Web-based Interactive Free-Viewpoint Streaming

A framework for high quality interactive free viewpoint navigation

Matthias Ueberheide[1], Felix Klose[1], Tilak Varisetty[2],
Markus Fidler[2] and Marcus Magnor[1]

[1]TU Braunschweig, Germany [2]Leibniz Universität Hannover, Germany
{ueberheide,klose,magnor}@cg.cs.tu-bs.de,
{tilak.varisetty,markus.fidler}@ikt.uni-hannover.de

ABSTRACT

Recent advances in free-viewpoint rendering techniques as well as the continued improvements of the internet network infrastructure open the door for challenging new applications. In this paper, we present a framework for interactive free-viewpoint streaming with open standards and software. Network bandwidth, encoding strategy as well as codec support for open source browsers are key constraints to be considered for our interactive streaming applications. Our framework is capable of real-time server-side rendering and interactively streaming the output by means of open source streaming. To enable viewer interaction with the free-viewpoint video rendering back-end in a standard browser, user events are captured with Javascript and transmitted using WebSockets. The rendered video is streamed to the browser using the FFmpeg free software project. This paper discusses the applicability of open source streaming and presents timing measurements for video-frame transmission over network.

Categories and Subject Descriptors

I.3.3 [**Computer Graphics**]: Picture/Image Generation-Miscellaneous; I.3.8 [**Computer Graphics**]: Applications; H.4 [**Multimedia Transport and Delivery**]: Miscellaneous

General Terms

Network, Streaming, Interactive, Free-Viewpoint Video

Keywords

Viewpoint; HTML5; Real-time streaming

1. INTRODUCTION

With the success of video platforms such as YouTube, video-on-demand services as Netflix and the increase in live streaming platforms such as Twitch, the internet infrastructure becomes ready for more involved media applications. Recently, interactive gaming services that enable server-side rendering of complex game graphics to thin-client consoles at home, started to appear. However, it is still extremely challenging to stream interactive applications due to the inherent constraints on low latency, network bandwidth and the requirement for real-time encoding. Existing frameworks and commercial products that enable applications such as Twitch, Gaikai, Sony Playstation NOW and Skype rely heavily on proprietary software and video codecs. In contrast, our goal in this paper is to evaluate and show the possibility of creating an interactive free-viewpoint experience using open standards and software. The client application runs completely inside a HTML5 compatible webbrowser and requires no further addons or plugins.

The main challenge in creating interactive video streaming applications, where the video content is directly controlled by the viewer, is keeping low latency response times. We explore different aspects of real-time, low latency streaming while only relying on open source APIs as backend and widely available consumer browsers as frontend.

Free-viewpoint video applications enable the viewer to interact with the video by freeing the camera from its usually fixed position and let the user explore the video [7]. The challenge in synthesizing new views from preprocessed data in real-time is the high volume of data required to represent the entire scene. Streaming these large volumes of input data to a client side renderer is impractical. Our system avoids high bandwidth data transfer by combining server-side rendering with low latency streaming.

We present an approach for image-based free-viewpoint rendering based on the work by Lipski et al. [2] and its implementation in a real-time rendering and real-world networking environment. We employ only open standardized protocols and open source codecs. This enables interactive video exploration in standard consumer browsers. The aim of this work is to demonstrate an interactive streaming application and implement a measurement technique to evaluate the performance of that application. The first contribution of this paper is to demonstrate an image based real time capable interactive free-viewpoint system. The second contribution of this paper is to give an overview of the challenges in real-time streaming using only open source software and provide results on the overall system performance in terms of delay. This work reflects the network characteristics of an interactive rendering application for Local Area Network environments.

MM'15, October 26–30, 2015, Brisbane, Australia.

DOI: http://dx.doi.org/10.1145/2733373.2806394.

Figure 1: (left),(right) Two input images and (middle) the generated in between view.

2. RELATED WORK

Video streaming over the internet can be categorized into two groups: VOD (video on demand) and live streaming [9]. One of the main differences is that for VOD it is possible for the client to buffer data to guarantee an uninterrupted playback while live streaming ideally requires no buffering at all. This requirement can be relaxed for live broadcasts that do not involve an interactive component, such as live sporting events. For interactive live streaming however, the video frames that are shown are generated in real-time and their content is influenced directly by the user. Well-known interactive systems are teleconferencing applications such as Skype, remote desktop sharing, and recently streamed gaming platforms such as Gaikai, Steam Homestreaming or Sony Playstation NOW. All of the named methods are only available commercially and heavily rely on proprietary software for encoding and display, which makes it difficult to use them in an open research environment.

Transmitting geometry and texture data required for a client to interactively render and show the information can be compressed efficiently for transmission [12]. However, this is mostly focused on sending static objects and scenes and is not suitable for continuous transmission of dynamic sequences.

Current research into interactive multi-view streaming [5, 13, 14] focuses on optimizing the encoding, storage and bandwidth requirements for transmitting and switching between multiple streams. The authors [1] have proposed a framework for streaming 3D objects to HTML5 browser without plugins. Our work focuses on streaming a single video generated interactively on the render-server side from multiple input videos of a scene containing arbitrary motion.

Shi et al. [10] propose a low latency remote rendering system that implements the real-time streaming of 3D data over the network to a mobile client.

Creating new viewpoints from multiple input streams is often referred to as free-viewpoint rendering. In the last years impressive results were created by depth-based rendering techniques such as [15] as well as purely image-based methods [3] that do not require explicit geometry. We base our free-viewpoint rendering on a hybrid approach presented recently by Lipski et al. [2] combining the strength of image-based and depth-based methods. While their rendering results have the high quality required, their system is not capable of creating interactive viewpoint changes in real-time. Video exploration of performances captured by multiple cameras has been done by Miller et al. [7]. They base view interpolation on proxy geometry from visual hulls of the captured object. First research into general scene interactive free-viewpoint video has been done by Meyer et al. [6], creating a desktop application for interactive free-viewpoint exploration.

To the best of our knowledge, the recent work does not illustrate the methodology for accurately mapping the video frame packets from the server to the client. We employ only open standardized protocols and open source codec implementations like the HTML5 video element, the Ogg Theora video codec and the WebSockets framework for user event transmission.

3. RENDERING PIPELINE

Image-based rendering relies directly on images to interpolate new views instead of modelling and subesequently rendering textured 3D geometry. For a new viewpoint of the scene a subset of the input frames is selected. These nearby frames are warped to the position of the desired view based on dense image correspondences and blended together for a convincing new virtual view [3]. The image correspondences are obtained automatically using a high quality image correspondence estimation method [4]. Dense correspondences additionally allow the computation of per pixel depth maps, with visible imperfections from incorrect correspondences. The hybrid approach by Lipski et al. [2] can be used to render consistent views despite the imperfections in scene depth. In this section we will give a short overview of the hybrid approach. The amount of data required by the renderer is discussed to motivate the use of server-side rendering.

3.1 Hybrid Approach

The hybrid free-viewpoint video approach relies on additionally available depth maps and uses 2D image correspondences to apply a warping in 3D space.

For every frame A the per-pixel depth D_A is known. Using the intrinsic parameters (e.g. focal length) and the extrinsic parameters as camera matrix M_A, each pixel X can be projected to its 3D space position x,

$$x = M_A^{-1} \left(\frac{r_f(X)}{||r_f(X)||} D_A(X) \right), \tag{1}$$

where $r_f(X)$ denotes the ray through pixel X.

The image-space correspondence maps a pixel in one frame to a pixel position in a second frame. Since the 3D positions for both is known from the respective depth maps and camera positions, a 3D space mapping can be computed. Using this 3D mapping allows the warping and blending of images in 3D. The new 3D position x' of a pixel x in frame A, can be computed from the its 3D correspondence W_{AB} from source frame A to target frame B.

$$x' = x + c \cdot W_{AB}(X). \tag{2}$$

The warp coefficient c determines how far the pixels from frame A are warped towards their positions in B. It can be obtained by an orthogonal projection of the virtual view position onto the source to target vector. The warps are applied and the 3D pixels reprojected respecting their depth. To allow for smooth blending between multiple views soft z-buffering is applied. For a more detailed description of the image synthesis process we refer to [2].

3.2 Bandwidth

Image-based free-viewpoint video requires a huge amount of data to be loaded for each rendered frame. In addition to the image color and depth data for all frames, the dense 2D image correspondences have to be stored for each image

pair that shall be used for rendering later on. Assuming for one camera to have a neighboring camera above, below, to the left and to the right results in four image pairs per input frame. To enable temporal interpolation as well, we add the correspondences to the next frame in the same camera and therefore require five correspondence maps per input frame. The correspondence map is saved as a two-channel image with two bytes per channel resulting in approximately 16.5MB per warp for full HD images. Although that includes source and target images, which can be shared between warps with the same source or target, in the worst case the warps do not share any images. A small scene with 10 cameras, 100 frames per camera and dense correspondences from each frame to five of its neighbors can reach up to 80GB. Streaming this amount of data to the client on demand is impractical with current bandwidth limitations for network traffic. Rendering the scene on a server and just transmitting the video, however, reduces the required bandwidth and allows even small devices to present interactive free viewpoint video.

Rendering the free viewpoint video in real-time on the server remains a challenging task as the full scene does not fit into GPU or main memory. Scheduling is required to decide which parts of the data should be available in RAM or GPU memory and which parts of data should remain on disk. The typical exploration of free viewpoint video utilizes continuous camera paths which leads to continuous access to the camera data. Thus the current position is used as a predictor to preload images and correspondences from surrounding cameras. This applies for the frame and warp data; the intrinsic and extrinsic data are uploaded on demand per warp computation but can remain in main memory as their memory footprint is small.

Our rendering application is able to produce 30fps on a computer with an Intel i7 CPU 3.2GHz, a NVidia GeForce GTX TITAN Black GPU and 12GB RAM. However, the client side framerate is mostly limited by network bandwidth and buffering.

4. NETWORK PIPELINE

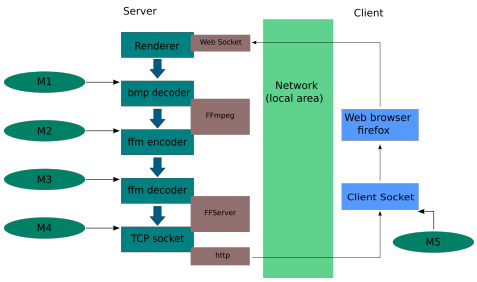

Figure 2: System overview

We developed a framework for streaming an image from the OpenGL-based renderer to a webbrowser-based client using the open source software ffmpeg and ffserver for encoding and streaming. The video stream is rendered in the browser using the HTML5 video element and WebSockets are used for the browsers communication with the server-side rendering application [8]. This allows us to perform our experiments in a Firefox browser without any modifications or plugins on the client side. In our experiments we run ffmpeg and ffserver in version 2.1, on the client side we performed our tests using Firefox 32.0 and Google Chrome 42.0.

Figure 2 illustrates the overall experimental set up. The OpenGL based renderer produces raw images which are transfered in memory to ffmpeg through the image2pipe interface. The very low overhead BMP format structure is used for the RGB frames before they are encoded in the ffm format used by the ffserver internally. The encoded stream is streamed via ffserver to the Firefox browser when the client connects via HTML5 video element to the ffserver feed. To avoid encoding delay, B-frames are disabled for the encoder. The user interaction, such as mouse and key events, is captured in Javascript and sent to the server using the WebSocket connection.

We rely completely on TCP/HTML streaming which has the advantage that TCP streaming over port 80 will not be blocked by typical firewall configurations. Streaming over TCP has other challenges in the form of the congestion control [11] and thus delays have to be expected.

To measure the delay introduced by every part of the pipeline each video frame is annotated with epoch times which are logged at each measurement point at the beginning of the components (see Figure 2).

The first measurement point $M1$ lies directly after the OpenGL renderer and before the encode image preprocessing. At the input of the ffm encoder component the frames and their timestamps are logged again at $M2$. The ffm encoded data stream is handed to the ffserver which has to decode the ffm format again after the third measurement point $M3$. Before the data is sent to the client via the TCP socket the delay is measured again at $M4$. Finally, the transmitted TCP packets are logged with the network analysis tool Wireshark on the client side. By logging the position from the OGG header on both, the sender and receiver side, each received packet can be matched to the timestamp when this particular frame was encoded. Comparing the sender side timestamps against the timestamps of the matching incomming packets on the client side, measured at measurement point $M5$, allows a fine granular delay measurement. To ensure the accuracy of this measurment the sytem clock of the client and server have been synchronized using NTP.

5. RESULTS

For the evaluation of our pipeline we measure the delay between the measurement points (see Figure 2). The experiment is performed over 100 Mbps link capacity over the LAN. Table 1 shows the average delay measured over 1000 frames per run.

The image preprocessing delay is calculated from $M1$ to $M2$ which includes the required image preprocessing (e.g. changes in color format for optimal encoding). Similarly the delay of the ffm encoder is measured from $M2$ to $M3$. The delay of ffm decoder includes the reinterpretation of the ffm container by the ffserver from $M3$ to $M4$. As buffering delay we measure time spent waiting for the next frame to fully load. The total delay is computed from $M1$ to $M5$.

We can observe that the delays remain constant over time, when we tested streaming large numbers of frames. However the buffered frames in the pipeline still constitute a significant part of the observable delay. Each individual part of the encoding and streaming pipeline keeps only approximately one frame in their buffers. Over the framework these sum

Run Nr	Img proc. delay [ms]	ffm en-coder [ms]	ffm de-coder [ms]	Buffer delay [ms]	Total delay [ms]
1	21	37	2	167	229
2	20	37	2	163	224
3	20	38	3	164	226
4	21	37	2	166	228
5	21	37	2	168	227
6	21	38	2	167	230
7	21	37	3	168	230
8	22	37	3	170	234
9	22	37	3	169	233
10	21	38	2	170	233

Table 1: Observed delay in our test system and network environment. Delays are average numbers over 1000 frames of video streamed in each test run

up to a total of 4-6 frames. At 30 frames per second each have a time impact of 33 ms for a total of the 165 ms which can be seen in Table 1. In comparison to the average buffering delay the delays introduced by decoding and encoding appear insignificant. The total delay through the complete pipeline, beginning with a rendered frame to the frame being shown in the browser, is given in the last column on table 1. While still allowing an interactive experience, much lower values are desired for realtime interaction.

6. CONCLUSION AND FUTURE WORK

We have presented a framework for real-time streaming of free-viewpoint renderings which relies solely on open standards and software. Our approach requires only a standard web browser which has to support the HTML5 video element for streaming the video data. A methodology to calculate the one way delay from the server to the client over a Local Area Network on a per frame basis has been demonstrated with insights into the application performance. In the future further investigation could be done on the buffering delays of the server component and optimizing the application performance in terms of delays to improve the interactive experience.

The recently standardized WebRTC is a promising alternative to the TCP/HTML video streams and we will investigate its application in open source based interactive streaming in the future.

7. ACKNOWLEDGMENTS

The research leading to these results has received funding from the European Union's Seventh Framework Programme FP7/2007-2013 under grant agreement no. 256941, Reality CG, and European Research Council StG 306644 for the project UnIQue.

8. REFERENCES

[1] K. Kapetanakis, S. Panagiotakis, and A. G. Malamos. Html5 and websockets; challenges in network 3d collaboration. In *Proc. 17th Panhellenic Conference on Informatics*, PCI '13, pages 33–38, New York, NY, USA, 2013. ACM.

[2] C. Lipski, F. Klose, and M. Magnor. Correspondence and depth-image based rendering: a hybrid approach for free-viewpoint video. *IEEE Trans. Circuits and Systems for Video Technology*, 24(6):942–951, June 2014.

[3] C. Lipski, C. Linz, K. Berger, A. Sellent, and M. Magnor. Virtual video camera: Image-based viewpoint navigation through space and time. *Computer Graphics Forum*, 29(8):2555–2568, Dec. 2010.

[4] C. Lipski, C. Linz, T. Neumann, M. Wacker, and M. Magnor. High resolution image correspondences for video post-production. *Journal of Virtual Reality and Broadcasting*, 9.2012(8):1–12, Dec. 2012.

[5] P. Merkle, A. Smolic, K. Muller, and T. Wiegand. Multi-view video plus depth representation and coding. In *Image Processing, 2007. ICIP 2007. IEEE International Conference on*, volume 1, pages I–201. IEEE, 2007.

[6] B. Meyer, C. Lipski, B. Scholz, and M. Magnor. Real-time free-viewpoint navigation from compressed multi-video recordings. In *Proc. 3D Data Processing, Visualization and Transmission*, pages 1–6, May 2010.

[7] G. Miller, A. Hilton, and J. Starck. Interactive free-viewpoint video. In *IEE European Conf. on Visual Media Production*, pages 50–59, 2005.

[8] D. Puranik, D. Feiock, and J. Hill. Real-time monitoring using ajax and websockets. In *Engineering of Computer Based Systems (ECBS), 2013 20th IEEE International Conference and Workshops on the*, pages 110–118, April 2013.

[9] A. Rao, Yeon-sup, Lim-Chadi, B. Arnaud, L. D. Towsley, and W. Dabbous. Network characteristics of video streaming traffic. In *Network Experiments and Technologies, ACMCoNEXT 2011 7th International Conference on*, December 2011.

[10] S. Shi. *A low latency remote rendering system for interactive mobile graphics*. PhD thesis, University of Illinois at Urbana-Champaign, 2012.

[11] H.-P. Shiang and M. van der Schaar. A quality-centric tcp-friendly congestion control for multimedia transmission. *Multimedia, IEEE Transactions on*, 14(3):896–909, June 2012.

[12] J. Sutter, K. Sons, and P. Slusallek. Blast: A binary large structured transmission format for the web. In *Proc. 9th International ACM Conference on 3D Web Technologies*, Web3D '14, pages 45–52, New York, NY, USA, 2014. ACM.

[13] X. Xiu, G. Cheung, and J. Liang. Frame structure optimization for interactive multiview video streaming with bounded network delay. In *Image Processing (ICIP), 2011 18th IEEE International Conference on*, pages 593–596, Sept 2011.

[14] G. C. Zhi Liu and Y. Ji. Optimizing distributed source coding for interactive multiview video streaming over lossy networks. *Circuits and Systems for Video Technology, IEEE Transactions on*, 23(10):1781–1794, Oct 2013.

[15] C. L. Zitnick, S. B. Kang, M. Uyttendaele, S. A. J. Winder, and R. Szeliski. High-quality video view interpolation using a layered representation. *ACM Trans. Graph.*, 23(3):600–608, 2004.