et al. 2003; Quax et al. 2006] both proposed similar video-based rendering frameworks for mobile users.

Dynamic Model & Unrestricted Interaction. Cloud gaming is the representative application of this last category. The emergence of cloud gaming services has moved the rendering computations of 3D video games to the cloud. Compared to other specialized remote rendering approaches, such as those introduced above, rendering video games remotely is a much more challenging task. First, the latest video games require very complex 3D graphics rendering to present the virtual world, avatars, and animation effects. Cloud gaming therefore requires advanced graphical hardware for rendering. Second, games need to be rendered at a high frame rate (e.g., 30 fps or higher) to accommodate the dynamically changing scenes and intensive motions. Last, but most importantly, long latency is intolerable. As we have mentioned in Section 1, 100 ms is the largest tolerable interaction latency for first person shooting games [Beigbeder et al. 2004].

Several companies have provided cloud gaming services and solutions, such as OnLive[2], which uses a video streaming approach that renders video games in the cloud and sends gameplay scenes as a 720p video stream to end users. Similar service providers include GaiKai[3], and G-Cluster[4]. Another newly started company Ciinow[5] claims to stream both video and graphics to the client. We believe these commercial cloud gaming systems are the state-of-the-art in remote rendering systems, and we will discuss them in greater technical detail in the next section.

Researchers in academia have been working on the same topic. Game@Large [Eisert and Fechteler 2008; Nave et al. 2008] proposed to send 3D graphics data to the client. The graphics rendering operations and textures are effectively compressed and transmitted using the RTP protocol. Jurgelionis et al. [2009] improved Game@Large with a hybrid approach. For "fat" clients, all graphics rendering operations and textures are streamed as the original Game@Large design. For "thin" clients that do not have enough 3D rendering resources, games are rendered on the server and the result images are compressed with H.264 [Wiegand et al. 2003] for the client. Winter et al. [2006] proposed another hybrid approach: video streaming is used for intensive motion sequences, and graphics data is delivered to render static scenes. [Tizon et al. 2011; Wang and Dey 2013] both studied the issues of cloud gaming for mobile devices. GamingAnywhere [Huang et al. 2013] is an open cloud gaming platform, which may be used by researchers, developers, and gamers to set up testbeds. Various customizations on GamingAnywhere are possible, such as dropping in an H.264/MVC [Vetro et al. 2011] codec to support stereoscopic games.

2.4. Classification and Summary

In this section, we have given a brief overview of different remote rendering designs. Here, we propose a classification method to summarize those designs and highlight their pros and cons. Our classification method is inspired by the survey for distributed graphics systems [Zhou 2006], in which different systems are classified according to the stage at which the rendering pipeline is split between the server and the client [Prohaska et al. 2004]. However, that method is unable to classify the complex remote rendering designs, e.g., the systems that generate environment maps or depth images. Thus, we propose a new classification method based on the type of data transmitted from the server to the client. It is sufficient to cover all remote rendering designs we have reviewed and characterize how rendering computations are distributed. For example, the remote rendering systems taking the image streaming approach belong to the **Image** category, because only the image frames of the rendering results are trans-

[2]http://www.onlive.com
[3]http://www.gaikai.com
[4]http://www.gcluster.com/
[5]http://www.ciinow.com

Table I. Summary of remote rendering systems

Data Type	Description
Original Model	**Server:** transmit all 3D data to client **Client:** perform 3D graphics rendering when all data is received **Pros:** general-purpose **Cons:** "fat" client; excessive bandwidth for complex models **Examples:** [Phil Karlton 2005; Eisert and Fechteler 2008]
Progressive Model	**Server:** transmit all 3D data progressively to client, based on viewpoint or multi-resolution representation **Client:** perform 3D graphics rendering upon the arrival of 3D data **Pros:** reduce the rendering "start" time; bandwidth control **Cons:** "fat" client; pre-processing to generate progressive models **Examples:** [Schmalstieg 1997; Lluch et al. 2005]
Simplified Model	**Server:** transmit simplified 3D models to client **Client:** perform 3D graphics rendering on the simplified models received **Pros:** reduce bandwidth usage; reduce rendering workloads on client **Cons:** rendering quality loss due to model simplification; pre-processing to generate simplified models **Examples:** [Duguet and Drettakis 2004; Li et al. 2011]
Model + Image	**Server:** transmit reformed (usually simplified) 3D models and the difference image to client **Client:** perform 3D graphics rendering and apply the difference image **Pros:** maintain rendering quality with low bandwidth and lightweight computation on client **Cons:** pre-processing to generate simplified models and render both original and simplified models for the difference image **Examples:** [Levoy 1995; Mann and Cohen-Or 1997]
Image	**Server:** perform 3D graphics rendering, transmit result images to client **Client:** display the images received **Pros:** high rendering quality; low bandwidth usage; no rendering workloads on client; source secure **Cons:** interaction latency **Examples:** [Commander 2007; Huang et al. 2013]
Environment Map	**Server:** perform 3D graphics rendering of the whole environment, generate an environment map (i.e., panorama), and transmit the environment map to client **Client:** project the received environment map to the correct viewpoint **Pros:** pros of **Image**; no latency for some types of user interaction (e.g., pan, tilt) **Cons:** extra workloads on server to generate environment maps; interaction latency for other unsupported user interactions **Examples:** [Chen 1995; Boukerche and Pazzi 2006]
Image + Depth	**Server:** perform 3D graphics rendering one or multiple times, extract depth maps together with result images, and send all result images and depth maps to client **Client:** display the result images received; if necessary, run IBR algorithms to synthesize images at new viewpoints **Pros:** pros of **Image**; reduce latency for most user interactions that only change rendering viewpoint **Cons:** extra workloads on server to generate multiple depth images; extra bandwidth needed to transmit all depth images; IBR artifacts; interaction latency for other unsupported user interactions **Examples:** [Chang and Ger 2002; Shi et al. 2012a]

mitted. It also indicates that the server performs all rendering computation, while the client only displays the received images. Interested readers are referred to Figure 6 and Table I for complete information on our classification.

3. CASE STUDY: CLOUD GAMING

In this section, we take OnLive as an example to explain in detail the key research problems of the remote rendering mentioned at the beginning of this article. We choose OnLive as our study case because the performance standard for cloud gaming is much higher than it is for any of the other remote rendering applications reviewed in Section

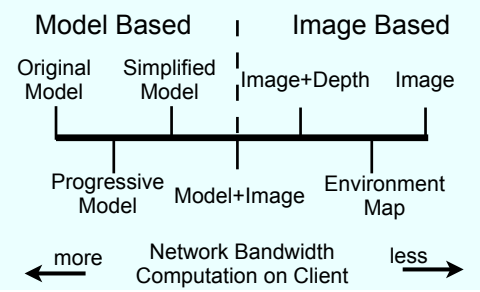


Fig. 6. Remote rendering classification

2. A cloud gaming system renders dynamic gaming contents at a high frame rate (e.g., 30 fps or more) and responds to all types of user interactions in real time. Such a system can easily serve other applications that are more tolerant to latency or refresh at a lower rate. For example, OnLive provides the service of rendering desktop applications to business customers using the same cloud gaming system.

OnLive is a proprietary cloud gaming system. Some technical system details are revealed in [Perlman et al. 2010] and the system performance has been measured and studied in [Claypool et al. 2012; Chen et al. 2014]. OnLive renders 3D games on a cloud server, extracts game scene images from the frame buffer, compresses the images with a customized video encoder, and streams the video to a game client through broadband networks. The game client simply receives, decodes, and displays the game video. Meanwhile, it collects user input signals (e.g., the events of mouse, keyboard, or game controller) and forwards them to the rendering server. Currently, the game client runs on PC, Mac, set-top box, or even mobile devices (e.g., iPad or Android tablets). Using such a rendering service, gamers can enjoy the most advanced video games without buying expensive game consoles. OnLive is highly optimized for satisfactory gaming experiences. However, it still struggles with two key problems: interaction latency and data transmission.

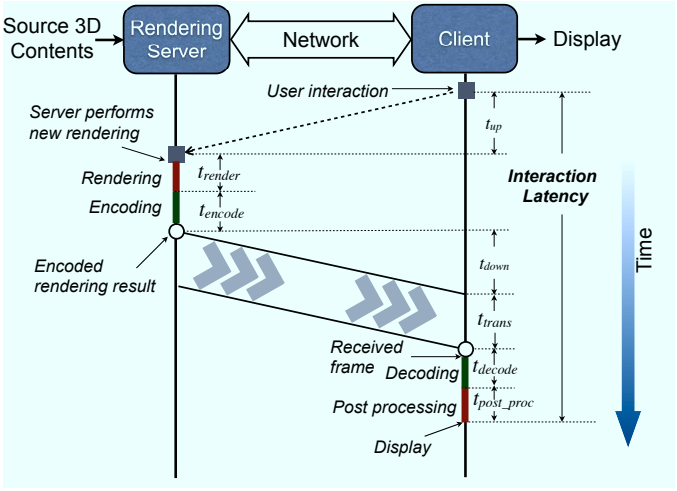


Fig. 7. Illustration of interaction latency

Interaction Latency. We first take a close look at the components of interaction latency. Figure 7 shows an illustration of the interaction latency of a video streaming-based remote rendering system. The overall interaction latency is comprised of the server processing time ($t_{render} + t_{encode}$), the client processing time ($t_{decode} + t_{post\_proc}$), the propagation delay ($t_{up} + t_{down}$), and the transmission delay (t_{trans}).

OnLive sets the target of interaction latency to be less than 80 ms, so as not to affect user experience even in motion intensive games [Beigbeder et al. 2004]. Their approach was to set strict requirements for networking. Users must have a broadband wired network connection to use OnLive services, which helps to control both propagation delay and transmission delay. Meanwhile, OnLive tries to save every single millisecond by optimizing every component of the system. Figure 8 elaborates the latency quota for each component in the interaction loop. In order to achieve this goal, OnLive has made special efforts not only on its own software, but also on its hardware and networking. For example, OnLive has tried to deploy cloud servers physically close to users because every 1000 mile of physical distance adds 25 ms round trip delay to the overall interaction latency [Perlman et al. 2010].

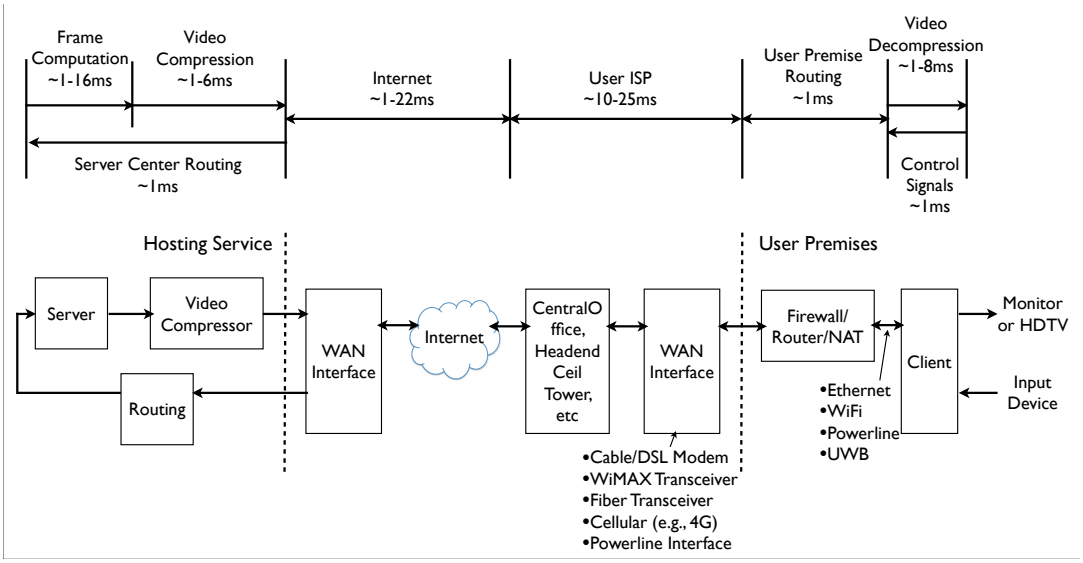
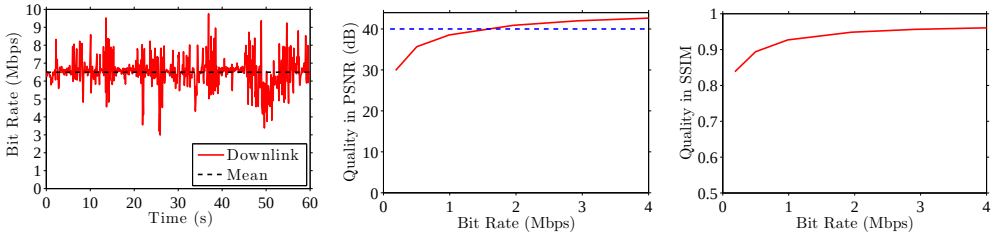


Fig. 8. Latency analysis of different system components in cloud gaming [Perlman et al. 2010]

Despite all of these efforts, OnLive is still unable to make any guarantees regarding latency. The current approach relies considerably on the connection quality and network condition. Due to the "best effort" nature of the Internet, latency can easily fluctuate due to background network traffic. Unlike other video streaming services, OnLive cannot buffer video frames for smooth playback because the buffer delay adds to the overall interaction latency. Therefore, any large jitter or packet loss rate may directly affect users' gaming experience.

Data Transmission. OnLive suffers from the coding inefficiency of using a real-time video encoder, i.e., to achieve the same video quality, OnLive consumes a higher bandwidth than the latest codecs. The minimum requirement is to have at least 3 Mbps broadband Internet connections to use the service. However, we have found that the actual bandwidth usage is much higher. We played *Unreal Tournament III: Titan Pack*,



(a) (b) (c)

Fig. 9. Sample results from a first-person shooting game: (a) OnLive streaming rate, (b) coding efficiency of a general purpose video coder in PSNR, and (c) in SSIM

a first-person shooting game, for 60 seconds, and captured all downlink network packets. From the network trace, we plot the OnLive streaming rate in Figure 9(a), which indicates that the real-time OnLive encoder produces a stream at a fairly high bit rate: 6.49 Mbps on average. As a comparison, we played a similar first-person shooting game locally on our laptop and captured a 720p raw video, encoded the raw video at several bit rates using x264 [Garrett-Glaser 2010] with typical coding parameters. We plot the resulting Rate-Distortion (R-D) curves on Peak Signal-to-Noise Ratio (PSNR) [Wang et al. 2001] and Structural SIMilarity (SSIM) [Wang et al. 2004] in Figures 9(b) and 9(c). These two figures show that x264 achieves almost perfect video quality, 40+ dB (PSNR) and 0.93+ (SSIM) at merely 1.5 Mbps, which is much lower than that used on OnLive. The experiment reveals that the state-of-the-art OnLive video encoder suffers from suboptimal coding efficiency, which increases the burden of network streaming and client processing. More details on these experiments can be found in [Shi et al. 2011a].

Moving to Mobile. Mobile gaming is playing an increasingly important role in the gaming industry as computing becomes more ubiquitous. Applying cloud gaming to mobile devices can readily boost the rendering quality of mobile games to the game console level and significantly reduce the complexity of developing the same game for different mobile devices.

However, applying the current cloud gaming system (e.g., OnLive) to mobile devices will the latency issue even more challenging. Mobile devices connect to the Internet through wireless connections like WiFi or mobile networks (e.g., 3G, 4G/LTE, etc.). Unlike wired Internet connections, wireless connections suffer from longer network delays. According to the experiments in [Huang et al. 2012], even the best technologies take more than a 60-ms round trip propagation delay, which alone exceeds the quota for all networking-related latency shown in Figure 8. This number does not count the factors of mobility, signal strength, or interference that can easily make the connection more fragile. Therefore, the current approach cannot be easily extended to wireless users.

In summary, latency and bandwidth are two issues that are not yet perfectly solved by the current remote rendering systems and will remain critical challenges for future systems serving mobile clients. In the next two sections, we will dig deeper into these two problems and survey proposed solutions.

4. LATENCY REDUCTION

With conventional optimization techniques, the overall interaction latency cannot break the lower bound of the network round trip delay. However, there is a bypass. It is possible to generate the visual response frames on the client before any server re-

sponses are received. In this case, the perceived interaction latency for the user is only the time it takes for the client to create a response frame (Figure 10). In this section, we survey the latency reduction techniques that are less affected by network delay.

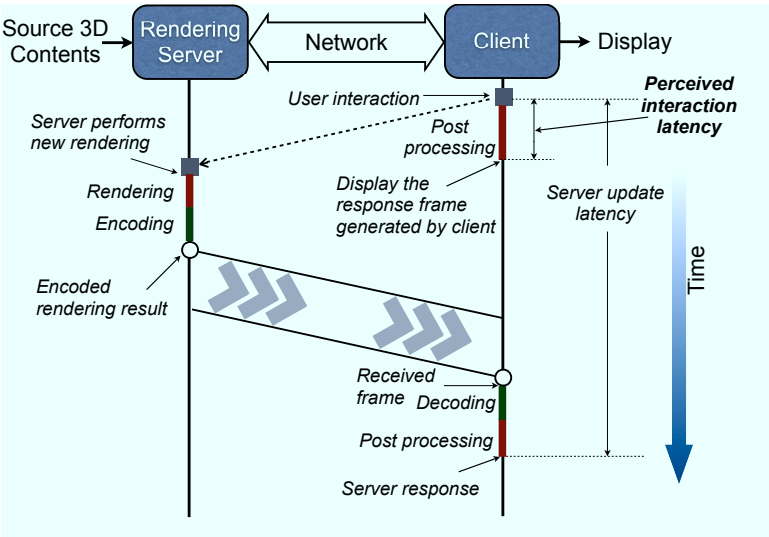


Fig. 10. Illustration of interaction latency reduction

4.1. Local Rendering and Dead Reckoning

The easiest method to create a response frame locally is to store 3D data on the client and perform 3D graphics rendering locally. Some VE walk-through [Lluch et al. 2005] and remote visualization systems [Duguet and Drettakis 2004] that stream meshes to the client can apply local rendering and reduce the interaction latency to the time needed for 3D graphics rendering on each client device. The cloud gaming provider Ciinow also claims that the integration of 3D graphics streaming [Dharmapurikar 2013b] actually reduces the latency[6].

One key issue in this approach is to keep the local rendering synchronized with the server. For visualization or walk-through applications, the server-client synchronization may only need to keep the rendering viewpoint consistent. Synchronization becomes more complicated for gaming applications because rendering game scenes requires knowledge of the game logic and user data (e.g., the speed and direction of moving objects).

Dead Reckoning, a strategy widely adopted in developing multi-player online games [Pantel and Wolf 2002], can be applied to mitigate the synchronization problem. The basic notion of dead reckoning is to agree in advance on a set of algorithms that can be used by the client to extrapolate the behavior of entities in the game. There should also be agreement on how far reality should be allowed to get from these extrapolation algorithms before a correction is issued. In the scenario of remote rendering, the correction can be issued with the server updates after approximately a network round trip delay.

[6]http://www.ciinow.com/cloud-gaming-service-technology/

4.2. Pre-Fetch

For a system that cannot process 3D graphics rendering on the client side, pre-fetch is an intuitive but effective latency reduction approach. A client (or server) predicts a user's future moves and requests the server to transmit the images rendered for all possible moves. If the prediction hits, the client can simply display the pre-fetched images and no latency is noticed. If the prediction fails, the client should flush the pre-fetch buffer and wait for the correct images to arrive from the server.

Hesina and Schmalstieg [1998] introduced a pre-fetch framework for a networked VE. Chen et al. [2008] discussed how to pre-fetch images of multiple resolutions to cover different levels of detail. Boukerche and Pazzi [2006] is another example but the pre-fetched images are panoramas which are sufficient to cover any pan/tilt movements. Different motion prediction methods and pre-fetch protocols are introduced in Chan et al. [2001] and Lazem et al. [2007]. Touch [1995] introduced a source-driven pre-sending approach for general communications over high-speed networks. Pre-sending helps to reduce the latency by half the time of a round trip delay when bandwidth is not the bottleneck.

Pre-fetch works best for navigating within a static environment, the image of which can be saved for future use without expiration. The performance of pre-fetch techniques completely relies on the hit rate of motion prediction and the size of the pre-fetch buffer. The cost is the extra network bandwidth needed to transmit all pre-fetched scene images, most of which might never be used. Pre-fetch is inappropriate for applications that render dynamic data, such as games. Such applications refresh quickly and each frame expires upon the arrival of the next frame. Therefore, a client needs to pre-fetch for every new frame, which significantly increases the network bandwidth usage.

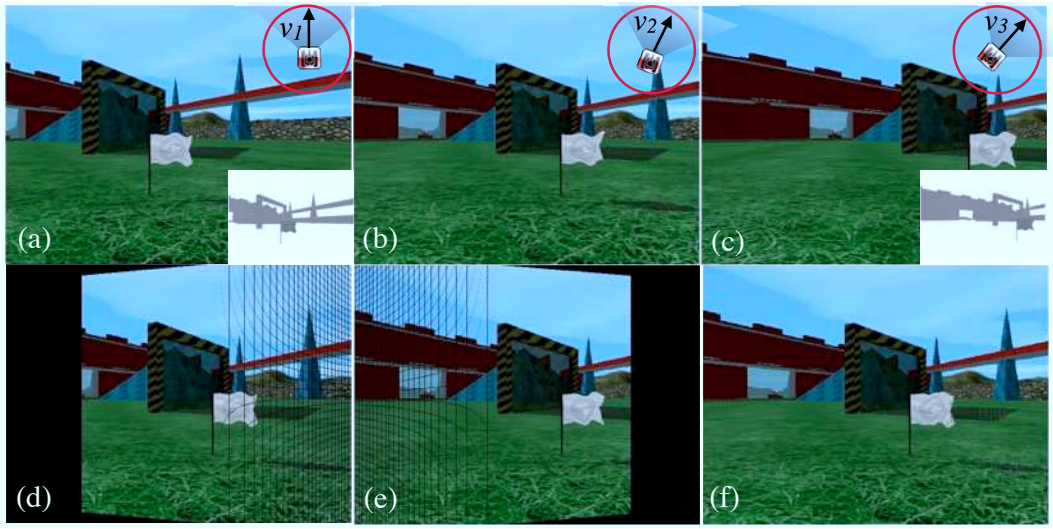


Fig. 11. (a) The game scene image and corresponding depth map, while playing game BZFlag (http://bzflag.org) at the current viewpoint v_1 (the tank looks in the 12 o' clock direction); (b) the game scene at the target viewpoint v_2 (the tank looks in the 1 o' clock direction); (c) the game scene image and corresponding depth map at the reference viewpoint v_3 (the tank looks in the 2 o' clock direction); (d) 3D image warping result of warping (a) to viewpoint v_2; (e) 3D image warping result of warping (c) to viewpoint v_2; (f) compose (d) and (e) [Shi et al. 2012a]

4.3. Image-Based Rendering

Image-based rendering falls between the 3D approach (local rendering) and the image approach (pre-fetch). It requires the server to send extra information with the rendering result images to the client. On the client side, when the user interaction changes the rendering viewpoint, the extra information can be used to run IBR algorithms to synthesize the display image at the new viewpoint. Therefore, the interaction latency is reduced to the time required to run IBR algorithms.

A well-known IBR algorithm is 3D image warping [McMillan 1997]. It takes a depth image, the viewpoint corresponding to the depth image, and the target viewpoint as inputs, and outputs the image at the target viewpoint (Figure 11). This algorithm can be efficiently executed on embedded CPUs/GPUs [Yoo et al. 2010] or on customized hardware [Popescu et al. 2000]. 3D image warping has been applied in several remote rendering systems [Chang and Ger 2002; Shi et al. 2012a; Mark 1999; Bao and Gourlay 2004]. For every frame, the server needs to send a depth map with a result image to the client and the interaction latency for any viewpoint change can be reduced to the time required to run 3D image warping on the client device.

However, using 3D image warping creates the problem of *exposure*. The synthesis result image usually has hole artifacts because there are insufficient pixels in the input image to fill the surface at a new viewpoint. Various techniques have been proposed to fill warping holes. Depth filters [Redert et al. 2002] and splat warping [Mark 1999] are effective for small holes. Super view warping [Bao and Gourlay 2004] and wide field-of-view warping [Mark 1999] can partially solve the occlusion problem. View compensation [Bao and Gourlay 2004] requires a server to generate the difference image between the warping result and the actual rendering result. LDI [Shade et al. 1998] and *Double Warping* [Shi et al. 2009; Shi et al. 2010] are both based on the idea of using multiple references to cover exposed holes.

Of all the techniques listed above, *Double Warping* is probably the most appropriate technique to reduce interaction latency in remote rendering systems. With this approach, a server renders not only the depth image at the current viewpoint, but multiple depth images at other reference viewpoints. The client runs 3D image warping algorithm for all received depth images and composes all the results. The reference viewpoints are carefully selected based on a prediction of camera movement [Mark 1999; Shi et al. 2010]. Note that the prediction here is slightly different from the motion prediction for pre-fetch. It does not need to give the accurate position of the future viewpoint, but only a possible direction/area. With the predicted direction/area of motion, references should be selected to cover that area so that a high-quality hole-free image can be synthesized by the client (Figure 11). Different reference selection algorithms have been studied in [Mark 1999; Shi et al. 2009; Shi et al. 2012a; Hudson and Mark 1999; Thomas et al. 2005]. *Double Warping* actually converts the network-dependent latency reduction problem to a content-based reference selection problem.

4.4. Summary

We have discussed three types of techniques that can successfully mitigate the influence of network delay on interaction latency. The pros and cons of each of these techniques are summarized in Table II. In practice, all three techniques can be combined in one system to achieve the best result. For example, in the scenario of cloud gaming, we can pre-fetch the panoramic image of the surrounding background environment, use 3D image warping algorithms to render moving foreground objects, and download 3D data on the client side to handle other user interactions like firing a weapon.

Table II. Summary of latency reduction techniques

Approach	Description
Local Rendering	**Pros:** works for any user interactions **Cons:** requires 3D graphics rendering on the client side **Best for:** lightweight rendering tasks, animations, non-viewpoint-changing actions
Pre-Fetch	**Pros:** simple image-based approach; no rendering computation on the client side **Cons:** works only for static scenes and discrete viewpoint-changing actions; relies on accurate motion prediction **Best for:** static background environment navigation
Image-Based Rendering	**Pros:** works for both static and dynamic scenes; image-based approach; lightweight computation workloads on the client side **Cons:** Works only for viewpoint-changing actions; hole artifacts **Best for:** foreground moving objects

5. DATA TRANSMISSION

We mainly focus on the data transmission module of remote rendering systems, and we classify all the studies in the literature into three groups. We first present the data compression techniques that are suitable for remote rendering systems. This is followed by the data streaming protocol and strategies that deliver the encoded data streams to clients. Finally, we present the latest work on QoS/QoE evaluations for remote rendering systems. The data transmission module has to satisfy stringent real-time constraints, and thus imposes a direct effect on the overall system performance.

5.1. Data Compression

In remote rendering systems, the data transferred between the server and client are compressed using various algorithms in order to reduce the network loads. We present the tools for common data compression, depth image compression, and joint coding.

Common Tools. There are many tools that can be used for compressing images, videos, and graphics, including lossless compression, lossy image/video compression, real-time video encoding, compound image encoding, and graphics encoding. We detail each of these common tools in the following paragraphs.

The lossless compression tools, such as LZO [Oberhumer 1996], BZIP [Seward 1996], and [Weinberger et al. 1996], harvest the repetitions in raw data for encoded data at a reduced bit rate. Uncompressing these encoded data results in perfect reconstruction identical to the input raw data. The lossless compression tools are general-purpose and can be used to compress texts, executables, binaries, audio, images and graphics. These lossless compression tools were used by the servers in early remote rendering systems [Ma and Camp 2000; Hege et al. 2000; Levoy 1995] to compress the rendered images before transferring them to the client.

Compared to the lossless compression tools, the lossy image compression tools drop the details that are unnoticeable to the human eye to further reduce bit rates. There are several early attempts to use lossy image compression tools for leveraging characteristics of human visual systems [Ikonomopoulos and Kunt 1985; Kunt et al. 1987; Egger et al. 1995; Campbell et al. 1986; Marcellin et al. 2000; Skodras et al. 2001; Du and Fowler 2007]. Although the lossy image compression tools may be used to compress videos as well, they cannot leverage the redundancy across multiple frames. The *hybrid* video coders [Wang et al. 2001] integrate lossy image compression tools with motion-compensation tools to exploit the temporal redundancy. The hybrid video coders have been standardized as, e.g., MPEG-4 [Schafer 1998] and H.264/AVC [Wiegand et al. 2003], which (or their variations) are used by several modern remote rendering systems [Noimark and Cohen-Or 2003; Lamberti and Sanna 2007; Jurgelionis et al. 2009; De Winter et al. 2006; Perlman et al. 2010; Shi et al. 2011a; Herzog et al. 2008; Huang et al. 2013]. The video coding tools proposed in [Tran et al. 1998; Bayazit

1999; Liu et al. 2007] adopted some or all of the real-time coding constraints discussed in [Reddy and Chunduri 2006; Schreier et al. 2006]. When extremely low coding delay, in the order of nanoseconds, is required, intra-frame video coders [Lee and Song 2008; Nadeem et al. 2010], which essentially compress individual video frames independently, can be used at the expense of significantly lower coding efficiency.

Compound image coding tools target compressing computer desktops consisting of less motions, but a combination of texts, graphics, and natural images. Shape Primitive Extraction and Coding (SPEC) [Lin and Hao 2005] segments each image into texts/graphics and natural image regions, and encodes the texts/graphics regions using a lossless compression algorithm, and the natural regions using JPEG. A similar idea was proposed by Maheswari and Radha in [2010]. Wang and Lin [2009] proposed to concurrently compress each macroblock using H.264 [Wiegand et al. 2003] and gzip [Gailly 1992], and choose the better compressed macroblock in the rate-distortion fashion; they later proposed a generalized system called United Coding (UC) [Wang and Lin 2012] based on multiple lossless coders, such as Run-Length Coding (RLE), gzip, and Portable Network Graphics (PNG). Two new H.264 intra modes were proposed in [Lan et al. 2010] to better exploit spatial redundancy in compound images. The first mode directly quantizes and entropy-codes the intra-predicted residues and the second mode uses adaptive vector quantization. Han et al. [2010] extended H.264 [Wiegand et al. 2003] by adding a preprocessor, an adaptive quantization algorithm, and a rate control algorithm.

Graphics coding tools are used to compress geometric graphics data, e.g., polygonal meshes, point clouds, and volume data. Compressing polygonal meshes was considered in an early geometry compression study [Deering 1995], which generated a linear stream of polygonal meshes before applying Huffman, delta, and VLC coding. In [Li and Kuo 1998], the polygonal meshes were represented by topological and geometrical data, and different coding tools were applied to them. Local compression of polygonal meshes was proposed for low-complexity mesh compression algorithms [Gumhold and StraBer 1998]. A spectral compression method projects polygonal meshes onto an orthonormal basis for progressive mesh streaming [Karni and Gotsman 2000]. The multi-resolution 3D Discrete Wavelet Transform (DWT) analysis was used for low bitrate polygonal mesh compression [Payan and Antonini 2002]. CODDYAC [Vasa and Skala 2007] studied the mesh compression problem from both spatial and temporal domains. The error resilience problem of polygonal mesh streaming was investigated in [Park et al. 2006]. The geometry compression algorithms for non-polygonal meshes were studied in [Gumhold et al. 1999]. The compression algorithms for normal maps were considered in [Shi et al. 2012b]. Many more mesh compression algorithms are proposed in the literature. Interested readers are referred to a survey [Peng et al. 2005].

Depth Compression. Depth images play an important role in image-based rendering. The compression of depth images has been well studied in the literature. For example, regions-of-interest and dynamic range adjustments were considered in [Krishnamurthy et al. 2001]. Surface shape prediction was used to compress some regions [Zanuttigh and Cortelazzo 2009]. LDI (Layered Depth Image) has been employed in some remote rendering systems [Chang and Ger 2002], which can be compressed by the algorithms introduced in [Duan and Li 2003; Cheng et al. 2007]. Other depth compression algorithms include [Sarkis and Diepold 2009; Penta and Narayanan 2005; Kum and Mayer-Patel 2005; Yang et al. 2006; Chai et al. 2002]. Studies [Milani and Calvagno 2010; Shi et al. 2011a; Oh and Ho 2006; Morvan et al. 2007] showed that the modified H.264 [Wiegand et al. 2003] encoders can efficiently encode depth images. For example, Oh and Ho [2006] computed candidate motion modes by utilizing texture

videos' motion information for shorter encoding time yet higher coding efficiency. Morvan et al. [2007] concurrently employed block-based and 3D warping prediction modes. The rendering qualities of different depth image compression algorithms were given in Merkle et al. [2009]. Interested readers are referred to a survey of depth compression algorithms [Shum et al. 2003].

Jointly Encoding. In a remote rendering system, the encoder usually runs on the same server as the rendering engine. Researchers have found that better coding performance can be achieved by jointly encoding the rendered images with the metadata extracted from the rendering engine. Noimark and Cohen-Or [2003] used graphics information to segment the background and foreground, applied different quantization parameters, detected scene cuts using motion information, and analyzed the optical flow to speed up motion estimation. All these steps help boost the performance of MPEG-4 encoding. Pajak et al. [2011] proposed compressing low-resolution depth images using H.264 [Wiegand et al. 2003] and streaming the H.264 videos along with augmentation information such as edges and motions. Upon receipt, the depth images are upscaled to their original resolution with the assistance of edges and motions. Shi et al. [2011a] retrieved the run-time graphics rendering contexts, including rendering viewpoints, depth images, and camera movements, from the 3D game engine. These contexts can be used to increase the compression ratio of cloud gaming video streams.

5.2. Data Streaming

For data streaming, we survey the streaming protocols and several streaming strategies.

Protocols. Several network protocols have been proposed to stream compressed videos over the Internet. For example, GLX [Phil Karlton 2005] was designed as an extended X11 protocol for streaming 3D rendering operations, and NX [BerliOS 2008] was proposed to improve the remote rendering efficiency of X11/GLX. Compression, caching, and round trip time suppression are included in the NX protocol to boost the overall speed of streaming XLib and OpenGL operations. Remote Frame Buffer (RFB) is an open network protocol used by VNC systems [Richardson et al. 1998] to stream frame buffer updates. THiNC, introduced in [Baratto et al. 2005], is another protocol that streams low-level drawing operations. Figure 12 compares the difference among these four protocols.

The network protocols proposed for video streaming [Li et al. 2013], such as RTP and DASH, may also be used in remote rendering systems. 3TP for graphics [AlRegib and Altunbasak 2005] is a network protocol for streaming 3D models over lossy channels on top of both TCP and UDP. 3TP combines source and channel characteristics to minimize end-to-end delay.

Progressive Streaming. Progressive streaming transmits lower-quality graphics/texture data, which are followed by refinements. Hoppe [1996] first proposed constructing progressive meshes. Rusinkiewicz et al. [2000; 2001] introduced QSplat to progressively divide large-scale dense graphic models for networked visualization. The cases where the geometry models are simple but the texture maps are complex have also been studied in [Cohen-Or et al. 1999] and [Marvie and Bouatouch 2003], in which the texture maps are organized based on user viewpoint for progressive transmission. [Yu et al. 2004; Tu et al. 2006] proposed using parallel Oct-trees to manage very large scale (in the range of tera-scale) unstructured meshes. This type of structure allows the rendering processor to process data blocks at different resolutions. While progressive streaming allows remote rendering systems to utilize

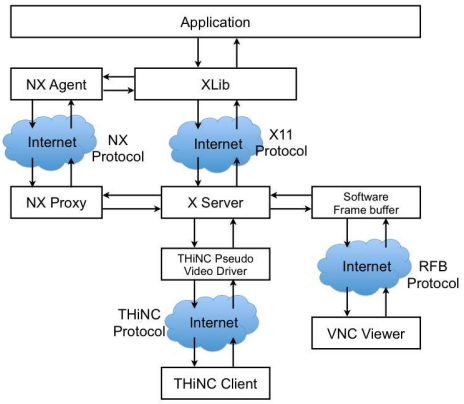


Fig. 12. Thin client and remote rendering [De Winter et al. 2006]

available bandwidth for higher data transmission quality, it does not actively change the data representations.

Adaptive Streaming. Adaptive streaming dynamically adjusts the data representations at the remote rendering server based on the current network conditions. Schneider and Martin [1999; 2000] proposed a performance model to evaluate client-server remote rendering systems. Their model selects a remote rendering approach based on three components: transmission time, quality, and interactivity. Pasman and Jansen [2003] developed a similar model for a client-server image-based rendering system to determine which graphical representation and graphics simplification technique should be used. Wang and Dey [Wang and Dey 2012] proposed an adaptive streaming approach that jointly changes rendering the encoding parameters to adapt to the network conditions for mobile cloud gaming. Ciinow [Dharmapurikar 2013a] proposed several bandwidth adaption schemes for different types of data. For cases in which large scale data is visualized, adaptive streaming should also consider the storage space available on the hard disk as well as network bandwidth [Malakar et al. 2010].

5.3. QoS/QoE Evaluation

The design space of data compression and data streaming in remote rendering systems is large. Therefore, the QoS and QoE measurement techniques for remote rendering systems are important for system evaluation. We survey a few representative measurement techniques below and discuss how they can be extended to better suit remote rendering systems in Section 6. Readers who are interested in more data streaming QoS/QoE measurement techniques are referred to [Chikkerur et al. 2011]. Serral-Gracia et al. [2010] surveyed the QoE measurement of video streaming systems. Stegmaier et al. [2003] discussed in depth how QoS parameters affect remote visualization. Pravati et al. [2011] studied the QoS and QoE measurements and performance monitoring for rendering VE remotely on mobile devices. Wang and Dey [2009] proposed a QoE model for mobile cloud gaming, which takes the game type, resolution, frame rate, video quality, delay, and packet loss into consideration, and their QoE measurement techniques were presented in Wang and Dey [2012]. VQ (Video Quality) [Nieh et al. 2003] is another metric to measure the actual video streaming performance of a thin-client system. Owing to the frame rate requirement of video playback, the system may actively drop video frames if they arrive late. Therefore, a slow motion benchmark that has a much lower frame rate requirement is applied to make sure that

no frame is dropped by the testing system. VQ was also borrowed in the image-based remote rendering systems [De Winter et al. 2006].

6. DISCUSSIONS

With the explosion of mobile devices and cloud computing, we truly believe the integration of these technologies will not only be limited to data sharing, but it will extend to distributed computation. Although mobile devices have more powerful, the abundance and flexibility of cloud resources still offer several incentives for mobile devices to the resources, including CPU (processing), GPU (rendering), energy, storage, and security [Abolfazli et al. 2014]. Therefore, we expect to see more seamless integration between mobile devices and cloud servers in computation-intensive tasks, and remote rendering will be an important part of this evolution. In this section, we want to share our vision of the future research directions on remote rendering.

Mobile Cloud Gaming. According to [Dinh et al. 2011], mobile cloud gaming is a major application of mobile cloud computing. We have already discussed the problems and difficulties of applying the current cloud gaming systems directly to mobile platforms in Section 3. New approaches are demanded to stream video games efficiently to mobile devices within strictly enforced latency restrictions.

After surveying all remote rendering related techniques, we believe the ideal mobile cloud gaming system should divide the rendering tasks based on rendering complexity and latency sensitivity, and process them differently. The tasks that are less sensitive to latency can be rendered on the server using an image-based approach. For the tasks that are sensitive to latency but easier to render, the 3D models can be streamed to the client to render locally. Only the tasks that are both sensitive to latency and complicated to render need to be treated specially. To be specific, we can choose to either generate a simplified 3D representation, or use IBR techniques for approximation, so that the rendering computation on the client can be managed with the available hardware resources. Finally, the image layers from different tasks are composed for the final display [Torborg and Kajiya 1996].

Remote Rendering Model, Library, and Service. Building a cloud gaming system alone may not solve all latency and bandwidth problems completely. This is because video games are designed to be rendered locally and game developers have no notion of large interaction latency in mind. In the long run, the success of cloud gaming relies on the adoption of the remote rendering model. Game developers should consider the network distributed architecture from the beginning stages of the development of new games. The concepts of two computers remotely connected by a delayed link, image streaming, graphics streaming, data synchronization, etc., should be integrated with the design of the game engine. At the same time, it is also necessary to encapsulate all the details of data compression, streaming protocols, IBR algorithms, etc., into, say, an Open Remote Graphics Library (similar to OpenGL), so that the developers can focus on game content. Another interesting topic is the provision of rendering as a cloud service so that any application with rendering requests can benefit from cloud computing.

Quantitative Performance Evaluations. Another interesting and important research topic involves defining a quantitative performance evaluation metric for remote rendering systems. Existing attempts [Wang and Dey 2009; 2012] studied the QoE model to evaluate the performance of remote rendering, but a quantitative metric like PSNR [Wang et al. 2001] or SSIM [Wang et al. 2004] that can be effectively calculated at run-time will be even more useful to help the rendering server to adjust encoding and streaming parameters for varying network conditions. For example, Shi et al. [2011b] proposed DOL (Distortion Over Latency) to evaluate the interactive performance of remote ren-

dering systems. DOL is defined to combine both interaction latency and rendering quality into one score to measure the interactive performance. However, the metric can be improved by integrating more components and validated with both extensive objective and subjective tests.

7. CONCLUSION

In this article, we have provided background knowledge on remote rendering systems, surveyed the existing remote rendering designs from different perspectives, summarized the technologies that are applied to build key components of a real system, and discussed the possible directions of future research. From the survey, we observe that the state-of-the-art remote rendering designs are usually customized to meet the requirements of specific applications. The direction of building a more general and scalable remote rendering system demands new frameworks, algorithms, and programming models that can guide developers in designing better distributed applications.

REFERENCES

ABOLFAZLI, S., SANAEI, Z., AHMED, E., GANI, A., AND BUYYA, R. 2014. Cloud-based augmentation for mobile devices: Motivation, taxonomies, and open challenges. *Communications Surveys Tutorials, IEEE 16,* 1, 337–368.

ALREGIB, G. AND ALTUNBASAK, Y. 2005. 3tp: An application-layer protocol for streaming 3-d models. *Multimedia, IEEE Transactions on 7,* 6, 1149–1156.

APPLE. 2009. Cgl reference. http://developer.apple.com/library/mac/documentation/Graphics Imaging/Reference/CGL_OpenGL/CGL_OpenGL.pdf.

BAO, P. AND GOURLAY, D. 2004. Remote walkthrough over mobile networks using 3-d image warping and streaming. *Vision, Image and Signal Processing, IEE Proceedings - 151,* 4, 329 – 336.

BARATTO, R. A., KIM, L. N., AND NIEH, J. 2005. Thinc: a virtual display architecture for thin-client computing. In *Proceedings of the twentieth ACM symposium on Operating systems principles*. SOSP '05. ACM, New York, NY, USA, 277–290.

BAYAZIT, U. 1999. Macroblock data classification and nonlinear bit count estimation for low delay H.263 rate control. In *Proc. of IEEE International Conference on Image Processing (ICIP'99)*. Kobe, Japan, 263–267.

BEERMANN, D. AND HUMPHREYS, G. 2003. Visual computing in the future: Computer graphics as a remote service. *University of Virginia, Computer Science Department, University of Virginia Technical Report CS-2003-16 25*.

BEIGBEDER, T., COUGHLAN, R., LUSHER, C., PLUNKETT, J., AGU, E., AND CLAYPOOL, M. 2004. The effects of loss and latency on user performance in unreal tournament 2003. In *Proc. of NetGames'04*. Portland, OR, 144–151.

BERLIOS. 2008. Freenx - nx components. http://openfacts2.berlios.de/wikien/index.php/Berlios Project:FreeNX - NX Components.

BETHEL, W. 2000. Visapult: A prototype remote and distributed visualization application and framework.

BHANIRAMKA, P., ROBERT, P., AND EILEMANN, S. 2005. Opengl multipipe sdk: a toolkit for scalable parallel rendering. In *Visualization, 2005. VIS 05. IEEE*. 119–126.

BOUKERCHE, A. AND PAZZI, R. W. N. 2006. Remote rendering and streaming of progressive panoramas for mobile devices. In *Proceedings of the 14th annual ACM international conference on Multimedia*. MULTIMEDIA '06. ACM, New York, NY, USA, 691–694.

CAMPBELL, G., DEFANTI, T. A., FREDERIKSEN, J., JOYCE, S. A., AND LESKE, L. A. 1986. Two bit/pixel full color encoding. *SIGGRAPH Comput. Graph. 20*, 215–223.

CEDILNIK, A., GEVECI, B., MORELAND, K., AHRENS, J., AND FAVRE, J. 2006. Remote large data visualization in the Paraview framework. In *Proceedings of the 6th Eurographics Conference on Parallel Graphics and Visualization*. EG PGV'06. Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 163–170.

CHAI, B.-B., SETHURAMAN, S., AND SAWHNEY, H. 2002. A depth map representation for real-time transmission and view-based rendering of a dynamic 3d scene. In *3D Data Processing Visualization and Transmission, 2002. Proceedings. First International Symposium on*. 107 – 114.

CHAN, A., LAU, R. W. H., AND NG, B. 2001. A hybrid motion prediction method for caching and prefetching in distributed virtual environments. In *VRST*. 135–142.

CHANG, C.-F. AND GER, S.-H. 2002. Enhancing 3d graphics on mobile devices by image-based rendering. In *Proceedings of the Third IEEE Pacific Rim Conference on Multimedia: Advances in Multimedia Information Processing*. PCM '02. Springer-Verlag, London, UK, 1105–1111.

CHEN, J., YOON, I., AND BETHEL, W. 2008. Interactive, internet delivery of visualization via structured prerendered multiresolution imagery. *IEEE Trans. Vis. Comput. Graph. 14,* 2, 302–312.

CHEN, K.-T., CHANG, Y.-C., HSU, H.-J., CHEN, D.-Y., HUANG, C.-Y., AND HSU, C.-H. 2014. On the quality of service of cloud gaming systems. *IEEE Transactions on Multimedia 16,* 2, 480–495.

CHEN, S. E. 1995. Quicktime vr: an image-based approach to virtual environment navigation. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*. SIGGRAPH '95. ACM, New York, NY, USA, 29–38.

CHENG, X., SUN, L., AND YANG, S. 2007. A multi-view video coding approach using layered depth image. In *IEEE 9th Workshop on Multimedia Signal Processing (MMSP'07)*. Chania, Crete, Greece, 143–146.

CHIKKERUR, S., SUNDARAM, V., REISSLEIN, M., AND KARAM, L. 2011. Objective video quality assessment methods: A classification, review, and performance comparison. *Broadcasting, IEEE Transactions on 57,* 2, 165–182.

CLAYPOOL, M., FINKEL, D., GRANT, A., AND SOLANO, M. 2012. Thin to win? network performance analysis of the OnLive thin client game system. In *ACM Workshop on Network and Systems Support for Games (NetGames)*. 1–6.

COHEN-OR, D., CHRYSANTHOU, Y. L., SILVA, C. T., AND DURAND, F. 2003. A survey of visibility for walkthrough applications. *Visualization and Computer Graphics, IEEE Transactions on 9,* 3, 412–431.

COHEN-OR, D., MANN, Y., AND FLEISHMAN, S. 1999. Deep compression for streaming texture intensive animations. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*. ACM Press/Addison-Wesley Publishing Co., 261–267.

COMMANDER, D. R. 2007. Virtualgl: 3d without boundaries the virtualgl project. http://www.virtualgl.org/.

CUMBERLAND, B. C., CARIUS, G., AND MUIR, A. 1999. Microsoft windows nt server 4.0 terminal server edition technical reference. Microsoft Press.

D'AMORA, B. AND BERNARDINI, F. 2003. Pervasive 3d viewing for product data management. *IEEE Comput. Graph. Appl. 23*, 14–19.

DE FLORIANI, L. AND MAGILLO, P. 2002. Multiresolution mesh representation: Models and data structures. *Tutorials on Multiresolution in Geometric Modelling*, 363–418.

DE WINTER, D., SIMOENS, P., DEBOOSERE, L., DE TURCK, F., MOREAU, J., DHOEDT, B., AND DEMEESTER, P. 2006. 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*. NOSSDAV '06. ACM, New York, NY, USA, 15:1–15:6.

DEERING, M. 1995. Geometry compression. In *Proc. of the 22nd Annual Conference on Computer Graphics and Interactive Techniques (SIGGRATH'95)*. Los Angeles, CA, 13–20.

DEFANTI, T., ACEVEDO, D., AINSWORTH, R., BROWN, M., CUTCHIN, S., DAWE, G., DOERR, K.-U., JOHNSON, A., KNOX, C., KOOIMA, R., KUESTER, F., LEIGH, J., LONG, L., OTTO, P., PETROVIC, V., PONTO, K., PRUDHOMME, A., RAO, R., RENAMBOT, L., SANDIN, D., SCHULZE, J., SMARR, L., SRINIVASAN, M., WEBER, P., AND WICKHAM, G. 2011. The future of the CAVE. *Central European Journal of Engineering 1,* 1, 16–37.

DHARMAPURIKAR, M. 2013a. Method and mechanism for efficiently delivering visual data across a network. US Patent App. 13/558,163.

DHARMAPURIKAR, M. 2013b. Method and mechanism for performing both server-side and client-side rendering of visual data. US Patent App. 13/349,422.

DIEPSTRATEN, J., GORKE, M., AND ERTL, T. 2004. Remote line rendering for mobile devices. In *Computer Graphics International, 2004. Proceedings*. IEEE, 454–461.

DINH, H. T., LEE, C., NIYATO, D., AND WANG, P. 2011. A survey of mobile cloud computing: architecture, applications, and approaches. *Wireless communications and mobile computing*.

DU, Q. AND FOWLER, J. 2007. Hyperspectral image compression using JPEG2000 and principal component analysis. *IEEE Geoscience and Remote Sensing Letters 4,* 2, 201–205.

DUAN, J. AND LI, J. 2003. Compression of the layered depth image. *IEEE Transactions on Image Processing 12,* 3, 365–372.

DUGUET, F. AND DRETTAKIS, G. 2004. Flexible point-based rendering on mobile devices. *IEEE Comput. Graph. Appl. 24*, 57–63.

EGGER, O., LI, W., AND KUNT, M. 1995. High compression image coding using an adaptive morphological subband decomposition. *IEEE 83,* 2, 272–287.

EILEMANN, S., MAKHINYA, M., AND PAJAROLA, R. 2009. Equalizer: A scalable parallel rendering framework. *IEEE Transactions on Visualization and Computer Graphics 15,* 3, 436–452.

EISERT, P. AND FECHTELER, P. 2008. Low delay streaming of computer graphics. In *Image Processing, 2008. ICIP 2008. 15th IEEE International Conference on*. 2704 –2707.

ENGEL, K., ERTL, T., HASTREITER, P., TOMANDL, B., AND EBERHARDT, K. 2000. Combining local and remote visualization techniques for interactive volume rendering in medical applications. In *Proceedings of the conference on Visualization '00*. VIS '00. IEEE Computer Society Press, Los Alamitos, CA, USA, 449–452.

FUNKHOUSER, T. A. 1995. Ring: a client-server system for multi-user virtual environments. In *Proceedings of the 1995 symposium on Interactive 3D graphics*. I3D '95. ACM, New York, NY, USA, 85–ff.

GAILLY, J. 1992. The gzip home page. http://www.gzip.org.

GARRETT-GLASER, J. 2010. x264: The best low-latency video streaming platform in the world. http://x264dev.multimedia.cx/archives/249.

GUMHOLD, S., GUTHE, S., AND STRABER, W. 1999. Tetrahedral mesh compression with the cut-border machine. In *Proc. of the Conference on Visualization (VIS'99)*. San Francisco, California, 51–58.

GUMHOLD, S. AND STRABER, W. 1998. Real time compression of triangle mesh connectivity. In *Proc. of the 25th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRATH'98)*. Orlando, FL, 133–140.

HAN, B., WU, D., AND ZHANG, H. 2010. Block-based method for real-time compound video compression. In *Proc. of the SPIE Security and Applications of Mobile Multimedia/Image Processing*. Orlando, Florida.

HEGE, H.-C., MERZKY, A., AND ZACHOW, S. 2000. Distributed visualization with opengl vizserver: Practical experiences. Tech. Rep. 00-31, ZIB, Takustr.7, 14195 Berlin.

HERZOG, R., KINUWAKI, S., MYSZKOWSKI, K., AND SEIDEL, H. 2008. Render2mpeg: A perception-based framework towards integrating rendering and video compression. In *Computer Graphics Forum*. Vol. 27. Wiley Online Library, 183–192.

HESINA, G. AND SCHMALSTIEG, D. 1998. A network architecture for remote rendering. In *DIS-RT*. IEEE Computer Society, 88–91.

HOPPE, H. 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.

HP. 2006. Remote graphics software. http://h20331.www2.hp.com/hpsub/cache/286504-0-0-225-121.html.

HUANG, C., HSU, C., CHANG, Y., AND CHEN, K. 2013. GamingAnywhere: An open cloud gaming system. In *Proc. of the ACM Multimedia Systems Conference (MMSys'13)*. Oslo, Norway.

HUANG, J., QIAN, F., GERBER, A., MAO, Z. M., SEN, S., AND SPATSCHECK, O. 2012. A close examination of performance and power characteristics of 4g lte networks. In *Proceedings of the 10th international conference on Mobile systems, applications, and services*. ACM, 225–238.

HUDSON, T. C. AND MARK, W. R. 1999. Multiple image warping for remote display of rendered images. Tech. rep., Chapel Hill, NC, USA.

HUMPHREYS, G., ELDRIDGE, M., BUCK, I., STOLL, G., EVERETT, M., AND HANRAHAN, P. 2001. Wiregl: a scalable graphics system for clusters. In *International Conference on Computer Graphics and Interactive Techniques: Proceedings of the 28 th annual conference on Computer graphics and interactive techniques*. Vol. 2001. 129–140.

HUMPHREYS, G., HOUSTON, M., NG, R., FRANK, R., AHERN, S., KIRCHNER, P. D., AND KLOSOWSKI, J. T. 2002. Chromium: A stream-processing framework for interactive rendering on clusters. *ACM Trans. Graph. 21,* 3, 693–702.

IKONOMOPOULOS, A. AND KUNT, M. 1985. High compression image coding via directional filtering. *Elsevier Signal Processing 8,* 2, 179–203.

JURGELIONIS, A., FECHTELER, P., EISERT, P., BELLOTTI, F., DAVID, H., LAULAJAINEN, J. P., CARMICHAEL, R., POULOPOULOS, V., LAIKARI, A., PERÄLÄ, P., DE GLORIA, A., AND BOURAS, C. 2009. Platform for distributed 3d gaming. *Int. J. Comput. Games Technol. 2009*, 1:1–1:15.

KARNI, Z. AND GOTSMAN, C. 2000. Spectral compression of mesh geometry. In *Proc. of the 22nd Annual Conference on Computer Graphics and Interactive Techniques (SIGGRATH'00)*. New Orleans, Louisiana, 279–286.

KOLLER, D., TURITZIN, M., LEVOY, M., TARINI, M., CROCCIA, G., CIGNONI, P., AND SCOPIGNO, R. 2004. Protected interactive 3d graphics via remote rendering. *ACM Trans. Graph. 23*, 695–703.

KRISHNAMURTHY, R., CHAI, B., TAO, H., AND SETHURAMAN, S. 2001. Compression and transmission of depth maps for image-based rendering. In *IEEE International Conference on Image Processing (ICIP'01)*. Thessaloniki, Greece, 828–831.

KUM, S.-U. AND MAYER-PATEL, K. 2005. Real-time multidepth stream compression. *ACM Trans. Multimedia Comput. Commun. Appl. 1*, 128–150.

KUNT, M., BENARD, M., AND LEONARDI, R. 1987. Recent results in high-compression image coding. *IEEE Transactions on Circuits and Systems 34,* 11, 1306–1336.

LAI, A. M. AND NIEH, J. 2006. On the performance of wide-area thin-client computing. *ACM Trans. Comput. Syst. 24*, 175–209.

LAMBERTI, F. AND SANNA, A. 2007. A streaming-based solution for remote visualization of 3D graphics on mobile devices. *IEEE Trans. Vis. Comput. Graph. 13,* 2, 247–260.

LAMBERTI, F., ZUNINO, C., SANNA, A., ANTONINO, F., AND MANIEZZO, M. 2003. An accelerated remote graphics architecture for pdas. In *Proceedings of the eighth international conference on 3D Web technology*. Web3D '03. ACM, New York, NY, USA, 55–ff.

LAN, C., SHI, G., AND WU, F. 2010. Compress compound images in H.264/MPGE-4 AVC by exploiting spatial correlation. *IEEE Transactions on Image Processing 19,* 4, 946–957.

LAZEM, S., ELTEIR, M., ABDEL-HAMID, A., AND GRACANM, D. 2007. Prediction-based prefetching for remote rendering streaming in mobile virtual environments. In *Signal Processing and Information Technology, 2007 IEEE International Symposium on*. IEEE, 760–765.

LEE, Y. AND SONG, B. 2008. An intra-frame rate control algorithm for ultra low delay H.264/AVC coding. In *Proc. of ICASSP'08*. Las Vegas, NV, 1041–1044.

LEVOY, M. 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.

LI, B., WANG, Z., LIU, J., AND ZHU, W. 2013. Two decades of internet video streaming: A retrospective view. *ACM Trans. Multimedia Comput. Commun. Appl. 9,* 1s, 33:1–33:20.

LI, J. AND KUO, C. 1998. A dual graph approach to 3D triangular mesh compression. In *International Conference on Image Processing (ICIP'98)*. Chicago, Illinois, 891–894.

LI, M., SCHMITZ, A., AND KOBBELT, L. 2011. Pseudo-immersive real-time display of 3d scenes on mobile devices. In *3D Imaging, Modeling, Processing, Visualization and Transmission (3DIMPVT), 2011 International Conference on*. IEEE, 41–48.

LIN, T. AND HAO, P. 2005. Compound image compression for real-time computer screen image transmission. *IEEE Transactions on Image Processing 14,* 8, 993–1005.

LIU, Y., LI, Z., AND SOH, Y. 2007. A novel rate control scheme for low delay video communication of H.264/AVC standard. *IEEE Transactions on Circuits and Systems for Video Technology 1,* 17, 68–78.

LLUCH, J., GAITÁN, R., CAMAHORT, E., AND VIVÓ, R. 2005. Interactive three-dimensional rendering on mobile computer devices. In *Proceedings of the 2005 ACM SIGCHI International Conference on Advances in computer entertainment technology*. ACE '05. ACM, New York, NY, USA, 254–257.

MA, K.-L. 2010. A new approach to remote visualization of large volume data. *SIGGRAPH Comput. Graph. 44,* 3, 5:1–5:2.

MA, K.-L. AND CAMP, D. M. 2000. High performance visualization of time-varying volume data over a wide-area network status. In *Proceedings of the 2000 ACM/IEEE conference on Supercomputing (CDROM)*. Supercomputing '00. IEEE Computer Society, Washington, DC, USA.

MAHESWARI, D. AND RADHA, V. 2010. Enhanced layer based compound image compression. In *Proc. of the Amrita ACM-W Celebration on Women in Computing in India (A2CWiC'10)*. Tamilnadu, India, 40:1–40:8.

MALAKAR, P., NATARAJAN, V., AND VADHIYAR, S. S. 2010. An adaptive framework for simulation and online remote visualization of critical climate applications in resource-constrained environments. In *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE Computer Society, 1–11.

MANN, Y. AND COHEN-OR, D. 1997. Selective pixel transmission for navigating in remote virtual environments. *Comput. Graph. Forum 16,* 3, 201–206.

MARCELLIN, M. W., BILGIN, A., GORMISH, M. J., AND BOLIEK, M. P. 2000. An overview of jpeg-2000. In *Proceedings of the Conference on Data Compression*. DCC '00. IEEE Computer Society, Washington, DC, USA, 523–.

MARK, W. 1999. Post-rendering 3D image warping: Visibility, reconstruction, and performance for depth-image warping. Ph.D. thesis, University of North Carolina at Chapel Hill, Department of Computer Science.

MARTIN, I. M. 2000. Adaptive rendering of 3d models over networks using multiple modalities. Tech. rep., IBM Research.

MARVIE, J. AND BOUATOUCH, K. 2003. Remote rendering of massively textured 3d scenes through progressive texture maps. In *The 3rd IASTED conference on Visualisation, Imaging and Image Processing*. Vol. 2. 756–761.

MCMILLAN, L. 1997. An image-based approach to three dimensional computer graphics. Ph.D. thesis, University of North Carolina at Chapel Hill, Department of Computer Science.

MERKLE, P., MORVAN, Y., SMOLIC, A., FARIN, D., MULLER, K., WITH, P., AND WIEGAND, T. 2009. The effects of multiview depth video compression on multiview rendering. *Elsevier Signal Processing: Image Communication 24,* 1-2, 73–88.

MILANI, S. AND CALVAGNO, G. 2010. A cognitive approach for effective coding and transmission of 3d video. In *Proceedings of the international conference on Multimedia*. MM '10. ACM, New York, NY, USA, 581–590.

MORVAN, Y., FARIN, D., AND WITH, P. 2007. Multiview depth-image compression using an extended h.264 encoder. In *Springer Advances in Image and Video Technology*. Delft, The Netherlands, 675–686.

NADEEM, M., WONG, S., AND KUZMANOV, G. 2010. An efficient realization of forward integer transform in H.264/AVC intra-frame encoder. In *Proc. of SAMOS'10*. Samos, Greece, 71–78.

NAVE, I., DAVID, H., SHANI, A., TZRUYA, Y., LAIKARI, A., EISERT, P., AND FECHTELER, P. 2008. Games@large graphics streaming architecture. In *Consumer Electronics, 2008. ISCE 2008. IEEE International Symposium on*. 1 –4.

NIEH, J., YANG, S. J., AND NOVIK, N. 2003. Measuring thin-client performance using slow-motion benchmarking. *ACM Trans. Comput. Syst. 21*, 87–115.

NOIMARK, Y. AND COHEN-OR, D. 2003. Streaming scenes to mpeg-4 video-enabled devices. *IEEE Comput. Graph. Appl. 23*, 58–64.

NVIDIA. 2009. Reality server. http://www.nvidia.com/object/realityserver.html.

OBERHUMER, M. 1996. Lzo – a real-time data compression library. http://www.oberhumer.com/opensource/lzo/.

OH, H. AND HO, Y. 2006. H.264-based depth map sequence coding using motion information of corresponding texture video. In *Springer Advances in Image and Video Technology*. Hsinchu, Taiwan, 898–907.

OHAZAMA, C. 1999. Opengl vizserver white paper. Silicon Graphics, Inc.

PAIK, S. Microsoft opengl information. http://www.opengl.org/resources/faq/technical/mslinks.htm.

PAJAK, D., HERZOG, R., EISEMANN, E., MYSZKOWSKI, K., AND SEIDEL, H. 2011. Scalable remote rendering with depth and motion-flow augmented streaming. In *Computer Graphics Forum*. Vol. 30. Wiley Online Library, 415–424.

PANTEL, L. AND WOLF, L. C. 2002. On the suitability of dead reckoning schemes for games. In *Proceedings of the 1st workshop on Network and system support for games*. ACM, 79–84.

PARAVATI, G., SANNA, A., LAMBERTI, F., AND CIMINIERA, L. 2011. An adaptive control system to deliver interactive virtual environment content to handheld devices. *Mobile Networks and Applications 16,* 3, 385–393.

PARK, S., KIM, C., AND LEE, S. 2006. Error resilient 3-D mesh compression. *IEEE Transactions on Multimedia 8,* 5, 885–895.

PASMAN, W. AND JANSEN, F. 2003. Comparing simplification and image-based techniques for 3d client-server rendering systems. *Visualization and Computer Graphics, IEEE Transactions on 9,* 2, 226 – 240.

PAUL, B., AHERN, S., BETHEL, W., BRUGGER, E., COOK, R., DANIEL, J., LEWIS, K., OWEN, J., AND SOUTHARD, D. 2008. Chromium renderserver: Scalable and open remote rendering infrastructure. *IEEE Transactions on Visualization and Computer Graphics 14*, 627–639.

PAYAN, F. AND ANTONINI, M. 2002. Multiresolution 3D mesh compression. In *International Conference on Image Processing (ICIP'02)*. Rochester, New York, 245–248.

PENG, J., KIM, C., AND KUO, C. 2005. Technologies for 3D mesh compression: A survey. *Journal of Visual Communication and Image Representation 16,* 6, 688–733.

PENTA, S. K. AND NARAYANAN, P. 2005. Compression of multiple depth maps for ibr. *The Visual Computer 21*, 611–618. 10.1007/s00371-005-0337-8.

PERLMAN, S. G., LAAN, R. V. D., COTTER, T., FURMAN, S., MCCOOL, R., AND BUCKLEY, I. 2010. System and method for multi-stream video compression using multiple encoding formats. US Patent No. 2010/0166068A1.

PHIL KARLTON, PAULA WOMACK, J. L. 2005. Opengl graphics with the x window system (version 1.4). http://www.opengl.org/documentation/specs/.

POPESCU, V., EYLES, J., LASTRA, A., STEINHURST, J., ENGLAND, N., AND NYLAND, L. 2000. The warpengine: An architecture for the post-polygonal age. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*. ACM Press/Addison-Wesley Publishing Co., 433–442.

PROHASKA, S., HUTANU, A., KAHLER, R., AND HEGE, H.-C. 2004. Interactive exploration of large remote micro-ct scans. In *Proceedings of the conference on Visualization '04*. VIS '04. IEEE Computer Society, Washington, DC, USA, 345–352.

QUAX, P., GEUNS, B., JEHAES, T., LAMOTTE, W., AND VANSICHEM, G. 2006. On the applicability of remote rendering of networked virtual environments on mobile devices. In *Systems and Networks Communications, 2006. ICSNC'06. International Conference on*. IEEE, 16–16.

REDDY, H. AND CHUNDURI, R. 2006. MPEG-4 low delay design for HDTV with multi-stream approach. M.S. thesis, Swiss Federal Institute of Technology, Lausanne (EPFL).

REDERT, A., DE BEECK, M. O., FEHN, C., IJSSELSTEIJN, W., POLLEFEYS, M., GOOL, L. J. V., OFEK, E., SEXTON, I., AND SURMAN, P. 2002. Attest: Advanced three-dimensional television system technologies. In *3DPVT*. IEEE Computer Society, 313–319.

RENAMBOT, L., RAO, A., SINGH, R., JEONG, B., KRISHNAPRASAD, N., VISHWANATH, V., CHANDRASEKHAR, V., SCHWARZ, N., SPALE, A., ZHANG, C., GOLDMAN, G., LEIGH, J., AND JOHNSON, A. 2004. SAGE: the scalable adaptive graphics environment. In *Proceedings of the WACE*.

RICHARDSON, T., STAFFORD-FRASER, Q., WOOD, K. R., AND HOPPER, A. 1998. Virtual network computing. *IEEE Internet Computing 2*, 33–38.

ROSS, P. 2009. Cloud computing's killer app: Gaming. *IEEE Spectrum 46,* 3, 14.

RUSINKIEWICZ, S. AND LEVOY, M. 2000. Qsplat: a multiresolution point rendering system for large meshes. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*. SIGGRAPH '00. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 343–352.

RUSINKIEWICZ, S. AND LEVOY, M. 2001. Streaming qsplat: a viewer for networked visualization of large, dense models. In *Proceedings of the 2001 symposium on Interactive 3D graphics*. I3D '01. ACM, New York, NY, USA, 63–68.

SARKIS, M. AND DIEPOLD, K. 2009. Depth map compression via compressed sensing. In *IEEE International Conference on Image Processing (ICIP'09)*. Cairo, Egypt, 737–740.

SCHAFER, R. 1998. MPEG-4: a multimedia compression standard for interactive applications and services. *Electronics and Communication Engineering Journal 10,* 6, 253–262.

SCHEIFLER, R. W. AND GETTYS, J. 1986. The x window system. *ACM Trans. Graph. 5*, 79–109.

SCHMALSTIEG, D. 1997. The remote rendering pipeline - managing geometry and bandwidth in distributed virtual environments. Ph.D. thesis, Institute of Computer Graphics and Algorithms, Vienna University of Technology, Favoritenstrasse 9-11/186, A-1040 Vienna, Austria.

SCHMIDT, B. K., LAM, M. S., AND NORTHCUTT, J. D. 1999. The interactive performance of slim: a stateless, thin-client architecture. In *Proceedings of the seventeenth ACM symposium on Operating systems principles*. SOSP '99. ACM, New York, NY, USA, 32–47.

SCHNEIDER, B.-O. AND MARTIN, I. M. 1999. An adaptive framework for 3d graphics over networks. *Computers & Graphics 23,* 6, 867 – 874.

SCHREIER, R., RAHMAN, A., KRISHNAMURTHY, G., AND ROTHERMEL, A. 2006. Architecture analysis for low-delay video coding. In *Proc. of ICME'06*. Toronto, Canada, 2053–2056.

SERRAL-GRACIÀ, R., CERQUEIRA, E., CURADO, M., YANNUZZI, M., MONTEIRO, E., AND MASIP-BRUIN, X. 2010. An overview of quality of experience measurement challenges for video applications in ip networks. In *Wired/Wireless Internet Communications*. Springer, 252–263.

SEWARD, J. 1996. bzip2 and libbzip2, version 1.0.5: A program and library for data compression. http://www.bzip.org.

SHADE, J., GORTLER, S., HE, L.-W., AND SZELISKI, R. 1998. Layered depth images. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*. SIGGRAPH '98. ACM, New York, NY, USA, 231–242.

SHI, S., HSU, C.-H., NAHRSTEDT, K., HART, J. C., AND CAMPBELL, R. H. 2011a. Using graphics rendering contexts to enhance the real-time video coding for mobile cloud gaming. In *Proceedings of the international conference on Multimedia*. MM '11.

SHI, S., JEON, W. J., NAHRSTEDT, K., AND CAMPBELL, R. H. 2009. Real-time remote rendering of 3d video for mobile devices. In *Proceedings of the 17th ACM international conference on Multimedia*. MM '09. ACM, New York, NY, USA, 391–400.

SHI, S., KAMALI, M., NAHRSTEDT, K., HART, J. C., AND CAMPBELL, R. H. 2010. A high-quality low-delay remote rendering system for 3d video. In *Proceedings of the international conference on Multimedia*. MM '10. ACM, New York, NY, USA, 601–610.

SHI, S., NAHRSTEDT, K., AND CAMPBELL, R. 2012a. A real-time remote rendering system for interactive mobile graphics. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMCCAP) 8,* 3s, 46.

SHI, S., NAHRSTEDT, K., AND CAMPBELL, R. H. 2008. View-dependent real-time 3d video compression for mobile devices. In *Proceeding of the 16th ACM international conference on Multimedia*. MM '08. ACM, New York, NY, USA, 781–784.

SHI, S., NAHRSTEDT, K., AND CAMPBELL, R. H. 2011b. Distortion over latency: Novel metric for measuring interactive performance in remote rendering systems. In *Proceedings of the 2011 IEEE International Conference on Multimedia and Expo, ICME'11*.

SHI, Y., WEN, B., DING, W., QI, N., AND YIN, B. 2012b. Realistic mesh compression based on geometry image. In *Picture Coding Symposium (PCS'12)*. Krakow, Poland, 133–136.

SHUM, H., KANG, S., AND CHAN, S. 2003. Survey of image-based representations and compression techniques. *IEEE Transaction on Circuits and System for Video Technology 13,* 11, 1020–1037.

SINGHAL, S. AND ZYDA, M. 1999. *Networked virtual environments: design and implementation*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA.

SKODRAS, A., CHRISTOPOULOS, C., AND EBRAHIMI, T. 2001. The JPEG 2000 still image compression standard. *IEEE Signal Processing Magazine 18,* 5, 36–58.

SMIT, F. A., VAN LIERE, R., BECK, S., AND FRÖHLICH, B. 2009. An image-warping architecture for vr: Low latency versus image quality. In *VR*. IEEE, 27–34.

SNELL, A. AND WILLARD, C. G. 2007. Ibm deep computing visualization. White Paper: http://www-06.ibm.com/systems/jp/deepcomputing/pdf/idc_white_paper.pdf.

STAADT, O. G., WALKER, J., NUBER, C., AND HAMANN, B. 2003. A survey and performance analysis of software platforms for interactive cluster-based multi-screen rendering. In *Proceedings of the workshop on Virtual environments 2003*. ACM, 261–270.

STEGMAIER, S., DIEPSTRATEN, J., WEILER, M., AND ERTL, T. 2003. Widening the remote visualization bottleneck. In *Image and Signal Processing and Analysis, 2003. ISPA 2003. Proceedings of the 3rd International Symposium on*. Vol. 1. 174 – 179 Vol.1.

STEGMAIER, S., MAGALLÓN, M., AND ERTL, T. 2002. A generic solution for hardware-accelerated remote visualization. In *Proceedings of the symposium on Data Visualisation 2002*. VISSYM '02. Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 87–ff.

TECHNOLOGY, M. I. 2007. Thinanywhere. http:// www.thinanywhere.com.

TELER, E. AND LISCHINSKI, D. 2001. Streaming of complex 3d scenes for remote walkthroughs. *Comput. Graph. Forum 20,* 3.

THOMAS, G., POINT, G., AND BOUATOUCH, K. 2005. A client-server approach to image-based rendering on mobile terminals. Tech. Rep. RR-5447, INRIA. January.

TIKHONOVA, A., CORREA, C., AND MA, K.-L. 2010. Explorable images for visualizing volume data. In *Proceedings of IEEE Pacific Visualization Symposium (PacificVis)*. 177–184.

TIZON, N., MORENO, C., CERNEA, M., AND PREDA, M. 2011. Mpeg-4-based adaptive remote rendering for video games. In *Proceedings of the 16th International Conference on 3D Web Technology*. ACM, 45–50.

TORBORG, J. AND KAJIYA, J. T. 1996. Talisman: commodity realtime 3d graphics for the pc. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*. SIGGRAPH '96. ACM, New York, NY, USA, 353–363.

TOUCH, J. 1995. Defining high-speed protocols: five challenges and an example that survives the challenges. *Selected Areas in Communications, IEEE Journal on 13,* 5, 828–835.

TRAN, T., LIU, L., AND WESTERINK, P. 1998. Low-delay MPEG-2 video coding. In *Proc. of VCIP'98*. San Jose, CA, 510–516.

TU, T., YU, H., RAMIREZ-GUZMAN, L., BIELAK, J., GHATTAS, O., MA, K.-L., AND O'HALLARON, D. R. 2006. From mesh generation to scientific visualization: An end-to-end approach to parallel supercomputing. In *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*. ACM, 91.

VASA, L. AND SKALA, V. 2007. CODDYAC: Connectivity driven dynamic mesh compression. In *Proc. of 3DTV Conference*. KosIsland, Greece, 1–4.

VETRO, A., WIEGAND, T., AND SULLIVAN, G. 2011. Overview of the stereo and multiview video coding extensions of the h.264/mpeg-4 avc standard. *Proceedings of the IEEE 99,* 4, 626 –642.

WANG, S. AND DEY, S. 2009. Modeling and characterizing user experience in a cloud server based mobile gaming approach. In *Global Telecommunications Conference, 2009. GLOBECOM 2009. IEEE*. IEEE, 1–7.

WANG, S. AND DEY, S. 2012. Cloud mobile gaming: modeling and measuring user experience in mobile wireless networks. *ACM SIGMOBILE Mobile Computing and Communications Review 16,* 1, 10–21.

WANG, S. AND DEY, S. 2013. Adaptive mobile cloud computing to enable rich mobile multimedia applications. *Multimedia, IEEE Transactions on 15,* 4, 870–883.

WANG, S. AND LIN, T. 2009. A unified LZ and hybrid coding for compound image partial-lossless compression. In *Proc. of International Congress on Image and Signal Processing (CISP'09)*. 1–5.

WANG, S. AND LIN, T. 2012. United coding method for compound image compression. *Multimedia Tools and Application*.

WANG, Y., OSTERMANN, J., AND ZHANG, Y. 2001. *Video Processing and Communications* 1st Ed. Prentice Hall.

WANG, Z., BOVIK, A., SHEIKH, H., AND SIMONCELLI, E. 2004. Image quality assessment: from error visibility to structural similarity. *Image Processing, IEEE Transactions on 13,* 4, 600–612.

WEINBERGER, M., SEROUSSI, G., AND SAPIRO, G. 1996. LOCO-I: A low complexity, context-based, lossless image compression algorithm. In *Proc. of Data Compression Conference (DCC'96)*. Snowbird, Utah, 140–149.

WIEGAND, T., SULLIVAN, G., BJNTEGAARD, G., AND LUTHRA, A. 2003. Overview of the H.264/AVC video coding standard. *IEEE Transactions on Circuits and Systems for Video Technology 13,* 7, 560–576.

WOO, M., NEIDER, J., DAVIS, T., AND SHREINER, D. 1999. *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 1.2* 3rd Ed. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

WOODWARD, C., VALLI, S., HONKAMAA, P., AND HAKKARAINEN, M. 2002. Wireless 3d cad viewing on a pda device. In *Proceedings of the 2nd Asian International Mobile Computing Conference (AMOC 2002)*. Vol. 14. 17.

YANG, S. J., NIEH, J., SELSKY, M., AND TIWARI, N. 2002. The performance of remote display mechanisms for thin-client computing. In *Proceedings of the General Track of the annual conference on USENIX Annual Technical Conference*. USENIX Association, Berkeley, CA, USA, 131–146.

YANG, Z., CUI, Y., ANWAR, Z., BOCCHINO, R., KIYANCLAR, N., NAHRSTEDT, K., CAMPBELL, R. H., AND YURCIK, W. 2006. Real-time 3d video compression for tele-immersive environments. In *Proc. of SPIE/ACM Multimedia Computing and Networking (MMCN'06)*.

YOO, W., SHI, S., JEON, W. J., NAHRSTEDT, K., AND CAMPBELL, R. H. 2010. Real-time parallel remote rendering for mobile devices using graphics processing units. In *ICME*. IEEE, 902–907.

YOON, I. AND NEUMANN, U. 2000. Web-based remote rendering with ibrac (image-based rendering acceleration and compression). *Comput. Graph. Forum 19,* 3, 321–330.

YU, H., MA, K.-L., AND WELLING, J. 2004. A parallel visualization pipeline for terascale earthquake simulations. In *Proceedings of the 2004 ACM/IEEE conference on Supercomputing*. IEEE Computer Society, 49.

ZANUTTIGH, P. AND CORTELAZZO, G. 2009. Compression of depth information for 3D rendering. In *Proc of 3DTV Conference: The True Vision - Capture, Transmission and Display of 3D Video*. Potsdam, Germany, 1–4.

ZHOU, H. 2006. A survey on ubiquitous graphics. Tech. rep., Hong Kong University of Science and Technology. From http://www.cse.ust.hk/ zhouhong/publications.html/.