Transport and Rendering Challenges of Multi-Stream, 3D Tele-Immersion Data

Herman Towles, Sang-Uok Kum, Travis Sparks, Sudipta Sinha, Scott Larsen and Nathan Beddes

Department of Computer Science
University of North Carolina at Chapel Hill

Abstract

For over four years, our colleagues at the University of Pennsylvania[1] and we have been developing an experimental 3D tele-immersion testbed with the goal of providing high-fidelity scene reconstruction coupled with life-size, view-dependent stereo display. The UPenn team has focused on stereo reconstruction algorithms, while UNC has worked on system architecture, network transport, and stereo rendering and display issues. This paper provides an overview of our testbed architecture and details the transport and rendering challenges and solutions we have explored.

1. Introduction

In 1999, we began our tele-immersion research focused on recreating the most compelling 3D visual experience possible. Removing the requirement for live scene reconstruction, our initial goal was to create a 3D portal into a distant office [2]. We first captured range and image data of a real office scene with mannequin using a scanning laser rangefinder, which we processed into a single static geometric model with high-fidelity textures. Using a life-size, head-tracked stereo projective display, we demonstrated an accurate, very compelling continuum between the local and remote offices as shown in Figure 1a.

Building on this display architecture, we began working in early 2000 with our colleagues at UPenn and Advanced Networks to develop a tele-immersion testbed that incorporated real-time acquisition and ran over Abilene between Philadelphia, Armonk, NY, and Chapel Hill [11]. Using a trinocular stereo algorithm [7, 8], the team developed a *3D camera* built around three 1394-digital color cameras and a quad-processor workstation running highly tuned, multi-threaded stereo reconstruction code. This first 3D camera output 320×240 resolution depth maps with color textures at frame rates of 1-3 Hz. To achieve these performance levels, background segmentation is used to reduce the number of active pixels to an average of approximately 20,000 per frame, and the disparity search range is limited to an effective reconstruction volume of approximately one cubic meter. This is enough volume to capture a single person in our 'across the desk' one-on-one teleconferencing scenario. At the display site, this reconstructed person (devoid of their actual local background) is 3D composited into one of several high-fidelity background scenes acquired off-line using the laser scanning system previously referenced. Figure 1b shows dual 3D tele-immersion sessions at the UNC display site.

Figure 1: (a) Portal to a static office. (b) Demonstration at UNC of dual 3D tele-immersion sessions from Armonk, NY and Philadelphia.

With a single 3D camera and stereo display system, this capture-display model is not unlike the 1-camera to 1-display paradigm of traditional conferencing systems. However, the major difference is that this data stream contains view-independent data (3D structure) and not just a 2D image from a fixed viewpoint. This 3D scene definition allows the remote display system to generate new novel views based on the local viewer's eye position with respect to the 3D display portal.

The scene description from a single 3D camera is generally inadequate unless the viewer's position matches that of the virtual camera position. Therefore, our initial testbed implementation was built with an array of five 3D cameras arranged at or near (seated) eye-height in a 120 degree arc. Today, we are working with our UPenn colleagues to expand this to fifteen 3D cameras operating at 640×480 resolution that are positioned to capture not only our collaborators, but the complete surrounding environment.

In practice, our tele-immersion testbed has an N-camera to 1-display architecture (Figure 2), where N cameras generate 3D image descriptions transported via N network flows to a single rendering-display node. This model has the advantage that it is straight forward to replicate or scale the

[1] Kostas Daniilidis, Jane Mulligan (now with University of Colorado, Boulder), and Nikhil Kelshikar

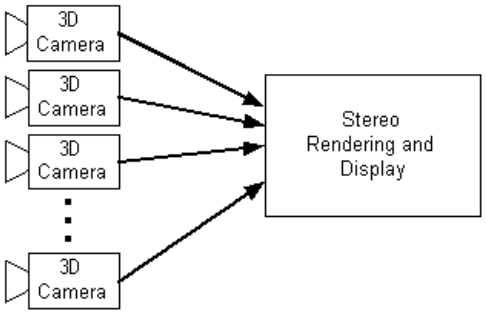

Figure 2: Tele-immersion framework with multiple view-independent streams mapped to the rendering-display node.

number of independently-operating, reconstruction cameras. However, this advantage is illusionary since scaling the performance of the single rendering node is a more significant challenge. The remainder of this paper will detail the transport and rendering challenges associated with increasing the number of 3D cameras in our testbed, and the solutions we have explored.

2. Transport Challenges

In our research, we have explored two variations on multi-stream, end-to-end data flow. The first is the simplest, and involves the model previously described with depth maps and RGB images flowing over independent TCP/IP streams from N acquisition/reconstruction nodes to a single rendering/display node. The three stages in this pipeline are shown in Figure 3a. The second case involves two transport stages as shown in Figure 3b. In this operational mode, image acquisition and 3D-scene reconstruction are separate stages with N distinct 2D-image streams connecting the two. This second operational mode was created when we modified our testbed to apply the 3000-processor Terascale Computing System at the Pittsburgh Supercomputing Center to the compute-intensive task of high-resolution, full-scene, stereo reconstruction [5].

In both 3D tele-immersion cases, the demand for bandwidth is considerable. Exact requirements for the system depend upon several factors: image resolution, encoding, frame rate, and the number of 3D cameras used. Table 1 describes the bandwidth for transporting raw video frames

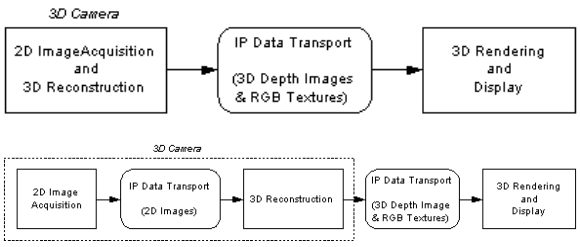

Figure 3: (a) 1-transport stage pipeline. (b) 2-transport stage pipeline where 3D camera functionality is split between two sites.

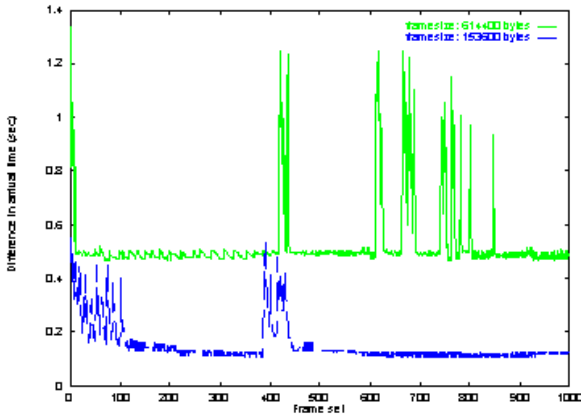

Figure 4: Difference in frame arrival times across two different size flows.

and 3D depth images given several configurations. Raw 2D images are always encoded at 8-bits per pixel for black and white cameras or color cameras (Bayer pattern color), and 3D reconstructed depth images are encoded at 40-bits per pixel (16-bit of 1/z range and 24-bits of color). In our initial testbed configuration operating with 5 streams at 320×240 resolution at 2 fps requires 31 Mbps of 3D image bandwidth. In terascale testing, we ran 9 full-resolution streams at 8 fps generating up to 880 Mbps of TCP/IP traffic to the rendering node.

No. Cameras & Resolution	2D Bandwidth Mbps	3D Bandwidth Mbps
5@320x240	18.4	30.8
5@640x480	73.8	122.8
9@640x480	132.8	221.4

Table 1: Uncompressed Bandwidth Requirements at 2 fps.

In general, the bandwidth requirements can easily exceed the link capacity of Abilene implying that data transport is a critical issue to this application. This means that compression of data over the link is key to decreasing overall throughput. In addition, our goal is to support the reliable transport of this data with as little latency as possible. Furthermore, we would like *frame* arrivals of the multiple streams to be highly synchronized since the rendering update cannot occur without a complete data set. Figure 4 shows that despite mechanisms insuring synchronized send initiation, frame arrivals are highly unsynchronized. The problem is most severe for the larger data frame sizes.

A closer analysis of frame synchronization reveals that lack of transport-level protocol coordination is the cause. Application flows use TCP because it offers in-order, reliable delivery of data. Because each data flow operates independently, bandwidth probe and backoff mechanisms lead to competition among flows. Figure 5 illustrates this problem. While flows receive roughly the same average band-

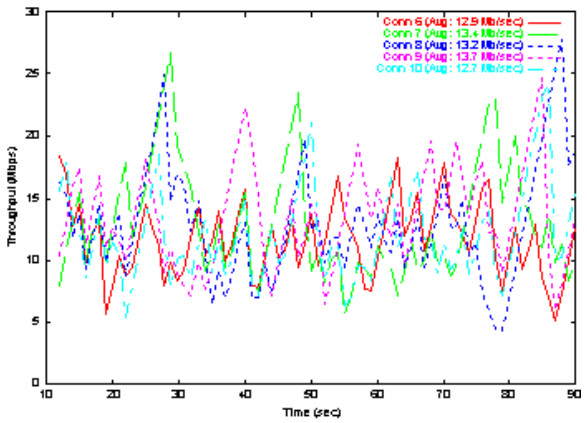

Figure 5: TCP flows competing for bandwidth.

width over a large time interval, small time intervals show a stark variation in throughput levels leading to vastly unsynchronized frame arrival times. Mechanisms for coordinating transport-level protocols are needed to distribute available bandwidth evenly across flows of the same application while still maintaining congestion responsiveness. This will improve the overall level of synchronization in frame transport.

The following sections will discuss our transport research related to compression of multi-stream data and the development of a new network coordination protocol.

2.1 Data Compression

Increasing the number of independently operating 3D cameras obviously increases the demand for both network bandwidth and rendering system performance. One way to alleviate this increased demand is to exploit spatial and temporal coherence in the multiple depth streams.

Using *temporal* coherence to compress each depth stream will reduce the required network bandwidth. However, it does not reduce the rendering load since the amount of data after decode is the same. *Spatial* coherence between depth streams can be used to reduce both the network bandwidth and rendering load requirements. It is this dual benefit that led us to initially focus on spatial compression methods for reducing the redundancy in data from multiple 3D cameras with overlapping views.

Driven by the need for real-time performance, scalability with the number of 3D cameras, good data reduction, and tunable network bandwidth (which is necessary in order to limit the bandwidth used as the total number of depth stream increases), a new *group-based*, real-time compression algorithm has been developed [6]. The key to compression performance and limiting inter-stream (camera) networking requirements in this algorithm is to compare each stream with only a single *reference stream* to create a new lower-bandwidth *differential stream* for transmission.

The algorithm works as follows. All depth streams are statically partitioned into disjoint groups, and a reference stream called the *center stream* is designated for each group.

The master reference stream for all differential encoding, designated the *main stream*, is the 3D camera stream which most closely matches the user's viewpoint. All points of the main stream are transmitted to the rendering process. The group that contains the main stream is designated the *main group*, and the center stream of all other groups is differentially compared with the center stream of the main group; while the center stream of the main group is compared with the *main stream*. All remaining streams of each group are differentially encoded with their respective group's center stream. You are referred to [6] for complete details.

Group partitioning is defined in a pre-process step and remains static. In practice, the main stream designation can change dynamically as the viewer's position moves, possibly even changing which group is designated the main group.

Testing to date has involved synthetic 3D camera views of a real-world scene - the office environment of our early tele-immersion experiments [2]. In these tests, a 5 to 1 compression for 22 depth streams was achieved. Since much of any scene is static background, huge bandwidth reductions should be possible by compressing the reference streams and the differential streams using temporal coherence. This could significantly reduce the network bandwidth, thus enabling an increase in the number of camera groups which leads to even better spatial compression.

2.2 Coordination Protocol and Reliable-UDP

As mentioned previously, the synchronization of frame arrivals is important to improve processing efficiency and thus providing an increase in overall frame rate. Synchronization at the sender-side does not achieve this goal because of network competition among our several TCP/IP flows. Our solution to this problem is a TCP-friendly UDP protocol that aggregately acts like N TCP flows, but provides bandwidth coordination and thus synchronization among N UDP flows.

To provide this synchronization and coordination, we developed a Reliable-UDP (RUDP) protocol that utilizes the Coordination Protocol (CP) described in [9]. The resulting CP-RUDP protocol gives individual bandwidth allocations to each of our N streams based on their individual progress in sending each frame. The aggregate bandwidth for all N streams does not exceed that of N TCP streams maintaining TCP-friendliness over Abilene.

The CP part of CP-RUDP provides congestion control across the link using the Aggregation Points (AP) described in [9]. RUDP determines the rate for each flow by locally calculating its proportion of the N-flows bandwidth using

link information provided by CP. This rate is calculated by the formula:

$$r_{send} = NR(b_L/b_A) \quad (1)$$

Where r_{send} is the send rate for an individual flow, N is the number of flows, R is the bandwidth available for an equivalent TCP flow, b_L is the number of bytes remaining in the frame being sent by this flow, and b_A is the number of bytes remaining for all frames being sent by all flows. Updated values for N, R, and b_A are all provided by the AP each time an RUDP acknowledgement (ACK) is received. Each CP-RUDP data packet sent by the sender contains the b_L value for this flow and is used by the AP to calculate the b_A value.

The result is that flows that get ahead are given less bandwidth and flows that get behind are given more bandwidth. In the end, the bandwidth is targeted such that each frame arrives at approximately the same time. Moreover, the congestion control provided by CP prevents the send rates from exceeding the network capacity at any given time. CP-RUDP provides the in-order streaming interface, with congestion-control, that TCP does, but adds the network flow synchronization that TCP cannot provide.

3. Rendering Challenges

The task of our tele-immersion rendering system is to create an interactive, view-dependent, stereo presentation of the remote scene description. Our design goals are simple:

- **High-performance, Interactive Architecture.** Tracking and rendering performance are paramount in creating a convincing sense of interactive presence.
- **Quality Surface Representation.** The reconstructed scene must be high-fidelity.
- **Scalable Performance.** Ideally the rendering system should scale with the acquisition/reconstruction system.

The following sections discusses many of the challenges and issues associated with achieving these goals.

3.1 High-Performance Architecture

To create a view-dependent presentation that is pleasing to the user, the rendering system must generate new frames at a minimum 20-30 Hz rate using the latest estimate of the local user's eye positions. Since the remote scene descriptions are updated independently at a slower rate (less than 10Hz today), our renderer is designed to buffer the scenes asynchronously and present the user with the latest complete scene available in the buffer upon receiving a redraw

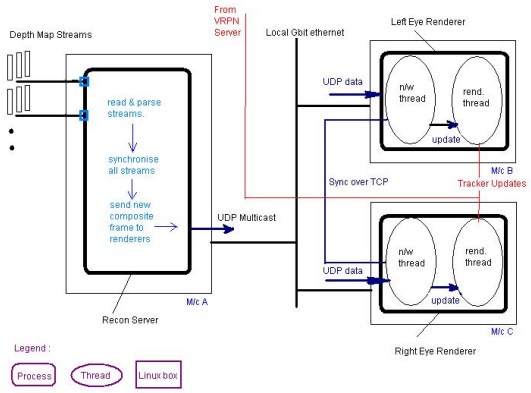

Figure 6: Tele-Immersion 3-PC Rendering Architecture.

request. This is accomplished with a multi-threaded implementation, which in its simplest form has one thread asynchronously receiving incoming 3D scene descriptions while a second thread is re-rendering the last complete scene using the latest estimate of the viewer's eye positions.

Rendering Cluster - Our stereo renderer is implemented using OpenGL on high-performance Linux PCs with NVIDIA graphics hardware. We have developed two configurations - a 1-PC rendering system and a 3-PC cluster architecture. The 1-PC system uses a dual-channel graphics card to drive the stereo projector pair in parallel. In the 3-PC architecture, two of the machines act as rendering machines, each implementing a left or right-eye renderer connected to one projector. With the approximately 2x performance advantage offered by the cluster configuration, the 3-PC architecture is our preferred renderer. Figure 6 shows the basic block diagram of this system.

In this configuration, the third machine serves as the rendering aggregation point (RAP) for the data broadcast from the remote reconstruction node. After frame re-synchronization and some data pre-processing this RAP node UDP multicasts the incoming scene description to the two rendering machines over a LAN.

Frame Re-Synchronization - On the acquisition end, the exposure of all cameras is guaranteed to be synchronous through the use of an external trigger signal at each camera. This is not only necessary for accurate stereo correlation between the multiple images of each 3D camera when there is motion in the scene, but is also important at the rendering node to avoid simultaneously presenting 3D depth data from different camera views that represent moving surfaces from different points in time. Therefore, the first processing step of the receive thread of the RAP node is to re-synchronize the data from all 3D camera streams based on a timestamp included with each data frame. This seems trivial, but in practice we have observed TCP streams falling one or more frames behind other streams even though the exposure of all cameras is synchronized.

We currently solve this by using a 5-frame ring buffer, but in the future a conditional transport-level protocol such as discussed in Section 2.2 is expected to better balance multi-stream throughput, which will reduce the buffering requirement and further decrease end-to-end latency.

Maximizing Refresh and Update Rates - The amount of data to render is formidable, but an equally daunting task is feeding the graphics pipeline.

Our tele-immersion experiments in 2000 using 5 camera streams at 320×240 resolution with foreground-background segmentation generated approximately 100,000 surface points per frame (or 20K per camera) to render. By 2002, our terascale computing experiments [5] were generating on average 230,000 surface points per camera per frame or as many as 2.1M surface points from a 9 camera array.

Our earliest implementation built a new OpenGL display list in host memory for each new incoming 3D frame. This process is inherently so slow that we amortized the build cost over as many as 6 frames to avoid a disturbing *stutter* or break in presence for the head-tracked viewer. A much better solution to this frame update issue, which also affords us a rendering performance increase, is to use Vertex Arrays and load data directly into AGP or graphics card memory.

No. Streams & Resolution	No. Surface Pts x1000	Refresh Rate, Hz	Update Rate, Hz
3@320x240	55	380	8.9
5@320x240	100	200	6.3
1@640x480	232	110	5.3
3@640x480	720	42	2.7
5@640x480	1,160	24	1.8
9@640x480	2,100	16	1.0

Table 2: 3-PC Rendering Performance using Vertex Arrays.

Table 2 summarizes our current performance levels with the multi-threaded architecture of Figure 6 with 2.4 GHz PCs and GeForce4 graphics. We are able to maintain a very interactive refresh rate of 24 fps for over 1.1M surface points from 5 full-resolution streams. Much less satisfying is the 1.8 fps limitation on scene update rate, which is well below that maximum 8 fps reconstruction rate demonstrated using the Pittsburgh Supercomputing Center's Terascale Computing System.

We currently are discussing a re-architecture of the RAP node to include a receive and multicast re-transmit thread per incoming stream, moving all stream parsing and data conversion into the rendering PC to be processed on the dual-processor host or directly by a Cg vertex program.

3.2 Quality Surface Representation

For each frame, the renderer receives multiple depth maps - one from each 3D camera. For each valid pixel in these images, there is an encoded range value and RGB color. Using the camera-specific projection matrix, each range value is converted into a surface point definition in a common Euclidian world coordinate system.

We initially explored techniques for triangulating these depth maps in real-time, but performance issues and disturbing visual artifacts such as *skinning* between disjoint surfaces or *pyramidal spiking* caused by temporally noisy data precluded using this solution. Instead, we have focused on rendering the multiple depth maps as a massive 3D point cloud. Each point from each depth map is rendered with its respective color as an opaque GL_POINT of size 2×2. While very fast, there is need for improved visual fidelity as all point splats are *screen-oriented* and drawn at a constant size regardless of depth in the scene. This latter characteristic can also result in a stippled transparency effect if the viewing geometry is such that adjacent surface points are rendered more than 4 pixels apart in screen coordinates.

The surface splatting work of [14, 10] offers some interesting alternatives that we have explored. The GL_POINT_SPRITE_NV extension renders splats that are automatically sized based on scene depth, which helps alleviate the stippling problem mentioned above. Point sprites always draw a quad in screen space, but by modulating the shape of the sprite's texture it is possible to create a sense of object-space orientation. Rather than computing this shape texture in real-time, a pre-computed set of textures representing different splat orientations can be de-referenced using the surface normal of the point.

This technique embodied in a Cg vertex program is being actively researched by our team.

3.3 Scalable Performance

Significant rendering performance gains over the last two years have come from simple hardware upgrades as we have replaced 933 MHz PCs and GeForce2 QuadroPro graphics with 2.4 GHz dual-processor machines and GeForce4 ti4600 graphics. At the same time, our rendering network adapters have been upgraded from 100 Mbps to GigE. And of course, the 3-PC architecture offers approximately a 2x performance advantage over the 1-PC renderer.

As the number of 3D cameras have scaled, we have discussed options for scaling the rendering hardware beyond two rendering PCs. Screen-space partitioning would certainly gain us primitive rendering performance, but would not scale linearly and does nothing to solve the equivalent performance gains needed for the RAP node or the local network used for UDP multicast. A more promising approach may be a data partitioning architecture that simply

divides the incoming data into equal parts to be rendered on separate PCs, and then uses a depth compositing solution such as Lightning-2 [12] or Hewlett-Packard's Sepia project [1].

In either case, scaling rendering software and hardware to keep pace with the simple replication of 3D cameras is a most difficult task. The best system solution will undoubtedly combine rendering performance gains with a reduction in data either through new compression techniques as outlined in Section 2.1 or more advanced algorithms that will compress the full-scene 3D description to more optimally match that defined by the rendered view position.

4. Conclusions and Future Work

The technical challenges in developing an effective 3D tele-immersion system are many, yet we and others worldwide [3, 4] continue to make good progress in identifying and resolving new issues. We are mutually buoyed by our collective success and the dream that one day we will create a truly convincing sense of tele-presence.

As our own tele-immersion research continues, we are actively exploring several new areas that show promise for improving both our rendering quality and rendering system scalability. On the quality front, we are investigating the use of object-oriented, surface splats with transparency modulation based on a new reconstruction algorithm (and 3D stream definition) that now provides a surface normal and confidence estimate per 3D surface point. On the scalability/performance side, we continue to research new methods of spatial and temporal, multi-stream data compression that are guided by view-dependency feedback. In addition, we are preparing to test the effectiveness of our new network coordination protocol (CP-RUDP) in balancing the needs for maximum bandwidth utilization against the network usage needs of competing TCP network streams.

Acknowledgments

The authors wish to thank Jaron Lanier, Amela Sadagic, Ketan Mayer-Patel, David Ott and Henry Fuchs for their leadership and technical contributions. This research has been supported by Advanced Network and Services (National Tele-Immersion Initiative) of Armonk, NY, the National Science Foundation (ITR/SI: Real-Time Long-Distance Terascale Computation for Full Bandwidth Tele-Immersion, IIS-0121293) and the Defense Advanced Research Projects Agency (3D Tele-Immersion over NGI).

References

[1] Byron Alcorn. "sv7: Blazing Visualization on Commodity Clusters", *Hot 3D Presentations, Graphics Hardware 2003*, July 2003.

[2] Wei-Chao Chen, H. Towles, L. Nyland, G. Welch and H. Fuchs. "Toward a Compelling Sensation of Telepresence: Demonstating a portal to a distant (static) office", *IEEE Visualization 2000*, October 2000.

[3] Eddie Cooke, Ingo Feldmann, Peter Kauff, Oliver Scheer. "A Modular Approach to Virtual View Creation for a Scalable Immersive Teleconferencing Configuration", *Proceedings of International Conference on Image Processing (ICIP 2003)*, September 2003.

[4] Markus Gross, et al. "blue-c: A Spatially Immersive Display and 3D Portal for Telepresence", *Siggraph 2003 Conference Proceedings*, July 2003.

[5] Nikhil Kelshikar, et al. "Real-time Terascale Implementation of Tele-Immersion", *International Conference on Computation Science*, June 2003.

[6] Sang-Uok Kum, Ketan Mayer-Patel and Henry Fuchs. "Real-Time Compression of Dynamic 3D Environments", *ACM Multimedia 2003*, November 2003.

[7] Jane Mulligan and Kostas Daniilidis. "Trinocular Stereo for Non-Parallel Configurations", *Proccedings of ICPR2000*, September 2000.

[8] Jane Mulligan and Kostas Daniilidis. "Trinocular Stereo: A New Algorithm and its Evaluation", *International Journal on Computer Vision*, pages 3-10, 1998.

[9] David Ott and Ketan Mayer-Patel. "A Mechanism for TCP-Friendly Transport-level Protocol Coordination", *Proceedings of USENIX 2002*, June 2002.

[10] Liu Ren, Hanspetter Pfister and Matthias Zwicker. "Object Space EWA Surface Splatting: A Hardware Accelerated Approach to High Quality Point Rendering", *Eurographics 2002*, September 2002.

[11] Amela Sadagic, H. Towles, L. Holden, K. Daniilidis and B. Zeleznik. "Tele-immersion Portal: Towards an Ultimate Synthesis of Computer Graphics and Computer Vision Systems", *4th International Workshop on Presence*, May 2001.

[12] Gordon Stroll, Pat Hanrahan et al. "Lighning-2: A High-Performance Display Subsystem for PC Clusters", *Siggraph 2001 Conference Proceedings*, August 2001.

[13] Herman Towles, W. Chen, R. Yang, S. Kum, H. Fuchs et al. "3D Tele-Immersion Over Internet2", *International Workshop on Immersive Telepresence (ITP2002)*, December 2002.

[14] Matthias Zwicker, Hanspetter Pfister, Jeroen van Baar, and Markus Gross. "Surface Splatting", *Siggraph 2001 Conference Proceedings*, August 2001.