Performance Evaluation of Scalable Distribution of Omni-Directional Video Sequences using H.264

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

ABSTRACT

In this paper, we present a scalable solution for distributing omni-directional video sequences to multiple viewers using advanced facilities provided by the H.264 codec. In contrast to traditional broadcast scenarios, in the context of omni-directional video, it is the consumer who defines the camera view direction and viewport size. This hints to the fact that a single generic stream will generally not suffice as input for the client viewer application or device, but that a customized stream is required for each user. Transcoding such content on the fly is widely regarded as a non-scalable solution, as will also be demonstrated in this paper. The proposed solution to the scalability problem consists of viewport selection in the compressed domain, reducing the complexity to the selection of specific parts in an existing bit stream. Experimental results are provided that demonstrate the feasibility of the solution and quantify the performance gain over a transcoding system.

Categories and Subject Descriptors

H.3.4 [**Systems and Software**]: Performance evaluation

1. INTRODUCTION AND RELATED WORK

In traditional broadcast scenarios, the director or operator determines the position, orientation and field-of-view of the cameras located within the scene to be captured. Although such a setup clearly works for specific genres (e.g. feature films and television series), there are times it would be beneficial to provide content viewers with the ability to adjust some of these parameters themselves. A prime example is found in sports events, where an omni-directional video camera can be placed in the center of a scene and captures the action in a 360-degree view. Although there is no control over the location of the camera, dynamic viewport selection in terms of vrtual camera orientation and field-of-view is clearly feasible in such a setup. The main advantage of this setup over the fixed camera alternative is that no part of the action will be missed, as there is no way for the camera to be pointing away from the action. Omni-directional video provides both directors (in post-production) and content consumers with the ability to determine the virtual camera direction and field-of-view. Similar examples can be envisaged for various usage scenarios, including game shows (like treasure hunt) and documentaries (e.g. exploration of cultural heritage sites).

Omni-directional camera systems typically produce six or more streams that need to be recorded and (possibly in post-production) stitched together to form a single 360-degree view on the action. In case of HD cameras, it should be obvious that the data rates are extremely high and the resulting video will be a sequence that far exceeds the bandwidth requirements for current HD distribution. However, this is not really an issue as long as the information is only used by the content producers, as dedicated hardware and networks are available to store and transmit the content. Neither are these issues the focus of this paper. Rather, the ways in which the end-user application can actively instruct the source to send only relevant information will be tackled. This will facilitate transmission over existing access networks using reasonable amounts of bandwidth. Findings are supported by experimental results using real implementations of the techniques proposed. Some more context on one of the optimizations presented below (H.264 slice removal) is presented in our earlier work[6]. In this paper, a more elaborate explanation on the technique of slice replacement is provided, along with a comparison of the findings against a baseline solution (using transcoding).

The distribution of omni-directional video sequences for entertainment purposes is currently a hot topic, shown by the fact that the European Commission is funding research in its 7th Framework Programme on ICT on precisely these issues, e.g. through the FascinatE project (no relationship to the ideas and 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] and MotionVR[5]. However, the fact that only still images are supported in low resolutions and their non-optimized way of transmitting data (they essentially transport the entire 360 degree panorama to each user, regardless of viewing angle) put them into a category of their own.

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 at-

tention paid to the actual transmission of these streams to end-users, therefore putting the paper outside of the focus of the discussion. In [7], 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. No codec-specific advanced feature optimizations are used, as the system is limited to the use of MPEG-1 and MPEG-2 codecs. The authors of [3] propose to use MPEG-7 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.

2. APPROACH

2.1 Transcoding content for each viewer

Using the traditional (transcoding) approach, the bit stream required for each viewer is to be generated at run-time. For optimization purposes, one may consider the original content to be stored in a non-compressed format. Because there are a number of factors to consider (i.e. camera direction and viewport size), a separate coder instance is needed for each content consumer. It is up to the encoding process to select the part of the video sequence relevant for each user (in the uncompressed domain). Re-use of already encoded parts of the sequence (caching) would be beneficial, as this may help in lowering the processing load. While it should be intuitively clear that transcoding is not a very scalable solution, it provides a useful baseline to compare optimizations against. As a summary, the main drawbacks of this approach are described below.

In case of parallel runs of the encoding process, each codec instance requires its own memory space and poses requirements on the CPU. Even if dedicated hardware is used, the number of simultaneously active codecs is practically limited by the number of acceleration boards supported on the bus or power limitations. Also, users may be interested in dynamically changing the size of the viewport (virtual camera field-of-view). As this change has an impact on the fundamental properties of the coding context, it requires a full reset of the codec environment, leading to a disruption in the user experience. Caching of generated bit streams is unfortunately not easily accomplished due to the variety of factors that can be changed by individual viewers. Combinations of different time indexes, viewport sizes and camera directions would quickly lead to a cache explosion. Also, the resulting compressed video stream can only be manipulated as a single entity. An entirely new and custom video stream needs to be encoded for each viewport size/camera direction combination that is requested. In section 3.2, experimental results are presented that quantify the scalability of such a solution under optimal conditions.

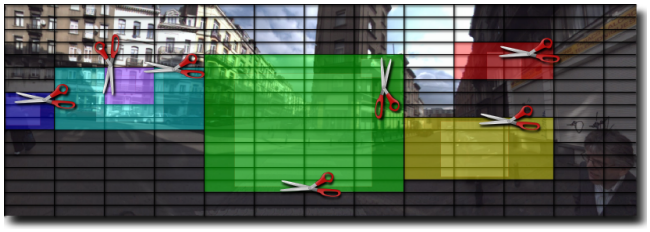


Figure 1: Viewport selection using rectangular grid pattern

2.2 Using H.264 slices

In this section, two alternative techniques are discussed that make use of H.264 features: slice removal and slice replacement. Although a more detailed discussion of the former is presented in our earlier work[6], a short summary is included to introduce the basic concepts to the reader. The latter (slice replacement) has been optimized when compared to our previous work and is described in more detail in this paper, as it is also the subject of the experiments in section 3.

Typically, a video frame is segmented into macroblocks, which are the fundamental units used in the compression and decompression stages. Similarities between these macroblocks are exploited to reduce the amount of information that needs to be retained when considering subsequent frames within a sequence. One of the features of the H.264 specification[4] (also known as MPEG4/AVC) that is essential to this work are the so-called 'slices', which allow macroblocks to be grouped into independently decodable units. Without them, the entire frame is the minimal encodable or decodable unit. Using slices, frames are subdivided into smaller regions that are under individual control of the codec. This is a useful feature on several levels: on the one hand, it enables the parallel decoding of several parts of the video frame; on the other hand, it enables decoding of parts of the video frame even when bit stream errors are appearing.

A rectangular grid is superimposed on the frames, consisting of several slices. In the solution described here, each slice is limited to a horizontal sequence of macro blocks (reason discussed below). The composition of this grid structure does not vary over time (compared to a generically encoded H.264 stream, in which this is possible). To be able to 'cut' a specific section out of the video sequence, a selection of multiple slices is required, based on the grid structure. Because the bits corresponding to this viewport region can be traced back to distinct slices (thanks to the regularity of the grid), manipulations can take place entirely within the compressed domain and are composed of cheap operations in terms of computing resources. H.264 includes a feature called FMO (Flexible Macroblock Ordering), which enables non-consecutive macroblocks to be allocated to an independently decodable slice group. This would allow the grid to be completely arbitrarily defined and not limited to the height of a single macroblock. Although this would be an ideal solution for the case presented here, there are no provisions for this feature in currently available real-time capable

decoders (needed for visualization). The workaround presented clearly introduces additional overhead (e.g. because of the large number of slices being defined), but is deployable in practice due to the fact that there is support for all required techniques in freely available codecs.

To visualize the concepts, figure 1 shows a panoramic video stream of which several clients request an individual view. Slices are represented by the rectangular areas. Note that both viewport size and location may be different for each client.

2.2.1 Slice removal

Using the approach described above, one might be tempted to just remove information about all unnecessary slices for each client and compose a new bit stream using the remaining information. However, such a solution does not work in practice, as decoders are unable to deal with such a degraded bit stream - additional control information is required to generate a valid stream for each client. Even when combining the remaining slices into a bit stream that is valid according to the specifications, many issues remain that make this solution very cumbersome to implement. For a more detailed discussion, the reader is referred to [6], which provides a detailed description.

2.2.2 Slice replacement

A different solution, which alleviates the aforementioned issues, uses replacement rather than removal of slices. Slices that would have been removed in the previous solution are instead replaced by an artificially generated slice of minimal size. This replacement slice consists of the same amount of macroblocks as the original slice it replaces, thereby filling in the gap that the removed slice would create. As the actual visual content of these replacement macroblocks is irrelevant, the goal is to try and find the smallest possible sequence of bits which represents a sequence of macroblocks while adhering to the H.264 specification. Note that the visual appearance is irrelevant as these macroblocks will not be shown in the client's viewport.

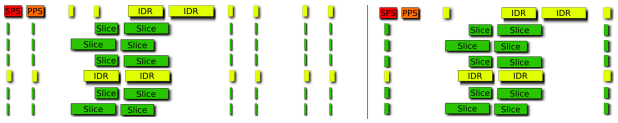
By analyzing the H.264 specifications bit stream syntax, it can be determined that for instantaneous decoder refresh (IDR) frames the smallest possible representation of one or more macroblock results in a corresponding number of plain gray macroblocks. As can be seen in table 1, the smallest possible representation of an intra-coded slice containing only the first macroblock of a frame, requires 22 bits. Due to the NAL unit header and startcode, 32 bits are added. As NAL units are byte-sized, two more bits need to be added resulting in 56 bits, or 7 bytes.

This is the smallest possible intra-coded slice, actual replacement slices will be larger as the first_mb_in_slice field value will be larger for every slice not located at the beginning of the frame. Furthermore, the frame_num value will be larger due to the chosen group of picture (GOP) size. A more realistic example of a typical empty IDR-slice, for a GOP size of 16, containing 32 macroblocks starting from macroblock 200, would still occupy only 40 bytes (36 bytes of actual slice content).

Similarly, the smallest possible representation of one or more macroblocks for inter-coded slices needs to be determined. Again, the visual result of decoding these bits is irrelevant, so there is an open choice as to which sequence of bits are legal according to the specification. The skip-bit

name	length (bits)
startcode	24
NAL_unit_header	8
first_mb_in_slice	1
slice_type	7
pic_parameter_set_id	1
frame_num	1
idr_pic_id	1
no_output_of_prior_pics_flag	1
long_term_reference_flag	1
slice_qp_delta	1
mb_type	5
intra_chrome_pred_mode	1
mb_qp_delta	1
lum16DC	1
total	54

Table 1: Intra-coded slice representation



(a) non-grouped (b) grouped

Figure 2: Slice replacement concept

feature of H.264 inter-coded slices requires only one bit per macroblock and is essentially a signal to the decoder that the currently to be decoded macroblock bit stream data can be skipped as it is identical to the corresponding macroblock in the previous frame. This is also illustrated in Figure 2(a).

As can be seen in table 2, the smallest possible representation of an inter coded slice containing only the first macroblock of a frame, requires 17 bits. Due to the NAL unit header and startcode, 32 bits are added, and as NAL units are byte-sized, seven bits will be added resulting in 56 bits, or 7 bytes. As with intra-coded slices, this is the smallest possible inter-coded slice; actual replacement slices will be larger as the first_mb_in_slice field value will be larger for every slice not located at the beginning of the frame. Finally, the macroblocks field will be longer depending on the number of macroblocks that need to be skipped. A more realistic example of a typical empty P-slice, for a GOP size of 16, containing 32 macroblocks starting from macroblock 200, would still occupy only 10 bytes (6 bytes of actual slice content).

Nevertheless, replacing each individual unneeded slice with a generated slice is still suboptimal and results in high overhead of headers. Especially if each slice would be sent individually, which would add more overhead due to RTP, UDP, IP and data link layer headers. To this end, the bit stream is restructured by generating slices containing spans of macroblocks from one unaltered slice to the next unaltered slice (see figure 2(b)). This technique can be applied to both IDR and P-frames.

name	length (bits)
startcode	24
NAL_unit_header	8
first_mb_in_slice	1
slice_type	7
pic_parameter_set_id	1
frame_num	1
num_ref_idx_active_override_flag	1
num_ref_idx_l0_active_minus1	1
ref_pic_list_modification_flag_l0	1
adaptive_ref_pic_marking_mode_flag_l0	1
slice_qp_delta	1
macroblocks_	5
stop_bit_	1
total	49

Table 2: Inter-coded slice representation

As slices are replaced instead of removing them, the structure and size of the frames remain constant, as do the positions of the slices and macroblocks. As the positions of the slices are unaltered, the first field in the slice header first_mb_in_slice needs no modification, implying that no bits need to be shifted and the entire slice can be sent unaltered, reducing CPU load and improving scalability. In contrast with the removal technique, the PPS and SPS header now require no modification, although the infrequency of their occurrence makes this advantage rather insignificant.

Instead of always generating the replacement slices on the fly, under certain circumstances, a caching mechanism can be used. The size of the cache is determined by the generated slice size (which has been shown to be rather small) and the number of entries in the cache. The number of entries in the cache depends on the cache cleanup strategy and the number of different slices that can be generated. The generation of slices depends on a limited number of parameters of which frame_num and first_mb_in_slice are the most relevant. The number of different frame_num values depends on the GOP size, and the number of different first_mb_in_slice values depends on the slice structure. As an example, for a GOP size of 16 and a regular grid-structured slice allocation of 8 by 32 slices, $16 * 8 * 32 = 4096$ different slices could be generated (if the slice regrouping feature would have been disabled). Combined with the knowledge that the generated slices are small, caching the generated slices is certainly feasible.

It can be concluded that this approach is very close to the space-efficiency-optimal case (when using the slice removal technique), but without the computational overhead and issues regarding the renumbering of slices and blocks. An additional advantage of this approach is that there is no longer a restriction to sending rectangular areas, which can be useful when the full frame contains an unwrapped cube map out of which one wants to cut the visible areas. In this case, a more or less circular or elliptical viewport might be better suited.

An issue that is still apparent with both approaches is the fact that the macroblocks to which motion vectors point may be either unavailable (with the first approach) or contain incorrect data (with the replacement approach). The problem occurs when the viewport moves when the client is receiving inter-coded pictures. The newly received inter-coded picture might refer to macroblocks in regions which have been replaced by flat gray macroblocks, resulting in incorrect motion compensation. A workaround for this issue consists of using a larger than strictly needed cropping area, combined with a limit on the viewport movement speed. This workaround is implemented in the test results detailed below in section 3.3. Also, to enable motion estimation in a reasonable part of the video sequence, the vertical resolution of the part of the image transmitted is higher than the actual viewport size. This enables the codec to search vertically for matching blocks (motion estimation) in a meaningful way, thereby reducing the need for extensive motion compensation. The same is true for the horizontal direction, allowing movements without immediate need for new reference information as indicated above. Two other disadvantages of this approach are the larger decoded picture buffer size at the decoder side and the need for the decoder to be able to handle a larger number of slices. However, tests have shown that most available decoders are capable of handling these conditions.

3. TEST RESULTS

It was already mentioned in section 2 that two viable approaches are envisaged that can be used in real-life scenarios: either transcode the content for each viewer or use the slice replacement technique. It was also already mentioned that the former is intuitively non-scalable, a fact that will be supported by figures in this section. An important remark to be made before looking at the charts is that the absolute numbers are not of primary importance. Rather, the trends that emerge are. This is due to the fact that performance of codecs is dependent on the available CPU resources, which may vary widely between hardware setups.

3.1 Test setup description

Instead of relying on mathematical models or extrapolations of small scale tests, the experiments use real codec implementations and are an accurate depiction of what is feasible under real-life conditions. The overall system setup is a cluster of low-end hardware, interconnected through a dedicated gigabit Ethernet LAN. Each node in the cluster is a Core2Duo E4600 system, running at 2.4GHz. As already indicated in the introduction, the pre-processing steps, where the camera images are warped and stitched together into a single panorama are integrated in external software. For practical reasons (preliminary visualization is based on a cylindrical projection), the sample data consists of a panoramic sequence instead of a truly omnidirectional stream. For the experiments, two input streams are used: one that is uncompressed and consists of raw YUV420 data (total resolution: 1536 by 1024, length 500 frames) and a compressed sequence (for the slice replacement technique). Although in practice resolutions would be significantly higher, the computing power of the hardware used would not yield representative information for the real-time transcoding benchmark under those conditions. It is important to note that the relative performance gain between the two approaches is the most important. The compressed sequence is constructed using an altered version of the H.264 JM Reference Codec, as it is the only (freely available and adaptable) one capable of generating streams that make use of the advanced features needed. Although JM does not encode in real-time, this is not a major issue as the streams can be generated in a pre-processing stage.

3.2 Transcoding approach

It was explained in section 2.1 that a transcoding server implementation would need to crop out the requested area for each client, encode it in a custom bit stream according to the specs and transfer it to that specific client. In this test setup, a simplified setup of the overall system is used, to ensure that the results are not skewed due to external factors (e.g. hardware/software interrupt overload by sending lots of traffic using the OS network stack and internal NIC) and to provide a best-case scenario to compare against. This way, the server is dedicated only to the generation of appropriate bit streams.

It should be noted that in this approach, the system could in essence choose from a variety of codecs to generate the compressed streams. However, as H.264 is currently the video codec with the broadest support for a wide range of resolutions, it is an obvious choice (also for comparative

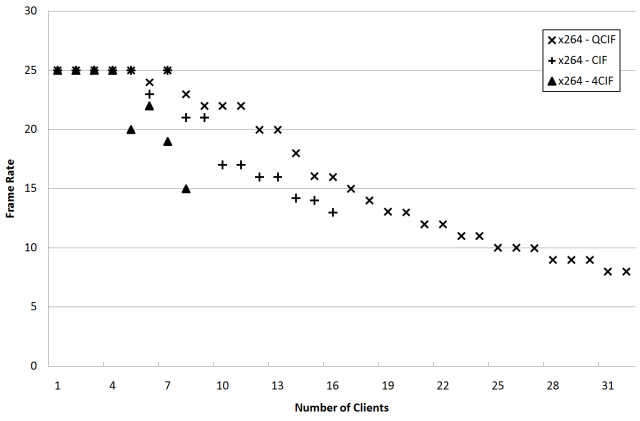


Figure 3: Performance of transcoding approach

reasons). The encoder of the popular x264 project is used (without alterations) for its performance and free availability. Video is stored on the server in uncompressed YUV420 format, so the encoding stage is the most important contributor to overall CPU resource usage. To get the highest performance out of the codec (in terms of frame encoding speed), the 'ultrafast' preset is used. This preset disables most advanced video coding features, including but not limited to: CABAC, B-frames, subpixel motion estimation, weighted prediction, macroblock partitioning, multiple reference frames and the in-loop deblocking filter. Although one could argue that the size of the generated bit streams is a factor larger than optimal, this is not really relevant as network transmission is not considered part of the test setup. On the server, several instances of x264 are concurrently actively cropping and encoding a number of streams. This parallelization can be exploited on multi-core systems, because each instance is running independently of the others. During the test runs, the encoding speed of each instance was recorded. Tests were repeated using an increasing number of parallel encoders, which facilitates the comparison of this transcoding approach to the slice replacement technique afterwards (using an identical number of clients). The tests were repeated for several commonly used resolutions (output video streams in CIF, QCIF and 4CIF) to provide appropriate estimates on the number of parallel encodings possible when targeting a specific device/resolution combination.

The encoders are using a file containing raw YUV420 pixel data as input (entire frames), out of which the requested viewport is cropped for each client (at different time indexes). This process involves a lot of disk seeking, but the only alternative – keeping the data in main memory –, is not feasible for raw video. Solid state disks might alleviate some of the overhead associated with disk seeking, although they suffer from a high economical cost. Another alternative solution to the disk seeking issues might be to use a compressed bit stream as the source video. However, this approach is non-scalable in itself, as an equal ratio of decoders to encoders would be required, due to the fact that clients are looking at the video sequence on distinct time indexes. Combining the decoding with the encoding steps would lead to even worse results than those presented here (in terms of CPU load) and would outweigh any possible advantages over the uncompressed storage scenario with regards to disk seek times. Overall, it should be noted that due to the large size of the raw input source video, scaling the number of clients leads to massive seeking overhead.

One could argue that a true integrated implementation of the above would improve the results, because of the possibility of reusing resources and sharing knowledge. On the other hand, on modern operating systems and hard drives, hard disk caches (in hardware) are already available. As streaming video clients access frames in a consecutive order, a custom cache implementation would not be able to take better decisions. Also, these hardware caches are small compared to the bit stream sizes. The chance for an effective cache hit is significantly lowered, as clients access the video at different time indexes. Regarding the CPU intensive parts, motion estimation data cannot be shared, as the individual encodings use different viewports.

Results of the tests are presented in figure 3. As can be seen, even for small output resolutions, scalability is very poor. This is even more apparent when considering larger viewport sizes. Although this approach is well-suited for parallelization, there is always a practical limit to the number of processing cores that any single machine may contain. For smooth video handling, a frame rate of 25 frames per second is required. This value will be used in benchmarking the optimization against the transcoding approach. Using the hardware described above, the setup is able to support a disappointing 6 clients at QCIF resolution or 4 clients using 4CIF, not taking into account any overhead induced by the network transmission (left out as explained before to create a best-case scenario for that approach).

3.3 Slice replacement approach

For the benchmark of this approach, a cluster of thirteen identical nodes is used. One node was used for running the server implementation, the others for the clients. The system is set up in such a way that the server selects the appropriate viewports from the (pre-computed) compressed bit stream. While the overall video sequence is the same for all clients, they each start requesting the video sequence on a distinct time index due to the delay in starting up the required number of processes on twelve machines. There is a communication channel (out of band) between the clients and the server, over which the information about the viewport location and dimension is exchanged. To make the test setup more realistic, the clients continuously move their viewport in a horizontal pattern, forcing the server to make separate selections for each connected client. Network overhead is now included in the frame rate calculations. The latter puts the slice replacement approach in a disadvantaged position, but helps to perform a best/worst case scenario comparison.

The test scenario is repeated using an increasing number of clients, evenly spread over the twelve cluster nodes. During each run, several factors are recorded. These include the frame rate at which the server is able to generate bit streams and the required bit rate for each client. The videos used to benchmark the approach were encoded using a modified version of the reference H.264 encoder. The video was encoded off-line with a grid-structured slice layout of 8 by 64 slices.

We repeated the tests using several resolutions, all the while measuring the performance. For the test cases presented here, the vertical resolution is always set to 512 (re-

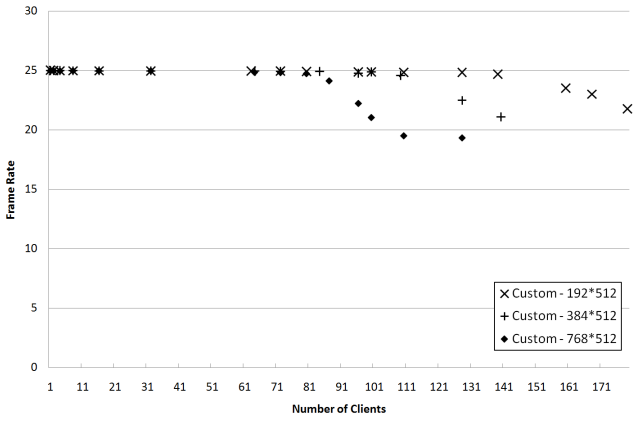


Figure 4: Slice replacement performance

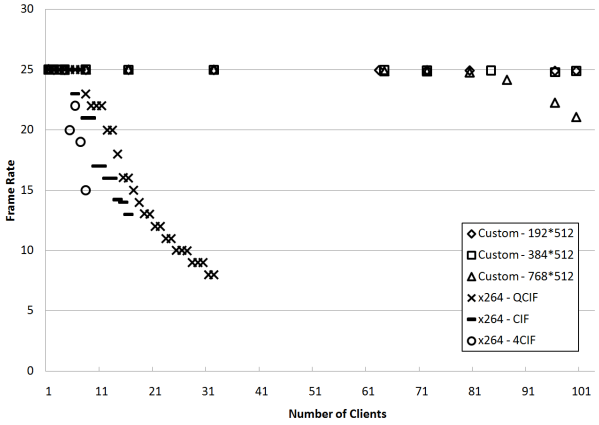


Figure 5: Performance comparison

call that the entire video sequence is composed of 1536 by 1024 pixels). This is of course just a ballpark figure (chosen rather large in this case, again as a worst-case scenario), and may be tweaked as required. In the horizontal direction, the values chosen are 192, 384 and 512. This enables the client software to visualize a viewport that is smaller than these dimensions (e.g. QCIF or CIF), while at the same time enabling (small) movements of the viewport without requiring additional slices from the server (both horizontally and vertically, although the latter is not exploited in these test runs).

Results are shown in figure 4. This time, the system can support about 130 simultaneous clients at 192x512 or about 80 at 768x512 resolution (enabling visualization crops to QCIF or CIF) using the same hardware setup as in the previous test. This results in a factor 15 to 20 increase over the transcoding approach (which did not include network overhead). As the system can easily be load-balanced (clearly needed for network transmission) and optimized in terms of multi-threading code, this shows that the setup is truly scalable towards practical user numbers. A more detailed chart is provided in figure 5, showing the drop-off point (below 25fps) for the most important resolutions. The transcoding approach is denoted by the x264 prefix, while the slice replacement approach is indicated by the 'custom' tags in the chart legend.

4. CONCLUSIONS AND FUTURE WORK

In this paper, an experimental validation was conducted on a novel way to distribute high quality omni-directional video streams to large numbers of users. By introducing a regular grid structure – superimposed on the video sequence – reference points are available that enable relatively low-cost cut/paste operations on the bit stream in the compressed domain. To regenerate a valid stream, non-required slices are replaced by gray values, which can be encoded in an efficient way and reduce the problems associated with the complete removal of this information. Tests were performed using implementations of both the transcoding and slice replacement techniques, which lead to the conclusion that a performance increase of at least a factor 15 is easily obtained.

When used for real-time events, the encoding part of the workflow is currently a stumbling block. However, alterations in the x264 codec (e.g. limiting the motion estimation to specific regions) are being investigated that would enable the use of this implementation on the encoding side (enabling near real-time processing). The goal of the xTV project is to deliver omni-directional streams to end-users through IP set top boxes. It is essential that the generated bit streams are standards compliant, to make sure that they can be decoded using hardware acceleration. Keeping to the standard and adapting to the quirkiness of these hardware decoders is an important challenge that weighs on every decision to be made for the practical implementation.

5. ACKNOWLEDGMENTS

Part of this work is funded by the IBBT xTV and IBBT ISBO Immersive Projects.

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 of *Computational Imaging and Vision*, 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, New York, NY, USA, 1995. ACM.

[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. *Multimedia Tools and Applications*, 40:321–339, 2008.

[4] H.264 : Advanced video coding for generic audiovisual services. World Wide Web, `http://www.itu.int/rec/T-REC-H.264`, 2009.

[5] MotionVR Technology Corporation. World Wide Web, `http://www.motionvrworldwide.com/`.

[6] P. Quax, F. Di Fiore, P. Issaris, W. Lamotte, and F. Van Reeth. Practical and scalable transmission of segmented video sequences to multiple players using h.264. In *Motion in Games 2009 (MIG09)*, LNCS series 5884, pages 256–267, 2009.

[7] 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.