Implementing a Distributed 3D Tele-Immersive System*

Wanmin Wu, Zhenyu Yang, Klara Nahrstedt
Department of Computer Science
University of Illinois at Urbana-Champaign
{wwu23, zyang2, klara}@uiuc.edu

Abstract

We present our implementation and evaluation of TEEVE, a distributed 3D tele-immersive system. TEEVE is among the first to support multi-stream/multi-site 3D tele-immersive environments with COTS hardware and software infrastructures. It promotes collaborative physical activities among geographically dispersed sites by immersing the 3D representations of remote participants into a joint 3D virtual space. In this paper, we describe our implementation of TEEVE and introduce the recent advances in TEEVE's different components. In particular, we present an implemented protocol for ViewCast-based semantic-aware data dissemination to support multi-site remote collaboration. We evaluate the TEEVE system by deploying it on the Internet. The experimental results demonstrate that it achieves stable visual quality, soft real-time delay, and efficient resource usage.

1 Introduction

TEEVE (Tele-immersive Environments for Everybody) is a distributed 3D tele-immersive (3DTI) system that has been under development jointly at University of California at Berkeley and University of Illinois at Urbana-Champaign since 2004. The system is designed to capture 3D full-body human motion in geographically distributed sites, aggregate the video data across the Internet, and visualize them in a joint "cyber-space" at each site. Through the shared visual context (i.e., the cyber-space), the participants can interact or collaborate in real time (Figure 1). Jung *et al.* [9] presented an overview of the system, with an emphasis on the "vision" components (e.g., image processing, 3D reconstruction). Yang *et al.* [18] detailed the design and implementation of the end-to-end data adaptation and transmission in the network between two sites.

Figure 1. Multi-site 3DTI collaboration

In this work, we describe the implementation of TEEVE and its most recent advances in different components. Moreover, the system was initially designed for two-site collaboration due to the huge demand of computing and networking resources. A slight increase in scale posed great challenges to the system [16]. We present our data dissemination protocol to support *multi-stream/multi-site* collaboration. We also examine the TEEVE system as a pipeline by deploying it on the Internet, and evaluate the system and networking performance of its different components. The experimental results demonstrate that it is scalable, stable, interactive, and resource-efficient.

The remainder of the paper is organized as follows. We describe the hardware and software components of TEEVE in Section 2, and provide the performance evaluation results in Section 3. We then discuss related work in Section 4, and finally conclude in Section 5.

2 The TEEVE System

TEEVE is meant to support real-time collaborative physical activities among multiple geographically distributed sites. In this section, we describe the details of the hardware components (Section 2.1) and software components (Section 2.2 - Section 2.4), respectively. When designing the TEEVE system, we made the following decisions that

*This research is supported by the National Science Foundation (NSF) under grants SCI 05-49242, CNS 05-20182, and NSF IIS 07-03756. The presented views are those of authors and do not represent the position of NSF.

distinguish us from previous work.

* *Build upon Commercial Off-The-Shelf (COTS) computing and networking infrastructures*. There existed other tele-immersive environments with very special purpose real-time operating systems, leased networks, or other special purpose and expensive hardware. But the cost is apparently too high for the system to work for "everybody" [1]. Our goal is to build the system with cheap, off-the-shelf infrastructures, so that it can be easily and widely deployed.

* *Capture full-body human motion in 3D representation*. Conventional video-mediated collaborative environments are mostly designed for desktop uses (e.g., conferencing, lecturing, etc.), hence minimal number of cameras are employed only to capture the upper bodies or the faces [1, 2, 7, 11]. Our goal is to broaden the application of the tele-immersive technology by supporting full-body capturing of the participant. The complete 3D representations of human body allow for a wide range of physical activities such as dancing, sports, and medical rehabilitation.

2.1 Hardware Components

Each local TEEVE environment (i.e., site) has three main tiers: *capturing*, *data dissemination*, and *rendering*, all of which use COTS hardware.

Capturing. Multiple *3D camera clusters* capture the local participant in 3D representation from a wide field of view. The clusters are mainly mounted on two different levels of metal frames (Figure 2), with the top ones (about 6 feet from floor) capturing the upper body, and the lower ones (about 3 feet from floor) capturing the lower body. We also mount two camera clusters on the middle level (about 4.5 feet from floor) that increase overlap among cameras to facilitate calibration.Each 3D camera cluster is hosted by a computer (*camera host PC*) which handles image grabbing and 3D reconstruction (Section 2.2).

A *trigger server* is used to synchronize all cameras to grab images at the same instants of time. This is done by periodically sending a hardware trigger signal from the parallel port on the server to a pair of general purpose I/O pins on each camera. We use the Multi-Camera Self-Calibration toolkit [14] for both intrinsic and extrinsic calibration of all cameras.

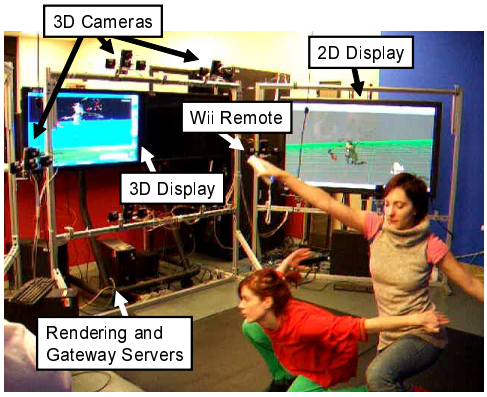


Figure 2. Hardware setup in TEEVE

Data Dissemination. We use a hierarchical topology for scalability. At the *local* level (i.e., within each site), the end hosts (i.e., camera host PCs, display host PCs) are managed by a *service gateway*, which is responsible for both outbound and inbound traffic. For outbound traffic, the gateway collects local streams from the camera host PCs, and disseminates them out to the network, i.e., other gateways. For inbound traffic, it receives the video streams from remote sites, and distributes both local and remote streams to the local displays. At the *global* level, all gateways are managed by a *session controller* which handles the membership and overlay topology construction for all gateway nodes.

Rendering. Multiple *displays* are placed in different positions and angles to present the scene, such that the participant can observe it even when she moves or turns (Figure 2). In our implementation, we employ two 61-in 2D plasma displays, and two 42-in 3D autostereoscopic displays for rendering the aggregated cyber-space for the users. Each display is connected to a PC (i.e., *renderer*) which mainly handles real-time rendering of the 3D video streams. It also provides users an interface to change the rendering viewpoint in the 3D cyber-space (Section 2.4). This is important because the power of the 3D data representation lies in its capability of allowing users to see the cyber-space from arbitrary view angles.

2.2 Capturing and 3D Reconstruction

Jung *et al.* [9] summarized the basic image processing and 3D reconstruction algorithms used in TEEVE. However, the achieved frame rate was less than 6 frames per second (fps). We take full advantage of today's multi-core computer architecture by parallelizing different algorithmic components.

Figure 3 (in comparison with Figure 7 in the original paper [9]) illustrates the basic idea of our approach. We use four threads for image capturing (one on each camera) and n threads for computing where n is the number of processor cores in the system. The camera threads and computing threads work in parallel, that is, once the current set of images becomes ready, the computing threads start to process them, and in the meanwhile the camera threads continue to capture the next set of images.

The camera threads grab images from the camera buffers, and perform some pre-processing including rectification, moments computation, background subtraction, and edge extraction. After pre-processing, the images are

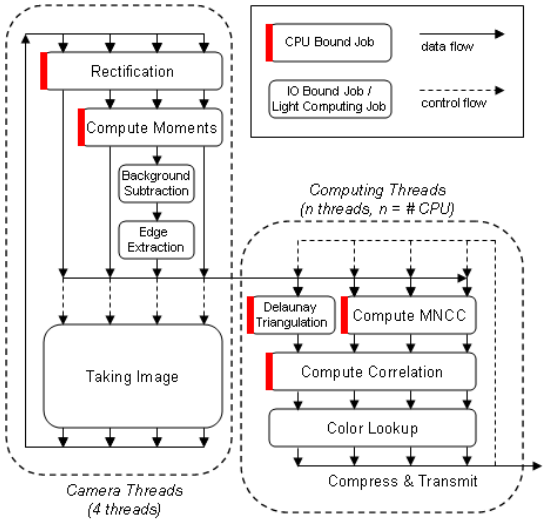


Figure 3. Parallel reconstruction

copied to the computing threads. Instead of dividing the images to only two halves as in the original algorithm, we divide the images into $n-1$ pieces for computing. There are three CPU-bound tasks among the computing threads: Delaunay Triangulation, Computing MNCC (Modified Normalized Cross Correlation measures), and Computing Correlation [9]. We parallelize Delaunay Triangulation with Compute MNCC because there is no dependancy between the two. Therefore, one thread is dedicated for triangulation, and the other for MNCC[1]. After this stage, we use all processor cores to compute correlation at each edge point obtained from edge extraction. Then the 3D image frame, i.e., the depth and color map acquired from the Computing Correlation and Color Lookup modules respectively, is compressed and transmitted to the service gateway. In our implementation, we use motion JPEG to compress the color map, and zlib to compress the depth map.

2.3 Data Dissemination

The camera host PCs continuously stream the reconstructed 3D video data to the local gateway for content delivery. The *data dissemination* component running on each service gateway is then responsible for efficient and adaptive data distribution across the Internet.

Supporting multi-stream/multi-site 3D tele-immersion is challenging due to the huge demand of networking and computing resources. The scalability of the 3DTI systems is not in the number of sites, but in the *streaming density* scale among a small number of sites. This is a very different scalability model from traditional peer-to-peer media streaming applications. In 3DTI environments, a slight increase in the number of sites significantly increases the streaming density in the system, due to the large number of I/O devices present in each site and the large amount of system/network resources each data stream requires.

ViewCast [19] was proposed as a *semantics-aware data dissemination model* to solve the problem. The basic idea is to select data for transmission based on their semantical importance, and construct a dissemination topology adaptive to network and system dynamics (e.g., available bandwidth, end-to-end latency, resource usage, etc.). More specifically, the semantical importance of the streams is dynamically determined with reference to the view selected by the user. Only a subset of streams that is contributing to this view will be transmitted. A substrate component manages the stream selection, overlay construction, session management, and data adaptation among sites, making it transparent to the users. The substrate also keeps track of available bandwidth, point-to-point latency in the network, and resource usage of all service gateways. More importantly, it maintains an adaptive application-level overlay to deliver the semantically important streams to the users based on their selected "views".

2.3.1 ViewCast Protocol

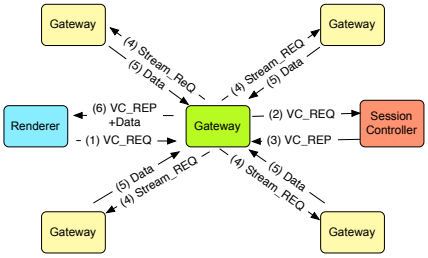


Figure 4. ViewCast protocol

Figure 4 illustrates the procedures from requesting a view to retrieving the contributing streams. At any given time, the streams transmitted always contribute the most the selected view. Therefore, every view change made by the user triggers a ViewCast request. Below we describe the basic steps in the protocol:

1. **Requesting Renderer (R_q) $\xrightarrow{VC\_REQ}$ Local Gateway (G_q).** On detecting a view change made by the user, the renderer sends a VC_REQ (ViewCast Request) message to the local gateway. Figure **??** shows the format of this request message. The message payload consists of a list of ViewCast request items (VC_REQ_Item), with each item representing a request for one 3D video stream. The renderer selects these streams by computing the *semantic importance*

[1]This is why we divide the images into $n-1$ pieces, not n.

of each stream with reference to the user-selected *rendering view*. The message is filled up by the renderer, and sent to the local gateway.

2. **Local Gateway (G_q) $\xrightarrow{VC\_REQ}$ Session Controller.** On receiving the VC_REQ message from a requesting renderer (R_q), the local gateway fills in the first two entries in the message header: (a) $gateway\_seq\_num$ - a monotonically increasing sequence number the gateway uses to keep track of its requests, and (b) $gateway\_addr$ - the IP address of itself. It then forwards this request message to the session controller.

3. **Session Controller $\xrightarrow{VC\_REP}$ Local Gateway (G_q).** On receiving the VC_REQ message from the requesting gateway G_q, the session controller sends a VC_REP (ViewCast Reply) message back. The message payload consists of a number of ViewCast reply items (VC_REP_Item), with each item representing a reply for one requested stream. The session controller generates these reply items by examining the current topology and network/system dynamics, and selecting a parent node (G_p^i), if possible, to serve the requesting gateway G_q with each requested stream in $VC\_REQ\_Item_i$. The message is filled up by the session controller, and sent to the requesting gateway (G_q).

4. **Local Gateway (G_q) $\xrightarrow{Stream\_REQ}$ Remote Gateways ($G_p^1, G_p^2, ..., G_p^m$).** The requesting gateway examines the VC_REP message, and then sends a $Stream\_REQ$ message to the "parent" gateway (G_p^i) specified in each $VC\_REP\_Item_i$ (refer to Figure 4). The $Stream\_REQ$ message contains the stream name and the IP address of its requesting gateway.

5. **Remote Gateway (G_p^i) $\xrightarrow{Data}$ Local Gateway (G_q).** The remote gateway (G_p^i) then resolves $Stream\_REQ$ by sending the requested stream to G_q.

6. **Local Gateway (G_q) $\xrightarrow{Data}$ Requesting Renderer (R_q).** Receiving the stream from the relay gateway G_p^i, the local gateway distributes this stream to the requesting renderer R_q, together with a ViewCast reply message (VC_REP). To this point, the ViewCast request is resolved.

2.3.2 Stream Selection by Renderer

The renderer selects a subset of most contributing streams to the user-selected view by computing the *semantic importance* of each stream with reference to the given view.

The rendering view, $\vec{V_r}$, is modeled as a vector in the 3D cyber-space. We define the *view* of a stream, $\vec{V_s}$, to be the normal vector of the imaging plane of the camera that produces the stream s, which can be obtained by the calibration parameters acquired via the initialization phase. The *importance* of a stream, $I_s^{V_r}$ with respect to a given rendering view, $\vec{V_r}$, is thereby the scalar product of the two vectors [20]: $I_s^{V_r} = \vec{V_r} \cdot \vec{V_s}$. The number of streams to select from each site is a tunable parameter. One can either select the top k number of streams for a given view, or a dynamic number of streams with importance larger than a threshold TH_{imp}. In our implementation, we take the second approach where $TH_{imp} = 0$ thus the selected streams cover 180° of the scene.

2.3.3 Overlay Management by Session Controller

The session controller maintains three types of information: (a) membership, (b) network and system dynamics, and (c) topology. For membership, it acquires a list of gateway nodes, and a list of participating camera streams via the session initialization protocol. It also keeps track of the network and system dynamics such as point-to-point latency and end-to-end available bandwidth between pairs of gateway nodes. Such dynamic information is needed for the overlay construction. Most importantly, the session controller maintains the overlay topology, i.e., how the gateway are connected on the application-level overlay for stream dissemination [16, 19].

2.4 Rendering

Since all cameras are pre-calibrated, the renderer does not compute the correlation among different video streams. In other words, it renders the streams independently after decompression. Note that all streams are progressively rendered into a single 3D representation. Figure 5 shows two screenshots of twelve streams in aggregation. The renderer provides users an interface for manipulating the view either with a Wii remote [15] or a mouse. The viewpoint selected

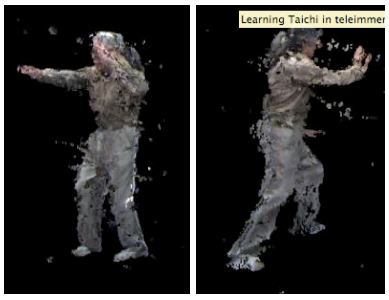


Figure 5. Aggregation of twelve 3D video streams of a participant playing Tai-Chi

via this interface triggers the ViewCast protocol (Section 2.3).

3 Performance

We evaluate TEEVE's different components including 3D reconstruction and 3D rendering with cameras and displays respectively. We also investigate the performance of the ViewCast protocol in a distributed setup of the system.

3.1 Experimental Setup

We perform experiments in four major sites, University of California at Berkeley (UCB), University of Illinois at Urbana-Champaign (UIUC) Siebel Center, University of Central Florida (UCF), and National Center for Supercomputing Applications (NCSA). We have set up the 3DTI environments (including cameras, triggers, displays) in UCB, UIUC, and NCSA, and only included UCF for testing purposes (i.e., installed gateway and renderer but no cameras)[2]. UCB, UIUC, and UCF are connected by Internet, while UIUC and NCSA are on the campus network. The session controller is placed at UIUC for administrative convenience.

Since the complexity of 3D data affects a number of metrics such as latency and visual quality, we use four sets of twelve pre-recorded 3D video streams (i.e., emulating twelve camera clusters) for all experiment (except the one evaluting 3D reconstruction which has to use real cameras). Each set captures a person practicing Tai-Chi (e.g., Figure 5). Real users are invited to watch the Tai-Chi lecture. During the session, the user can rotate his/her view. The view change trace of each user is recorded so that we can perform more experiments by feeding in user input automatically.

The camera host PCs have 2.80 GHz CPU, 2GB RAM, running Windows XP. The cameras capture imagery at the resolution of 320x240, and at the frequency of 15 fps (limited by the bandwidth of the FireWire interface on the PC). The display host PCs have 3GHz CPU, 1GB Ram, running Windows XP. The service gateways have the same configuration as display host PCs. Within a TEEVE environment, the PCs share an 1Gbps Ethernet switch. All of our code is written in C and C++. The renderer code heavily uses the OpenGL library. It also relies on an nVidia GeForce 6800XT GPU for graphics processing.

3.2 Latency

Latency is a critical measure in real-time streaming applications. Timely delivery has to be guaranteed for the interactivity among remote participants. We break down the latency from the end-to-end path (i.e., from a camera host PC in one site to a display host PC in another site), with the results shown in Section 3.2.1. We also more carefully examine the three main components that introduce latency: reconstruction (Section 3.2.2), transmission (Section 3.2.3), and rendering (Section 3.2.4).

3.2.1 Overview

Figure 7 gives a rough overview of the latency breakdown from one end to the other. The values are computed as the average over about three thousand frames. More specifically, it measures the time from the moment where twelve cameras are triggered to capture frames at the same instant of time (which constitute a *macro-frame*) to the moment where the macro-frame is rendered to the remote users in real time.

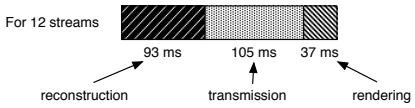


Figure 7. Latency on display host PC

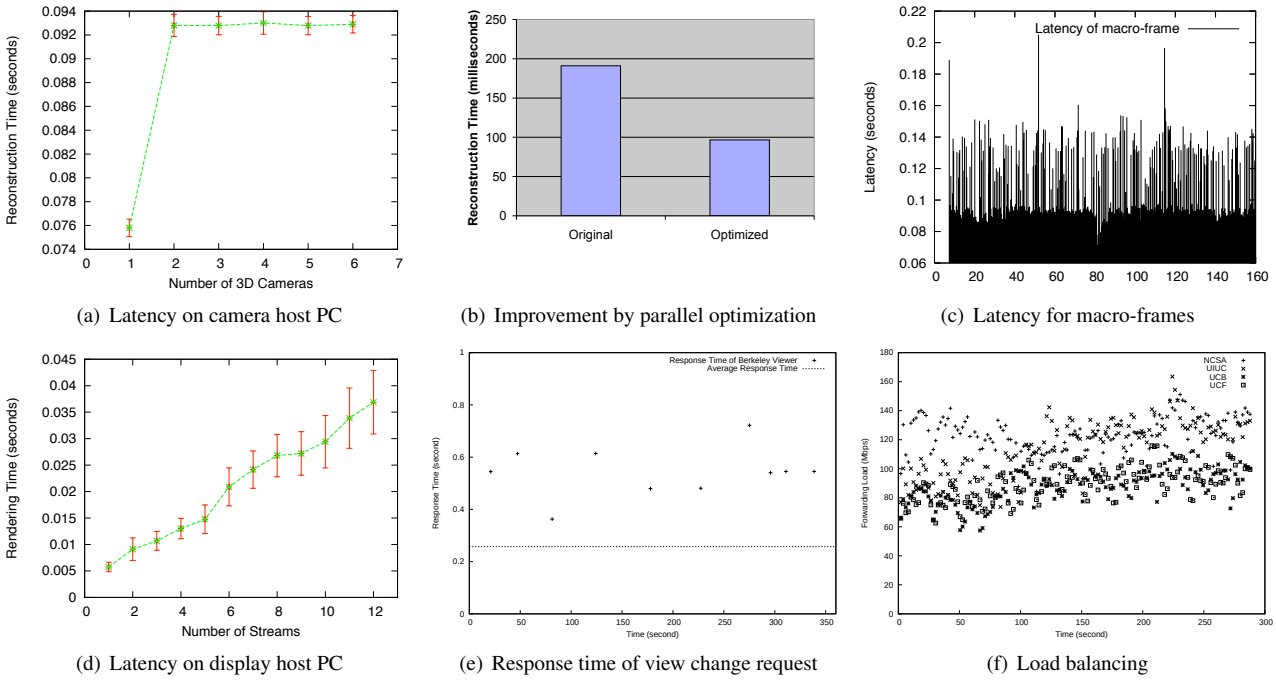
3.2.2 Reconstruction Latency

Figure 6(a) shows the latency incurred at each camera host PC for a single frame, which is mainly due to image processing and 3D reconstruction. For this experiment, six camera clusters are calibrated and used. Since the reconstruction latency largely depends on the complexity of the scene, we use a "benchmark" object, a $35'' \times 24''$ checkerboard with 9×7 number of $2.5''$ squares, as the foreground object for all experiments. The averages and standard deviations shown in Figure 6(a) are taken over about three thousand frames.

We note that with only one camera cluster running, the latency on the host PC is about 76 ms, and with two or more clusters, the latency grows to about 93 ms. The difference (about 17 ms) is due to the synchronization overhead, where the trigger waits for ACKs from all hosts before firing the next trigger signal. We observe that this additional synchronization overhead does not scale with the number of camera clusters. Besides, the variance remains very small (about 1 ms) as the number of cameras increases.

Our experiments also show that the parallelization of the image capturing and 3D reconstruction code (Section 2.2) speed up the computation significantly. Figure 6(b) shows the average reconstruction time for a frame with one participant in the scene. The reconstruction time is reduced to about a half with the parallel optimization.

[2]We are unfortunately not able to use PlanetLab [13] for experiments because our data rate would exceed the permitted quota. However, we emphasize that the scale of the system is not in the number of sites, but in the streaming density among a small number of sites.



(a) Latency on camera host PC

(b) Improvement by parallel optimization

(c) Latency for macro-frames

(d) Latency on display host PC

(e) Response time of view change request

(f) Load balancing

Figure 6. Experimental Results

3.2.3 Transmission Latency

We measure the one-way delay from UIUC to UC Berkeley (farthest). Without synchronized clocks, we use estimation by making the UC Berkeley gateway sending back an ACK message on the receipt of a data frame. Assuming that the two-way latencies are approximately the same, we can acquire the estimation of one-way latency by halving the time difference.

More formally, a macro-frame is defined as the union of N frames taken at the same instant of time by N cameras at a site [20]. The one-way delays for macro-frames are shown in Figure 6(c). The average value is about 105.1 ms, which guarantees interactivity.

3.2.4 Rendering Latency

Figure 6(d) shows the latency incurred at each display host PC for a single frame, which is mainly due to 3D data visualization (Section 2.4). Twelve pre-recorded streams are used for the experiment. We fix the rendering viewpoint for all testing runs to guarantee consistency. The averages and standard deviations shown in Figure 6(d) are taken over about three thousand frames.

We observe that the rendering time grows almost linearly with the number of 3D video streams. This is expected because we render the data as point clouds, and the number of pixels is proportional to the number of streams. Furthermore, we note that the standard deviation of rendering time also grows with the number of streams, which indicates the accumulative nature of the variances in the rendering latency. These results confirmed the challenges of multi-stream/multi-site 3DTI environments. As the system grows, the renderers have to be able to handle all those streams in real time.

3.3 ViewCast Evaluation

3.3.1 Response Time

We measure the response time of the ViewCast request, and profile a representative snapshot of 5-minute session time in Figure 6(e). The response time is defined as $T_2 - T_1$, where T_1 is the time when the renderer triggers a view request to the local gateway, and T_2 is the time when the renderer receives all the streams from the other sites. We find that among the four sites, UC Berkeley experiences the maximal latency which is about 600 ms (round trip). This is because the session controller is placed at UIUC for administrative convenience. The average response time taken over all sites including UC Berkeley is about 250 ms. The users report that they do not experience noticeable delay when changing the view. This reflects the advantage of the 3D data representation - the streams are progressively retrieved and rendered, hence the visual quality is gradually improved as the contributing streams arrive.

3.3.2 Load Balancing

Since every gateway node is heavily loaded with data dissemination, load balancing is important to utilize the resources efficiently. The topology constructed among multiple sites is an application-level overlay, where each gateway node is a source, a receiver, as well as a forwarder of multiple streams. Figure 6(f) shows how the forwarding workloads are distributed among different sites. UIUC and NCSA achieve an average forwarding load of about 110 Mbps, and UCF and UCB have almost 90 Mbps. The discrepancy of loads between two sites is approximately between 1% and 20%.

3.3.3 Impact of View Change

Most importantly, we concern the quality degradation incurred by the view change. The quality is measured using the metrics of *peak signal-to-noise ratio* (PSNR), which is calculated by comparing with the base case rendering of full content streaming as follows,

$$PSNR = 20\log_{10}\left(\sqrt{\frac{\sum_{i=1}^{h}\sum_{j=1}^{w}[I_a(i,j) - I_o(i,j)]^2}{h \times w}}\right) \quad (1)$$

where I_a and I_o denote the rendered 2D image using ViewCast and full content streaming respectively. The w and h denote the width and height of the rendered image.

We aggregate all the view changes made by the users at four sites for the Tai-Chi video, and measure how the visual quality varies with the changes. Figure 8 shows the results for four rendering sites with the view changes made by all users (represented by vertical lines). We note that the visual quality remains relatively stable between 30 and 40 dB for all sites. It is also interesting to note that when multiple users change views at the same time (where the vertical lines become thick in Figure 8), the visual quality would typically have a dip. However, the variance is usually within the bound of 5 dB.

4 Related Work

Existing tele-immersive systems can be roughly divided into two categories: (a) single-stream per site, and (b) multi-stream per site.

Virtual Auditorium [2], Digital Amphitheater [6], VIRTUE [11] and Coliseum [1] are systems that belong to the first catogery. They either capture video directly in 2D or convert 3D video streams to a synthesized 2D-view stream for transmission. There, it is natural to use the "all-to-all" dissemination scheme, that is, sending the stream from each site to all other sites, so that a comprehensive representation of the immersive scene can be constructed.

The UNC tele-immersive system [12] and TEEVE fall into the second category where multiple 3D video streams are produced and transmitted for a much richer context in the reconstructed 3D cyber-space. For multi-stream/multi-site systems, it would be impossible to send all streams from each site to all other sites, as the scale of the systems grows. Ott *et al.* proposed the Coordination Protocol as the transport-level solution for the multi-streaming problem between two tele-immersive sites. However, it does not consider the challenges for multi-site connection. Additionally, it may not be easy to deploy because it needs to modify the kernels of all end hosts, and even the first-hop router in each site.

5 Conclusion

In summary, we presented the design and implementation of TEEVE, a distributed 3D tele-immersive system. In particular, we presented a protocol that is implemented based on the ViewCast model for semantic-aware data dissemination among multiple sites. We evaluated the TEEVE system as a pipeline on various metrics such as latency, visual quality, view change resilience, and response time. Our results demonstrated that TEEVE enabled multi-site collaboration with stable visual quality, soft real-time latency, and efficient resource usage.

6 Acknowledgement

We would like to thank Sahan Gamage, Dongyan Xu, Gregorij Kurillo, Ruzena Bajcsy, Rahul Malik, Peter Bajcsy, and Hongliang Gao for helping us to set up the testbeds.

References

[1] H. Baker, N. Bhatti, D. Tanguay, I. Sobel, D. Gelb, M. Goss, W. Culbertson, and T. Malzbender. The coliseum immersive teleconferencing system. *ACM Trans. on Multimedia Computing, Communications, and Applications*, 2005.

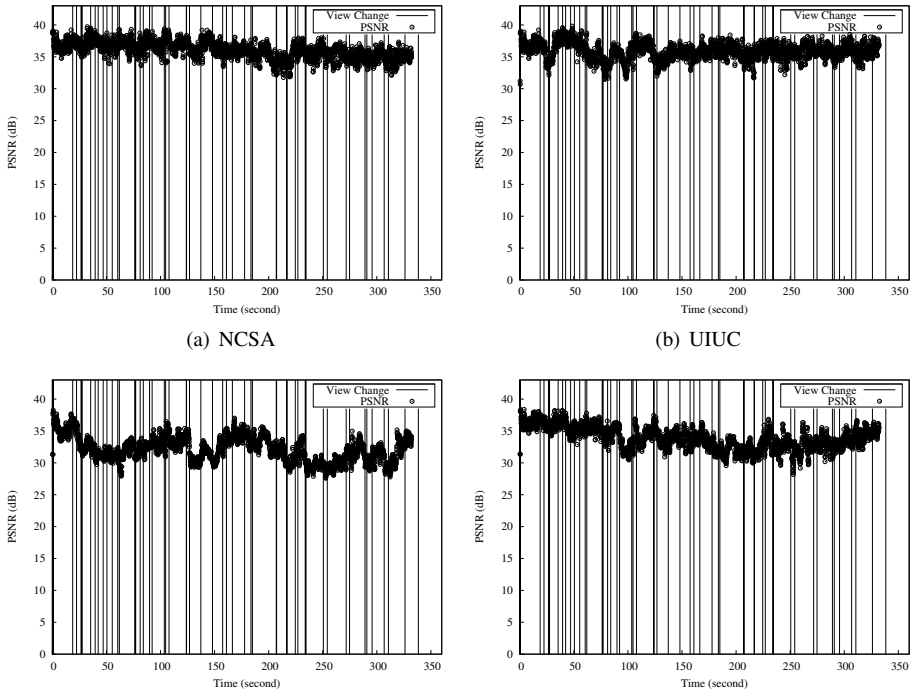
[2] M. Chen. Design of a virtual auditorium. In *Proceedings of the ninth ACM international conference on Multimedia*, 2001.

[3] R. Cheng and K. Nahrstedt. Empirical study of 3d video source coding for autostereoscopic displays. In *Proc. of ACM Multimedia*, pages 573–576, 2007.

[4] S. Deering. Multicast routing in internetworks and extended lans. In *Proc. of ACM SIGCOMM*, August 1988.

[5] Dragonfly2, Point Grey Research Inc. http://www.ptgrey.com/products/dragonfly2/index.asp.

[6] L. Gharai, C. Perkins, R. Riley, and A. Mankin. Large scale video conferencing: A digital amphitheater. In *Proc. 8th International Conference on Distributed Multimedia Systems*, San Francisco, CA, 2002.



(a) NCSA (b) UIUC

(c) UCF (d) UCB

Figure 8. Visual quality variance with view change

[7] M. Hosseini and N. D. Georganas. Design of a multi-sender 3d videoconferencing application over an end system multicast protocol. In *Proc. of ACM Multimedia*, 2003.

[8] Internet2. http://www.internet2.edu/about/.

[9] S. Jung and R. Bajcsy. A framework for constructing real-time immersive environments for training physical activities. In *Journal of Multimedia*, volume 1, pages 9–17, 2006.

[10] S. Jung and R. Bajcsy. Learning physical activities in immersive virtual environments. In *Proceedings of the 4th IEEE Int. Conference on Computer Vision Systems*, New York City, January 5-7th, 2006.

[11] P. Kauff and O. Schreer. An immersive 3d video-conferencing system using shared virtual team user environments. In *Proceedings of the 4th international Conference on Collaborative Virtual Environments (CVE '02)*, pages 105–112, New York, 2002.

[12] D. E. Ott and K. Mayer-Patel. An open architecture for transport-level protocol coordination in distributed multimedia applications. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMCCAP)*, 3, 2007.

[13] PlanetLab. http://www.planet-lab.org/.

[14] T. Svoboda, D. Martinec, and T. Pajdla. Multi-camera self-calibration a convenient multi-camera self-calibration for virtual environments. In *PRESENCE: Teleoperators and Virtual Environments*, volume 14, pages 407–422, 2005.

[15] M. Tamai, W. Wu, K. Nahrstedt, and K. Yasumoto. A view control interface for 3d tele-immersive environments. In *Proc. of IEEE International Conference on Multimedia & Expo (ICME)*, Hannover, Germany, 2008.

[16] W. Wu, Z. Yang, I. Gupta, and K. Nahrstedt. Towards multi-site collaboration in 3d tele-immersive environments. In *Proc. of the 28th International Conference on Distributed Computing Systems (ICDCS)*, Beijing, China, 2008.

[17] W. Wu, Z. Yang, and K. Nahrstedt. A study of visual context representation and control for remote sport learning tasks. In *Proc. of World Conference on Educational Multimedia, Hypermedia & Telecommunications (ED-MEDIA)*, Vienna, Austria, 2008.

[18] Z. Yang, K. Nahrstedt, Y. Cui, B. Yu, J. Liang, S. hack Jung, and R. Bajcsy. TEEVE: The next generation architecture for tele-immersive environments. In *Proc. of the 7th IEEE International Symposium on Multimedia (ISM)*, Irvine, CA, 2005.

[19] Z. Yang, W. Wu, K. Nahrstedt, G. Kurillo, and R. Bajcsy. Viewcast: View dissemination and management for multi-party 3d tele-immersive environments. In *Proc. of of 15th ACM International Conference on Multimedia*, 2007.

[20] Z. Yang, B. Yu, K. Nahrstedt, and R. Bajcsy. A multi-stream adaptation framework for bandwidth management in 3d tele-immersion. In *Proc. of the 16th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV'06)*, Newport, Rhode Island, May 22-23 2006.

[21] Z. Yang, B. Yu, W. Wu, R. Diankov, and R. Bajcsy. A study of collaborative dancing in tele-immersive environment. In *Proc. of the 8th IEEE International Symposium on Multimedia (ISM)*, San Diego, CA, 2006.