A Real-Time Remote Rendering System for Interactive Mobile Graphics

SHU SHI, KLARA NAHRSTEDT, and ROY CAMPBELL, University of Illinois at Urbana-Champaign

Mobile devices are gradually changing people's computing behaviors. However, due to the limitations of physical size and power consumption, they are not capable of delivering a 3D graphics rendering experience comparable to desktops. Many applications with intensive graphics rendering workloads are unable to run on mobile platforms directly. This issue can be addressed with the idea of remote rendering: the heavy 3D graphics rendering computation runs on a powerful server and the rendering results are transmitted to the mobile client for display. However, the simple remote rendering solution inevitably suffers from the large interaction latency caused by wireless networks, and is not acceptable for many applications that have very strict latency requirements.

In this article, we present an advanced low-latency remote rendering system that assists mobile devices to render interactive 3D graphics in real-time. Our design takes advantage of an image based rendering technique: 3D image warping, to synthesize the mobile display from the depth images generated on the server. The research indicates that the system can successfully reduce the interaction latency while maintaining the high rendering quality by generating multiple depth images at the carefully selected viewpoints. We study the problem of viewpoint selection, propose a real-time reference viewpoint prediction algorithm, and evaluate the algorithm performance with real-device experiments.

Categories and Subject Descriptors: H.5.1 [**Information Interfaces and Presentation**]: Multimedia Information Systems—*Video*

General Terms: Design, Measurement

Additional Key Words and Phrases: Remote rendering, interaction latency, 3D image warping, mobile devices

ACM Reference Format:

Shi, S., Nahrstedt, K., and Campbell, R. 2012. A real-time remote rendering system for interactive mobile graphics. ACM Trans. Multimedia Comput. Commun. Appl. 8, 3s, Article 46 (September 2012), 20 pages.
DOI = 10.1145/2348816.2348825 http://doi.acm.org/10.1145/2348816.2348825

1. INTRODUCTION

The recent explosion of mobile devices is changing people's computing behaviors. While more and more applications are ported to mobile platforms, some 3D graphics applications that require intensive computation or network bandwidth are not capable of running on mobile yet. Some good examples include the state of art 3D video games (e.g., Electronic Games [2007]), the visualization of very complex 3D

This work was supported by the National Science Foundation under Grant CNS 05-20182 and CNS 07-20702.
Authors' present addresses: S. Shi, Ricoh Innovations, Inc., 2882 Sand Hill Road, Suite 115, Menlo Park, CS 94025-7057; email: shushi@rii.ricoh.com; K. Nahrstedt and R. Campbell, University of Illinois at Urbana-Champaign, 201 N. Goodwin Ave., Urbana, IL 61801; email: {klara, rhc}@illinois.edu.

DOI 10.1145/2348816.2348825 http://doi.acm.org/10.1145/2348816.2348825

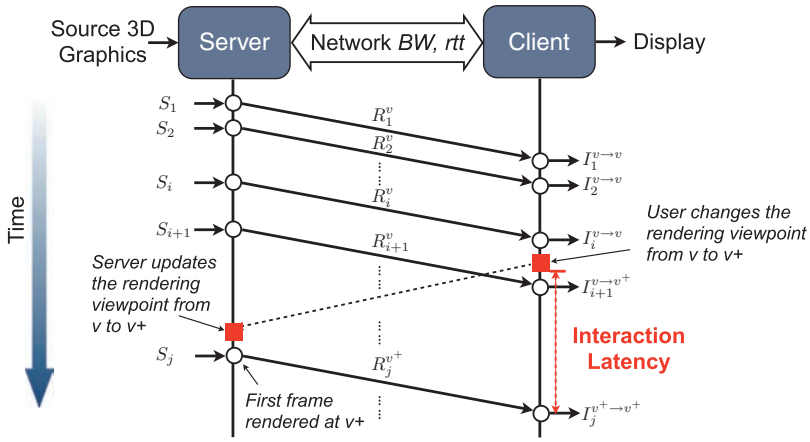

Fig. 1. The framework of a remote rendering system and an illustration of interaction latency.

models [ScanView 2003], the 3D tele-immersive system that renders the 3D video of real-world objects captured by multiple depth cameras [Kurillo et al. 2008], and so on. These applications share two features: (1) users can interact with the application by changing the rendering viewpoint; (2) the 3D graphics contents are very complex and rely on a powerful GPU to render in real-time.

Although the mobile GPU has been adequately optimized over years for lightweight graphics rendering, it is still far away from providing a similar 3D rendering experience to desktops. Due to the restrictions of physical size and power consumption, it is very difficult to fill this gap even in the near future. Therefore, how to render heavyweight 3D graphics applications on mobile devices remains a big challenge. Moreover, the wireless networks available for most mobile devices have limited network bandwidth. Streaming a large amount of 3D data (e.g., game scenes, 3D video frames, etc.) to mobile devices can easily overwhelm the limited wireless bandwidth.

Remote rendering is a simple but effective solution. A workstation with enough computing and networking resources (e.g., a cloud server) is used as a rendering server. It renders 3D graphics contents and sends the extracted result images to mobile clients. The client can simply display the received 2D images. Figure 1 shows an illustration. The idea of remote rendering has a history. It appeared early when PC was not powerful enough to render 3D graphics [Azuma 1997; Singhal and Zyda 1999]. Sharing a dedicated graphics workstation over network to provide rendering services stimulated the research of remote rendering [Ohazama 1999]. Nowadays, the same approach can be applied to assist 3D graphics rendering on mobile devices. The state-of-art cloud gaming service represented by OnLive[1] renders the most advanced 3D video games in the cloud and streams the game scenes as video to game players.

However, for the applications with frequent user interactions, such a remote rendering framework brings a new problem of *interaction latency*, which in our research is defined as the time from the generation of user interaction request till the appearance of the first updated frame on the mobile client. From Figure 1, it indicates that the interaction latency should be no less than the network round trip time between the rendering server and the client. The long latency can significantly impair the user experience in many application scenarios [Beigbeder et al. 2004]. For example, the cloud gaming service hosted by OnLive requires an interaction latency less than 80 ms so that the game player does not notice the delay. However, the interaction latency of OnLive streaming completely depends on the

[1]OnLive: http://www.onlive.com.

network condition. When cellular networks are used to stream game videos, the interaction latency can hardly meet the requirement the intrinsic network latency is already beyond the limit.[2]

In this article, we propose a real-time remote rendering system for interactive mobile graphics. The server of the proposed system sends the carefully selected depth images to the mobile client. On the client side, if any user interaction changes the rendering viewpoint, the mobile client runs 3D image warping with the received depth images to synthesize an image at the updated rendering viewpoint, and displays this synthesized image before receiving any updates from server. Therefore, the interaction latency of the proposed remote rendering system is reduced to the time of image synthesis on mobile, which is independent of network. This article makes two major contributions. First, we study the remote rendering system from a new perspective and transform the network-related latency reduction problem into a content-based reference prediction problem. Second, we find a computation efficient method to create an image-based representation of complex graphics models, which is sufficient to support high-quality rendering of the original model when the rendering viewpoint changes in a limited range.

The major difference between the proposed system and other remote rendering systems, like OnLive, is the data generated on the server. Instead of sending one color image per frame to the client, our system renders the 3D scene from multiple rendering viewpoints and sends multiple depth images to the client. The extra information allows the client to create high quality synthesis locally when the viewpoint is changed. The interaction latency is reduced significantly at the cost of extra computation on the server and more network bandwidth for streaming all extra depth images.

For the rest of this article, we first propose a simple model for remote rendering systems (Section 2), and use the model to analyze different remote rendering designs in the history (Section 3). Then we present our depth image based remote rendering system, explain how to transform the latency reduction problem into a reference prediction problem, and discuss a couple of reference prediction algorithms in Section 4. The experiment results are evaluated in Section 5. At last, we discuss, the future work and conclude this article in Section 6.

2. REMOTE RENDERING MODEL

In this section, we build a model to characterize the key components in designing a remote rendering system. Table I summarizes the symbols used in this article.

For every 3D scene, the server takes S as input, processes some operations, and generates a result R^{v_s}:

$$R^{v_s} = proc\_s(S, v_s),$$

where v_s is the rendering viewpoint maintained on the server. R^{v_s} is usually encoded before network streaming

$$Rc^{v_s} = enc(R^{v_s}).$$

The encoded data is sent into the network, that has a bandwidth BW and a round-trip time rtt.[3] On the other side, the client runs decoding to retrieve the data. In this article, we assume that the distortion

[2]The measurement studies [Riva and Kangasharju 2008; Marquez et al. 2008] reported 700+, 300+, and 200+ ms latencies in GPRS (2G), EDGE (2.5G), and UMTS (3G) cellular networks. Although the latest 4G LTE network provides low-latency connections, such video streaming approaches may still suffer from the latency issue when the signal is weak or the network is congested.

[3]For simplicity, we add all network related factors to rtt, such as the retransmission due to packet loss, control overheads, the latency caused by traffic congestion, the waiting time in a streaming buffer, etc.

Table I. Notations

Symbol	Description
S	The source 3D content frame. For the applications like 3D video that have sequential frames (Figure 1), the subscript i is used to identify different frames.
v	A rendering viewpoint. v^+ denotes a different rendering viewpoint from v. v_s and v_c are specifically used to denote the rendering viewpoint managed on the server and client sides, respectively.
R^v	A rendering server output frame. Depending on which remote rendering approach is taken, the output can have different representations.
Rc^v	The encoding result of R^v.
I^v	A rendering result image frame. It is generated by rendering the source 3D content S at the rendering viewpoint v.
$I^{v \to v^+}$	It is generated by processing R^v at the viewpoint v^+. The exact process depends on which remote rendering approach is taken.
D^v	A rendering result depth frame. It is generated by rendering the source 3D content S at the rendering viewpoint v, and extracting the pixel depth value from *z-buffer*.
$W^{v \to v^+}$	A warping result frame. It is generated by warping depth image $\langle I_i^v, D_i^v \rangle$ to a different rendering viewpoint v^+ with the 3D image warping algorithm.
BW	The bandwidth of the network connection.
rtt	The round trip time of the network connection.
FPS	The frame rate. FPS_s is the frame rate of source rendering and FPS_c is the frame rate of client display.
err_{disp}	The threshold of the image difference when evaluating the rendering quality.
err_{resp}	The threshold of the image difference when evaluating the response time.

caused by encoding/decoding is neglectable. So we have:

$$R^{v_s} = dec(Rc^{v_s})$$

After the client has R^{v_s}, it needs to generate the display image:

$$I^{v_s \to v_c} = proc\_c(R^{v_s}, v_c)$$

where v_c is the rendering viewpoint maintained on the client. For most time, v_c should be equal to v_s. However, as illustrated in Figure 1, there exists a period of viewpoint inconsistency when user interaction happens.

This simplified model can be used to analyze system performance. In our research of latency reduction, we mainly focus on two requirements in designing a remote rendering system: (1) the client is expected to present a good quality rendering result, as if the client has the same computing resources as the server; (2) reduce the interaction latency so that the system should respond to user interactions as fast as possible. These two requirements can be represented by two metrics using our remote rendering model.

The first metric *rendering quality* refers to the rendering quality when there is no viewpoint inconsistency (i.e., $v_s = v_c = v$). We use $\mathbf{Q}$ to define the metric as following:

$$\mathbf{Q} = diff(I^{v_s \to v_c}, I^{v_c}),$$

where I^v is the result image of rendering S at the viewpoint v. $diff(I_1, I_2)$ returns the difference of two images. In our research, we calculate the mean square error (MSE) of two input images

$$I^v = render(S, v).$$

Please note that I^v may not be actually generated in the system, but only used to evaluate the *rendering quality* in our model. In order to present a good rendering quality on the client, we expect $\mathbf{Q_i}$ to be

zero or at least smaller than a threshold

$$\mathbf{Q} < err_{disp}. \tag{1}$$

The second metric *average response time* reflects how fast the system is responsive to the user interaction that changes the rendering viewpoint (i.e., the user interaction changes the rendering viewpoint on the client from v to v^+. Before the update request is received by the server, $v_s = v$, $v_c = v^+$). We define the *response time* T_{resp} as following:

$$T_{resp}(v \rightarrow v^+) = \begin{cases} T(proc\_c) & \text{if } diff(I^{v \rightarrow v^+}, I^{v^+}) < err_{resp} \\ T(proc\_c) + T(proc\_s) + T(enc) + T(dec) + rtt + \frac{\|Rc^{v^+}\|}{BW} & \text{else,} \end{cases} \tag{2}$$

where the function $T(func)$ returns how long it takes to execute $func$, and $\|Rc^{v^+}\|$ is the frame size. When the user interaction changes the client rendering viewpoint from v to v^+, the client should update the viewpoint change to server. At the same time, the client can run $proc\_c$ again based on the existing R_i^v and generate $I^{v \rightarrow v^+}$. If the image $I^{v \rightarrow v^+}$ is already similar to the standard rendering result I^{v^+}, we consider the *response time* is only the execute time of $proc\_c$. Otherwise, the client needs to wait until the updated frame Rc^{v^+} is sent back from server. This is the interaction latency problem we mentioned in Section 1. Based on Eq. (2), we can derive the *average response time* as $\mathbf{E}(\mathbf{T_{resp}})$:

$$\mathbf{E}(\mathbf{T_{resp}}) = T(proc\_c) + \mathbf{p} \cdot \left(T(proc\_s) + T(enc) + T(dec) + rtt + \frac{E(\|Rc\|)}{BW} \right), \tag{3}$$

where $E(\|Rc\|)$ is the average size of the data frames generated on the server and $\mathbf{p}$ is the *penalty probability* that $I^{v \rightarrow v^+}$ is not similar to I^{v^+}

$$\mathbf{p} = P(diff(I^{v \rightarrow v^+}, I^{v^+}) \geq err_{resp}) \tag{4}$$

The goal is to have the *average response time* less than the largest tolerable latency $delay_{max}$. From Eq. (3), we find that the most effective method to reduce $\mathbf{E}(\mathbf{T_{resp}})$ is to reduce the *penalty probability* $\mathbf{p}$. If $\mathbf{p}$ can be reduced to zero, the interaction latency becomes independent of network (rtt). Therefore, we can present the latency requirement as:

$$T(proc\_c) < delay_{max} \quad \text{and} \quad \mathbf{p} \rightarrow 0. \tag{5}$$

Furthermore, for the rendering applications that dynamically generate source data and refresh screen at a high frame rate (e.g., games, 3D video, etc.), there are a few more timing requirements. Assuming FPS_s is the frame rate of source data generation and FPS_c is the refresh rate on the client (FPS_s should be equal or larger than FPS_c), we have:

$$T(proc\_s) < \frac{1}{FPS_s}, \quad T(proc\_c) < \frac{1}{FPS_c}, \quad \frac{\|Rc^v\|}{BW} < \frac{1}{FPS_c}. \tag{6}$$

In the next section, we use our model to analyze different remote rendering systems proposed in the history, and compare their performance in latency reduction based on the requirements defined in Eqs. (1), (5), and (6).

3. REMOTE RENDERING SYSTEM ANALYSIS

The server-client based remote rendering and remote visualization systems proposed in the literatures can be classified into two major categories according to what type of data is generated on the server: the model-based approach and the image-based approach (Figure 2). In this section, we analyze these approaches with the remote rendering model introduced in the previous section and compare their pros and cons.

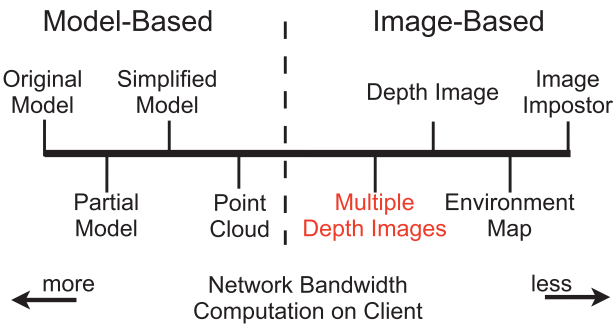

Fig. 2. Classify remote rendering and remote visualization systems based on data types.

3.1 Model-Based Approach

We consider the remote visualization system that sends 3D models (triangle meshes or point clouds) to the client a model-based approach. The server of this design shares the computation of generating and processing source data while the client needs powerful hardware for 3D graphics rendering.

Original Model. For this approach, the application runs on the server, but all 3D models are forwarded to the client for rendering. It is useful for the application that requires heavy computation in generating rather than rendering these models. This approach can be implemented within system libraries so that any graphics applications can be easily ported to a server-client based remote visualization system without modifying any source code. GLX [Karlton et al. 2005], Game@Large [Eisert and Fechteler 2008], and THiNC [Baratto et al. 2005] are good examples.

Partial Model. One problem of Original Model is that for the scenes with very complex geometry and textures, it takes excessive network bandwidth and long time to transmit all 3D models before the client can start rendering. However, in many scenarios, only a subset of the original model is actually needed. The systems introduced in Engel et al. [1999] have been designed for the remote visualization of medical volume data. The server only sends the client a slice data of the whole volume every time depending on user attention. Schmalstieg [1997] is another example of rendering virtual environment for remote walk-through. The 3D models of the whole virtual environment can be divided into zones based on the user's position in the virtual world. The zone directly viewable to the user should be transmitted at the full resolution with high priority; the zone close to the currently viewable zone can be streamed at the reduced resolution; and the zone far away from the current viewpoint may be transmitted later if there is not enough network bandwidth. Compared with *Original Model*, this approach can release the network contention and reduce the start time on the client side at the cost of performing extra computation on the server side to divide models into subsets.

Simplified Model. This approach is proposed to target "thin" client scenarios. The system introduced in Levoy [1995] sends the client a simplified model and an image of the rendering difference between the simplified model and the original model. The client renders the simplified model and adds the difference image to the rendering result. The system needs only a lightweight rendering engine on the client, requires much less network bandwidth for streaming models, and maintains the same rendering quality. In return for the benefits, the server is heavily loaded to generate a simplified model, render both models and compare the difference.

Point Cloud. This is a special case of Simplified Model to generate a point cloud representation of the original model [Duguet and Drettakis 2004]. The idea was proposed to target the mobile clients

with small display screens. The size of the original model is proportional to the number of the polygons in the mesh, which can be far more than the available display pixels for complex scenes. The point cloud generated on the server, on the contrary, has a size comparable to the display screen resolution and therefore can be much easier for streaming and rendering.

3.2 Image-Based Approach

An image-based remote rendering system renders all 3D models on the server and sends images to the client. Unlike the model-based approaches, the client does not need any graphical hardware for 3D graphics rendering.

Image Impostor. Most remote rendering systems [Ohazama 1999; Lamberti and Sanna 2007; Noimark and Cohen-Or 2003] take this simple approach. The server renders all 3D models and transmits the rendering results as 2D images to the client. The client only displays the received images on the screen and sends the user interaction requests back to the server. Considering the image impostor only needs a resolution comparable to the display resolution, the required network bandwidth for streaming images is bounded regardless of the complexity of original 3D models. Moreover, the 2D images can be efficiently compressed with the advanced image and video coding tools to save streaming bandwidth. The only drawback of this approach is the interaction latency issue explained in Section 1.

Environment Map. Environment map has been widely used in 3D game development to simplify the rendering of far away background objects. In the scenario of rendering virtual environment [Chen 1995; Boukerche and Pazzi 2006], it can be effectively applied to reduce the interaction latency. Given the position of the viewpoint, the server generates a panoramic environment map with a 360-degree coverage of the whole world. On the client side, the environment map can be projected to the standard view image at any view direction, which allows the user to freely pan the virtual camera without further rendering assistance from server. Therefore, the latency for any panning motion only depends on the time of running view projection on the client. But for other user interaction that changes the viewpoint position, the client still needs to wait for the server to send new environment maps.

Depth Image. In this approach, the server renders the 3D models and generates a depth image (with both color map and depth map). When the client receives this depth image, it can directly display the color map if the rendering viewpoint remains unchanged. Otherwise, the client runs 3D image warping [McMillan 1997] on the received depth image, and display a synthesized image immediately after the user changes viewpoint. This approach can be considered as a simple version of *Point Cloud* because every pixel of the depth image represents a 3D point in the space. The difference is that running 3D image warping requires much less computation than rendering a point cloud with 3D graphics pipeline. This approach has been applied in several systems: [Chang and Ger 2002; Bao and Gourlay 2004; Smit et al. 2009; Zhu et al. 2011].

3.3 Model Analysis

We analyze these remote rendering approaches with the model introduced in Section 2 to find the best solution for rendering interactive graphics on mobile devices. Table II lists the details.

Original Model performs perfectly in rendering quality because it is actually a local rendering approach. However, $T(proc\_c)$ depends on how fast the client can render the whole 3D model. It does not meet the requirement of Eq. (5) if the mobile client is not capable of rendering complex 3D graphics. Partial Model has the same problem since the complexity of 3D models is not reduced. Simplified Model and Point Cloud reduce the complexity of 3D models for the client to render, but may not meet real-time requirements because the computation workload on the server side is increased. Moreover,

Table II. Summary of Remote Rendering Systems

System Design	Server Operation: proc_s	ClientOperation :proc_c
Original Model	return S	$render(S, v_c)$
Partial Model	generate $S' = subset(S, vs)$	$render(S', vc)$
Simplified Model	generate $\langle \hat{S}, I_\Delta \rangle$, $\hat{S} = simplify(S, v_s)$, $I_\Delta = render(S, v_s) - render(\hat{S}, v_s)$	if $v_c = v_s$ $render(\hat{S}, v_c) + I_\Delta$ else $render(\hat{S}, v_c)$
Point Cloud	generate $P = point\_cloud(S, v_s)$	$render(P, v_c)$
Image Impostor	generate $I^{v_s} = render(S, v_s)$	return I^{v_s}
Environment Map	generate $E^{v_s} = env\_map(S, v_s)$	if $v_c.pos = v_s.pos$ $project(E^{v_s}, v_c)$ else do not update display
Depth Image	generate $\langle I^{v_s}, D^{v_s} \rangle$, $I^{v_s} = render(S, v_s)$, D^{v_s} is the depth map of I^{v_s}	if $v_c = v_s$ return I^{v_s} else $warping(\langle I^{v_s}, D^{v_s} \rangle, v_s \rightarrow v_c)$

System Design	Server Proc. Time: T(proc_s)	Client Proc. Time: T(proc_c)
Original Model	0	$T(render)$, depending on S complexity
Partial Model	$T(subset)$, depending on the complexity of S and the subset selection algorithm	$T(render)$, depending on S' complexity
Simplified Model	$T(simplification) + 2 \cdot T(render)$, depending on the complexity of S and the model simplification algorithm	$T(render)$, depending on $\hat{S}$ complexity
Point Cloud	$T(point\_cloud)$, depending on the complexity of S and the point cloud generation algorithm	$T(render)$, depending on P complexity
Image Impostor	$T(render)$	0
Environment Map	$T(env\_map) \propto T(render)$ (e.g, $T(cubical\_env\_map) = 6 \cdot T(render)$)	$T(project) \propto client\_resolution$
Depth Image	$T(render)$	$T(warping) \propto client\_resolution$

System Design	Rendering Quality: Q_i	Penalty Probability: p
Original Model	0	0
Partial Model	depending on how S' is created	$P(diff(render(S', v_c), I^{v_c}) \geq err_{resp})$
Simplified Model	0	$P(diff(render(\hat{S}, v_c), I^{v_c}) \geq err_{resp})$
Point Cloud	depending on how P is created	$P(diff(render(P, v_c), I^{v_c}) \geq err_{resp})$
Image Impostor	0	1
Environment Map	0	$P(v_c.pos \neq v_s.pos)$
Depth Image	0	$P(v_c \notin V_{single})$, where $V_{single} = \{v_i \mid diff(W^{v_s \rightarrow v_i}, I^{v_i}) < err_{resp}\}$

the performance and complexity of 3D simplification algorithms can vary significantly for different source models while two core metrics ($\mathbf{Q}$ and $\mathbf{E}(\mathbf{T_{resp}})$) both rely on whether the simplified model has the similar rendering result to the original model. In some cases, Simplified Model and Point Cloud may need to pre-process the model data offline to create the required simplified models.

All image-based approaches are perfect for rendering quality because the image rendered on the server can be directly displayed if there is no viewpoint change. $T(proc\_c)$ can also meet the requirement since the computation complexity only depends on the display resolution. Therefore, we only look at the *penalty probability* $\mathbf{p}$, and see whether it can be reduced to zero.

Image Impostor has the highest penalty probability. If we assume the image rendered at different viewpoints are different,

$$diff(I^{v_s}, I^{v_c}) > err_{disp},\ if\ v_s \neq v_c$$

Then, $\mathbf{p} = 1$ and every user interaction suffers a full network round trip delay. *Environment Map* only performs well for the user interaction that does not change the viewpoint position, $\mathbf{p} = P(v_c.pos \neq v_s.pos)$. The penalty probability of the Depth Image approach is determined by the quality of 3D image warping. We use $\langle I^v, D^v \rangle$ to denote the depth image rendered at the viewpoint v and $W^{v \rightarrow v^+}$ to denote

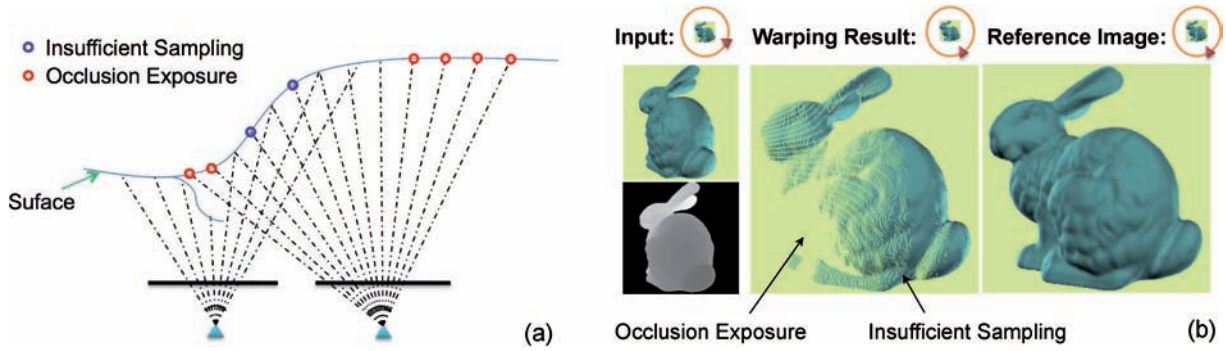

Fig. 3. (a) Illustrate the generation of *occlusion exposure* and *insufficient sampling*; (b) Example of two type of holes in a warping result image.

the result image of warping $\langle I^v, D^v \rangle$ to a new viewpoint v^+

$$W^{v \to v^+} = warping(\langle I^v, D^v \rangle, v \to v^+).$$

We define a viewpoint set V_{single} as:

$$V_{single} = \{v_i \mid diff(W^{v_s \to v_i}, I^{v_i}) < err_{resp}\}. \quad (7)$$

If the user interaction changes v_c to any viewpoint within V_{single}, the warping result is similar to rendering the original model at this viewpoint. According to Eq. (2), $T_{resp} = T(warping)$. So $\mathbf{p} = P(v_c \notin V_{single})$. If V_{single} is large enough to cover all the viewpoints that the user interaction can possibly change to, $\mathbf{p}$ is reduced to zero.

4. SYSTEM DESIGN

In this section, we present the design of our low-latency real-time remote rendering system.

4.1 Warping Artifacts Removal

From the previous analysis, Depth Image is more favorable than other approaches. However, the average response time of Depth Image mainly depends on whether V_{single} is large enough to cover all user motions. According to the definition in Eq. (7), the viewpoint set V_{single} can be enlarged by removing 3D image warping artifacts.

3D image warping is a classic image-based rendering technique proposed in McMillan and Bishop [1995]. The algorithm takes a depth image (including both color and depth maps), the viewpoint at which the depth image is rendered, and the target viewpoint as input. The output is the color image at the target viewpoint. For every pixel (u_1, v_1) in the input image, the algorithm can very efficiently calculate the new coordinate (u_2, v_2), and then copy the pixel color to the new position in the target image. Refer to McMillan [1997] for more details.

"Hole" artifacts are inevitably introduced by 3D image warping because the output image can only use the pixels from the input image. There are actually two types of holes. The first is caused by *occlusion exposure*: the objects that are completely occluded or out of view range in the input image become exposed. The second type of holes is generated due to *insufficient sampling*, which usually happens on a surface with varying gradient. Figure 3 shows an example of these two types of warping errors.

Many hole filling methods have been developed. Depth filter [Redert et al. 2002] and splats [Mark 1999] can help fill the small holes caused by insufficient sampling. Super view warping [Bao and

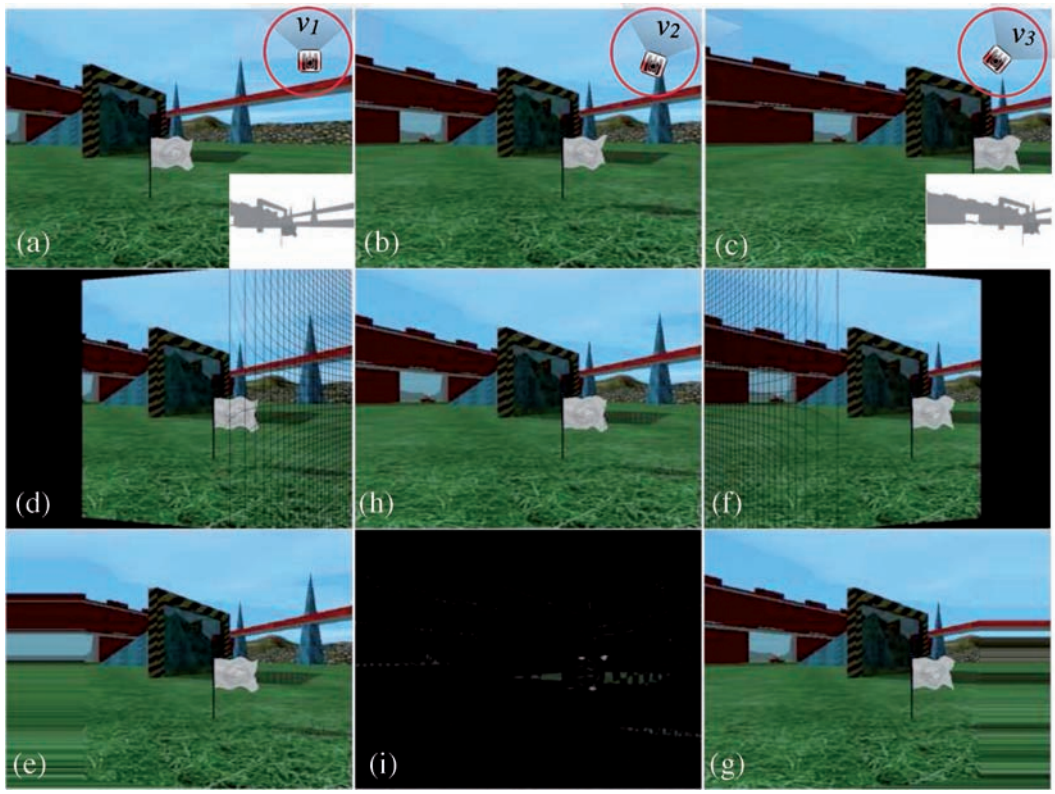

Fig. 4. (a) the depth image frame $\langle I^{v_1}, D^{v_1}\rangle$; (b) the image frame I^{v_2}; (c) the depth image frame $\langle I^{v_3}, D^{v_3}\rangle$; (d) $W^{v_1\to v_2}$ without hole filling; (e) $W^{v_1\to v_2}$ with hole filling; (f) $W^{v_1\to v_3}$ without hole filling; (g) $W^{v_1\to v_3}$ with hole filling; (h) the warping result of double warping $W^{v_1\to v_2} \bigcup W^{v_1\to v_3}$, with hole filling; (i) the difference between (h) and (b).

Gourlay 2004] and wide field-of-view warping [Mark 1999] are useful for out-of-view exposure. Multiple references [Mark 1999], view compensation [Bao and Gourlay 2004], and layered depth image (LDI) [Shade et al. 1998] all use the images from different viewpoints to compensate warping artifacts and can be used to fill either type of holes.

In our remote rendering system, we select the method of multiple references to remove warping artifacts because it does not need offline processing (LDI) or interactive communication (viewpoint compensation). Figure 4 illustrates how two references can significantly improve the warping result quality. The graphics scenes are extracted from a 3D tank shooting game BZFlag.[4]

4.2 System Overview

Now we present our *Multi Depth Image* design: a remote rendering system that generates multiple depth images to help mobile clients rendering interactive graphics in real-time. Figure 5 shows the system framework.

On the server side, a reference viewpoint selection module selects a viewpoint set $\{ref_k\}$ (we let $ref_0 = v_s$ all the time). The rendering engine not only needs to render the source 3D data at ref_0 (v_s), but at all viewpoints in $\{ref_k\}$, so that we get a set of depth images $\{\langle I^{ref_k}, D^{ref_k}\rangle\}$. On the client side, if

[4]BZFlag: http://www.bzflag.org.

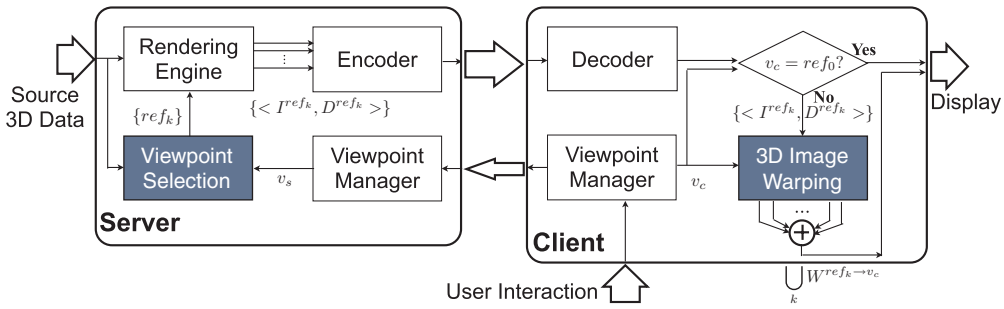

Fig. 5. The system framework of *Multiple Depth Images* approach.

no user interaction happens, or $v_c = ref_0$, then I^{ref_0} is directly displayed on the screen. Otherwise, the client warps every received depth image to the new viewpoint and composites all the warping results for a display image. The process is represented by: $\bigcup_k W^{ref_k \to v_c}$. The warping result is expected to have much less holes if the reference viewpoints are appropriately selected (Figure 4).

The model analysis shows that the proposed system meets the design requirements. Like other image-based remote rendering systems, $\mathbf{Q}$ is always 0. Even though it takes more time to warp multiple depth images on the client, $T(proc\_c)$ is still proportional to the display resolution rather than the complexity of source 3D models. Similar to the Depth Image approach, we can define V_{multi}:

$$V_{multi} = \left\{ v_i \mid diff\left(\bigcup_k W^{ref_k \to v_i}, I^{v_i} \right) < err_{resp} \right\} \tag{8}$$

and $\mathbf{p} = P(v_c \notin V_{multi})$. $\|V_{multi}\|$ ($\|V\|$ denotes the size of V) is mainly determined by how reference viewpoints $\{ref_k\}$ are selected. Therefore, the problem of minimizing $\mathbf{p}$ to reduce latency turns into a new problem of selecting reference viewpoints to maximize $\|V_{multi}\|$.

There are key differences between our *Multi Depth Image* design and other systems that also use multiple references for 3D image warping. The related systems, such as Mark [1999] and Zhu et al. [2011], are designed to replace 3D graphics rendering with 3D image warping. Therefore, the reference viewpoints are selected after knowing the viewpoint change. The criteria of reference selection is to maximize the warping quality. However, the goal of our system is to reduce the interaction latency, which requires the reference viewpoints to be selected even before the change happens. The selection criteria is to maximize the coverage and maintain adequate warping quality for the covered viewpoints. Minor artifacts can be tolerated because the warping result image is only displayed for a short period before the server updates arrive.

Compared with other image-based remote rendering approaches, our system needs more network bandwidth to transmit more data to the client. Fortunately, the server generates standard depth images and many existing compression techniques can be applied in our system. Depth image compression has been widely studied. The project ATTEST [Redert et al. 2002] suggests using the standard video codec H.264 to compress depth data. The proxy-based compression proposed by Penta and Narayanan [2005] and the skeleton-based compression introduced in Lien et al. [2007] both use the predefined models to reduce data redundancy. Kum and Mayer-Patel [2005] studied the inter-stream compression using reference stream. We also proposed a method of using graphics contexts to assist the real-time encoding of color-plus-depth image sequences [Shi et al. 2011a], which we believe can perfectly match the remote rendering system introduced in this article.

Table III. Motion Patterns

Pattern	Description	Direction
Translate	*Pos*: change a unit distance toward $\pm Dir \times Up$ or $\pm Up$; *Dir*: unchanged; *Up*: unchanged.	*Left, Right, Up, Down*
Orbit	*Pos*: change a unit angle in the circle with the tangent toward $\pm Dir \times Up$ or $\pm Up$; *Dir*: remains pointing to the circle center; *Up*: unchanged.	*Left, Right, Up, Down*
Zoom	*Pos*: change a unit distance toward $\pm Dir$; *Dir*: unchanged; *Up*: unchanged.	*Forward, Backward*
Pan & Tilt	*Pos*: unchanged; *Dir*: unchanged; *Up*: change a unit angle in the circle with the tangent toward $\pm Dir \times Up$.	*Left, Right, Up, Down*

4.3 Reference Selection

The major challenge of our system is how to select a set of reference viewpoints to maximize the coverage. A rendering viewpoint is a set of three vectors: $\{pos, dir, up\}$, which stand for camera position, look-at direction, and up direction. Fortunately, we do not need to cover all possible combinations of the viewpoint vectors, but only the viewpoints that mobile users can possibly change to. According to the applications we have studied, two assumptions can be made to simplify the reference selection problem: (1) The user motion that changes the rendering viewpoint follows specific patterns (the most frequent patterns are summarized in Table III). (2) The viewpoint change is discrete and bounded in distance. These two assumptions can significantly reduce the search space of all possible viewpoints and simplify the reference selection problem to one dimension. Instead of selecting a set of viewpoints to maximize the coverage of all possible viewpoints, we can select one reference viewpoint for every motion pattern to maximize the coverage of all viewpoints along this motion path.

In order to formally describe this simplified version of reference selection problem, we use v_i^M to denote the viewpoint that is i unit distance away from v_s (the current rendering viewpoint maintained on the server) following motion pattern M

$$v_i^M = v_s + i \cdot M, \quad 0 \le i \le MAX_M. \tag{9}$$

The goal of the simplified reference selection problem is to find a reference viewpoint v_r^M along the path of motion M that maximizes the size of $V_{double}(r, M)$:

$$V_{double}(r, M) = \left\{ v_i^M \;\middle|\; diff\left(W^{v_s \to v_i^M} \bigcup W^{v_r^M \to v_i^M}, I^{v_i^M} \right) < err_{resp} \right\}. \tag{10}$$

For every motion pattern M_k the rendering system supports, one reference viewpoint $v_r^{M_k}$ should be selected. Therefore, we get a set of reference viewpoints $\{v_r^{M_k}\}$ as the multi depth image representation of the original 3D scene. The number of the depth images equals to the number of motion patterns supported in the system.

Figure 6 illustrates how reference viewpoints are selected. Figure 6(a) shows a remote visualization application that only allows the camera to orbit the visualized model. In this case, two reference viewpoints (the motions with the same pattern but different direction are considered as two different patterns) are selected to cover the possible viewpoints in the orbit. Figure 6(b) shows a video game scenario, in which the player can move the tank on the battlefield to shoot enemies. Considering four motion patterns are supported in the game (move forward, move backward, pan left, and pan right), five reference viewpoints will be selected to cover all possible viewpoints the player can change to.

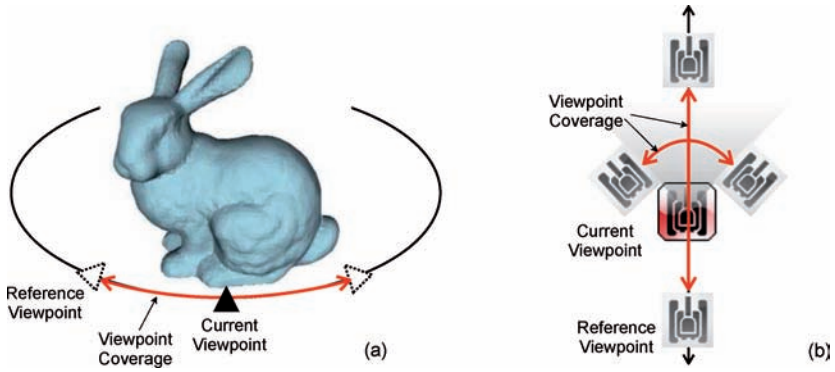

Fig. 6. (a) The illustration of selecting two reference viewpoints for the visualization of *bunny*; (b) The illustration of selecting four reference viewpoints for the tank shooting game.

4.4 Algorithms

Now we focus on the algorithms that select the appropriate reference viewpoint for the specified motion pattern. Given v_0^M (according to Eq. (9), $v_0^M = v_s$) and v_r^M as the references, the viewpoints v_i^M with $0 \le i \le r$ take the most benefits from both depth images (illustrated in Figure 4). For the viewpoints v_i^M with $i > r$, the image quality of warping two references is no better than $W^{v_r^M \to v_i^M}$ because they are not covered by v_0^M. Therefore, we can exclude all viewpoints v_i^M with $i > r$ from $V_{double}(r, M)$.

Meanwhile, r can not be very large. Otherwise, the viewpoints close to $v_{\frac{r}{2}}^M$ are too far away from both references and cannot be well covered for high quality warping. Therefore, selecting the appropriate reference viewpoint is to find the maximum r that still has every viewpoint v_i^M with $0 \le i \le r$ belonging to $V_{double}(r, M)$

$$\max_r \left\{ V_{double}(r, M) = \left[v_0^M, v_r^M \right] \right\}, \tag{11}$$

where $[v_0^M, v_r^M] = \{v_i^M \mid 0 \le i \le r\}$. According to Eq. (11), the selection of v_r^M provides a range of viewpoints $[v_0^M, v_r^M]$. As long as the viewpoint change falls into this range, the *penalty probability* is zero. We propose two algorithms to find the maximum r.

4.4.1 *Full Search Algorithm.* A baseline Full Search Algorithm can be easily developed based on Eq. (10) and (11). The search starts from $r = 2$ and for every candidate of v_r (for simplicity, we do not specify M in algorithm discussion because M remains constant for one algorithm execution), all $v_i \in [v_0, v_r]$ are verified. If every v_i can be warped in high quality, the algorithm increases x and continues. The pseudocode description of the algorithm is presented here. Obviously, Full Search can find the optimal reference but takes too much computation. In order to find the maximum r, the Full Search Algorithm needs to render $r_{max} + 1$ frames and run 3D image warping approximately r_{max}^2 times. In our experiments, it usually takes minutes to finish. Therefore, we need a much more efficient algorithm that can be practically used in a real system.

4.4.2 *Reference Prediction Algorithm.* If the optimal result is not required, we can predict a sub-optimal reference based on an error function $F(r)$ that predicts the maximum warping error for any viewpoint in the range $[v_0, v_r]$ if v_r is selected as the reference. The function is defined as follows:

$$F(r) = \sum_{w=0}^{W-2} \sum_{h=0}^{H-1} f(w, h, r), \quad 1 < r \le MAX_M, \tag{12}$$

ALGORITHM 1: Full Search Algorithm

$\langle I^{v_0}, D^{v_0} \rangle = render(S, v_0)$
$\langle I^{v_1}, D^{v_1} \rangle = render(S, v_1)$
for $r = 2; r \leq MAX_M; r$++ **do**
  $\langle I^{v_r}, D^{v_r x} \rangle = render(S, v_r)$
  for $i = 1; i < r; i$++ **do**
    $err = warping(\langle I^{v_0}, D^{v_0} \rangle, v_0 \rightarrow v_i) \bigcup warping(\langle I^{v_r}, D^{v_r} \rangle, v_r \rightarrow v_i) - I^{v_i}$
    if $err > err_{resp}$ **then**
      return $r - 1$
    end
  end
end
return MAX_M

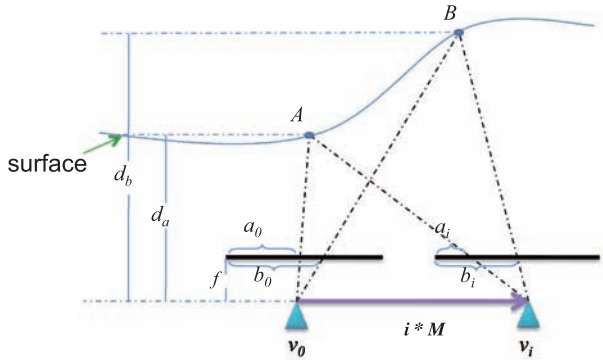

Fig. 7. Warping hole size analysis.

where W and H are the width and height of the generated image. $f(w, h, r)$ is the pixel pair error function for the adjacent pixels (w, h) and $(\hat{w}, \hat{h})$[5] in I^{v_0} and defined as:

$$f(w,h,r) = \begin{cases} 0 & \text{if } h_{max}(w,h) < 1 \text{ or } r \leq i_{max}(w,h) \\ \lfloor h_{max}(w,h) \rfloor \cdot (I^{v_0}(\hat{w},\hat{h}) - I^{v_0}(w,h))^2 & \text{else.} \end{cases} \quad (13)$$

$I^{v_0}(w, h)$ returns the intensity of pixel (w, h). h_{max} and i_{max} are explained in details as follows.

The warping error can be estimated on the base of pixels. The holes are introduced because the pixels that are adjacent in the original picture are not adjacent any more after 3D image warping. For every pair of adjacent pixels (w, h) and $(\hat{w}, \hat{h})$, we can build a hole size function $h(i)$, $2 \leq i \leq MAX_M$ to represent the size of the hole caused by warping this pair of pixels to viewpoint v_i. For example, Figure 7 shows a scenario that the viewpoint translates from v_0 right to v_i horizontally. A and B are two points on the surface visible at both v_0 and v_i. d_a and d_b are the depth value of A and B to the image plane. a_0 and b_0 are the horizontal coordinates[6] of the corresponding pixels of A and B in I^{v_0}. If $b_0 = a_0 + 1$, then A and B are adjacent in I^{v_0}. The horizontal coordinates a_i and b_i in I^{v_i} can be

[5]$(\hat{w}, \hat{h})$ denotes the adjacent pixel to (w, h) and for different motion patterns it has different definitions. For example, for horizontal movements, $\hat{w} = w + 1, \hat{h} = h$; and for vertical movements, $\hat{w} = w, \hat{h} = h + 1$. For the pixels on the image edge that do not have valid $(\hat{w}, \hat{h})$, $f(w, h, r)$ always returns 0.

[6]The vertical coordinates are neglected here because there is no viewpoint movement in the vertical direction.

ALGORITHM 2: Reference Prediction Algorithm

$\langle I^{v_0}, D^{v_0} \rangle = render(S, v_0)$
allocate and reset array $F[MAX_M]$
for $w = 0$; $w < W$; w++ **do**
  for $h = 0$; $h < H$; h++ **do**
    calculate $h_{max}(w,h)$, $i_{max}(w,h)$ for I^{v_0}
    if $h_{max}(w,h) < 1$ **then**
      continue
    end
    $F[i_{max}(w,h)+1] = F[i_{max}(w,h)+1] + h_{max}(w,h) \times (I^{v_0}(w,h) - I^{v_0}(adj_w(w), adj_h(h)))^2$
  end
end
for $r = 2$; $r <= MAX_M$; r++ **do**
  $F[r] = F[r] + F[r-1]$
  if $F[r] > err_{disp}$ **then**
    return $r-1$
  end
end
return MAX_M

represented as:

$$a_i = a_0 - \frac{i \cdot M \cdot f}{d_a}, b_i = b_0 - \frac{i \cdot M \cdot f}{d_b}$$

where M is the unit vector of viewpoint translation and f is the distance from center-of-projection to the image plane. The hole size function $h(i)$ in this case is:

$$\begin{aligned} h(i) &= (b_i - a_i) - (b_0 - a_0) \\ &= M \cdot f \cdot i \left(\frac{1}{d_a} - \frac{1}{d_b} \right). \end{aligned} \tag{14}$$

We define h_{max} as the max value that $h(i)$ can reach within the given viewpoint range ($i \in [2, MAX_M]$) and i_{max} as the viewpoint at which h_{max} is achieved. In our example scenario, if $d_a < d_b$, the hole size increases monotonically with i and reaches the max value when $a_i = 0$. Thus:

$$i_{max} = \frac{d_a \cdot a_0}{M \cdot f} \tag{15}$$

$$h_{max} = a_0 \left(1 - \frac{d_a}{d_b} \right). \tag{16}$$

Back to the definition of pixel pair error function in Eq. (13). For a given pixel pair, $h_{max} > 1$ means this pixel pair may create a visible hole after 3D image warping. In order to fill this hole, the reference image at v_r should have all the missing pixels so that i_{max} is the best candidate for v_r. However, if $i_{max} < r$, then the reference viewpoint v_r does not have enough pixels to cover the warping hole at $v_{i_{max}}$. The error score (equivalent to Mean Square Error) is calculated based on the pixel difference and hole size because we are assuming some hole filling techniques are applied to fill the holes with the available neighbors.

With the error function, we propose our reference prediction algorithm. First, h_{max} and i_{max} are calculated for every pair of adjacent pixels in I_{v_0}. Then, the error function $F(r)$ is tested for every reference candidate. If the total error score is larger than the pre-set threshold, the algorithm stops and return the previous candidate as the predicted reference. Compared with Full Search, Reference Prediction

Fig. 8. Remote rendering prototype system: (a) *Bunny* visualization; (b) BZFlag on mobile.

does not generate the optimal result but requires much less computation: only I^{v_0} is rendered, no 3D image warping is performed, and the algorithm complexity is equivalent to scaning the whole image ($O(W \times H)$).

5. EVALUATION

We have built a prototype system to evaluate the proposed remote rendering design. The server runs on a workstation that has an AMD Phenom II 3.2GHz quad-core CPU, 4GB memory, an Nvidia GeForce 9800GT GPU, and 1 Gbps ethernet connection to the university network. There are three rendering modes.

—*Image Impostor.* Only the 2D image at the current rendering viewpoint is rendered.

—*Depth Image.* The depth image at the current rendering viewpoint is generated.

—*Multidepth Image.* For every rendering frame, the server not only renders the depth image at the current rendering viewpoint, but more depth images at the reference viewpoints selected by either Full Search Algorithm or Reference Prediction Algorithm.

The mobile client has been implemented for Apple iOS platforms. Three different setups are selected to run experiments.

—*Simulator.* The mobile client runs on an iPhone simulator hosted by an Apple laptop. The host machine has a dual-core CPU, 2GB memory, and connects to the university network through 802.11n Wi-Fi.

—*Wi-Fi.* The mobile client runs on an Apple iPhone 4 smartphone. The phone connects to the university network through 802.11g Wi-Fi.

—*3G.* The mobile client runs on an Apple iPhone 4 smartphone. The phone connects to Internet through AT&T 3G (HSDPA/UMTS) network.

We also developed two applications for experiments (Figure 8).

—*Bunny.* The server renders a static bunny model that consists of 69,450 triangles. The only user interaction pattern allowed is to orbit right so that only two depth images are generated for Multi Depth Image mode.

—*BZFlag.* The server renders a 3D tank shooting game. Four motion patterns are allowed (Figure 6(b)) so that five depth images should be created for every rendering frame. However, according to some optimization techniques discussed in Section 6, the server sends no more than two depth images to the client for every frame.

Table IV. Bandwidth and Rendering Frame Rate

	Bunny	*BZFlag*		
	All Networks	*Simulator*	*Wi-Fi*	*3G*
Image Impostor	64 Kbps @ 1 fps	2880 Kbps @ 15 fps	1920 Kbps @ 10 fps	1344 Kbps @ 7 fps
Depth Image	168 Kbps @ 1 fps	4080 Kbps @ 15 fps	static: 2176 Kbps @ 8 fps motion: 1632 Kbps @ 6 fps	1360 Kbps @ 5 fps
Multi Depth Image	336 Kbps @ 1 fps	static: 3424 Kbps @ 15 fps motion: 4240 Kbps @ 15 fps	static: 2080 Kbps @ 8 fps motion: 1312 Kbps @ 4 fps	static: 1232 Kbps @ 5 fps motion: 1312 Kbps @ 4 fps

Table V. Network Latency (rtt) and Interaction Latency (IL)

	rtt	*Image Impostor*	*Depth Image*	*Multi Depth Image*
Simulator	8 ms	40 ms	17 ms	18 ms
Wi-Fi	43 ms	255 ms	161 ms	177 ms
3G	248 ms	449 ms	144 ms	161 ms

For both applications, the rendering resolution is 480×320. The color images are compressed with JPEG and the depth images are compressed with ZLIB. TCP protocol is used to guarantee the reliable transmission.

First, we evaluate the rendering rate and network bandwidth of the prototype system. Table IV lists the network bandwidth usage and the rendering frame rate. For the application Bunny, the server only renders a new frame when the viewpoint is changed. For the application BZFlag, the application loops in rendering the game scene even if there is no user interaction. The target frame rate for the remote rendering system FPS_c is set to 15. In our experiments, only the Simulator setup can achieve this frame rate. For the rendering modes that require 3D image warping, we notice that the rendering frame rate drops further when the user interacts more frequently with the application. This is because the client needs to decode depth images, run 3D image warping, and receive more references.

Second, we evaluate the interaction latency of our system. Table V summaries the results. *rtt* is counted by *ping*. Interaction latency is recorded by adding timing codes into the client program. The interaction latency of Image Impostor increases with *rtt* but for other two modes, the interaction latency is independent of network latency because it only depends on how fast the client can run 3D image warping. The large latency numbers of the real device experiments using depth images are mainly because the nonoptimized image decoding and 3D image warping computation in our implementation. There is enough room to reduce the latency by optimizing the local computation.

Last, we compare the performance of the proposed multi depth image scheme and reference selection algorithms. The test sequence is collected from the Bunny application. Figure 9(a) plots the image quality of warping results. The horizontal axis indicates the viewpoint position and the vertical axis is the PSNR of the warping image at the given viewpoint. For the line of One Reference, only v_0 is used as the reference viewpoint. The PSNR drops fast as the target viewpoint moves away from the reference. For the line of Two Reference, v_0 and v_{60} are used as references. High warping quality is maintained for all viewpoints between two references. If we set 35 dB as the threshold err_{resp}, the size of V_{single} is only 3 while the size of V_{double} is 63.

Figure 9(b) presents how reference viewpoint is selected. The horizontal axis stands for the reference candidates. For the line of Full Search, the vertical axis indicates the PSNR of the worst warping image inside the reference range:

$$PSNR(r) = \min\left\{\forall v_i \in [v_0, v_r],\ psnr\left(W^{v_0 \to v_i} \bigcup W^{v_r \to v_i}\right)\right\}$$

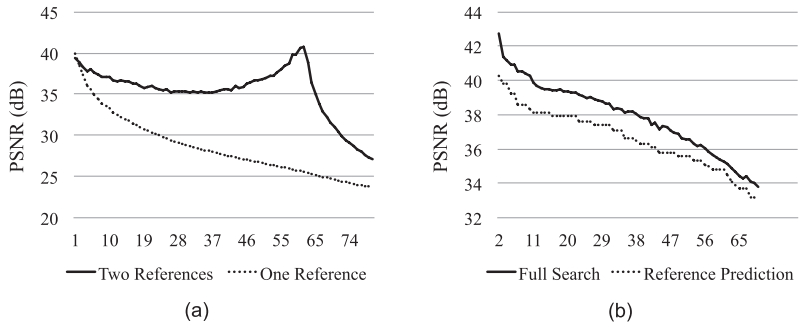

Fig. 9. (a) The warping quality comparison between one depth image as reference and two depth images as references. (b) The comparison of reference viewpoint selection algorithms.

For the line of Reference Prediction, the vertical axis is the PSNR equivalent score of $F(r)$ as defined in Eq. (12). If we set 35 dB as the threshold of err_{resp}, we get 62 as the output of Full Search Algorithm and 54 as the output of Reference Prediction Algorithm.

Although Reference Prediction Algorithm does not generate an optimal result, it is much faster in execution. In our experiments, it takes a couple of minutes to finish the whole Full Search Algorithm and generate the optimal result but only a few mini-seconds to run Reference Prediction.

6. DISCUSSION

In this section, we mainly discuss the limitations of our rendering system and how we can improve in the future.

First, the proposed Multi Depth Image remote rendering approach reduces the interaction latency at the cost of sending extra depth images to the client. Although we have discussed in Section 4.2 that different depth image compression techniques can be applied to save bandwidth, the number of reference frames can still be overwhelming for the applications that support many motion patterns. In our implementation of rendering BZFlag, we have tried several methods to avoid creating too many reference depth images: (a) only one set of reference depth images are created if the rendering viewpoint is not changed; (b) if the viewpoint is moving, the server waits until the viewpoint moves out of the coverage range of the previous reference and then generates a reference for the current motion pattern; (c) the reference depth images are queued for streaming to make sure that no more than two depth images are transmitted for every rendering frame. Moreover, we can develop more techniques to reduce the overall number of reference depth images. For example, not all motion patterns have the same priority. The server can predict the user's next move based on history or the user behavior pattern, and only generates the references for the motions that are most likely to happen.

Second, using multiple depth images only helps solving the "hole" artifacts caused by 3D image warping. However, some 2D objects (e.g., texts, frames, etc.), special rendering effects (e.g., shadows) and graphics animation cannot be warped correctly. In addition, 3D image warping can not handle the moving foreground objects that are not controlled by the rendering viewpoint. In order to remove these artifacts, the remote rendering system should be co-designed with the original application to get extra tags and contexts. The background and foreground should be separated into layers like the approaches taken in Zhu et al. [2011].

Last, the quality measurement is also an important research topic in the future. Currently, we use MSE in our system to measure the warping quality. However, MSE may not be the best metric to use here because it can not distinguish the warping artifacts that are difficult to perceive from the artifacts

that cause serious visual distortion. Moreover, for the rendering application like games, the user pays more attention to the quality of the video sequence, rather than a single image. We have made some initial efforts in building a new metric for interactive performance evaluation in Shi et al. [2011b] and we believe this is an interesting area to explore.

7. CONCLUSION

In this article, we have built a model to analyze the interaction latency problem of different remote rendering designs, and proposed a novel design that can be effectively applied to assist rendering interactive 3D graphics on mobile devices. Compared with previous remote rendering systems, our design features in network-independent interaction latency and high rendering quality. Our work answered three questions: what type of rendering results can be used to render the original scene at a different viewpoint: multiple depth images and 3D image warping; which references should be selected to achieve the balance of high warping quality and large motion coverage: Eq. (10) and Eq. (11); and how to efficiently select references in real-time: Reference Prediction Algorithm.

REFERENCES

AZUMA, R. T. 1997. A survey of augmented reality. *Presence: Teleoper. Virtual Environ. 6,* 4, 355–385.

BAO, P. AND GOURLAY, D. 2004. Remote walkthrough over mobile networks using 3-D image warping and streaming. *Vis. Image 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 20th ACM Symposium on Operating Systems Principles (SOSP'05)*. ACM, New York, 277–290.

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 *Proceedings of NetGames'04*. 144–151.

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, 691–694.

CHANG, C.-F. AND GER, S.-H. 2002. Enhancing 3D graphics on mobile devices by image-based rendering. In *Proceedings of PCM'02*. 1105–1111.

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, 29–38.

DUGUET, F. AND DRETTAKIS, G. 2004. Flexible point-based rendering on mobile devices. *IEEE Trans. Comput. Graph. Appl. 24,* 4, 57–63.

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

ENGEL, K., WESTERMANN, R., AND ERTL, T. 1999. Isosurface extraction techniques for web-based volume visualization. *IEEE Visualiz.*

EPIC GAMES. 2007. Unreal tournament 3. www.unrealtournament.com/.

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

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

KURILLO, G., VASUDEVAN, R., LOBATON, E., AND BAJCSY, R. 2008. A framework for collaborative real-time 3D teleimmersion in a geographically distributed environment. In *Proceedings of the 10th IEEE International Symposium on Multimedia (ISM'08)*. 111–118.

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.

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, 21–28.

LIEN, J.-M., KURILLO, G., AND BAJCSY, R. 2007. Seleton-based data compression for multi-camera tele-immersion system. In *Proceedings of the 3rd International Conference on Advances in Visual Computing (ISVC'07)*. Vol. 1, Springer-Verlag, Berlin, Heidelberg, 714–723.

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

MARQUEZ, J., DOMENECH, J., GIL, J., AND PONT, A. 2008. Exploring the benefits of caching and prefetching in the mobile web. In *Proceedings of WCITD'08*.

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

MCMILLAN, L. AND BISHOP, G. 1995. Plenoptic modeling: an image-based rendering system. In *Proceedings of the 22nd Annual Conference on Computer Graphics and Internetive Techniques (SIGGRAPH'95)*. 39–46.

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

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

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

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 *Proceedings of 1st International Symposium on 3D Data Processing Visualization Transmission (3DPVT)*. IEEE Computer Society, 313–319.

RIVA, O. AND KANGASHARJU, J. 2008. Challenges and lessons in developing middleware on smart phones. *IEEE Computer 41,* 10, 77–85.

SCANVIEW. 2003. A system for remote visualization of scanned 3D models. http://graphics.stanford.edu/software/scanview/.

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

SHADE, J., GORTLER, S., HE, L., AND SZELISKI, R. 1998. Layered depth images. In *Proceedings of the 28th Annual Conference on Computer Graphics and Internetive Techniques (SIGGRAPH'98)*. 231–242.

SHI, S., HSU, C.-H., NAHRSTEDT, K., AND CAMPBELL, R. 2011a. Using graphics rendering contexts to enhance the real-time video coding for mobile cloud gaming. In *Proceedings of the 19th ACM International Conference on Multimedia (MM '11)*. ACM, New York, NY, USA, 103–112.

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 IEEE International Conference on Multimedia and Expo (ICME'11)*.

SINGHAL, S. AND ZYDA, M. 1999. *Networked Virtual Environments: Design and Implementation*. ACM Press/Addison-Wesley Publishing Co., New York, NY.

SMIT, F. A., VAN LIERE, R., BECK, S., AND FRÖHLICH, B. 2009. An image-warping architecture for Vr: Low latency versus image quality. In *Proceedings of the IEEE Symposium on Virtual Reality (VR)*. IEEE, 27–34.

ZHU, M., MONDET, S., MORIN, G., OOI, W. T., AND CHENG, W. 2011. Towards peer-assisted rendering in networked virtual environments. In *Proceedings of the 19th ACM International Conference on Multimedia (MM '11)*. ACM, New York, 183–192.

Received January 2012; revised May 2012; accepted June 2012