Toward 3D-IPTV: Design and Implementation of a Stereoscopic and Multiple-Perspective Video Streaming System

Goran Petrovic[a], Dirk Farin[a] and Peter H. N. de With[b]

[a]Eindhoven University of Technology, Den Dolech, 5600 MB, Eindhoven, The Netherlands;
[b]Eindhoven University of Technology and LogicaCMG, Den Dolech, 5600 MB, Eindhoven, The Netherlands

ABSTRACT

3D-Video systems allow a user to perceive depth in the viewed scene and to display the scene from arbitrary viewpoints interactively and on-demand. This paper presents a prototype implementation of a 3D-video streaming system using an IP network. The architecture of our streaming system is *layered*, where each information layer conveys a single coded video signal or coded scene-description data. We demonstrate the benefits of a layered architecture with two examples: (a) stereoscopic video streaming, (b) monoscopic video streaming with remote multiple-perspective rendering. Our implementation experiments confirm that prototyping 3D-video streaming systems is possible with today's software and hardware. Furthermore, our current operational prototype demonstrates that highly heterogeneous clients can coexist in the system, ranging from auto-stereoscopic 3D displays to resource-constrained mobile devices.

Keywords: 3D-TV, IPTV, stereoscopic video, view interpolation, depth image, multimedia streaming

1. INTRODUCTION

A strong growth in camera surveillance and security systems and the emergence of 3D-TV has spawned interest in building large multi-camera recording systems.[1] A number of consumer-grade cameras are used to synchronously record a scene from multiple viewpoints. In a *multiple-perspective* viewing scenario, a scene can be displayed from different viewpoints (angles) interactively and on-demand. A user either selects a new viewpoint, or his movements are continuously tracked and the displayed content automatically adjusted. *Stereoscopic video* is a special case of multiple-perspective viewing, where the depth effect is rendered with the help of a specialized display device (head-mounted glasses or an auto-stereoscopic display). For brevity, we refer to multiple-perspective and stereoscopic applications jointly as *3D-video* and make clear distinctions where appropriate.

This paper focuses on the design and implementation of a 3D-video streaming system in IP networks. In doing so, we assume that suitable multi-camera recording systems can be constructed, as exemplified in recent work on this topic (e.g.,[2,3,4]). Moreover, we believe that the state-of-the-art video coding standards (MPEG-4/H.264) – although not specifically tailored for the compression of 3D-video data – are at least readily applicable to fast prototyping and building operational streaming systems. Armed with these two assumptions, we design and implement an efficient 3D-video streaming architecture while addressing the key practical trade-off in 3D-video streaming: *the balance between the computation load and the network bandwidth.*

The remainder of the paper is structured as follows. Section 2 surveys the recent related work and puts our work in context. Section 3 presents our choice of a 3D-video data format, introduces the layering concept and describes the rendering algorithm. In Section 4 we detail on the software components, their integration in the system and briefly report on the end-to-end system performance. Section 5 highlights the main points of the presented work.

Further author information: (Send correspondence to G.P.)
G.P.: E-mail: g.petrovic@tue.nl, Telephone: +31 40 247 3708
P.H.N.de.W.: E-mail: p.h.n.de.with@tue.nl, Telephone: +31 40 247 2540

2. RELATED WORK

Our work builds upon recent research in several application areas related to networked 3D-video systems, most notably stereoscopic video streaming over the Internet, tele-immersion and multi-view streaming systems.

2.1 Stereoscopic video transmission

Stereoscopic video transmission over the IP networks was first considered in the *Smile!* teleconferencing system.[5] *Smile!* is an end-to-end system for capturing, encoding, transmission and display of stereo streams. The system features separate JPEG coding of the two streams, standard protocols for real-time transmission[6] and active shutter-glasses for display. The main focus of this work is on the signalling extension to the RTCP .[6] This extension is necessary for the association of the two streams as belonging to the same session and distinction between them within the session. More recent system studies employ state-of-the art video codecs for compression (e.g., H.264) and investigate the bitrate savings achievable when encoding the two streams in different qualities or spatio-temporal resolutions[7].[8] The test-case implementation in Section 4.1 of this paper bears technical similarity to recently published studies on stereoscopic streaming [7].[8] However, in this paper we discuss additional system aspects and suggest a way of implementing both the stereoscopic and the multiple-perspective video within the same framework.

2.2 Immersive teleconferencing

A delivery model where a 3D-video sender transmits all available camera streams to the receiver is common in *immersive teleconferencing* systems,[9] which employ multi-camera set-ups at each participant's site. The choice of the delivery model is clearly dictated by the application – to create a continuous immersive feeling, all available streams must be rendered at the same time. Due to high bandwidth requirements, such systems are typically implemented over high-bandwidth WAN networks (e.g., Internet2). Still, efficient compression[10] and congestion control algorithms[11] are important in practice. Adaptive streaming strategies have also been proposed to exploit unequal importance of the individual streams and maximize the user QoS under a given bandwidth constraint,[12].[3] The same delivery model has also been considered for multi-view stereosopic 3D-TV applications.[2] Matusik and Pfister[2] present an end-to-end system for multi-view stereoscopic rendering of a dynamic scene. The camera streams are encoded using MPEG-2, and rendered on a multi-projector system with lenticular screens. The focus of their work is on the projector system design and rendering, rather than on the encoding and transmission aspects. Our work targets a different application space. Instead of focusing on immersive (and computationally expensive) scene rendering, our main goal is to enable interactive exploration of a remote scene.

2.3 Multi-view streaming systems

In *multi-view streaming systems* (or equivalently, *Free-Viewpoint Television*[13]), a remote scene can be displayed from arbitrary viewpoints interactively and on-demand. The setup of Kimata *et al.*[14] consists of a large number of closely spaced cameras so that arbitrary perspective views can be created by view interpolation in the *ray-space*. When the receiver requests a viewpoint change, the server transmits only the views required to create the desired virtual viewpoint. A similar system, supporting unicast and multicast of multiple viewpoints and a number of multi-view special effects, is presented by Lou *et al.*[4] Still, without view interpolation, the set of available viewpoints in this system is limited to the original cameras. The lack of global Internet support for IP-multicast can be alleviated using a peer-to-peer overlay network for multi-view video delivery.[15] In our earlier work,[16] we propose a framework for layered transmission of multi-stream data and also suggest the application of network overlay architectures for higher server and network efficiency. Additionally, the proposed framework[16] introduces virtual viewpoint creation (view interpolation) as a new functionality to be implemented in a network overlay, e.g., in a group of collaborating overlay nodes. The system described in this paper complements and extends the prior work in that our system proposal includes an implementation of the view interpolation module, which is efficient and flexible enough to be employed at either the sender, or at an intermediate node (e.g., proxy/peer), or at the receiver.

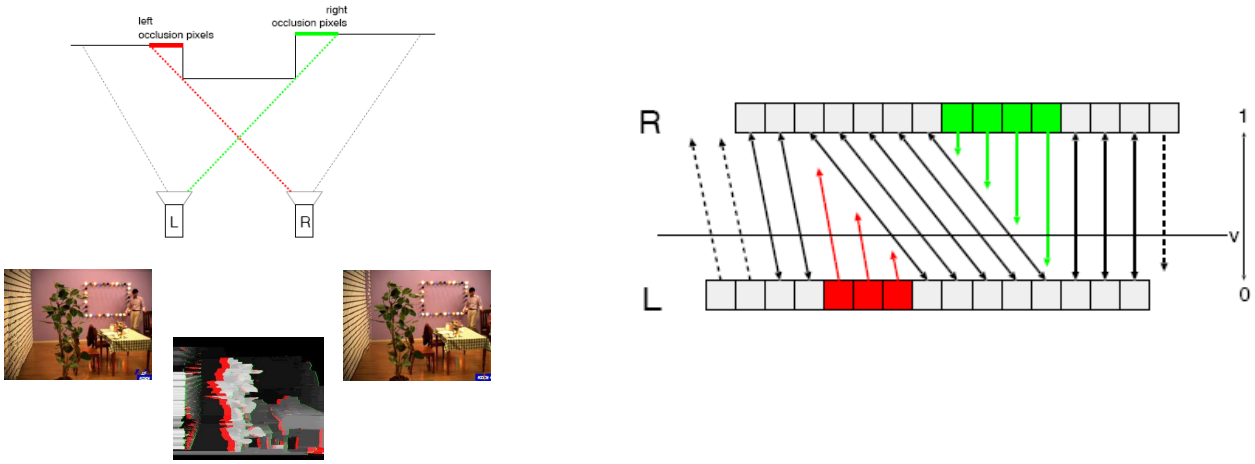

Figure 1. Visualization of the interpolation process. Arrows indicate how pixels move in the parallax motion. An intermediate view corresponds to a horizontal cut at a position v between the two extreme views. Double-headed arrows indicate that the pixel color is interpolated between two views, while the data along single-headed arrows is simply copied.

3. SYSTEM DESIGN

3.1 Addition of depth (disparity) maps

3D-Video rendering algorithms typically assume some form of geometric model for the scene in order to generate virtual views more efficiently. The geometric model used in our system consists of *multiple depth maps*, one for each original camera.[17] For every pixel in a video frame, a depth map conveys the distance between the camera plane and the nearest surface point in the scene. Using a single video frame and its depth map, multiple nearby perspective views can be rendered by re-projecting the pixel values driven by the depth map. This class of rendering algorithms is often referred to as Depth-Image Based Rendering (DIBR) in the literature.[13] For an overview of algorithms for automatic extraction of depth maps from video frames, the interested reader is referred to a recent survey .[13] DIBR is particularly effective for narrow-angle viewpoint changes (small positional difference between the original and a virtual view), where the distortions in virtual views are usually imperceptible. For wide-angle perspective changes, rendering distortions come in the form of holes in the texture of the virtual views. This situation illustrates the "disocclusion" problem, where certain regions of the scene occluded in the original view become visible in the virtual view. As no algorithmic solution to the disocclusion problem exists, a practical rendering algorithm must combine information from multiple original cameras. If the number of cameras aimed at a certain part of the scene is large enough, the scene regions occluded in one camera will be visible in one of the neighboring cameras. The rendering algorithm then proceeds by blending the frames from the selected set of cameras, while taking into account the visibility relationships between them.

3.2 View interpolation

View interpolation refers to a set of rendering algorithms that render virtual views of the scene by blending a number of original views. The view-interpolation algorithm integrated into our system[18] enables a *continuous horizontal blending* between pairs of original cameras. For each pair of adjacent cameras, the input to the algorithm includes: (1) the left-camera stream; (2) the right-camera stream; (3) per-pixel disparity (depth) maps; (4) occlusion information (pixels flagged as visible to one of the cameras only). The algorithm creates a virtual view at an arbitrary position between the cameras by weighting their respective contributions at every pixel location(Fig. 3). To achieve real-time performance, the view-interpolation module partly relies on hardware acceleration, i.e., it uses the OpenGL API[19] to take advantage of the hardware-rendering capabilities of today's PC graphics cards.

The implemented interpolation algorithm allows to generate intermediate views along a chain of "weakly" calibrated cameras. In the case of weak calibration, only a projective transform can be specified between adjacent

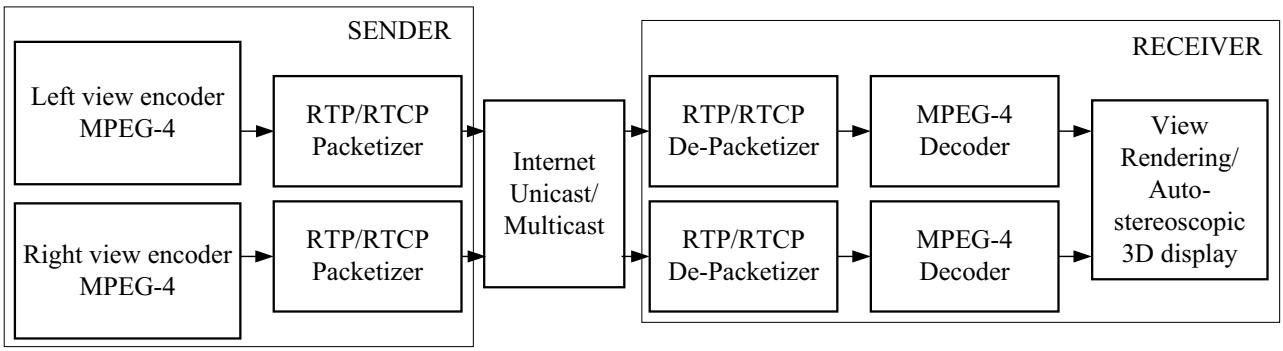

Figure 2. Example of a dual-layer stereoscopic video streaming system.

cameras, defined by the fundamental matrix. Therefore, we estimate the epipolar geometry from the input images directly. As a preprocessing step, our interpolation algorithm estimates the epipolar geometry for each pair of adjacent cameras and rectifies the input images to obtain horizontal epipolar lines. Next, the disparity images are estimated for each pair of cameras in the rectified coordinate system. The rectification step makes the situation convenient for disparity estimation, since the parallax motion becomes a one-dimensional phenomenon in the horizontal direction. The disparity image is estimated using a 1D dynamic-programming algorithm with a cost function that reduces the inconsistencies of disparity estimates between scan lines. The details of the computation of the rectification transform and disparity estimation can be found in Ref.[18]

3.3 Layered streaming

Information layering is a well-known design guideline for the content representation in adaptive streaming systems. It is commonly used to simplify the problem of receiver heterogeneity by allowing the system to select the number of layers to transmit based on receiver preferences or capabilities. As a result, representation scalability is achieved, where the rendering quality depends on the selected number of layers (e.g., *scalable* video coding).

Application of the information layering concept in our system means that each information layer conveys a single video stream or a scene-description stream. This design decision is based on two observations related to the 3D-video rendering process. First, the usage of scene-geometry description for rendering (e.g., depth maps in DIBR, occlusion information in view interpolation) is a norm in computer graphics, but is uncommon in video streaming. It is therefore useful only to the receivers which support this rendering option. Second, the quality of interpolated virtual frames depends on the number of original frames and depth maps that are blended together. In view of a large number of possible combinations, we believe that optimized decisions are best implemented at run-time, for each individual session independently.

Following the layering principle, the *number* of layers used can be extended to include multiple viewpoints, which we call a *multiple-perspective streaming system*. The user can select the number of different camera streams to receive simultaneously. Similarly, in the scene-descriptive case, the number of layers can be increased to improve the quality by sending additional descriptive data, such as occlusion or objects masks. As an example of a combined scheme, it is possible to transmit several camera streams and a selection of supporting depth maps, where the depth maps can be located at intermediate positions. In its optimal form, the actual transmission scheme at any given time will be adaptively determined by the view interpolation algorithm, taking into account the user navigation patterns and the resources available.

3.4 Interactivity

Another important system aspect is the inherent *interactivity*. In the system we envisage, some camera streams will be transmitted frequently, and others not at all, entirely driven by the user interest. Therefore, an on-demand transmission is desired to reduce the overall bandwidth cost. As our initial implementation experiences suggest, these bandwidth savings appear sufficient to implement 3D-video streaming systems in today's networks.

4. IMPLEMENTATION AND RESULTS

Our prototype implementation includes software modules that demonstrate both aspects of 3D-video streaming introduced in this paper: (a) stereoscopic video streaming, (b) monoscopic video streaming with remote multiple-perspective rendering. For the experimental implementation of stereoscopic video streaming, we currently instantiate two layers. Depending on the stereoscopic image-rendering algorithm available at the receiver end, either a *depth* stream and a *texture* stream (either left or right),[16] or both the *left* and the *right* camera streams are transmitted (Figure 2). We currently support these two 3D-video modalities as separate applications. Their integration into an end-to-end system is ongoing and planned for the next implementation stage. In this paper, we can only report on the implementation for the case where all the data required by the view interpolation is locally available. Therefore, our current multiple-perspective streaming prototype can be thought of as a client-server application in which the server creates the interpolated views, compresses and streams them to the client in real time (Fig. 3).

4.1 Stereoscopic streaming

4.1.1 Sender

At the sender, the pre-recorded left and right camera views are independently encoded using an open-source MPEG-4 SP encoder,[20] and stored in separate files. When a streaming session is started, the compressed bitstreams are read frame-by-frame, packetized in RTP packets using the open-source Live555 library,[21] and sent to the receiver. According to IETF recommendations for the carriage of MPEG-4 elementary streams (ESs) over IP networks[6],[22] the two streams are transmitted as separate RTP sessions, i.e., using different RTP/RTCP ports *. The sender's packet scheduler implements a round-robin discipline – it intersperses transmissions of the left and the right video stream on a frame basis.

4.1.2 Receiver

The receiver system includes components for RTP depacketization, decoding and display. The RTP depacketizers assemble the received RTP packets into compressed MPEG-4 frames and pass them on to the corresponding left and right stream decoders. Both decoders run in the same thread and an application-level coordination is implemented to ensure fair treatment of both streams. The display module receives pairs of decoded frames and combines them to create a *stereoscopic image*, in a format specific to the auto-stereoscopic display we use in the system.[23]

The streaming experiments were performed on a high-speed LAN in our department. We achieve real-time end-to-end system performance with both half-SDTV (Standard Definition broadcast TV, 720×288) and VGA-type resolution sequences (e.g., 800×600), while running the receiver application on a 3GHz desktop PC. The depth effect while viewing a 3D scene is compelling with both synthetic and real-world sequences (captured in our lab). The synchronization between the two streams is maintained over all system components, providing a working end-to-end prototype for stereoscopic video streaming with glitch-free display.

4.2 Multiple-perspective streaming

4.2.1 User interaction

The viewpoint control to support interactive scene viewing is implemented using standard PC input-devices. User sends a request for a viewpoint change by moving the mouse pointer inside of the video display window. The generated events are transmitted to the sender, including the current pointer position in display window coordinates (currently, the "x-coordinate" only).

*Definition of an RTP Payload Format for the transmission of stereoscopic streams is beyond the scope of our work.

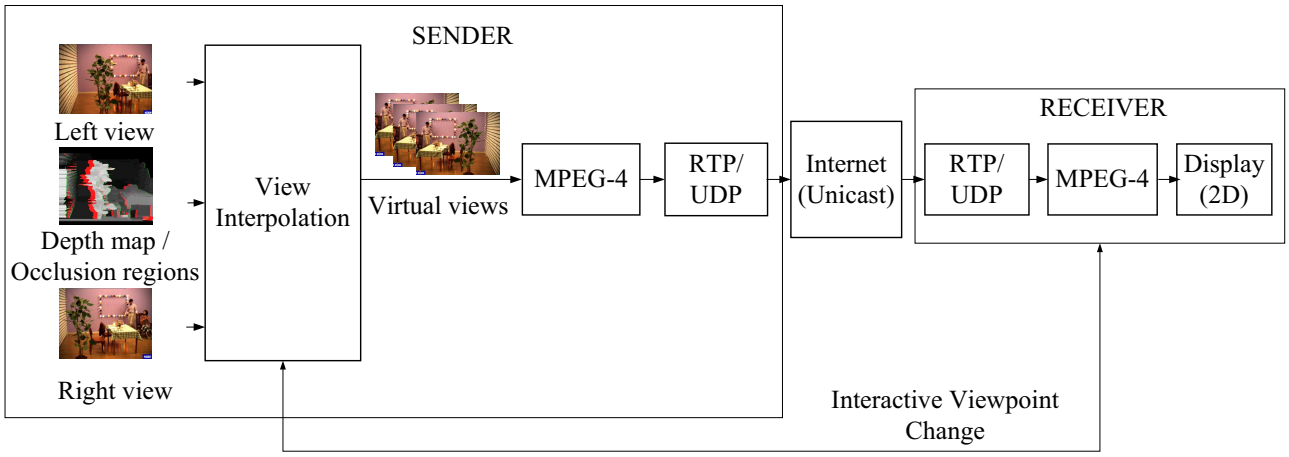

Figure 3. Multiple-perspective video streaming.

4.2.2 Sender

The sender maintains a mapping between the client's display window coordinates and the coordinates of the capturing cameras. A routine in the "listening thread" at the sender determines if the receiver coordinates match one of the original cameras and if so, transmits its video stream. Otherwise, it starts the view-interpolation procedure.

The intermediate views are interpolated in the rectified coordinate system, using the disparity estimates, and projected back into the original coordinate system prior to display.[18] Carrying out the interpolation on rectified images has the advantage that only horizontal pixel shifts are involved. The interpolation consists of compensating for both the translational and the rotational motion of the virtual camera. We compensate for the translational motion using the estimated disparities. The compensation of the rotational component is simulated by gradually interpolating the rectification transforms.

Our *interpolation algorithm* operates on two rectified original views and a disparity map, and distinguishes between these cases:

A. Pixels are visible in both views. These pixels have a corresponding disparity value d assigned to it. The pixel position in the intermediate view can be obtained by scaling the disparity by the inter-view position v, giving a new disparity value $d' = vd$. The color values between the left and right view are also interpolated according to v, as indicated in Fig. 1 by double-headed black arrows.

B. Pixels in occlusion areas that are only visible in one view (see marked areas in Fig. 1). These pixels do not have a disparity value assigned to them, as they are explicitly detected as occluded in the disparity estimation process. We observe from marked areas in Fig. 1 that the pixels in the left view are occluded by the objects positioned at their right side, while the pixels in the right view are occluded by the objects at their left side. Assuming that occluded objects are parallel to the image plane, we extend the disparities of the background into the occlusion area. Hence, for pixels in the left view, the missing disparity values are copied from the left side, while for the right view, they are copied from the right side (colored arrows in Fig. 1). Since these pixels are only visible in one view, we cannot interpolate the color values between the two views, but only copy them.

C. Pixels in one view whose corresponding pixels are outside the valid image area in the other view. For these pixels, no disparity was computed (see Fig. 1). Again assuming that objects are mostly parallel to the image plane, we extend the disparity of the border pixels into the undefined region (see dashed lines in Fig. 1). Similar to the previous case, the pixel color cannot be interpolated, since the pixel is only visible in one view.

The interpolated virtual-viewpoint frames are read back from the GPU frame-buffer, performing a color-space conversion for every frame on-the-fly (RGB to YUV). The resulting frames are passed on to the encoder and

Figure 4. View interpolation of two middle views from two surrounding cameras.

subsequently to the streaming library, using the same components as in Section 4.1. Figure 4 illustrates two original camera frames and one interpolated virtual frame between them.

4.2.3 Receiver

As only a monoscopic video stream (containing a sequence of virtual and/or original frames) is transmitted, the receiver application can be any media-player with streaming support (e.g.,[24]). Our choice of remote rendering design is geared towards enabling multiple-perspective video on receivers constrained in bandwidth and rendering resources. Consequently, we have run tests with low-resolution multi-view sequences (e.g., CIF). The sender is a 3-GHz Desktop PC with a state-of-the-art graphics card †. The receiver (a 400-MHz iPAQ PDA) uses a VLC client[24] for decoding and display.[25] The viewpoint change is effective with a delay of 500 ms (mainly the buffering delay in the player software), thus demonstrating the interactive properties of our system.

5. CONCLUSIONS AND FUTURE WORK

We have presented a scalable architecture for the delivery of 3D-video streaming, allowing both multiple-perspective and stereoscopic viewing. First, we have cast the multi-perspective transmission as an on-demand layered streaming problem and implemented a stereoscopic streaming prototype using standard transport protocols and compression techniques. Second, we have addressed the continuous navigation requirement of 3D-video systems by incorporating an interactive view rendering and interpolation algorithm at the sender end. Our current operational prototype demonstrates that highly heterogeneous clients can coexist in the system, ranging from auto-stereoscopic 3D displays to resource-constrained mobile devices.

Our current implementation of multiple-perspective streaming assumes that all the data required for view interpolation is locally available. A natural extension is to explore the scenario where view interpolation is performed on the data streamed from a remote location. In the context of the implemented view interpolation scheme, two video streams, one depth stream and a sequence of occlusion masks would have to be delivered to the view interpolation process in real-time. As our model assumes best-effort IP networks and an unreliable transport protocol (UDP), an interesting avenue for future research would be to explore adaptive streaming algorithms. The adaptation in a 3D-video system will have to apply optimizations that cut across its multiple layers of information.

ACKNOWLEDGMENTS

The "Objects" video sequence used in this work was kindly provided by the KDDI Corporation.

†Commercially available under Nvidia GeForce 7600GS.

REFERENCES

1. 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*, pp. 1–10, June 2002.
2. W. Matusik and H.-P. Pfister, "3D TV: A scalable system for real-time acquisition, transmission, and autostereoscopic display of dynamic scenes," in *Proc. SIGGRAPH'04*, pp. 814–824, Aug. 2004.
3. E. Lamboray, S. Würmlin, and M. Gross, "Data streaming in telepresence environments," *IEEE Transactions on Visualization and Computer Graphics* **11**, pp. 637–648, Nov. 2005.
4. J. Lou, H. Cai, and J.Li, "A real-time interactive multi-view video system," in *Proceedings of the 13th annual ACM international conference on Multimedia*, pp. 161–170, Nov. 2005.
5. M. Johanson, "Stereoscopic video transmission over the Internet," in *IEEE Workshop on Internet Applications*, July 2001.
6. H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "RTP: A transport protocol for real-time applications," *RFC 3550, IETF* , July 2003.
7. A. Aksay, S. Pehlivan, E. Kurutepe, C. Bilen, T. Ozcelebi, G. B. Akar, M. R. Civanlar, and A. M. Tekalp, "End-to-end stereoscopic video streaming with content-adaptive rate and format control," *Signal Processing: Image Communication* **22**, pp. 157–168, Feb. 2007.
8. H. Kalva, L. Christodoulou, L. M. Mayron, O. Marques, and B. Furht, "Design and evaluation of a 3D video system based on H.264 view coding," in *Proc. (NOSSDAV)*, May 2006.
9. H. Towles, S.-U. Kum, T. Sparks, S. Sinha, S. Larsen, and N. Beddes, "Transport and rendering challenges for multi-stream 3D tele-immersion data," in *NSF Lake Tahoe Workshop on Collaborative Virtual Reality and Visualization*, Oct. 2003.
10. S. Kum and K. Mayer-Patel, "Real-time multidepth stream compression," *ACM Transactions on Multimedia Computing, Communications, and Applications* **1**, pp. 128–150, May 2005.
11. D. E. Ott and K. Mayer-Patel, "Coordinated multi-streaming for 3D tele-immersion," in *Proceedings of the 12th annual ACM international conference on Multimedia*, pp. 596–603, Oct. 2004.
12. Z. Yang, B. Yu, K. Nahrstedt, and R. Bajcsy, "A multi-stream adaptation framework for bandwidth management in 3D tele-immersion," in *Proc. NOSSDAV*, May 2006.
13. A. Smolic and P. Kauff, "Interactive 3D video representation and coding technologies," *Proc. of the IEEE* **93**, pp. 98–110, Jan. 2005.
14. H. Kimata, M. Kitahara, K. Kamikura, and Y. Yashima, "System design of free viewpoint video communication," in *Proc. Fourth Intern. Conf. on Computer and Information Technology (CIT04)*, 2004.
15. E. Kurutepe, M. R. Civanlar, and A. M. Tekalp, "Interactive transport of multi-view videos for 3DTV applications," *Journal of Zhejiang University: Science A* **7**(5), pp. 830–836, 2006.
16. G. Petrovic and P. H. N. de With, "Near-future streaming framework for 3D-TV applications," in *IEEE International Conference on Multimedia and Expo (ICME)*, pp. 1881–1884, July 2006.
17. Y. Morvan, D. Farin, and P. H. N. de With, "Design considerations for a 3D-TV video coding architecture," in *IEEE International Conference on Consumer Electronics (ICCE)*, pp. paper 6.4–5, Jan. 2008.
18. D. Farin, Y. Morvan, and P. H. N. de With, "View interpolation along a chain of weakly calibrated cameras," in *IEEE Workshop on Content Generation and Coding for 3D-Television*, June 2006.
19. "OpenGL." `http://www.opengl.org/`.
20. "FFmpeg Multimedia System." `http://ffmpeg.mplayerhq.hu/`.
21. "Live555 Streaming Media." `http://www.live555.com/`.
22. Y. Kikuchi, T. Nomura, S. Fukunaga, Y. Matsui, and H. Kimata, "RTP payload format for MPEG-4 audio/visual streams," *RFC 3016, IETF* , Nov. 2000.
23. "SeeReal Technologies." `http://www.seereal.com/`.
24. "VLC media player." `http://www.videolan.org/vlc/`.
25. S. Chen, J. Lukkien, I. Radovanovic, M. Tjiong, R. Bosman, R. Verhoeven, and G. Petrovic, "VICSDA: Using virtual communities to secure service discovery and access," in *Proc. of QShine*, July 2007.