A High-Quality Low-Delay Remote Rendering System for 3D Video

Shu Shi, Mahsa Kamali, Klara Nahrstedt, John C. Hart, and Roy H. Campbell
Department of Computer Science
University of Illinois at Urbana-Champaign
201 N Goodwin Ave., Urbana, IL 61801, USA
{shushi2, mkamali2, klara, jch, rhc}@illinois.edu

ABSTRACT

As an emerging technology, 3D video shows a great potential to become the next generation media for tele-immersion. However, streaming and rendering this dynamic 3D data in real-time requires tremendous network bandwidth and computing resources. In this paper, we build a remote rendering model to better study different remote rendering designs and define 3D video rendering as an optimization problem. Moreover, we design a 3D video remote rendering system that significantly reduces the delay while maintaining high rendering quality. We also propose a reference viewpoint prediction algorithm with super sampling support that requires much less computation resources but provides better performance than the search-based algorithms proposed in the related work.

Categories and Subject Descriptors

H.5.1 [**Multimedia Information System**]: Video

General Terms

Design, Measurement

Keywords

3D Video, Remote Rendering, 3D Warping, Mobile Devices

1. INTRODUCTION

3D video has attracted more and more research interests in recent years due to its bright future in the next generation tele-communication and tele-immersion applications. In this paper, we study how to design a remote rendering system for 3D video rendering on mobile devices. The 3D video referred here is different from the stereo video played in 3D movie theaters. The stereo video provides the audience a 3D depth illusion by forcing different eyes to see different views, while the 3D video in our research, also named as the free viewpoint video [21], allows the audience to interactively select different rendering viewpoints or change different virtual

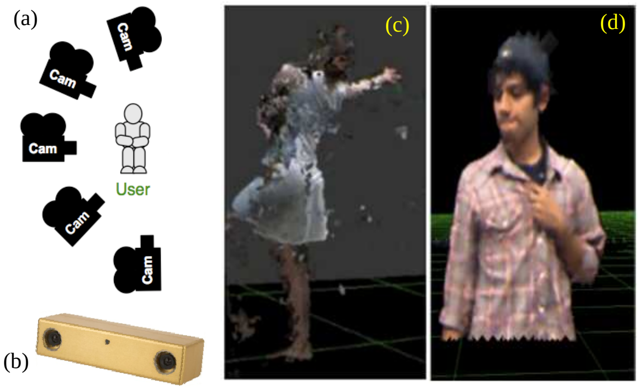

Figure 1: (a) Camera Setup for 3D Video Capturing; (b) BumbleBee 2 Stereo Camera (c) Point Cloud Based 3D Video (d) Mesh Based 3D Video

background, just like dealing with computer graphics. Although different approaches to 3D video have been proposed [22, 11, 23], all share the same feature that 3D video is comprised of sequential frames and each frame is built from the static images that are captured by cameras from multiple viewpoints. For example, Figure 1 illustrates how 3D video is generated in our tele-immersion system, TEEVE [2]. Multiple stereo cameras [1] are calibrated and placed in different positions to capture the depth image of real-world objects. The depth images from different cameras are collected to construct either a point-cloud [23] or triangle meshes [12] as the 3D video frame in real-time. The system is currently capable of generating up to 20 3D video frames per second.

There are two major challenges for rendering 3D video on mobile devices. First, 3D video requires very large network bandwidth for transmission. The 3D video stream size is proportional to the total number of capturing cameras. According to [23], a quality real-time immersive communication using point cloud based 3D video needs a Gbps bandwidth, which is far beyond the capacity of wireless networks (cellular 3G/4G, Wi-Fi 802.11b/g) supported by most mobile devices. Second, although the most advanced smartphones are currently equipped with hardware to support OpenGL ES, rending 3D video frames is a very computation intensive task. Moreover, the rendering of every frame has a strict deadline because the frame expires when the next frame arrives.

In our previous work [20], we proposed to use remote rendering system for real-time 3D video rendering on mobile

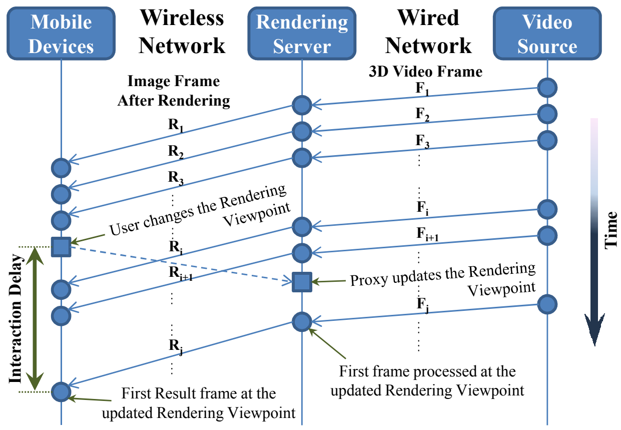
Mobile Devices
Wireless Network
Rendering Server
Wired Network
Video Source
3D Video Frame
Image Frame After Rendering
F_1
F_2
F_3
F_i
F_{i+1}
F_j
R_1
R_2
R_3
R_i
R_{i+1}
R_j
User changes the Rendering Viewpoint
Proxy updates the Rendering Viewpoint
First frame processed at the updated Rendering Viewpoint
First Result frame at the updated Rendering Viewpoint
Interaction Delay
Time

Figure 2: Interaction Delay

devices. A powerful workstation is introduced as the rendering server (proxy) that pre-renders the 3D video frame to 2D images for mobile clients. The remote rendering system solves both bandwidth and computation problems, but a new problem of *interaction delay* is brought into the scope. Since 3D video allows the viewer to interactively change the rendering viewpoint, when the mobile client receives a user interaction request, it has to forward the request to the proxy server and waits at least a round trip network delay for the future frames that are processed at the updated rendering viewpoint. Figure 2 gives an intuitive illustration. Since the *interaction delay* has great impacts on the user experience of interactive applications [8], we must consider how to reduce the *interaction delay* to an acceptable level when designing a remote rendering system.

In this paper, we build a remote rendering model to study different remote rendering systems in literatures and *interaction delay* problems. We propose using a new reference viewpoint prediction algorithm for 3D warping based remote rendering system. We prove that our algorithm requires far less computation resources but provides much better performance in terms of *viewpoint range* (explained in Section 3.4) than the search-based algorithms suggested in [20]. We believe the work of this paper makes two major contributions. First, we study the remote rendering system from a new perspective and transform the network-related *interaction delay* problem into a content-based reference prediction problem. Second, we find a computation efficient method to create an image-based representation for any complex graphic models. The image-based representation is sufficient to support high-quality rendering of the original model when the rendering viewpoint changes in a limited range. Besides, we will also discuss how our remote rendering system can be extended to support general graphic content, such as games.

In the rest of the paper, we first review previous remote rendering designs in Section 2. We define the remote rendering model. Section 4 concentrates on reference viewpoint prediction algorithm and Section 5 evaluates the algorithm performance. Issues of optimization and extension are discussed in Section 6. Section 7 concludes the paper.

2. RELATED WORK

In general, previously proposed server-client based remote rendering systems can be divided into two major categories

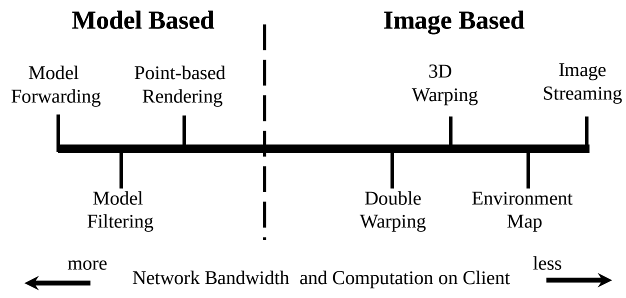

Figure 3: Remote Rendering Systems

based on what type data is sent from the rendering server to the client: the model-based approach and the image-based approach (Figure 3). In a model-based remote rendering system, the server transmits the original or simplified 3D models (meshes or point cloud) to the client. The client needs to have enough local graphical capabilities to render received models. On the contrary, the image-based remote rendering system depends on the server to render 3D models and requires no special graphical computation on the client. In this section, we briefly review the designs in both categories and we will compare their pros and cons in Section 3.3.

Model Forwarding & Filtering: Strictly speaking, model forwarding is not a remote but local rendering approach. The server forwards the original 3D model based upon request and leaves all 3D rendering to the client. Model filtering requires the server to filter the data before transmission. A subset of original data or the simplified low-resolution 3D model are sent to the client for rendering. Such systems [10, 17] have been mainly designed for medical image visualization.

Point-Based Rendering: Duguet and Drettakis [9] suggested using a powerful server to transform the mesh-based graphic model to a point-based representation, the density of which is decided by the screen size of mobile devices. Since most mobile devices have only small screens, rendering the simplified point-based model requires much less computing resources on mobile devices compared with original mesh-based rendering.

Image Streaming: In this simple remote rendering approach, the server renders the 3D model at the viewpoint specified by the client. The generated 2D image can be efficiently compressed with mature image/video coding tools for network transmission. The client simply displays the received image frame. The only disadvantage of this approach is the *interaction delay* as we have mentioned in Section 1. Lamberti and Sanna [13] surveyed on image streaming systems and discussed in depth about timing factors that affect system performance.

Environment Map: Chen [6] first introduced environment map to remote rendering in his Virtual VR system. The server generates an environment map (e.g., a panoramic cylindrical projection map) of the 3D scene. The environment map contains enough information to represent the scene from any viewpoint at a fixed position. Thus it allows the client to pan and tilt the virtual camera freely in a wide range. Boukerche and Pazzi [4] studied how to stream environment maps to mobile devices in a remote virtual world walk-through system.

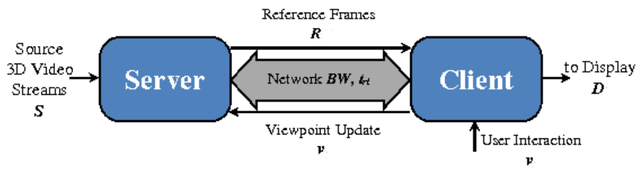

Figure 4: Server-Client Remote Rendering System Architecture

3D Warping: 3D warping proposed by McMillan [15] is a classic image-based rendering technique adopted in many remote rendering applications [5, 3]. Compared with the image streaming approach, the server only generates one more depth map of the rendered scene, but the client can warp the received depth image (including both color and depth maps) to arbitrary viewpoint. 3D warping solves the problem of *interaction delay* but brings the new problem of warping error.

Double Warping: Mark [14] first suggested double warping. The idea of using multiple image to compensate holes appeared even before 3D warping [7] . Besides the depth image rendered at the current viewpoint, the server also generates another depth image at a selected reference viewpoint, and sends both depth images to the client. The client runs 3D warping twice for two depth images and composite the results. The double warping can greatly reduce the warping errors caused by single warping if the reference viewpoint is appropriately selected. Shi [20] proposed algorithms to select the optimal viewpoint for the remote rendering system using double warping.

Before we discuss more about warping error and double warping, we first introduce our remote rendering model.

3. REMOTE RENDERING MODEL

In this section, we build a generalized model to study remote rendering systems. Then we analyze systems reviewed in Section 2 with our model and conclude the research problem of designing a 3D video remote rendering system.

3.1 System & Data Model

We simplify the system architecture used for our remote rendering model to only three components: server, client, and network (Figure 4). The server receives the source 3D content, processes some operations on the content to generate a reference frame, and transmits the reference frame to the client through the network, that has a bandwidth BW and a round-trip time rtt[1]. The client receives the reference frame from the server and tries to render the 3D scene image corresponding to the current rendering viewpoint. Meanwhile, the client also receives user inputs. In this system prototype, we only consider user inputs that change the rendering viewpoint. Once the rendering viewpoint on the client is changed, a viewpoint update message is sent to the server. However, since it takes at least a one-way network delay for the update message to be transmitted, there will be a duration when the rendering viewpoint stored on the server is inconsistent with that on the client.

[1]For simplicity, we add all network related factors to rtt, such as the retransmission due to packet loss, the latency caused by traffic congestion, the waiting time in a streaming buffer, etc. Thus, we consider that rtt can have a huge variance even in wired networks and may go unbounded in unreliable wireless networks.

We now define the system behaviors described above as follows. We use S to denote the source 3D content. If the source content is static, S refers to the whole set of 3D models (can be either meshes or point clouds). If the source content is dynamically updated (like 3D video), S refers to only one frame of the dynamic content, and FPS_{src} is used for the update rate of the source content. In this case, S has a lifespan of $\frac{1}{FPS_{src}}$. The rendering viewpoint v is a set of three three-dimensional vectors: $\{pos, dir, up\}$, which stand for camera position, look-at direction, and up direction. Due to the possible viewpoint inconsistency between the server and the client, v_s and v_c are used to denote the rendering viewpoint on the server and client, respectively.

Given the rendering viewpoint v_s, the server processes the source content S to generate a reference frame R_s.

$$R_s = proc\_s(S, v_s)$$

If the source content is dynamic, there is an extra time constraint that $proc\_s$ must finish before S expires.

$$time\_s(proc\_s) < \frac{1}{FPS_{src}} \quad (1)$$

where $time\_s$ records the time that the server needs to execute $proc\_s$.

The generated reference frame R_s is usually encoded (compressed) before network transmission.

$$Rc_s = encode(R_s)$$

On the client side, decoding(decompression) is needed to reconstruct the reference frame.

$$R'_s = decode(Rc_s)$$

The distortion caused by encoding/decoding should be minimal.

$$diff(R_s, R'_s) < \epsilon$$

Similarly, if the source content is dynamic, there also exists a time constraint for network transmission.

$$\frac{\|Rc_s\|}{BW} < \frac{1}{FPS_{src}} \quad (2)$$

where $\|Rc_s\|$ indicates the size of the encoded reference frame Rc_s.

After the client reconstructs R'_s, it needs to render the image D^s_c to display at the current rendering viewpoint v_c.

$$D^s_c = proc\_c(R'_s, v_c)$$

If there has been no viewpoint change and $v_s = v_c$, we have

$$D_s = D^s_s = proc\_c(R'_s, v_s)$$

The time constraint of dynamic content rendering applies to the client, too.

$$time\_c(proc\_c) < \frac{1}{FPS_{src}} \quad (3)$$

where $time\_c$ records the time that the client needs to execute $proc\_c$.

3.2 Remote Rendering Problem

Although different systems are designed for different applications, two goals are shared by all systems we have reviewed: remote rendering is expected to have the same rendering quality as local rendering, and the system should respond to user interaction requests as soon as possible. These two goals can be represented by two metrics in our remote rendering model.

The first metric *rendering quality* refers to the source content rendering quality when there is no viewpoint inconsistency between the server and the client ($v_s = v_c$). The *rendering quality* of remote rendering should be similar to local rendering:

$$\mathit{diff}(D_s, D_s^{ref}) < err_{disp} \tag{4}$$

where D_s^{ref} is considered as the local rendering result.

$$D_s^{ref} = render(S, v_s)$$

Please note that D_s^{ref} is not actually generated in the system, but only used to evaluate the *rendering quality* in our model.

The second metric *response time* reflects how fast the system is responsive to the user interaction that changes the rendering viewpoint on the client from v_s to v_c. We define the *response time* as following:

$$time\_resp(v_s, v_c) = \begin{cases} time\_c(proc\_c) \\ \qquad if \quad \mathit{diff}(D_c^s, D_c^{ref}) < err_{resp} \\ time\_c(proc\_c) + rtt + \frac{\|Rc_c\|}{BW} \quad else \end{cases} \tag{5}$$

Although the viewpoint is changed to v_c, the reference frame available on the client is still R'_s. Therefore the client can only generate D_c^s. If the image D_c^s is already similar to the local rendering result D_c^{ref}, we consider the *response time* is only the execute time of *proc_c*. Otherwise, the client needs to send the new viewpoint v_c to the server and wait until the updated reference frame R'_c is sent back. Thus the *response time* includes some extra components. Moreover, we define the average *response time*:

$$avg\_time\_resp = time\_c(proc\_c) + time\_penalty \tag{6}$$

and *time_penalty* can be represented as:

$$time\_penalty = p \cdot (rtt + \frac{E(\|Rc)\|)}{BW}) \tag{7}$$

where p is the *penalty probability* of that D_c^s is not similar to D_c^{ref}

$$p = P\left(\mathit{diff}(D_c^s, D_c^{ref}) \geq err_{resp}\right) \tag{8}$$

and $E(\|Rc)\|)$ is the average size of compressed reference frames. Due to Eq (3), *time_c(proc_c)* is considered as a fixed portion, and we focus on reducing *time_penalty*. $\frac{E(\|Rc)\|)}{BW}$ is limited by Eq (2), but rtt can vary significantly frame by frame in different networks. We find that the *penalty probability* is not affected by network. We can reduce p with different remote rendering system designs, so that the *response time* is also reduced.

In the rest of this section, we will use this model to analyze the remote rendering designs reviewed in Section 2 and find out what we can improve for 3D video remote rendering. For simplicity, we may omit the encoding/decoding procedure in some discussions. In that case, we consider $R'_s = R_s$.

3.3 Systems Analysis

Table 1 elaborates how our remote rendering model can be used to analyze different systems. Here we show some highlights.

Model Forwarding performs perfectly in *rendering quality* because it is actually a local rendering approach. The *response time* depends only on how fast the client can render the whole 3D model. It is not appropriate to use model forwarding for rendering large size 3D contents because both network bandwidth and computation on the client are proportional to the size of S. **Model Filtering** and **Point-Based Rendering** are similar in performance. Compared with model forwarding, the network bandwidth and computation on the client are reduced. But both *rendering quality* and *response time* depend on how S' or P are constructed.

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. They are also efficient in network bandwidth and client computation, which means they are more appropriate for rendering large size 3D contents on less powerful mobile clients. The major difference is the *response time*. **Image Streaming** has the worst performance. If we assume the image rendered from different viewpoints are different,

$$\mathit{diff}(D_s^{ref}, D_c^{ref}) > err_{disp}, \ if \ v_s \neq v_c$$

The *penalty probability* is 1 so that every interaction suffers from the full network delay.

Environment Map approach requires the server to generate an environment map $env\_map(S, v_s)$ at the viewpoint v_s. This generation is supported in OpenGL. For example, generating a cubical environment map is equivalent to rendering S six times for six faces of the cube. The client can efficiently project the environment map to a new viewpoint v_c if $v_c.pos = v_s.pos$. We have

$$proj(env\_map(S, v_s), v_c) = D_c^{ref} \quad if \ v_c.pos = v_s.pos$$

Thus, environment map can reduce the *penalty probability* to the probability that the user interaction changes the camera position.

3D Warping system generates a depth image with both color and depth maps $< D_s^{ref}, depth_s >$ on the server. The client can warp the depth image to arbitrary v_c. The function $warp(v_s, v_c)$ used in Table 1 means to warp the depth image generated at v_s, that is $< D_s^{ref}, depth_s >$ to the new viewpoint v_c. The complexity of warping function depends only on the image resolution. The *penalty probability* of 3D warping depends on warping errors. For a given v_s, there exists a viewpoint set V_{warp}

$$V_{warp} = \left\{\forall v_i, \mathit{diff}(warp(v_s, v_i), D_i^{ref}) < err_{resp}\right\} \tag{9}$$

The *penalty probability* can be reduced to probability that v_c does not fall into V_{warp}. Thus the *response time* of 3D warping is actually decided by the size of V_{warp}.

Double Warping can provide a V_{dwarp} that is larger than V_{warp}.

$$V_{dwarp} = \left\{\forall v_i, \mathit{diff}(dwarp(v_s, v_p, v_i), D_i^{ref}) < err_{resp}\right\} \tag{10}$$

Compared with 3D warping, the larger V_{dwarp} increases the probability that v_c falls into, and can further reduce the *penalty probability* and *response time*. The function in Eq

Table 1: Summary of Remote Rendering Systems

System Design	$\mathbf{proc\_s(S, v_s)}$	Ref. Frame $\mathbf{R_s}$	$\mathbf{proc\_c(R_s, v_c)}$
Model Forwarding	return S	S	$render(S, v_c)$
Model Filtering	apply filter to S	S', $S' \subseteq S$	$render(S', v_c)$
Point Rendering	transform S to point cloud	P	$render(P, v_c)$
Image Streaming	$render(S, v_s)$	D_s^{ref}	display only
Environment Map	generate environment map	$env\_map(S, v_s)$	$if\ v_c.pos = v_s.pos \quad proj(env\_map(S, v_s), v_c)$ $else \quad do\ nothing$
3D Warping	$render(S, v_s)$, extract $depth_s$	$< D_s^{ref}, depth_s >$	$if\ v_c = v_s \quad return\ D_s^{ref}$ $else \quad warp(v_s, v_c)$
Double Warping	$render(S, v_s)$, extract $depth_s$ select reference viewpoint v_p $render(S, v_p)$, extract $depth_p$	$< D_s^{ref}, depth_s >$ $< D_p^{ref}, depth_p >$	$if\ v_c = v_s \quad return\ D_s^{ref}$ $elseif\ v_c = v_p \quad return\ D_p^{ref}$ $else \quad dwarp(v_s, v_p, v_c)$

System Design	$\mathbf{time(proc\_s)}$	$\mathbf{time(proc\_c)}$	$\mathbf{\|R_s\|}$
Model Forwarding	0	$1 \times time\_c(render)$, $\propto \|S\|$	$\propto \|S\|$
Model Filtering	depends on filtering algo.	$1 \times time\_c(render)$, $\propto \|S'\|$	$\propto \|S'\|$
Point Rendering	depends on point transformation algo.	$1 \times time\_c(render)$, $\propto \|P\|$	$\propto \|P\|$
Image Streaming	$1 \times time\_s(render)$	0	$\propto screen\_resolution$
Environment Map	$6 \times time\_s(render)$	$\propto screen\_resolution$	$\propto screen\_resolution$
3D Warping	$1 \times time\_s(render)$	$\propto screen\_resolution$	$\propto screen\_resolution$
Double Warping	$time\_s(ref\_viewpoint\_sel) + 2 \times time\_s(render)$	$\propto screen\_resolution$	$\propto screen\_resolution$

System Design	$\mathbf{diff(D_s, D_s^{ref})}$	Penalty Probability
Model Forwarding	0	0
Model Filtering	$< err_{disp}$	$P(diff(render(S', v_c), D_c^{ref}) \geq err_{resp})$
Point Rendering	$< err_{disp}$	$P(diff(render(P, v_c), D_c^{ref}) \geq err_{resp})$
Image Streaming	0	1
Environment Map	0	$P(v_c.pos \neq v_s.pos)$
3D Warping	0	$P(v_c \notin V_{warp})$, where $V_{warp} = \left\{\forall v_i, diff(warp(v_s, v_i), D_i^{ref}) < err_{resp}\right\}$
Double Warping	0	$P(v_c \notin V_{dwarp})$, where $V_{dwarp} = \left\{\forall v_i, diff(dwarp(v_s, v_p, v_i), D_i^{ref}) < err_{resp}\right\}$

(10), $dwarp(v_s, v_p, v_c)$ warps both $< D_s^{ref}, depth_s >$ and $< D_p^{ref}, depth_p >$, to the new viewpoint v_c and composite the result images. The overhead is that the server needs to select a reference viewpoint v_p and generate two depth images for both v_s and v_p.

3.4 3D Video Remote Rendering

According to the introduction before, remote rendering 3D video for mobile devices has three challenges. First, the huge size of the original 3D video streams makes it very difficult to transmit through wireless networks and render on mobile devices in real-time. Second, the dynamic nature of 3D video requires both the server and the client to intensively compute every 3D video frame before fixed deadline. Third, the wireless networks used by mobile devices are not reliable and can cause long *rtt*. Obviously, the image-based approaches that require less network bandwidth and client computation are more appropriate for 3D video rendering than model-based approaches. Considering the pros and cons of each image-based approach discussed above, we believe a hybrid solution can be the best for 3D video.

Due to the setup of 3D video capturing (Figure 1(a)), only a limited range of viewpoints are necessary for 3D video rendering. Thus the 3D video interaction can be simplified to four modes [20, 6] and we list here how each interaction mode changes viewpoint.

- *Zoom*: Move *pos* along *dir*
- *Pan & Tilt*: Change *dir* horizontally or vertically.
- *Translate*: Change *pos* horizontally or vertically.
- *Orbit*: Move *pos* along a circle and keep *dir* pointing to the center of the circle.

When the client is at *pan & tilt* mode, the server generates an environment map as the reference frame. When the client is at other interaction modes (*translate*, *orbit*, or *zoom*), we take the approach of double warping. Since environment map already achieves minimum *response time* for *pan & tilt* (the *penalty probability* is 0 because *pan & tilt* does not change *pos*), we focus on how to improve double warping in this paper.

According to Table 1, we can reduce the *response time* of double warping by increasing V_{dwarp}. Thus, we change the problem of designing a 3D video remote rendering system to an optimization problem: how to find the reference viewpoint v_p that maximizes $\|V_{dwarp}\|$ ($\|V_{dwarp}\|$ represents the number of viewpoints contained in V_{dwarp}).

Before studying how v_p should be selected, we take a look at viewpoints contained in V_{dwarp}. We assume that all viewpoint changes are discrete. v_c can be represented by the following equation because it is transformed from v_s.

$$v_c = v_s + c \cdot T$$

where T is the vector for one step viewpoint change (T has different values for different interaction modes). We only care about the viewpoint $v_i = v_s + i \cdot T$ because they are candidates for v_c. Given the reference viewpoints v_s and v_p ($v_p = v_s + p \cdot T$, $p > 0$), intuitively, we think only the viewpoint v_i with $0 \leq i \leq p$ can benefit from double warping. If $i < 0$ or $i > p$, the image quality of $dwarp(v_s, v_p, v_i)$ can be no better than $warp(v_s, v_i)$ or $warp(v_p, v_i)$. This intuitive conclusion has been supported in [14]. Therefore, for the study of double warping in the rest of this paper, we omit the viewpoints that do not benefit from double warping and consider $V_{dwarp}(v_s, v_p) \subseteq [v_s, v_p]$, and

$$[v_s, v_p] = \{v_i | v_i = v_s + i \cdot T, \quad 0 \leq i \leq p\}$$

Thus

$$\|V_{dwarp}(v_s, v_p)\| \leq \|[v_s, v_p]\|$$

$\|[v_s, v_p]\|$ is the size of $[v_s, v_p]$, also referred as the *viewpoint range* of two reference viewpoints. We can further simplify the 3D video rendering problem to find the reference viewpoint v_p that

$$\max_p \{\|V_{dwarp}(v_s, v_p)\| = \|[v_s, v_p]\|\} \tag{11}$$

None of previous researches thoroughly solved this problem. [20] suggested using search to find v_p that satisfies Eq (11). However, it requires too much computation on the server and contradicts with the real-time requirement in Eq (3). In the next section, we introduce our reference viewpoint prediction algorithm and show how we can generate reference frames efficiently for double warping.

4. ENHANCED DOUBLE WARPING

In this section, we analyze how warping errors are generated and show how double warping can be improved with a computationally efficient reference viewpoint prediction algorithm.

4.1 Warping Error Analysis

The 3D warping algorithm takes a reference depth image (including both color and depth maps), the reference viewpoint at which the depth image is generated, and the target viewpoint as input, and computes the color image at the target viewpoint as output. For every pixel (u_1, v_1) in the reference depth image, the algorithm calculates the image coordinate (u_2, v_2), and copies the color from (u_1, v_1) to the pixel (u_2, v_2) in the target image. The algorithm is very efficient. It processes each pixel only once and the image coordinate calculation just takes a few arithmetic operations. Refer to [15] for more details about 3D warping.

3D warping is not perfect. "Hole" artifacts are introduced when the scene objects that do not exist in the input image become visible to the new rendering viewpoint. There is no pixel in the input image to reference when drawing the new image. It has been referred as the *exposure* problem. However, by looking into how exactly holes are generated, we find that holes can actually be divided into two categories.

The first category of holes are caused by *occlusion exposure*. The objects that are completely occluded or out of view range in the input image become exposed. The second category of holes is named as *insufficient sampling*. It happens when warping a small resolution image to a large resolution, or more frequently, on a surface with varying gradient. Figure 5 shows an example of these two types of warping errors.

The classification also helps to understand previous techniques proposed to fill holes. Recent progress on sparse-representation and data-driven hole filling appears to be

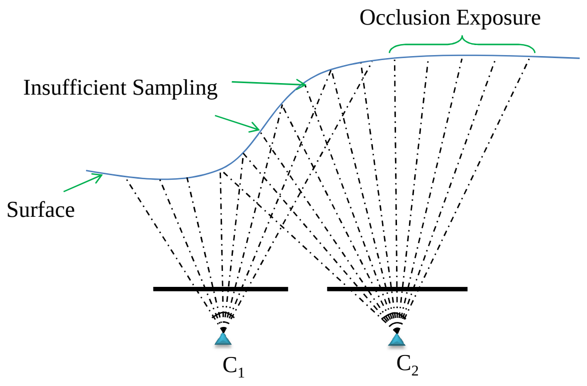

Figure 5: Occlusion Exposure and Insufficient Sampling

state of the art [16], but, too slow for real-time applications. Depth filter [18] and splat warping [14] are only useful for the small holes caused by insufficient sampling. Super view warping [3] and wide field-of-view warping [14] can partially solve the occlusion exposure but do not help with insufficient sampling at all. Double warping [14, 20], view compensation [3], and LDI [19] are effective for both holes. Although using different names, both view compensation and LDI are actually the same as double warping in nature. Images from different viewpoints are used to compensate warping errors. However, double warping best suits a 3D video remote rendering system because it does not need offline processing (LDI) or interactive communication (viewpoint compensation). Figure 6 illustrates how double warping works. We use a static bunny model instead of 3D video frames for demonstration in this paper because the holes introduced in 3D video capturing (Figure 1) can be misleading when we analyze the holes caused by 3D warping.

4.2 Reference Viewpoint Prediction

From the above analysis of warping holes, we find that the holes are generated when there are not enough pixels in the reference image. Double warping can fill holes because the pixels missing in one reference image can be found in the other one. Therefore, given one reference viewpoint (v_s, also referred as main reference viewpoint), the expected reference viewpoint v_p should provide pixels missing in the main reference image to compensate holes generated by warping.

We try to answer the following two questions first. Which viewpoints can serve to compensate most insufficient sampling holes caused by warping the depth image at v_s? Which viewpoints can serve to compensate most occlusion exposure holes of the depth image at v_s? Intuitively, for the first question, we can warp the main reference frame to all possible target viewpoints and find the viewpoints which generate the insufficient sampling hole with the largest size. These viewpoints are the best candidates because they provide most pixels to fill insufficient sampling holes. For the second question, we try to find the viewpoints which are far away from the the main reference viewpoint so that two reference viewpoint do not overlap. Therefore, the intersection of two viewpoint sets can be the best candidate of the reference viewpoint we try to predict.

We start from the analyzing *translate* mode. Assuming the viewpoint only translates to right horizontally along the

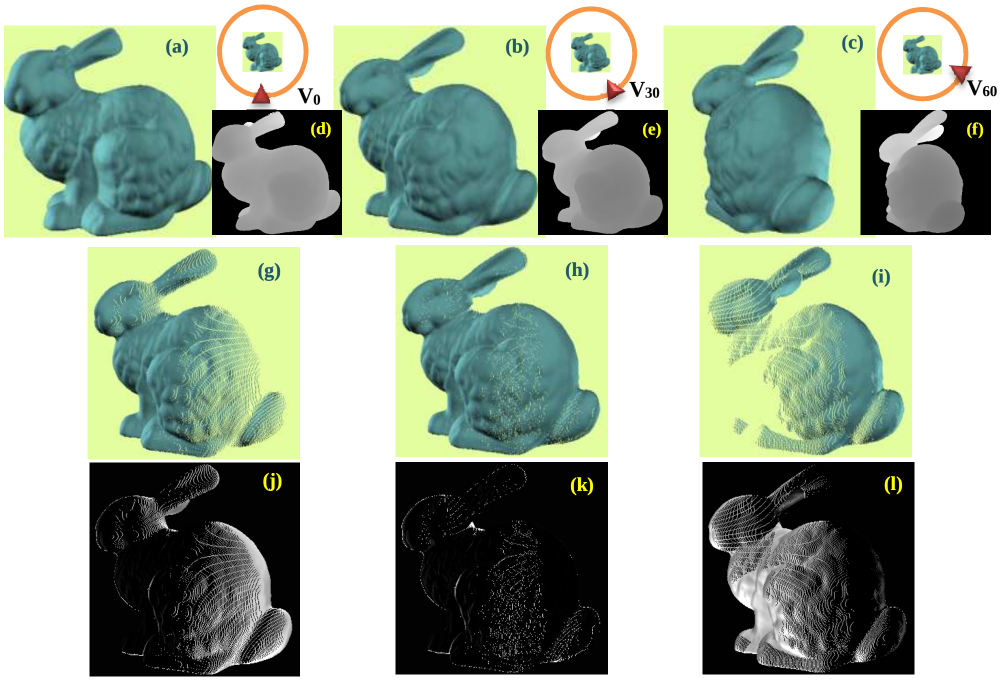

Figure 6: (a)(b)(c) Color maps rendered at viewpoints v_0,v_{30}, and v_{60}; (d)(e)(f) Depth map of the image rendered at viewpoint v_0,v_{30}, and v_{60}; (g) Image warped from v_0 to v_{30}; (h) Image double warped from v_0 and v_{60} to v_30; (i) Image warped from v_{60} to v_{30}; (j)(k)(l) The difference between (g)(h)(i) and (b), respectively

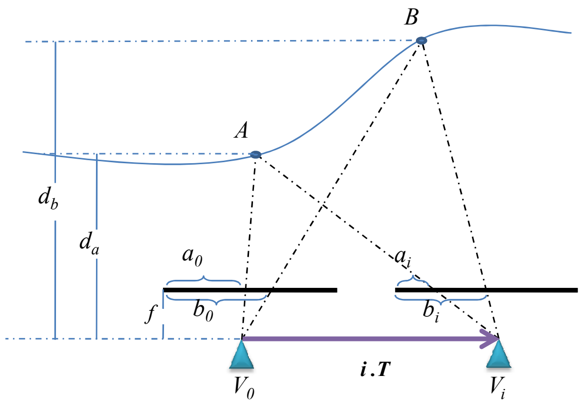

Figure 7: Warping Hole Size Analysis

axis that is parallel with the image plane, the viewpoint v can be represented by the center-of-projection C in this case. As shown in Figure 7, object points X and Y are visible to the viewpoint at C_0, and the image coordinates are **a**, **b**, respectively. Since there is no movement in the vertical direction, we use only a_0 and b_0 to denote the projection of X and Y on the image plane. d_a and d_b are the depth value of X and Y. For any new viewpoint

$$C_i = C_0 + i \cdot T,\ i > 0$$

where T is the unit vector of camera translation, a_i and b_i are the new image coordinates of X and Y. We can calculate a_i and b_i as:

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

f is the distance between the center-of-projection and the image plane. The warping holes can be generated between X and Y, if $b_i - a_i \geq b_0 - a_0$, and the hole size is represented by

$$h = T \cdot f \cdot i \left(\frac{1}{d_a} - \frac{1}{d_b} \right) \tag{12}$$

Since T, f, d_a, d_b are all know parameters, the hole size h is a only decided by i. Therefore, if $d_a < d_b$, the warping from C_0 to C_i can generate a hole between X and Y, and the hole has the maximum size h_{max} when the viewpoint moves to C_{imax} where the projection of X reaches the left boundary of the image plane.

$$i_{max} = \frac{d_a \cdot a_0}{T \cdot f} \tag{13}$$

$$h_{max} = a_0 \left(1 - \frac{d_a}{d_b} \right) \tag{14}$$

C_{imax} is also the farthest viewpoint from C_0 that can help compensate holes caused by insufficient sampling between X and Y. We select this viewpoint as one candidate of the reference viewpoint to predict.

We propose our reference prediction algorithm in Table 2. The general idea is to calculate h_{max} and i_{max} for every adjacent pixels in the image if they both have valid depth values. If h_{max} is large enough to create a hole, we add corresponding i_{max} to the candidate list. The algorithm output v_p is elected from the candidate list. Conservatively, the minimum value in the candidate list should be selected as v_p so that all possible insufficient sampling warping errors can be covered. However, since the goal of our system design is to maximize the reference viewpoint v_p rather than

Table 2: Reference Viewpoint Prediction Algorithm

```
$Tf = T_{unit} \cdot \frac{width}{2} / \tan\left(\frac{foc}{2}\right)$
$< img_0, d_0 > = render(S, C_0)$
FOR $u = 1$ TO $width$
   FOR $v = 1$ TO $height$
      IF $d_0(u, v) \leq 0$
         CONTINUE
      END
      IF $u \neq width$ && $d_0(u+1, v) \leq 0$
         CONTINUE
      END
      IF $u \neq width$
         $h_{max} = u \cdot (1 - d_0(u, v)/d_0(u+1, v))$
         IF $h_{max} < 0.5$
            CONTINUE
         END
      END
      $i_{max} = u \cdot d_0(u, v)/Tf$
      $add\_to\_list(candidate\_list,\ i_{max})$
   END
END
RETURN $k$th-min($candidate\_list$)
```

error free rendering, we can be more aggressive by selecting the kth-min value in the candidate list as the reference viewpoint. We will compare different k selections in the evaluation section.

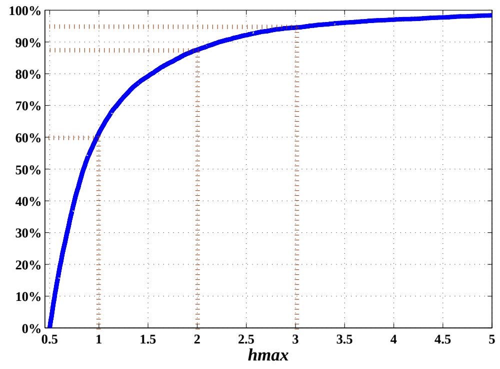

Figure 8: Cumulative Distribution of h_{max}

4.3 Super Sampling

Because the algorithm neglects the size difference of holes, we are still not satisfied with the solution mentioned above. It is possible that a lot of i_{max} candidates in the candidate list are generated by small holes. In order to verify this hypothesis, we calculate the cumulative distribution of h_{max} of the bunny model. Figure 8 indicates that 60% holes have the size less than 1. Therefore, we propose a super sampling approach which intends to fill small holes caused by insufficient sampling. We render the main reference depth image at a higher resolution for super sampling pixels. For example, rendering at a double size gives one more pixel to each adjacent pixel pair in the original resolution, which can potentially eliminate all insufficient sampling holes with $h_{max} \leq 1$.

Table 3: Reference Viewpoint Prediction Algorithm with Super Sampling

```
$Tf = T_{unit} \cdot \frac{width}{2} / \tan\left(\frac{foc}{2}\right)$
$< img_0^{2^n}, d_0^{2^n} > = render(S, C_0, 2^n \times width, height)^1$
FOR $i = 0$ TO $n - 1$
   $img_0^{2^i} = down\_scale(img_0^{2^n \times}, 2^i \times width, height)^2$
   $d_0^{2^i} = down\_scale(d_0^{2^n \times}, 2^i \times width, height)$
END
FOR $u = 1$ TO $width$
   FOR $v = 1$ TO $height$
      IF $d_0^1(u, v) \leq 0$
         CONTINUE
      END
      IF $u \neq width$ && $d_0^1(u+1, v) \leq 0$
         CONTINUE
      END
      IF $u \neq width$
         $h_{max} = u \cdot (1 - d_0^1(u, v)/d_0^1(u+1, v))$
         $i = \lfloor \log(h_{max} + 1) \rfloor$
         IF $i < n$
            $add\_resolution(img_0^1, u, v, img_0^{2^{(i+1)} \times})^3$
            $add\_resolution(d_0^1, u, v, d_0^{2^{(i+1)} \times})$
         END
      END
      $i_{max} = u \cdot d_0^{1\times}(u, v)/Tf$
      $add\_to\_list(candidate\_list,\ i_{max})$
   END
END
RETURN $k$th-min($candidate\_list$)
```

The overhead of super sampling is that it increases the size of the reference frame and requires more network bandwidth for streaming. There are several improvements to overcome this. First, when the viewpoint is moving horizontally, it is not necessary to increase the vertical resolution. Second, we do not store the whole main reference frame in the larger resolution, but only add super sampling pixels to the pixel pairs which can cause small holes. Table 3 shows an improved version of reference viewpoint prediction algorithm with super sampling support. The algorithm is scalable up to $2^n \times width$ super sampling.

Our reference viewpoint prediction algorithm is efficient. Except for the 3D rendering at the main reference viewpoint, which can be reused for reference frame generation, the rest part of the algorithm barely needs any computation. Adding super sampling requires more memory space to store depth images at multiple resolution and extra computation for downscaling. However, those memory and computation requests are both proportional to the image resolution, which is decided by the display screen size of the client device.

The derived algorithm only applies when the viewpoint translates to right horizontally. For other directions (e.g., left, up, or down), the algorithm can be derived in a similar procedure but with different Eq (13)(14). Our algorithm can also be modified for different user interaction modes, such as *orbit* and *zoom*. However, the hole size function for those modes becomes more complicated than Eq (12). We need to approximate h_{max} and i_{max} for different modes. The

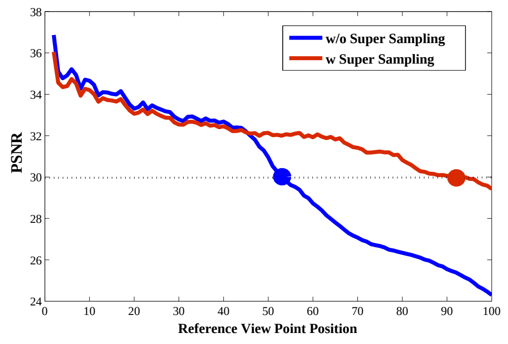

Figure 9: Optimal Rendering Viewpoint

other parts of the algorithm and super sampling can apply to different user interaction modes without modification.

5. EVALUATION

We have built the framework for remote rendering 3D video and streaming to both Nokia N800 tablet and iPhone in our previous work [20]. As a summary, the proxy server receives original 3D video, generates two depth images as the reference frame, and sends to the mobile client through 802.11g Wi-Fi. Only one mobile client is supported for now, and we achieve the frame rate up to 10 fps for the resolution of 320×240. Please refer to original paper for details on system setup and networking statistics. In this section, we mainly evaluate how our reference viewpoint prediction algorithm compares with previous search-based algorithms. We use the same bunny model shown in Figure 6 instead of noisy 3D video frames so that the PSNR results are more convincing. The 3D model is rendered at the resolution of 250×250.

We first generate Figure 9 to find the optimal reference viewpoints as the baseline for further comparison. Let the main reference viewpoint $v_s = v_0$. The *x-coordinate* indicates the viewpoint v_x is x steps away from v_s. The *y-coordinate* indicates the PSNR of the worst image that double warping can generate, or formally expressed as

$$f_{PSNR}(x) = \min\{\forall v_i \in [v_s, v_x], psnr(dwarp(v_s, v_x, v_i))\}$$

The dashed line is the threshold of err_{resp} in Eq (10). The red and blue lines stand for double warping with and without the support of super sampling pixels. We can observe that both lines remain in high PSNR region when x is relatively small. As x increases, the line that corresponds to double warping without $2 \times width$ super sampling support drops much faster. It is because two reference viewpoints are too far away from each other to cover holes caused by insufficient sampling. The intersection points of two lines with the threshold line are considered as the optimal reference viewpoints that satisfy Eq (11).

[1]$render(S, C, width, height)$ renders the source S at the viewpoint C and generates the depth image $< img, dep >$ at the resolution $width \times height$.

[2]$down\_scale(img, width, height)$ downscales the image to the new resolution $width \times height$.

[3]$add\_resolution(img, u, v, img')$ adds super sampling pixels from img' to img between pixel (u, v) and $(u + 1, v)$.

Table 4: Viewpoint Prediction Results

Algorithm & Configuration	Viewpoint Range	PSNR (dB)
w/o Super Sampling, $k = 1$	40	32.68
w/o Super Sampling, $k = 5\%$	48	31.47
w/o Super Sampling, $k = 15\%$	56	29.52
Optimal, w/o Super Sampling	52	30.27
w. Super Sampling, $k = 1$	49	32.12
w. Super Sampling, $k = 5\%$	53	32.01
w. Super Sampling, $k = 15\%$	63	31.88
Optimal, w. Super Sampling	91	30.04
Full Search	6	34.92
Fast Median Search	16	33.99

Table 4 compares the *viewpoint range* calculated by our proposed algorithm with the optimal results derived from Figure 9. We select both algorithms with and without $2 \times$ *width* super sampling support, and for each algorithm we select different kth-min. We also compare our results with the search-based algorithms (full search and fast median search) from [20]. Since the search algorithm takes very long time to calculate the optimal results, we add a time constraint to the algorithm execution by assuming the FPS_{src} is 15 frame per second. All algorithms are forced to stop after 66 ms. Thus the results also reflect the computation complexity of different algorithms. For each algorithm execution, we record the predicted reference viewpoint v_p and $f_{PSNR}(p)$.

We can draw two conclusions from the results. First, when super sampling is not considered, the results of our reference prediction algorithm are far better than the results of search based algorithms, and close to the optimal value. According to Eq (6), it also indicates that the double warping remote rendering system using our new algorithms has lower *interaction delay* when compared with original system because the new viewpoint has a higher probability to fall into V_{dwarp}. Second, using super sampling can further increase the performance of our algorithm. However, our algorithm is conservative in prediction when super sampling is supported. It is mainly because we assume the $2 \times width$ super sampling pixels can only remove holes with $h_{max} \leq 1$. A more aggressive selection of h_{max} threshold or k can help generate some results closer to the optimal.

6. DISCUSSION

We have not addressed in previous sections how the viewpoint motion can be predicted. The algorithm in Section 4 applies to the viewpoint motion that moves to right. However, what if the user decides to move left or zooming out? Previous work [7, 14] tried to predict the viewpoint motion based on history and [20] proposed to combine both history and pattern for motion prediction. Since we have proposed a computationally efficient algorithm, we can run the algorithm multiple times for different directions or different modes, and generate the depth image for each predicted reference viewpoint. It takes more bandwidth to stream all depth images but keeps the system performance less dependent on motion prediction, which can vary a lot in performance for different users.

Although the whole remote rendering system is designed for 3D video streaming, we believe our system can be extended to render general graphic contents for mobile devices as well, such as 3D graphic games. The reference frame selection algorithms can be implemented at the end of graphic pipeline on the rendering server to generate depth images as the reference frame for mobile devices. However, the proposed remote rendering system can only be applied to games that only allow players to change rendering viewpoint. Other games with rich user interaction, such as shooting games, can not be supported in our system.

7. CONCLUSION

In this paper, we have built a generalized model to study different remote rendering systems, and used the model to analyze a specific problem of rendering dynamic 3D video streams for mobile devices. Generally, our work answered three questions: given a reference frame, how to correctly render the scene to a different viewpoint: double warp from two reference viewpoints to compensate warping errors; how to select reference frames for best performance: Eq (11); and how to select reference frames efficiently: reference viewpoint prediction based on depth analysis and super sampling. For the future work, we are going to study the remote rendering system that supports broadcasting the same content to multiple mobile clients. We will try to improve reference frame selection algorithm for more complex motions and allow users to request similar views to share the same reference frames.

8. ACKNOWLEDGMENTS

This work was supported by the National Science Foundation under Grant CNS 05-20182 and CNS 07-20702. Any opinions, finding and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

9. REFERENCES

[1] Bumblebee 2 camera. http://www.ptgrey.com/products/bumblebee2/.

[2] Teeve project. http://cairo.cs.uiuc.edu/teleimmersion/.

[3] P. Bao and D. Gourlay. Remote walkthrough over mobile networks using 3-D image warping and streaming. *Vision, Image and Signal Processing, IEE Proceedings*, 151(4):329–336, Aug. 2004.

[4] A. Boukerche and R. Pazzi. Remote rendering and streaming of progressive panoramas for mobile devices. In *Multimedia '06*, pages 691–694. ACM, 2006.

[5] C.-F. Chang and S.-H. Ger. Enhancing 3D graphics on mobile devices by image-based rendering. In *PCM '02*, pages 1105–1111, London, UK, 2002.

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

[7] S. E. Chen and L. Williams. View interpolation for image synthesis. In *SIGGRAPH '93: Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, pages 279–288, 1993.

[8] M. Claypool and K. Claypool. Latency can kill: precision and deadline in online games. In *MMSys '10: Proceedings of the first annual ACM SIGMM conference on Multimedia systems*, pages 215–222, New York, NY, USA, 2010. ACM.

[9] F. Duguet and G. Drettakis. Flexible point-based rendering on mobile devices. *IEEE Trans. Computer Graphics and Applications*, 24(4):57–63, 2004.

[10] K. Engel, R. Westermann, and T. Ertl. Isosurface extraction techniques for web-based volume visualization. In *IEEE Visualization*, 1999.

[11] H. Kim, I. Kitahara, R. Sakamoto, and K. Kogure. An immersive free-viewpoint video system using multiple outer/inner cameras. In *3DPVT*, pages 782–789, 2006.

[12] G. Kurillo, R. Vasudevan, E. Lobaton, and R. Bajcsy. A framework for collaborative real-time 3D teleimmersion in a geographically distributed environment. In *ISM*, pages 111–118, 2008.

[13] F. Lamberti and A. Sanna. A streaming-based solution for remote visualization of 3D graphics on mobile devices. *IEEE Trans. Vis. Comput. Graph.*, 13(2):247–260, 2007.

[14] W. R. Mark. Post-rendering 3D image warping: Visibility, reconstruction, and performance for depth-image warping. In *Ph.D. Dissertation.* University of North Carolina at Chapel Hill, Department of Computer Science, 1999.

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

[16] H. Mobahi, S. R. Rao, and Y. Ma. Data-driven image completion by image patch subspaces. In *PCS'09: Proceedings of the 27th conference on Picture Coding Symposium*, pages 241–244, Piscataway, NJ, 2009.

[17] S. Prohaska, A. Hutanu, R. Kahler, and H.-C. Hege. Interactive exploration of large remote micro-ct scans. In *VIS '04: Proceedings of the conference on Visualization '04*, pages 345–352, Washington, DC, USA, 2004. IEEE Computer Society.

[18] A. Redert, M. O. de Beeck, C. Fehn, W. IJsselsteijn, M. Pollefeys, L. J. V. Gool, E. Ofek, I. Sexton, and P. Surman. Attest: Advanced three-dimensional television system technologies. In *3DPVT*, pages 313–319. IEEE Computer Society, 2002.

[19] J. Shade, S. Gortler, L. He, and R. Szeliski. Layered depth images. In *SIGGRAPH*, pages 231–242, 1998.

[20] S. Shi, W. J. Jeon, K. Nahrstedt, and R. H. Campbell. Real-time remote rendering of 3D video for mobile devices. In *MM '09: Proceedings of the seventeen ACM international conference on Multimedia*, pages 391–400, New York, NY, USA, 2009. ACM.

[21] A. Smolic, K. Müller, P. Merkle, C. Fehn, P. Kauff, P. Eisert, and T. Wiegand. 3D video and free viewpoint video - technologies, applications and mpeg standards. In *ICME*, pages 2161–2164. IEEE, 2006.

[22] S. Würmlin, E. Lamboray, O. G. Staadt, and M. H. Gross. 3D video recorder: a system for recording and playing free-viewpoint video. *Comput. Graph. Forum*, 22(2):181–194, 2003.

[23] Z. Yang, K. Nahrstedt, Y. Cui, B. Yu, J. Liang, S.-H. Jung, and R. Bajcsy. Teeve: The next generation architecture for tele-immersive environment. In *ISM*, pages 112–119. IEEE Computer Society, 2005.