Evaluation of Distribution of Panoramic Video Sequences in the eXplorative Television Project

Peer-reviewed author version

QUAX, Peter; ISSARIS, Takis; VANMONTFORT, Wouter & LAMOTTE, Wim (2012) Evaluation of Distribution of Panoramic Video Sequences in the eXplorative Television Project. In: Proceedings of The 22nd ACM Workshop on Network and Operating Systems Support for Digital Audio and Video NOSSDAV 2012.

DOI: 10.1145/2229087.2229100
Handle: http://hdl.handle.net/1942/14265

Evaluation of Distribution of Panoramic Video Sequences in the eXplorative Television Project

Peter Quax Panagiotis Issaris Wouter Vanmontfort Wim Lamotte
Hasselt University / tUL / IBBT
Expertise Center for Digital Media
Wetenschapspark 2, 3590 Diepenbeek, Belgium
{peter.quax, takis.issaris, wouter.vanmontfort, wim.lamotte}@uhasselt.be

ABSTRACT

In this paper, a scalable solution is presented for distributing panoramic video sequences to multiple viewers at high resolution and quality levels. In contrast to traditional broadcast scenarios, panoramic video enables the content consumer to manipulate the camera view direction and viewport size. By segmenting the panoramic input video into a set of separate sequences, transporting them over standard delivery channels and recombining them at end user side, bandwidth utilization is optimized and the quality of the video that is visualized is increased. The proposed solution, called the segmentation approach, is thoroughly explained and evaluated versus a single-stream solution with regards to several metrics, including bandwidth utilization, encoding speed, objective quality levels and seeking performance.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous

Keywords

Distributed Video Coding, Evaluation

1 Introduction and related work

One of the goals of the eXplorative Television project [4] is to design and evaluate an architecture that is capable of distributing and visualizing panoramic video content on relatively low-end hardware, such as-top boxes and tablet devices. At the same time, the solution should be designed in such a way that the existing distribution infrastructure (consisting mainly of clusters of web servers and a complete end-to-end IP delivery chain) can be utilized and that the load on this back-end (in terms of bandwidth and encoding capabilities) is in line with standard HD transmissions. For capturing panoramic or omni-directional video, either off-the-shelf or custom-built hardware can be used. The output of this stage is a sequence of six or more streams that are to be recorded and (possibly in post-production) stitched together to form a single 360-degree view on the action. Users are able to interact with the video sequences through control over a virtual camera, of which the view direction and zoom

NOSSDAV'12, June 7–8, 2012, Toronto, Ontario, Canada.

Figure 1: Example of a panoramic frame generated by the capturing process

factor can be adjusted on the fly. This is fundamentally different from traditional video broadcasts, where a director pre-determines the camera direction and field-of-view. What is specific to the xTV project is that the captured content is to be delivered to the end user in the highest resolution and quality possible, utilizing all camera capabilities available at recording side. To make the solution as generically applicable and future-proof as possible, the architecture was designed to be independent of the codec, container and delivery protocol used.

The European Commission is funding research in its 7th Framework Programme on ICT on panoramic video, e.g. through the FascinatE and FinE projects (no relationship to the results presented here). Traditionally, the technology has been used for the generation of user controllable still image panoramas. Prime examples of this are QuickTime VR [2], MotionVR[6] and Google StreetView. However, the fact that only still images are supported in low resolutions and - for the former two- their non-optimized way of transmitting data (they essentially transport the entire 360 degree panorama to each user, regardless of viewing angle) limits their usage possibilities. Other existing (commercial) solutions utilize a single camera to capture the scene and compress the resulting sequence into a low resolution output stream [5]

The Panocast [9] system and derived PanoMobi solution utilize a spherical video recording as a background to increase the feeling of presence when virtual humans (avatars) are presented to a user. In their solution, the authors capture and process the spherical image to a single image, out of which several viewports are 'cut' at run-time, enhanced with 3D graphics (e.g. the avatar and other objects) and

encoded using standard toolchains. As these stages are to be done separately for each user, the system is only capable of handling about 10-15 users on a single server and the output resolution is low. The authors of [3] propose to use an MPEG-7 description to more efficiently compress and distribute panoramic videos. However, the solution proposed does not take into account real-time video streams, but rather image sequences that are updated from time to time. Besides this, the entire panoramic sequence (including all still images) is transmitted as a JPEG(2000) still, making it non-suitable for distribution to large groups of viewers. Automated stitching of images is also included in the solution, but is out of scope for this discussion.

The authors of [1] present pre-processing optimizations to make H.264 codecs better at handling omni-directional video sequences. By performing image warping and resampling, the images are aligned in such a way that intra/inter prediction becomes more viable. There is however no attention paid to the actual transmission of these streams to end users, therefore putting the paper outside of the focus of the discussion. In [8], a system is presented that is able to select viewports from a panoramic video stream, called the Region of Interest (RoI) by the authors. The system focuses on the efficient detection of the whereabouts of the main actors in a video sequence (e.g. a speaker during a lecture) in both the compressed (using P-frame analysis) and uncompressed domain (through feature tracking). However, the tracking is not under direct control of the end user, but is controlled automatically through the system, thereby enabling the transmission of the same video stream to all viewers. This is an essential difference to the system proposed in this paper.

2 Approach

2.1 High level description

In this section, a novel technique is discussed that lowers the requirements for both the back-end and the end user application and is referred to as the 'segmentation approach' in the remainder of this paper. To make the overall architecture as generically applicable as possible, great care was taken to ensure that the solution can make use of existing codecs, containers and delivery mechanisms. This will, for example, enable delivery to devices that have specific requirements in terms of hardware capabilities (e.g. only a specific codec profile that can be decoded using hardware support) or delivery using existing hardware/software in the back-end (e.g. a commodity set of HTTP servers). As an example, the main configuration used for testing the implementation was a Motorola Xoom tablet with a Tegra2 chipset, using full software decoding with FFmpeg (Baseline Profile), a standard Ubuntu installation with Apache2 on a low-end laptop for the back-end and a 802.11G wireless network. The stages of the pipeline are discussed separately for reasons of clarity, in practice most of these are integrated into a single processing flow to optimize the performance.

2.1.1 Stream preparation

First of all, it should be mentioned that the capturing and processing of the frames is out of the scope of this paper. What is to be delivered as input to the processing mechanism is a set of still images, composed of a stitched projection of multiple cameras. In the demonstration case, such images are generated by a Point Grey Ladybug3 camera [7], but can equally be obtained using a custom-developed setup. Typical resolutions for these input images are around 3840 by 2160 pixels (or 4 times Full-HD). An example is shown in figure 1. Higher resolutions are possible using custom camera setups and have also been tested using the described approach but not included in this paper. Once the images are fed into the pipeline, a mosaic is overlaid; segments are cut out from the original sequence based on the boundaries indicated by the mosaic. While the size of these segments can be varied as desired (n.b. the complete frame height/width divided by the segment size needs to be an integer number) , experiments have shown that a size of about 250 by 200 pixels provides good results for a normal viewport on a panoramic sequence. The final viewports will, not surprisingly, consist of a number of stitched segments. Available in the back-end is a number of encoder instances that are continuously waiting for input from the segmentation process. It is important to note that these encoders can work in parallel, and that the process can easily be distributed across multiple machines, as each segment is independent of the others.

2.1.2 Stream encoding

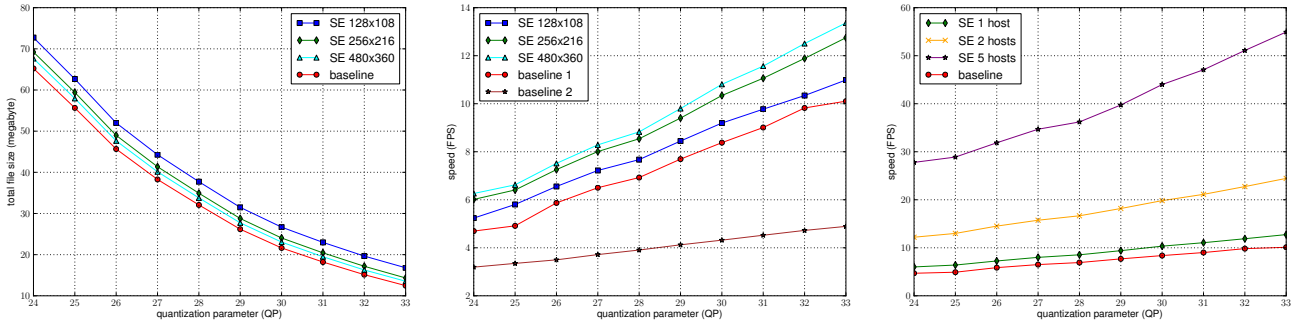
Once delivered to the encoders, each segment is encoded using off-the-shelf technology. As in most cases this content will be viewed on mobile devices such as tablets or on low-end hardware such as set-top-boxes, the codec parameters need to be tweaked for optimal decoding afterwards. Therefore, not all advanced (and often bandwidth-saving) techniques may be utilized. In most cases, multiple versions of the same sequence will be generated using varying codec settings. The resulting encoded frames are encapsulated in a container format for storage. The solution is also capable of splitting the video stream into a sequence of fragments of pre-determined duration (more details in the next section), resulting in output that is fully compliant with the HTTP live streaming specification.

2.1.3 Stream delivery

Although the proposed technique can make use of any delivery mechanism currently available, the focus in this paper is on transmission using HTTP. There are several reasons for this choice: first and foremost the ease through which this can be integrated into existing systems in the back-end (using standard web server technology) and the lack of issues with NAT and firewalls. Existing knowledge on web server clustering can be leveraged to ensure the scalability of such a solution. Another clear benefit is that existing CDN technology can be used to deliver content and HTTP proxies can be used to cache contents, speed up delivery and lower the overall requirements on the web server back-end. Two main mechanisms for delivery over HTTP have been integrated and will be discussed further in the evaluation section of this paper: partial downloads of MKV/MP4 container formats (as implemented in the libavformat toolset) and HTTP live streaming (popular on Apple platforms, but also used in the Android OS). Other container formats, such as NUT and MPEG-TS (standard, not HTTP live encapsulated) have also been investigated, but results are not included in the discussion.

2.1.4 Stream decoding and visualization

At client-side, the software determines the size and location of the (virtual) viewport on the entire omni-directional video sequence. These parameters may be based on the processing capabilities of the devices, but also on network parameters



(a) File size comparison (200 frames) (b) Speed comparison (segment size) (c) Speed comparison (hosts)

Figure 2: Quantification of segmentation overhead

or just the available screen space. It is up to the software to instruct the back-end to deliver the right segments, to decode each of them, make sure that they are exactly synchronized and subsequently visualize them in such a way that boundaries between the segments become invisible. To increase the interactivity of the application, segments at the edges of the viewport may be pre-cached, even if not directly needed for visualization.

3 Implementation and test results

3.1 Baseline for comparison

To obtain a baseline against which to compare the proposed solution, the x264 implementation of the ITU-T H.264 / MPEG-4 AVC codec standard was used to generate an encoding of the entire frame in its original resolution (3840 by 2160 pixels). This is similar to the way in which panoramic sequences are delivered to end users in a previously developed solution; however in that case (for practical reasons) the streams needed to be downsized to 1920x1080 resolution before being encoded (resulting in viewports with very low resolution and quality).

For the baseline condition, the H.264 codec parameters were chosen in such a way that they are representative of what the target hardware is typically capable of in terms of decoding (either through hardware assistance or in software). Although under most conditions, a full frame would not be decodable or deliverable in real time due to its size, the frames should intuitively be more easily and efficiently compressed (vs the proposed segmentation approach) and therefore provides a good baseline to compare against. As test sequence for all conditions, a recording was used that was made at the Arras Main Square Festival 2011, with a Ladybug3 camera located on the front stage.

Performance tests on the back-end (encoding and distribution) were run on a cluster of Dell Poweredge 2970 servers that incorporated a Dual 2.3 GHz Quad-core AMD opteron CPU and 8 GB of memory. To avoid any influence of disk I/O delay (which, as a sidenote, is not to be underestimated but not detailed here due to space constraints), the image sequences were copied to a RAM disk before providing them to the encoding software (where relevant).

3.2 Notes on quality and codec settings

To explain the choices made in the comparisons described below, one should understand that it is not possible to target the codec towards a specific bit rate in the segmentation approach. To do so would result in segments that may vary widely in visual quality. In case they are subsequently stitched back together to form a single high-resolution image, the outcome might look like a patchwork of individual segments rather than a normal composed frame. Therefore, the bitrate is allowed to vary freely, but the quality setting (determined by the Quantification Parameter or QP) is kept at a specific value. As stated in the introduction, one of the main delivery targets are tablet devices, therefore the codec parameters are adjusted to fit those conditions (e.g. not all H.264 features are enabled and a lower profile is chosen, simplifying the decoding process).

Parameter	Value
H.264 Profile	Main
Preset	medium
Tune	fastdecode
GOP size	16
Segment size	256x216
Number of frames	200

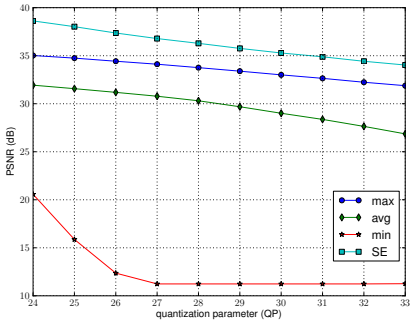
Table 1: Test conditions used in section 3.3

3.3 Overhead associated with segmentation

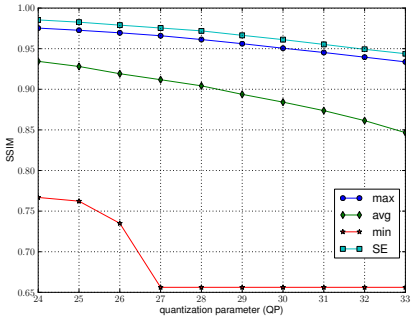
For the results presented in this section, the settings detailed in table 1 were used unless otherwise noted.

When compared to the baseline condition (single encoding of the entire sequence), there will clearly be an overhead factor when cutting the sequence into segments and encoding them individually. Intuitively, this is due to the fact that motion vector ranges are limited to the segment sizes and headers need to be duplicated for each segment. In figure 2(a), the total encoded (stored) file size for 200 frames is shown for various QP settings and for various (representative) segment sizes using the settings as specified in table 1. When calculating the overhead percentage due to segmentation versus the baseline, values of 3 to 8 percent are obtained for large segments and 11 to 28 percent for very small segment sizes. The most realistic conditions are obtained using medium size segments (6 to 14 percent). Unsurprisingly, the overhead is largest when considering very small segments. Note also that for delivery, only a subset of segments is required and not the entire file (as is the case in the baseline condition), so this overhead mainly impacts the storage infrastructure.

An additional overhead might be expected when comparing the speed of the encoding. These results are presented in



(a) PSNR



(b) SSIM

Figure 3: Quality comparison

figure 2(b). In this chart, both baseline data series utilize the same encoder : FFmpeg using libx264. The difference between them is that "baseline 1" represents (full) frames being encoded in parallel using separate threads, while "baseline 2" uses threads to encode slices in parallel (to reduce latency). For both baseline conditions, the FFmpeg codec was allowed to utilize all 8 available cores on the server. It can be seen that even under the simplest condition, the proposed solution provides better results. The segment size (and thus, the total number of segments) is inversely related to the performance. This is mainly due to the fact that the separate codec instances can more easily be distributed over the CPU cores versus the threaded approach in FFmpeg. However, as the number of parallel encodings rises, the overhead due to context switches increases as well. A thread pool as large as the number of available cores is used in the implementation.

As stated before, the segmentation solution also parallelizes easily over multiple hosts, as one can delegate encoding of multiple segments to distinct machines. In case of this test, this was done by subdividing the input sequence into horizontal bands with a height that is a multiple of the individual segment height. As all segments can be independently encoded, these bands are transmitted to other hosts in the cluster in order to be encoded. Results are presented in figure 2(c). Note that a similar approach in case of a single FFmpeg instance would be much harder to achieve, as the encoding depends on information that may or may not be available if the input sequence is split. Using 5 hosts, real-time performance is easily achieved using high quality settings (QP=24) and a segment size of 256 by 216 pixels.

3.4 Quality comparison

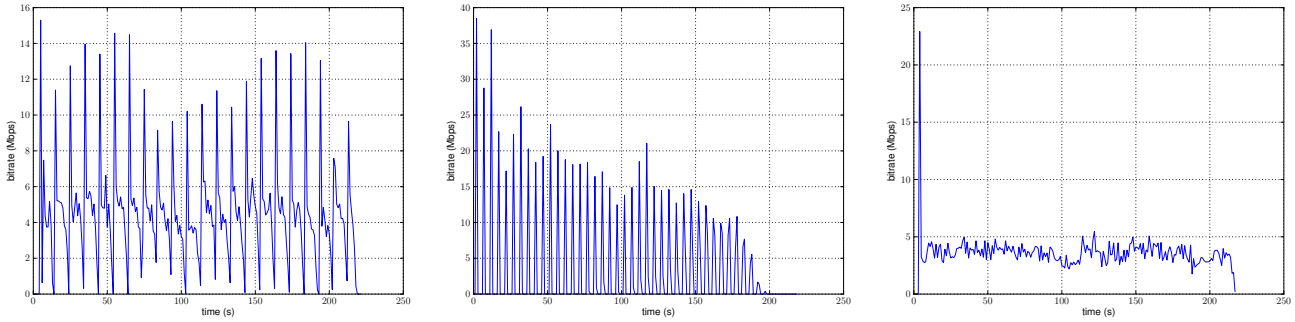
As the goal of the proposed solution is to provide the end users with the highest possible quality under specific conditions (bandwidth, CPU resources, screen space), it is vital to compare the quality level that can be obtained with segmentation versus the baseline. For these tests, bandwidth was chosen as the main criterium, as the primary target for deployment of the application is tablet devices. These devices are practically incapable of decoding the sequence generated by the baseline condition (3840 by 2160 pixels) in real-time, thereby nullifying the usefulness of any comparison based on processing power.

Parameter	Value
Segment size	256 x 216
Horizontal segments in viewport	4
Vertical segments in viewport	3
Number of frames	3454
Quantization parameter (QP)	25
Total sequence resolution	3840 x 2160
Total sequence file size HTTP Live	1012 MB
Total sequence file size MKV	759 MB
Total sequence file size MP4	758 MB

Table 2: Test conditions

To achieve the results in the plot, several conditions were compared. In the charts presented in figure 3, the SE line denotes the PSNR or SSIM value calculated on the segmentation approach. The reader is reminded that for the segmentation approach, the QP is varied instead of using a CBR approach. In practice, this means that the bandwidth will vary over time and will depend on the actual *contents* of the viewport of the user at any given time. An example will be described using frames like figure 1 : under certain conditions (e.g. the user is looking at a piece of the sky), the segments can be compressed to a high degree and the bit rate requirements will be (very) low. In other cases (e.g. looking at the constantly moving crowd of a concert), there is a lot of movement and compression will be relatively poor. As the viewport typically contains multiple segments and the viewport is subject to movement, conditions will average out over time. In table 2, the viewport size is indicated for the test scene in this section.

To compare the quality obtained by the segmentation approach versus the baseline, three bandwidth conditions were chosen: one where compression of segments in the viewport was the highest (cf. the above description, using the least amount of bandwidth, therefore 'min'), one where compression was lowest (highest bandwidth requirements, therefore 'max') and an average condition. The codec used in the baseline condition was instructed to encode the full 3840 by 2160 pixel frames using target bandwidths equal to these values. Note that the x264 codec used in the baseline can do a CBR encoding, as there is no risk of creating patches of different quality levels. It is very important to state again that, when using the segmented approach, only the actual pixels within the viewport need to be transferred (which is just a fraction of the amount of pixels versus the entire frame in the baseline), allowing the codec instances associated with the segments to use more bits per pixel versus the single codec baseline approach. This results in a higher image quality for the segmented approach than can be obtained using the baseline setup under the same bandwidth utilization conditions. The effect is clear both using the PSNR and SSIM metrics (see figures 3(a) and 3(b)).



(a) HTTP Live Streaming (b) MKV over HTTP (c) MP4 over HTTP

Figure 4: Bandwidth comparison

3.5 Bandwidth utilization

To clarify some of the results shown below, it is essential to point out that the bandwidth evaluation was performed on a non-restricted local area network. An abstraction from the access network technology is required (wireless/wired networks and WAN technologies), which is accomplished by provisioning a high capacity back-end link for testing. The only practical limitation is the buffer sizes allocated within the software and the TCP/IP stack limitations (sender and/or receiver buffer sizes). This approach will expose the differences in streaming strategies to the maximal extent possible.

Figure 4 shows results from a test run using the parameters detailed in table 2. Mean bandwidth figures are: 4.5 Mbps for HTTP live streaming (HLS), 3.6 Mbps for MP4 over HTTP and 3.5 Mbps for MKV over HTTP. However, the way in which this bandwidth is consumed is fundamentally different. This is due to the internal workings of the different delivery mechanisms and will be explained in the following paragraphs. Note that the results presented here are different from a generic testing scenario of the streaming technologies, as multiple streams are actually requested at the same time (which is not normally the case).

In figure 4(a), the HTTP live delivery mechanism is used (HLS). This technology requires video sequences to be cut into pieces of equal duration. Most often, this duration is kept short, to eliminate the need for long downloads. This also facilitates seeking, as only the file containing the required frame (which can easily be tracked by matching the timestamp vs the duration) needs to be downloaded. In practice, the sequences in this case are 10 seconds long. The list of files needed for a complete sequence is stored in an index file, which is also placed on the HTTP server. The spikes in the bandwidth chart clearly show this behavior, as every 10 seconds a new file has to be downloaded. Note here that the behavior is amplified by the fact that each 10 seconds, an additional file is required for all 12 segments that make up the viewport. Downloading finishes rather quickly, as buffers are read by the program quite soon after the data arrives (making room for more data to be delivered over TCP).

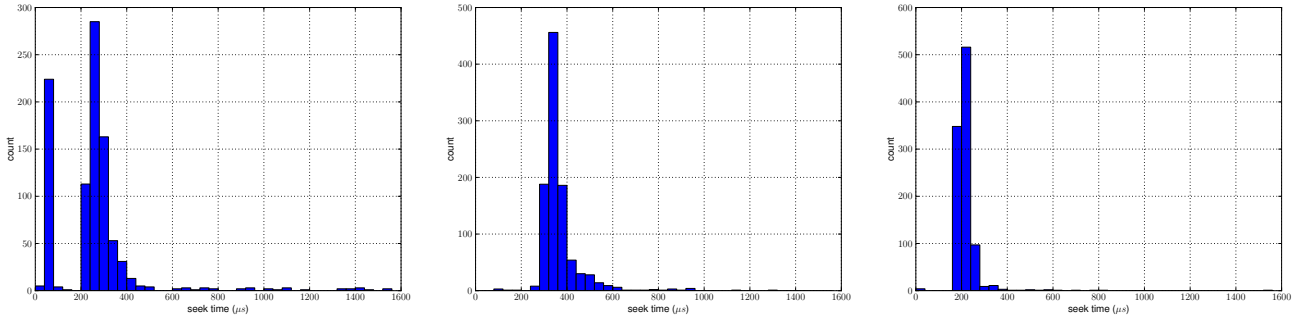
Figure 4(b) shows results from the same test run, using the standard HTTP delivery method and the MKV container. The standard delivery method does not require the sequences to be cut into units of short duration; each segment is contained within a single .mkv file. If needed, the delivery mechanism supports partial HTTP downloads, in order to quickly obtain data e.g. in the middle of the sequence. An example would be if a seek operation to a future time stamp was initiated in the software. For this test case, the content was played back continuously and the feature was therefore not needed. As can be seen, the behavior is quite different from the HLS case. Here, the application requests the entire MKV file for a specific segment and fills its buffer to the maximum. Once enough data is obtained, the buffer is flushed in its entirety to the application, which will start decoding the frames within. Once additional frames are required that are not yet within the buffer, the application performs additional read operations and the buffer will fill up again.

In contrast, the MP4 container, when used with the same delivery mechanism, shows a completely different behavior (see figure 4(c)). After filling the initial buffer, the application will read only as much information as is needed for a single frame to be decoded. The window size is then adjusted accordingly and the TCP throttling mechanism will enable a relatively small amount of data to be delivered. It should come as no surprise that the bandwidth chart here shows a much smoother download characteristic (which may or may not be desired).

3.6 Synchronization and seeking

To ensure perfect synchronization between the segments that make up the final viewport to be rendered on the screen of the end user, exact seeking within each stream is clearly required. FFmpeg, the open source library containing the most extensive set of codec and container format support (used throughout the implementation of the proposed system) provides generic methods for seeking. Unfortunately, some of them are either completely broken, present unexpected or inconsistent results or consistently seek to the wrong moment in time. Additionally, seeking methods are implemented in such a way that they provide a pointer to the closest I-frame - a solution that is fine for general use, but not for frame-precise synchronization as required here.

To cope with these issues, the available seeking methods have been extended to enable single-frame precision. First, a rough seek operation is performed using the standard functionality, after which the exact frame is retrieved by either slowing down (in case a pointer is received to a 'future' I-frame) or speeding up ('past' I-frame) the decoding process. Although it is impossible to directly show the exact frame, at least visualizing something in the same GOP will ensure that the user is not presented with missing picture elements in the



(a) HTTP Live Streaming (HLS) (b) MKV over HTTP (c) MP4 over HTTP

Figure 5: Seeking speed comparison

visualization. By using the described technique, the exact frame can be found and decoded quickly (typically within a few milliseconds after discovery of the reference I-frame). The work presented here also includes an implementation of a seeking method to be used in HLS, something that was not previously possible using standard FFmpeg functionality. This functionality is now part of the official distribution.

Seeking is further sped up by indices in the container formats, facilitating tracking the whereabouts of a specific I-frame within the sequence (vs a complete binary search).

Figure 5 presents histograms showing the seeking speed comparison between the various container formats. A set of 1000 randomly selected time stamps was used within a panoramic video sequence of 3.5 minutes, uniformly distributed over the duration of the file. The application was instructed to do a seek operation to the nearest I-frame. Included in the duration was the time to retrieve the data from the server and demultiplex the video frame where required; decoding of the exact frame is left out of the equation.

Note that for HLS, the separate index file is used that details the different files that make up the entire sequence for a segment (split into 10 second durations). Once the correct fragment is found and downloaded, the generic seeking method is used within the 10 second fragment (direct seeking cannot be performed due to the unavailability of an index in the MPEG-TS container in FFmpeg). For the MKV and MP4 containers, the integrated index is utilized.

The difference in seek times between the various methods may be explained by the internal buffer sizes used by the libavformat library. As figure 4 illustrates, FFmpeg's behavior varies greatly when utilizing different demultiplexers over HTTP. For HLS, if the frame requested is within the current fragment, seeking is near instantaneous. In the other methods, buffer sizes are markedly smaller and a partial HTTP download is needed to first retrieve the requested data.

4 Conclusion and future work

In this paper, the segmentation approach was proposed as a means to deliver panoramic video sequences to end users at a resolution and quality level that is as close to the recorded video material as possible. At the same time, attention was paid to the integration of existing technologies, bandwidth utilization, the performance in the back-end infrastructure (encoding and serving of the sequences) and interactivity levels (i.e. seeking speed). A thorough evaluation has shown that the encoding performance is better than the baseline approach, especially as the process can be easily parallelized. The quality level that can be obtained under a fixed bandwidth condition is increased markedly. Three different delivery mechanisms have been compared in terms of their bandwidth utilization characteristics and seeking performance. While MP4 over HTTP delivers the overall 'best' results, the availability of software in the back-end also has to be taken into account (e.g. re-use of an HLS delivery infrastructure for on-demand video).

Future work consists of a formal subjective quality comparison of various codec settings, as PSNR/SSIM metrics may not yield realistic results for the specific type of video sequences in the panoramic use case. Also, the implementation is to be tested on a larger scale in a living labs testbed.

5 Acknowledgments

Part of this work is funded by the IBBT ICON xTV Project. The authors would like to thank Telenet and Androme NV for creating the end user application.

6 References

[1] I. Bauermann, M. Mielke, and E. Steinbach. H.264 based coding of omni-directional video. In *Computer Vision and Graphics*, volume 32, pages 209–215. Springer, 2006.

[2] S. E. Chen. Quicktime vr: an image-based approach to virtual environment navigation. In *Proceedings of SIGGRAPH'95*, pages 29–38.

[3] A. Glowacz, M. Grega, P. Romaniak, M. Leszczuk, Z. Papir, and I. Pardyka. Compression and distribution of panoramic videos utilising mpeg-7-based image registration. *Springer MTAP*, 40:321–339, 2008.

[4] IBBT xTV Project. `http://www.ibbt.be/en/projects/overview-projects/p/detail/xtv-2`.

[5] Kogeto Dot. http://kogeto.com/dot.php.

[6] MotionVR Technology Corporation. http://www.motionvrworldwide.com/.

[7] PointGrey Ladybug 3. `http://www.ptgrey.com`.

[8] X. Sun, J. Foote, D. Kimber, and B. S. Manjunath. Panoramic video capturing and compressed domain virtual camera control. In *Proceedings of ACM Multimedia 2001*, pages 329–347. ACM.

[9] B. Takacs, A. Beregszaszi, and G. Komaromi-Meszaros. Panocast: A panoramic multicasting system for mobile entertainment. *Information Visualisation, International Conference on*, 0:883–887, 2007.