End-user viewpoint control of live video from a medical camera array

Jeffrey R. Blum, Haijian Sun, Adriana Olmos, and Jeremy R. Cooperstock
Centre for Intelligent Machines, McGill University, Montreal, Quebec, Canada
Email: {jeffbl | hsun | aolmos | jer}@cim.mcgill.ca

***Abstract*—The design and implementation of a camera array for real-time streaming of medical video across a high speed research network is described. Live video output from the array, composed of 17 Gigabit Ethernet cameras, must be delivered in low-latency, simultaneously, to many students at geographically disparate locations. The students require dynamic control over their individual viewpoints not only from physical camera positions, but potentially, of a real-time interpolated view. The technology used to implement the system, the rationale for its selection, scalability issues, and potential future improvements, such as recording and offline playback, are discussed.**

I. Introduction

The camera array described in this article is deployed at McGill's Medical Simulation Centre as one of the tools in a geographically distributed medical teaching application, in which remote students participate in live cadaveric dissections and other surgical procedures. In these scenarios, the main requirements we had to support were live delivery of audio and video, coping with possible occlusion of one or more cameras by the instructor's head or hands, and a dynamically varying "best vantage point" throughout the procedure. Our system was designed as part of the Health Services Virtual Organization[1] (HSVO) project to generate, capture and distribute all of the physical camera data, with accompanying audio, and offer the possibility of real-time interpolation of virtual viewpoints. Access to these capabilities was required simultaneously, at low latency, by dozens of geographically distributed users, through an effective user interface as suited to the medical training context. To meet these goals within reasonable budgetary constraints, we prototyped and tested multiple potential solutions before settling on our final architecture. The entire solution for capturing, storing and streaming the content is referred to collectively as the *camera array server*, which streams video to multiple *client players* running remotely.

The remainder of this article is organized as follows. Section II summarizes related work, explaining the limitations of existing camera arrays for our purposes. Section III describes our architecture in further detail. Section IV presents the technical issues affecting the user's experience, including discussion of rejected designs. Section V concludes the paper.

II. Related Work

Traditionally, most video streaming has been limited to a single, non-interactive perspective seen by all users, where discrete viewpoints are decided by the video producer. Recognizing these limitations, significant research effort has been applied to improve the user experience, both by providing users some control over their individual views, and of smooth transitions between viewpoints. Efforts to produce higher quality interpolated viewpoints have included the creation of a 3D scene model using cameras positioned around a sporting event [1], [2]. Systems emerging from this work, such as "Eye Vision", have been deployed commercially for sporting events to produce flythroughs of a scene by showing frames from cameras at intermediate locations between the initial and final camera positions. Later work used physical camera views and also interpolated additional viewpoints between physical cameras. Other systems not only interpolated new viewpoints, but allowed the end user viewpoint control when playing back recorded scenes stored on the client computer [3].

The Real-Time Interactive Multi-View Video System [4] used a set of 32 cameras arranged closely together in an arc to provide remote users individual viewpoint control. Although it did not interpolate viewpoints between camera positions, it offered features such as perspective control at a specific "frozen moment" in time, and a semi-continuous "view sweep." This solution, using a specialized client playback engine, was capable of rendering 640×480 resolution video at 30 frames-per-second. More generally, the problem of transmitting multi-perspective video or 3D scenes and allowing users viewpoint control for exploration of the 3D world, has motivated efforts such as Free-Viewpoint Television [5]. This effort has included an extension to the H.264 video standard for combining multiple camera perspectives into a single encoded stream, which can then be used on the client side to render new viewpoints [6].

Last, camera arrays are often constructed for specific needs, such as for high-speed videography [7]. To reduce data traffic and storage, their system employed video compression, which may be integrated into the cameras, or directly coupled to them. Such compression may have implications for video quality, especially important if post-processing is necessary. Many camera arrays tend toward a grid layout of closely spaced cameras, which is often desirable for the rendering of synthetic viewpoints or for enhancing imaging performance [8], [9]. This is especially the case when light field rendering techniques are employed [10]. Other camera array architectures have attempted to provide a generic solution [11], albeit at potentially prohibitive costs, e.g., requiring all machines to

[1] http://www.hsvo.ca

Component	Qty
Basler scA1000-30gc GbE camera	17
Pentax 8.5mm c-mount lens	16
Varifocal Tamron 4-12mm lens for overview	1
Baumer TZG01 ToF camera	1
Recording/encoding servers (Intel Core i7 920)	4
Hauppauge WinTV-HVR 1250 analog capture card	1
Interpolation server with NVidia 265 GPU	1
SMC 8648T 48 port GbE managed TigerSwitch	1
Adobe Flash Media Streaming Server (FMSS)	1
Shure SLX14/85 wireless microphones	2
Shure receiver, mixer, stand microphone	1
Kino Flo Diva-Lite 401 Kit (2 lights)	1

TABLE I
EQUIPMENT LIST

connect through a 10 GbE network.

The systems described above sucessfully cover some, but not all, of the requirements identified in Section I. In particular, since this project was not solely a research demonstration system, but intended for deployment into an active medical teaching environment for ongoing use, we wanted to create a system that used as many off-the-shelf components as possible. This decision not only reduced development and test time, but also simplified installation and ongoing maintenance. As one example, the desire to have end users connect to the system simply by entering a website address in their browser guided us toward existing video streaming solutions, rather than using custom compression or streaming protocols, despite the potential benefits to latency, bandwidth, and video quality they might have provided. Systems described above that used lossy compression or low resolution when capturing camera data did not meet our requirement to capture and record uncompressed high quality video suitable for a medical environment. Likewise, those that required expensive components to achieve a more general system architecture would make it difficult to maintain a low enough system cost for practical deployment. This motivated us to design and implement a new camera array system architecture, whose physical configuration and logical structure we discuss next.

III. ARCHITECTURE

As illustrated in Figure 1, the architecture supports the necessary services of the camera array, which consist of encoding the raw video data into a format suitable for delivery to clients, image-based rendering to interpolate new views where no camera exists, streaming of the video (and audio) to clients, and although not yet fully implemented, recording of the video data for later on-demand viewing. The hardware to support these needs is listed in Table I.

In addition to the 17 viewpoints from the physical cameras, the system also interpolates virtual viewpoints between the physical camera positions. Since the cameras are already placed relatively close together, these interpolated viewpoints are likely to be most useful in helping users maintain context while moving their viewpoint around the procedure.

A. Cameras and mounting

Although Firewire cameras are commonly used for machine vision applications, we opted instead for Gigabit Ethernet cameras. This offered the benefit of multicast support directly from the cameras, thereby allowing the video output to be received simultaneously by multiple machines, which can each dynamically select the streams they wish to receive. For our application, this means that raw video can be streamed directly both to the recording/encoding machines and to the interpolation machine(s), thus simplifying the architecture. Moreover, this avoids the need to include an intermediary for relaying the data to multiple destinations, thereby reducing video latency. As the system scales, increasing the number of cameras or adding more machines to process the video can be as simple as plugging the additional equipment into the Gigabit Ethernet switch and reconfiguring the software to take advantage of the new capabilities.

For video interpolation, precise inter-camera synchronization is crucial, since motion in the scene would otherwise induce matching errors, leading to artifacts in the synthesized view. We found that software synchronization of the Basler cameras via their Ethernet interface was not sufficiently reliable, so we adopted a hardware triggering solution, connecting all 17 cameras to a custom timing circuit developed in our lab. This circuit can also trigger an optional time-of-flight (ToF) camera, which outputs a low-resolution depth map that can be used by the interpolation algorithm, but is not directly viewed by the end-user.

In anticipation of the need to capture, digitize, record and stream a camera feed from a medical device such as an endoscope, we also included an analog video capture card.

The array's physical structure consists of two concentric rings, each holding eight cameras, pointed downward and slightly inward. The rings are attached solidly to the ceiling to prevent movement and vibration. This arrangement, shown in Figure 2, provides useful perspectives of a medical procedure from a full range of angles, effectively allowing remote viewers to "walk around" the surgical environment. Screw mount clamps attach the cameras to the supporting structure rigidly, since any post-calibration movement can significantly degrade the quality of interpolation. One additional camera with a shorter focal length lens is positioned in the center of the array to provide a wide angle overview of the viewable scene that aids the user in maintaining context.

B. Servers

The camera array architecture is designed for live streaming of its output over a high-speed network, as well as future recording of the live sessions to enable "on-demand" viewing. Although the camera array is connected to the high speed CANARIE network, the available bandwidth is still insufficient to send uncompressed video while supporting more than a very limited number of users, so a compressed encoding and streaming solution was required. The steps required for receiving, recording and encoding the video data are shown

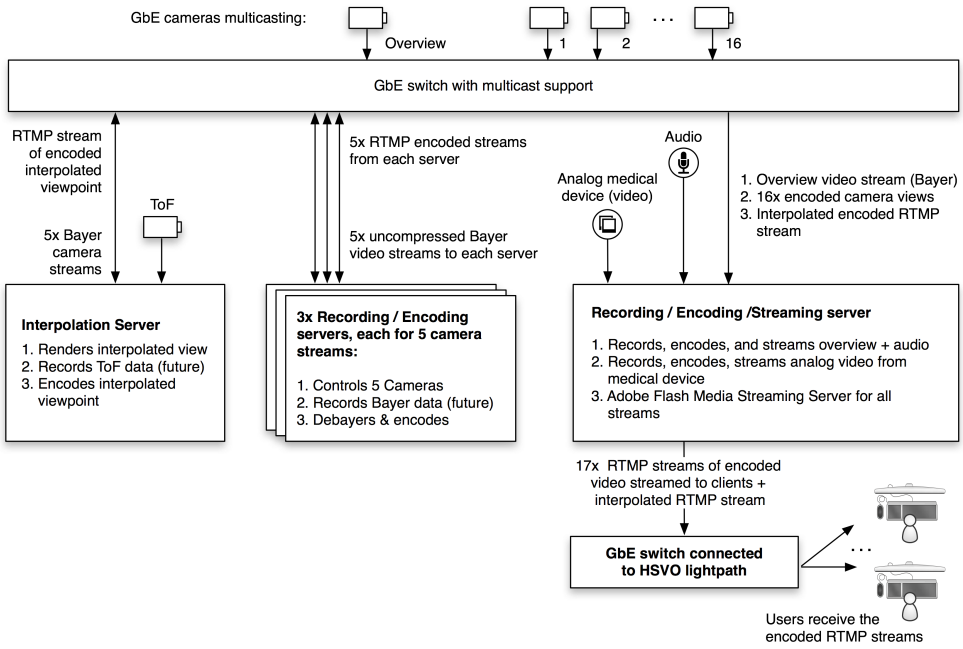

Fig. 1. Camera Array Architecture

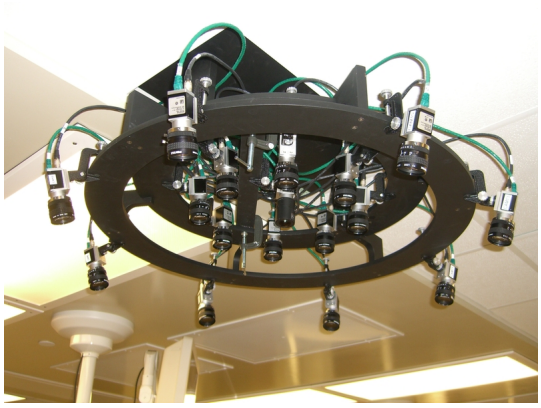

Fig. 2. Camera mounting ring, replacing a ceiling tile

in Figure 3 and described in further detail in the remainder of this section.

Our full 17 camera array requires five servers:

- Three machines each encode the output of five CCD cameras (15 total). Although this service is not yet integrated, these machines will also record the raw camera data.
- One machine encodes (and will also record) the 16th CCD camera plus either the wide-angle overview camera in the center of the ring or an optional analog camera feed from medical equipment via an analog capture card. This machine also captures and encodes the associated audio and streams the encoded video data from all cameras to clients via Adobe Flash Media Streaming Server (FMSS). The overview/medical equipment stream is recorded in compressed VP6 format after encoding, along with synchronized audio, providing a single video file for later viewing.
- One machine is responsible for interpolating a synthetic stream at a desired viewpoint and encoding the result. In addition, the ToF camera, if included in the array, would be captured and recorded directly by this machine.

C. Encoding and Streaming

The encoder receives raw Bayer video frames from the cameras, demosaics the data, and compresses the video for streaming to clients. Bayer is the preferred format since it requires half the network bandwidth of already demosaiced formats such as YUV 4:2:2, and three times less than RGB. Although we investigated and tested several different encoding and streaming solutions, we rapidly narrowed our choices down to Adobe Flash and its open-source competitor, Ogg Theora. After testing both, we rejected Theora due to high startup delay, as well as video quality factors and lack of cross-platform support. Choosing Flash allows us to benefit from its ability to run on multiple platforms and web browsers, as well as its ActionScript development environment, which we used to create a rich client application that receives, decodes, and manipulates the streamed video. Flash Player uses either VP6 or H.264 formatted video.

Although H.264 offers superior video quality, its computational requirements for encoding were much higher than VP6. That said, the more significant factor influencing our choice between VP6 and H.264 was latency. Although some cameras encode H.264 video directly on the camera itself, presumably with very low latency, we require the original uncompressed camera data for interpolation. Moreover, only for H.264 video, the Adobe Flash Player buffers a minimum of 64 frames

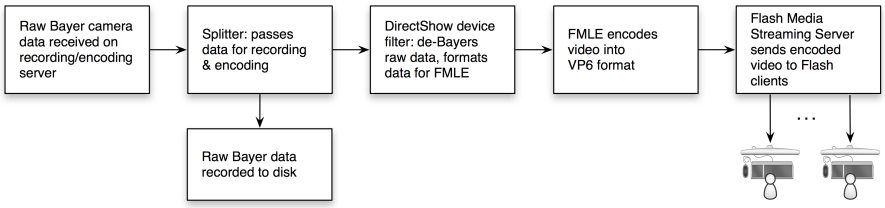

Fig. 3. Steps for recording, encoding and streaming video from Ethernet cameras.

	VP6	**H.264**
Latency (zero client buffer)	112 ± 25 ms	1152 ± 25 ms
Latency (default client buffer)	232 ± 25 ms	3432 ± 25 ms
CPU (%)	13%	16-25%

TABLE II
ADOBE FMLE VP6 VS. H.264 ENCODING LATENCY AND TOTAL CPU LOAD. CONFIGURATION: INTEL CORE I7 920 CPU, WINDOWS XP 32-BIT OPERATING SYSTEM; DALSA GENIE C1024 ETHERNET CAMERA AT 1024×768 RESOLUTION, 20 FPS, ENCODED AT 1 MBIT/S.

unless all buffering is turned off.[2] This limitation imposes a choice between (possibly) jittery video with no client buffer, or approximately two seconds of latency at 30 fps, with higher latency at lower frame rates, neither of which is desirable for our system. We measured both CPU load and latency in a head-to-head performance comparison, shown in Table II, with encoding implemented in Adobe Flash Media Live Encoder (FMLE). To determine the latency, we displayed both the original and the encoded video, streamed through a Wowza streaming server, on the same monitor, and used a high-speed Redlake MotionMeter camera to measure the time difference between an electronic flash in the two views.[3] CPU percentage is approximate since it was estimated manually from the Windows Task Manager process list. Although H.264 video quality is generally considered superior to VP6 at any given bitrate, we are operating on a high-capacity research network that can easily support an increase of bitrate to compensate, and thus, VP6 is a significantly better choice, at least when using FMLE as the encoder. If streaming over a more modest commodity Internet connection, this decision would need to be reevaluated. However, even at the lower processor demands of VP6, the encoding task for 17 or more cameras must be distributed across multiple encoder machines.

FMLE requires that video data be provided via Microsoft DirectShow devices. Although Basler provides a DirectShow device filter that we used for early testing, we needed to implement a custom version[4] that allows us to intercept and save the raw Bayer video before passing it to FMLE.

Flash Media Streaming Server (FMSS), running on the machine handling the overview stream, streams all of the encoded video to clients.

[2]http://adobe.com/products/flashmediaserver/flashmediaencoder/faq

[3]Our measurements can only be assumed to be precise within a 50 ms window, given a frame rate of 20 fps with the cameras we used.

[4]Based on Vivek N.'s example at http://tmhare.mvps.org/downloads.htm

D. Recording

Although not yet deployed due to issues discussed in this section, a number of factors motivate capture of uncompressed data instead of encoded streams. First, we were unsure how lossy compression would impact the interpolation algorithm, and second, we can use the original data for higher-quality offline encoding than would be possible for real-time streaming, e.g., using two-pass encoding techniques, or more CPU intensive codecs such as H.264.

To record the large quantities of uncompressed camera data (16 streams between 88-180 Mbps each), we considered both solid state disks (SSDs) and RAID systems. However, we opted for a simpler and much less expensive distributed solution. Since encoding the received video is entirely CPU-bound, disk I/O capacity on each server remains largely free for recording, which is not CPU-intensive, so each machine can be tasked, in parallel, with losslessly recording the video streams it is simultaneously encoding. Unfortunately, our current implementation based on these assumptions drops frames and reduces the frame rate, and is thus not deployed. Further work is required to determine why the frames are being dropped, the results of which will impact the number of disks required on each encoding/recording machine.

IV. USER EXPERIENCE

The client player, implemented in Adobe Flash and shown in Figure 4, provides the user's experience of the camera array capabilities. It simultaneously displays a small overview video stream and a larger view of the camera currently selected by the user. The main interface elements are:

1) the selected viewpoint, dynamically changeable
2) a wide-angle overview of the entire procedure, or a view from an endoscope or other medical device, depending on the procedure being viewed
3) the option to expand the player into full screen mode
4) zoom and volume controls

A virtual knob is used to select the desired viewpoint; when the user releases the mouse controlling the knob, the main view updates to the new video stream. The design choices and scenarios around the interface are discussed in a separate publication [12], and the rest of this section describes the technical choices that support this user interface.

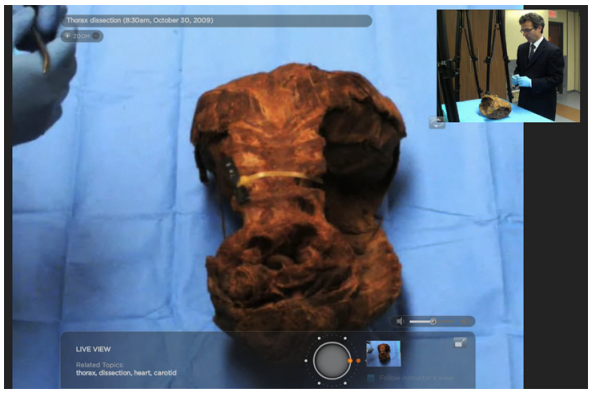

Fig. 4. Multiple angle viewer, player user interface, with a six camera array.

A. View Switching Architecture

A key consideration of our design was to achieve high responsiveness and fluidity when users select a new viewpoint. This entails an efficient switching mechanism between video streams requested by the client, without introducing significant delays or video glitches as the client updates to the new position. One of the most agonizing decisions we faced in the design of our architecture was the strategy for managing stream switching by clients as the users change viewpoints. Although the client is only responsible for decoding the video it receives, the manner in which the server handles encoding data from multiple cameras has a significant impact on the end user's experience. We now describe the encoding architectures we considered.

1) Continuous encoding of all cameras: The first design is to encode all of the camera streams continuously so they are always available immediately from the RTMP streaming server. This implementation is potentially much more CPU intensive than encoding only the subset of streams currently being viewed by users, but simplifies the architecture and allows us to expand to many more simultaneous live users without limiting which views are available to each client. As such, this is the currently deployed approach. Note that although all streams are available, the client application only receives the ones desired, and is responsible for unsubscribing from any streams no longer needed when the user requests a new view. Fortunately, dropping one RTMP stream and subscribing to a new one is very fast, provided that the streams are already published and available from the streaming server.

The problem, however, is that FMLE restricts the keyframe rate to a maximum of one per second, which results in a brief lag when switching streams. During this time, the client waits for the next available keyframe before commencing display of the newly selected video stream. The net delay is tolerable, but precludes the ability to provide clients with a completely fluid transition between camera views.[5] Nonetheless, we favoured this simple and easily scalable design over the more complex alternatives described below. Note that avoiding this delay by having the client receive all possible streams at all times is undesirable due to the resulting enormous bandwith increase.

2) Dedicated stream per client: To support a faster view transition from the client perspective, the server could instead allocate a unique, dedicated RTMP stream to each client, which the client maintains while it is active. When the user changes views, the server need only switch the raw video stream being sent to the DirectShow filter feeding the encoding pipeline for that particular client. In testing, this resulted in extremely fast switching between streams, but introduced a number of other problems, such as redundant encoding if two users request the same camera view, thus wasting valuable encoding resources dealing with the same content twice. Most significantly, this design limits the number of viewers to the number of streams the system can encode simultaneously, which is determined by the number and capacity of the encoding computers. Although this would be acceptable if we were deploying for a known, small number of simultaneously connected clients, this is not the case for our system. Another condition that would motivate further consideration of the dedicated stream per client option is if encoding costs for the full array become excessive, as might arise if using H.264 encoding, or in the case of an array containing a significantly larger number of cameras. Again, however, in this case, the maximum number of simultaneous clients would be limited by the number of sustainable encoded streams.

3) Hybrid switching: A hybrid architecture compromises between a dedicated stream per client and the full encoding of all cameras. Such a hybrid allows for more simultaneously connected clients than the number of encoded streams by limiting the number of camera views available to end users, while retaining *some* of the advantages of rapid switching between views. In this model, each machine is assigned responsibility for a fixed subset of the cameras, $[C_i, C_{i+k}]$, limited by its disk writing and network bandwidth capacities. This is to permit the continuous recording of raw camera data for all views, independently of which are being encoded. In parallel, the machine can encode some dynamic subset of these cameras, $j, j \leq k+1$, determined by its CPU capacity, thereby able to serve up to j client requests for different streams, or more if multiple clients request the same view. In this architecture, the server tracks which cameras are being streamed. When the server receives a client request for a view change from camera C_m to C_n, there are four possibilities:

1) C_m is not currently being encoded, but is assigned to a server already at full capacity: client request denied.
2) C_m is already being encoded, i.e., it is being viewed by another client: the client connects to the existing stream, incurring the client-side RTMP stream switching delay, as with the architecture from Section IV-A1.
3) C_m is not currently being encoded, but both C_m and C_n are assigned to the same server, and no other clients are subscribed to C_m: this is the best case, as that server can simply switch one of its raw camera feeds from C_m to C_n without disrupting the existing encoding pipeline for the requesting client. The view change is then extremely

[5] An exception is the interpolated view, discussed in Section IV-B.

fast, just as in the architecture of Section IV-A2.

4) Otherwise: this is the worst of the successful cases, as the system must start the encoding and streaming of C_n, and the client must also switch to the new RTMP stream.

The requirement to record the raw data for all views, coupled with limited bandwidth to each encoding machine, prevents us from allocating cameras dynamically to machines. This leads to the possibility where the encoding capacity of one server sits idle, while another server refuses client requests due to CPU saturation. Without the recording constraint, any server can encode the data from any camera, although the time required to unsubscribe from one multicast stream, then subscribe to the new one, may result in a switching delay perceived on the client. Of course, with sufficient bandwidth to each encoding machine, it would be possible to subscribe to more simultaneous streams, even if they were not being encoded, reducing this potential switching cost.

Note that if sufficient capacity exists to encode all streams, rather than only a subset, then this hybrid architecture becomes an optimized version of the first architecture. The worst case is that all cameras are encoded and every time a different camera is selected, this requires switching streams at the client side. The best case, which is likely with a small number of users, is very fast switching via stream switching on the server side.

B. Interpolation

As previously discussed, interpolated viewpoints between camera positions help the user maintain context. In the currently deployed 17 camera system, a fixed set of five cameras is used to generate interpolated viewpoints anywhere in the scene. Since the processing of the interpolated viewpoint requires more steps than simply encoding and streaming the physical camera viewpoints, there is an unavoidable additional delay in the interpolated stream. To decrease this latency, the interpolation algorithm [13] is implemented in OpenGL and performed on a Graphics Processing Unit. Although we have achieved real-time interpolation using this approach, we are continuing to tackle the problem of added latency in delivering the raw camera data to the interpolation engine in the required format, as well as in delivering the interpolated view to the client. Nonetheless, moving the interpolated virtual camera position using the client interface is extremely fast, since the client does not need to switch streams in order to see the new position. Instead, the interpolated viewpoint is simply updated on the server upon the client's request.

C. Audio

All audio associated with the camera array is presently one-way to the clients via a directional microphone worn by the instructor. Audio is synchronized and streamed with the wide-angle overview video during both live and recorded sessions, since all clients continuously receive this stream. Interaction among users, e.g., as students at remote sites work together to solve a medical problem, or between participants and the instructor, as anticipated when students wish to pose questions, requires use of a separate videoconferencing system.

V. Conclusion and Future Work

We have designed and deployed an architecture for an array of 17 Ethernet cameras, streaming live video of a medical scenario over a high-speed network. The system allows each client to select a physical camera position or an interpolated view between the physical camera positions. The live system has been demonstrated in a medical environment, streaming 800×600 resolution video at 24 frames per second to sites throughout Canada, as well as Cork, Ireland, during a teaching session involving students and instructors at multiple locations. Future work includes implementing recording of the raw video and using it to create on-demand sessions, as well as exploring hardware compression to reduce cost.

Acknowledgment

This work was funded by Canada's Advanced Research and Innovation Network (CANARIE) under a Network Enabled Platforms research contract, as part of the HSVO project. Members of the Shared Reality Environment lab at McGill University and the staff at the Arnold and Blema Steinberg Medical Simulation Centre at McGill University provided tremendous support when testing and deploying the system.

References

[1] O. Grau, A. Hilton, J. Kilner, G. Miller, and J. Starck, "A free-viewpoint video system for visualization of sport scenes," *SMPTE Motion Imaging J.*, vol. 116, no. 5-6, pp. 213–219, 2007.

[2] T. Kanade, P. Rander, S. Vedula, and H. Saito, "Virtualized reality: Digitizing a 3D time-varying event as is and in real time," in *Mixed Reality, Merging Real and Virtual Worlds*, H. T. Yuichi Ohta, Ed. Springer-Verlag, 1999, pp. 41–57.

[3] N. Inamoto and H. Saito, "Fly-through viewpoint video system for multi-view soccer movie using viewpoint interpolation," in *Visual Communications and Image Processing*, vol. 5150. SPIE, 2003, pp. 1143–1151.

[4] J.-G. Lou, H. Cai, and J. Li, "A real-time interactive multi-view video system," in *Proc. 13th annual ACM international conference on Multimedia*. New York: ACM, 2005, pp. 161–170.

[5] M. Tanimoto, "Free-viewpoint television," in *Image and Geometry Processing for 3-D Cinematography*, ser. Geometry and Computing, R. Ronfard and G. Taubin, Eds. Springer, 2010, vol. 5, pp. 53–76.

[6] Y. Chen, Y.-K. Wang, K. Ugur, M. M. Hannuksela, J. Lainema, and M. Gabbouj, "The emerging MVC standard for 3D video services," *EURASIP J ADV SIG PR*, vol. 2009, pp. 1–13, 2009.

[7] B. Wilburn, N. Joshi, V. Vaish, M. Levoy, and M. Horowitz, "High-speed videography using a dense camera array," in *Computer Vision and Pattern Recognition*. IEEE, 2004, pp. 294–301.

[8] B. Wilburn, N. Joshi, V. Vaish, E.-V. Talvala, E. Antunez, A. Barth, A. Adams, M. Horowitz, and M. Levoy, "High performance imaging using large camera arrays," *ACM Trans. Graph.*, vol. 24, pp. 765–776, 2005.

[9] Y. Taguchi, K. Takahashi, and T. Naemura, "Real-time all-in-focus video-based rendering using a network camera array," in *3DTV Conference: The True Vision - Capture, Transmission and Display of 3D Video, 2008*, 2008, pp. 241 –244.

[10] J. C. Yang, M. Everett, C. Buehler, and L. McMillan, "A real-time distributed light field camera," in *Proceedings of the 13th Eurographics workshop on Rendering*, ser. EGRW '02. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, 2002, pp. 77–86.

[11] C. Lei and Y. Yang, "Design and implementation of a cluster based smart camera array application framework," in *Second ACM/IEEE Intl. Conf on Distributed Smart Cameras*, 2008, pp. 1–10.

[12] A. Olmos, K. Lachapelle, and J. R. Cooperstock, "Multiple angle viewer for remote medical training," in *Second ACM Int. Workshop on Multimedia Technologies for Distance Learning*. ACM, 2010.

[13] S. Pelletier and J. R. Cooperstock, "Real-time free viewpoint video from a range sensor and color cameras," *Machine Vision and Applications (in review)*.