Interactive Zoom and Panning from Live Panoramic Video

Vamsidhar Reddy Gaddam[1], Ragnar Langseth[1], Sigurd Ljødal[1], Pierre Gurdjos[2], Vincent Charvillat[2], Carsten Griwodz[1], Pål Halvorsen[1]

[1]Simula Research Laboratory & University of Oslo, Norway
[2]Universite de Toulouse, France

[1]{vamsidhg, ragnarla, sigurdlj, griff, paalh}@ifi.uio.no
[2]{pgurdjos, vincent.charvillat}@enseeiht.fr

ABSTRACT

Panorama video is becoming increasingly popular, and we present an end-to-end real-time system to interactively zoom and pan into high-resolution panoramic videos. Compared to existing systems using perspective panoramas with cropping, our approach creates a cylindrical panorama. Here, the perspective is corrected in real-time, and the result is a better and more natural zoom. Our experimental results also indicate that such zoomed virtual views can be generated far below the frame-rate threshold. Taking into account recent trends in device development, our approach should be able to scale to a large number of concurrent users in the near future.

Keywords

panorama video, zoom, panning, real-time

1. INTRODUCTION

The growing availability of high-speed Internet access has gone along with a growth in interactive and immersive multimedia applications, and panoramic video is a feature that is becoming more and more popular in various scenarios for its ability to increase immersion. We look at it in the context of arena sports like soccer, which have always provided opportunities for innovation in broadcasting. The challenges in providing interactive experiences in broadcasting to a large audience span several fields. One such challenge is to provide a real-time pannable and zoomable virtual camera to several thousands or even millions of users.

This type of challenges has attracted the attention of several researchers [10, 12, 3, 5, 8, 4, 13]. For example, Carr *et al.* [5] demonstrated recently a hybrid robotic/virtual camera for event recording. It uses a robotic camera to follow features and assist a virtual camera. Such a system can provide a smooth and aesthetic camera, but it is restricted to *one* virtual camera per a robotic camera. So, in case of a multi-user system, all clients receive the same video. Furthermore, Mavlankar *et al.* [8] describe a pan-tilt-zoom streaming system, but the system presented merely crops from a larger video. For a large space like a soccer stadium, such a system introduces widely varying quality from one end to the other end (this was also the case in our earlier approach [9]). If the perspective nature is maintained (straight lines remain straight) in the panorama, a large amount of redundant data is transferred on the close end.

To provide a better perceived quality when zooming and panning into a panorama video and keep redundant data transfer low, we present a system that uses a cylindrical panorama as an intermediate representation. The approach followed by us to create the intermediate representation is one of the numerous choices available. One can use a reflective sphere and a high resolution camera to capture the panoramic texture video, but the 4K/8K cameras are not yet available at reasonable prices. Then, a virtual view is generated where the perspective of the delivered video is corrected before it is presented to the client. The client has full freedom to pan, tilt and zoom using the system, and the system supports several algorithms for pixel interpolation. Furthermore, in order to support a large number of concurrent users, we aim for a lightweight system to generate the views in real-time. Therefore, we have implemented several versions running on both CPUs and GPUs, and we discuss the system's ability to scale to a large number of users – both to spectators within a stadium as well as users outside.

Furthermore, we generate a virtual view frame in about 10 ms on a GPU, and our system supports real-time interactions. If processed on the server side, we support a limited number of users per machine. However, taking into account existing work to transfer panorama video in real-time [6] and the fact that GPUs are becoming a commodity also on mobile phones, we work on an adaptation of our system that creates the virtual camera on the client side. This system will trade bandwidth for server processing power by broadcasting the panorama video to all clients instead of computing and transmitting a personalized view for each client.

The rest of our paper is organized as follows: Section 2 briefly describes the example soccer scenario with our real-world installation. In section 3, we describe our system, and we present our experiments and evaluation results in section 4. Section 5 provides some discussions before we conclude the paper in section 6.

2. EXAMPLE SCENARIO

Our prototype is currently installed at Alfheim, a soccer

NOSSDAV'14, March 19 – 21 2014, Singapore, Singapore
Copyright 2014 ACM 978-1-4503-2706-0/14/03 ...$15.00.
http://dx.doi.org/10.1145/2578260.2578264

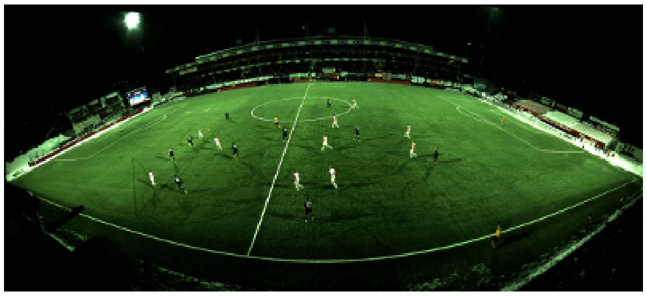


Figure 1: Generated panorama video.

stadium in Tromsø, Norway. In this scenario, we use the panorama video in a sport analysis system [7, 9] where video events must be generated in real-time. The camera array is integrated with a player-position sensor system and an expert annotation system. The integrated system enables users to search for events, follow one or more players in the video, and automatically query for video summaries.

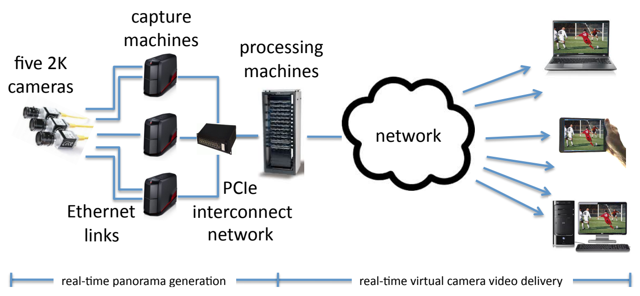
The scenario poses an array of challenges. To stitch the individual cameras into a panorama video (see figure 1), the camera shutters must be synchronized. This was accomplished by building a custom trigger box[1]. The cameras must again be synchronized with the sensor system to correctly identify the corresponding video frames and sensor records [7]. Furthermore, the high-rate video streams must be transferred and processed in real-time [11].

Even though the system was originally meant for the coaches, a lot of this functionality is also interesting for outside users. Our goal is to enable spectators in the stadium and supporters at home to use a part of the system with individual videos. This means that every user must be able to interact with the system in real-time to generate their own personal view (e.g. generating a virtual camera following the ball) from a single viewpoint.

Until recently, our virtual views were limited to cropping and scaling. For a better user experience, we need perspective correction, and to scale the system up to thousands of spectators in a stadium, we need a scalable way of generating and delivering the individual views.

3. SYSTEM OVERVIEW

Our system is divided into a panorama generation part and a video delivery part, which supports user-controlled interactive virtual cameras. A sketch of our system is given in figure 2. In this section, we describe these two parts.


capture machines
five 2K cameras
processing machines
network
Ethernet links
PCIe interconnect network
real-time panorama generation
real-time virtual camera video delivery
Figure 2: Setup.

[1]Hardware design and software open sourced at `https://bitbucket.org/mpg_code/micro-trigger-box`

3.1 Real-time Panorama Video Capture

As we have earlier described the real-time version of the panorama video pipeline [11], we only highlight the changes made to increase the resolution (better and more cameras) and to distribute the capture and processing.

The capture pipeline (left-most part of figure 2) consists of several sub-components. First, we have a *camera reader* which fetches raw video frames from the (frame-synchronized) cameras over a point-to-point Ethernet network. Depending on the chosen output format from the cameras, we have a *debayering* component interpolating the frames from a Bayer pattern to full YUV 4:2:2 used later in the pipeline. To move data to the panorama processing machine, we use a *Dolphin* high-speed interconnect component transferring the uncompressed video data from the recording machines to the single processing machine. On the processing machine, we first have a *frame synchronizer*, which retrieves a full set of synchronized frames from the separate camera streams. It delivers each set of frames to the *panorama stitcher*, which generates cylindrical panorama frames. Finally, we have a *video encoder*, which encodes the panorama video in H.264 for immediate delivery to clients and for storage for on-demand operations.

3.1.1 Video Capture

To capture the entire soccer field, we use five Basler industry vision cameras [2], each of which delivers a maximum resolution of 2046×1086 pixels at 50 frames per second over Gigabit Ethernet. We use an 8mm lens [1] with virtually no image distortion, allowing us to bypass the lossy debarreling step of our previous pipeline [11]. To maximize the panorama resolution, the cameras are rotated by 90° (see figure 3), giving a vertical field-of-view (fov) of 66°. Furthermore, the cameras are mounted in a circular pattern, i.e., pitched, yawed and rolled to look directly through a point 5cm in front of the lenses, in an attempt to reduce parallax effects. As a result of this, only minor adjustments are required before the images can be stitched together. The video capture system also determines the required exposure, which requires frequent changes due to quickly changing light conditions outdoors. Auto-exposure is performed on the center camera once every 4 seconds , and the camera reader module broadcasts the resulting exposure parameters to the other camera readers.

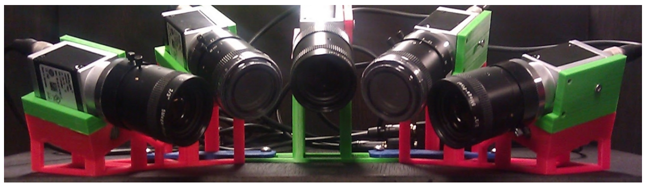


Figure 3: Mounted camera array

3.1.2 Distributed Processing

The new setup requires many more resources (particularly bandwidth) per camera, and we therefore had to distribute capturing as shown in figure 2. To transfer the video streams to the panorama processing machine, a single gigabit Ethernet network is insufficient. We therefore use Dolphin Interconnect cards, which use PCIe as the interconnect protocol and allow for ultra-low latency Direct Memory Access (DMA) across several machines. We transfer each frame with a single DMA transfer at a rate of 19.41 Gbps and a total latency of 1.26 ms per frame.

3.1.3 *Cylindrical Projections*

Our new pipeline generates a cylindrical stitched panorama as shown in figure 1. The idea is to have the cameras in the center of a virtual cylinder where each source image can be considered a (cropped) plane that is tangential to the cylinder and orthogonal to its camera's viewing axis. Each pixel of the cylinder is then computed as the (interpolated) pixel value of the ray from the camera center through the pixel intersecting the image plane.

The radius (r) of the cylinder is determined by the width (W_s) of the source images and its field of view (fov):

$$r = \frac{W_s}{2 * tan(\frac{fov}{2})} \tag{1}$$

The viewing axes of the cameras form a plane L (orthogonal to the rotation axis of the cylinder). The angle between the viewing axes of two neighbouring cameras is $\approx 28.3°$, with the 0°angle assigned to the center camera. For brevity, we say also that the angle α of a camera is the angle α of its corresponding source image. The unrolled cylinder forms a Cartesian coordinate system, where $(0, 0)$ corresponds to the intersection of the center camera's viewing axis with the cylinder and the X axis corresponds to the intersection of L and the cylinder.

Every pixel coordinate (T_x, T_y) on the unrolled cylinder determines the corresponding horizontal (θ) and vertical (ϕ) angles of a ray from the camera center through this coordinate.

$$\theta = \frac{T_x}{r} \quad \text{and} \quad \phi = arctan\left(\frac{T_y}{r}\right) \tag{2}$$

To determine the pixel(s) in the source image for every pixel (T_x, T_y), the source image with the closest α value to θ is selected and α is subtracted, essentially centering the coordinate system on that camera's viewing axis. Then, the point x', y', z' in 3D space where the ray intersects the image plane is determined by:

$$z' = r \quad \text{and} \quad x' = tan(\theta) * z' \tag{3}$$

$$y' = tan(\phi) * \sqrt{z'^2 + x'^2} \tag{4}$$

This relationship is visualized in figure 4.

This algorithm requires each image to be perfectly aligned and rotated. The camera mount seen in figure 3 provides a

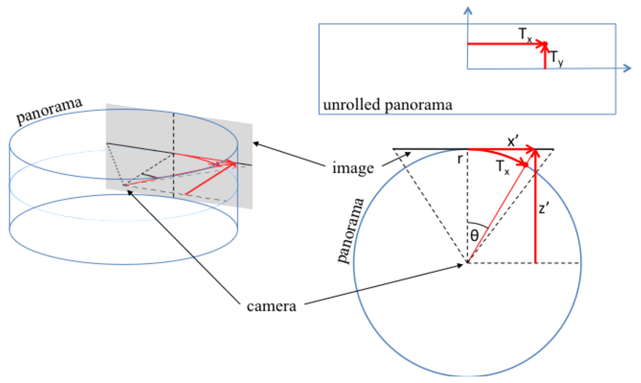


Figure 4: Creating panorama pixels from a captured image

good alignment, but there are small deviations that must be corrected. A small per-camera vertical correction is easily applied with no additional complexity. For each source image, a standard rotational matrix is created to compensate for rotation around the camera's x, y and z axis.

The Cartesian coordinates of the pixels on the cylinder are multiplied with this rotational matrix, rotating the cylinder before we project onto it. Then we can find θ and ϕ based on the new coordinates:

$$\theta = arctan\left(\frac{x}{z}\right) \quad \text{and} \quad \phi = arctan\left(\frac{y * sin(\theta)}{x}\right) \tag{5}$$

The computational complexity of equations in 5 is significantly greater than that of eq. 2.

3.2 Live Panoramic Zoom and Panning

Using the cylindrical panorama as an intermediate representation, our system can generate an arbitrary virtual camera view from the position of the camera array. The user has the full freedom to pan, tilt and zoom. As shown in figure 5, the virtual camera view is corrected to a perspective view very similar to that of a physical camera.

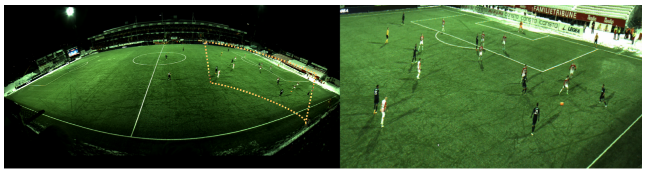


Figure 5: Panorama video with labeled ROI (left) and the virtual camera generated (right). It can be observed that it is not a simple crop from the bigger video.

This enables the virtual camera video delivery part shown on the right in figure 2. It consists of fetching and decoding the panoramic video file, creating the virtual camera and handling user input. The implementation uses one thread for most of the process including decoding the current frame, moving the data from host to device, creation of the virtual camera and rendering the texture onto a display. Fetching of video segment data and user input are handled by different threads.

3.2.1 *Video Handling*

For streaming, we use HTTP segment streaming (with plans for adaptive HTTP streaming). The segments of the panoramic videos are served by an Apache server along with a manifest file. The manifest file is used to inform the clients when the next file is ready for download. The viewer checks the manifest file periodically and downloads the next segment when ready.

As soon as the panoramic video segment is transferred, it is kept ready for processing. This process runs in the background without blocking either the display thread or the user input thread. At the moment, the system demands a large network bandwidth due to the full resolution panorama video. Nevertheless the panoramic video is compressed using H.264, saving quite a lot of space.

3.2.2 *Virtual Camera*

Panning and zooming are performed by a virtual perspective camera. The cylindrical panoramic texture generated as

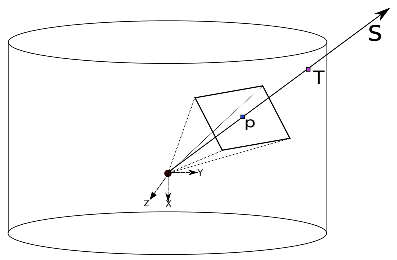


Figure 6: The intersection of the ray from the virtual view with the unit cylinder

described in section 3.1.3 is viewed through a virtual camera and the output is rendered on the screen.

When we use the virtual camera, the operation is to fetch the pixels of the image formed on the camera from the cylindrical texture. This can be better seen in figure 6.

A pin-hole camera for a point projection from a 3D point P to image point q can be written in the following manner:

$$\lambda q = [K|0_3] \begin{bmatrix} R & 0 \\ 0_3 & 1 \end{bmatrix} \begin{bmatrix} 0_3^T & -C \\ 0 & 1 \end{bmatrix} P \tag{6}$$

where R is the general (3×3) 3D rotation matrix as a function of θ_x, θ_y and θ_z, the rotation angles around the x, y and z axes respectively and K is the camera intrinsic matrix built with focal length(f).

Let p be the current pixel. So we need to find the ray that passes from the camera center C to the pixel p. The ray can be represented by:

$$s = \lambda R^{-1} K^{-1} p \tag{7}$$

Then the intersection of this ray with the unit cylinder gives us the exact position on the cylindrical texture. The intersection point can be found as follows:

$$T_x = \left(\frac{W_p}{FOV}\right) \left\{arctan\left(\frac{-s(1)}{s(3)}\right)\right\} + \frac{W_p}{2} \tag{8}$$

$$T_y = \left(\frac{1}{2} - \frac{s(2)}{\sqrt{s(1)^2 + s(3)^2}}\right) H_p \tag{9}$$

where W_p, H_p and FOV are the width, height and the field of view of the panoramic texture respectively. (T_x, T_y) are the coordinates on the unrolled cylindrical texture as described in section 3.1.3. When these calculations are performed with sub-pixel accuracy, it is not necessary that the intersection always lands at one pixel. So there is a need for interpolation from the surrounding pixels. A few experimental results are presented in the next section in this respect.

3.2.3 Implementations

A straightforward implementation (CPU) of such a viewer is to loop through all the pixels in the virtual view and find the positions where the rays land on the panoramic texture. The heavy operations include an inverse tangent and a square root in every pixel calculation. Since the operations are well suited for parallelization, we have ported the program to a GPU.

A simple port (GPU1) performs the calculation of the ray intersection and the fetching of the corresponding pixel on the GPU. So, in the initial implementation the videos are decoded on the CPU, the frames are transferred to the GPU, and calculations and fetching operations are performed on the GPU. Then the virtual image is created and transferred to the host for displaying/storing purposes.

Since it is possible to render OpenGL textures written by an NVidia CUDA kernel directly from the GPU to the screen, the current implementation (GPU2) uses that feature. A texture area is defined in advance and bound to the screen buffer. When the fetching operations are complete, the output is not transferred to the host, but written to the bound texture buffer on the GPU. Then this texture is displayed directly on the screen, saving the transfer overhead from device to the host. Other optimizations of GPU2 include the use of CUDA texture buffers instead of global memory on the GPU for the panorama frames to speed up the fetching operations owing to hardware-accelerated interpolation.

3.2.4 Operation

The user can pan, tilt or zoom using the virtual camera. When the panning operation is performed, θ_x is modified. θ_y is modified when a tilting operation is performed on the camera. The zoom is realized by modifying the focal length f of the virtual camera.

4. EXPERIMENTS

To test our prototype, we have performed a number of experiments during the development of the system. In this section, we present the results.

4.1 Real-time generation of panorama video

We have earlier shown that we are able to generate panorama video in real-time [11], and we therefore do not focus our evaluation on this part of the system. However, we have replaced the four 1K-cameras with five 2K-cameras, i.e., generating more than the double number of pixels. Nevertheless, using the setup in figure 2 distributing the capturing and processing, we are still able to process the panorama frames in real-time, i.e., generating a 25 fps video live. In average, we are able to generate the frames in 34.58 ms on an Intel i7 CPU, and we are currently working towards delivering a higher frame rate. In this respect, the only bottleneck is the processing. To deliver higher rates over the 1 Gbps Ethernet link, the cameras must use a Bayer representation which must be converted to YUV on the host, and at higher frame rates, both the stitcher and x264 encoder may become bottlenecks. We are therefore currently moving the new pipeline back to the GPU we had in [11], but for our current 25 fps prototype, the current setup is sufficient for real-time panorama generation.

4.2 Zooming and panning

After the panorama video has been generated, it is ready for consumption by local or remote clients. The video must be retrieved, processed for zooming and panning and processed for scaling (interpolation).

4.2.1 Video Download

First, the video must be fetched to the processing machine of the client. The time taken for downloading is measured and the results for a few video segments are presented in figure 7. Each of the segments has a playout duration of 3 seconds and varying file size. The experiments are performed using three different networks: 1) All on the same machine using localhost; 2) Client and server on the same local network; and 3) Client is 1.500 km avay from the server over a wide-area network. Figure 7 shows the average times taken for 23 files and their sizes in the background to illustrate the effect of the sizes. The plot also demonstrates a horizontal cut-off line which is the limit for smooth playout.

Usually, we run the system over the norwegian research backbone network (Uninett with 19 hops). We have, however, also tested it locally. It can be observed that even when the client and server are separated by a real network, which is subject to unknown traffic shaping and congestion on the Tromsø side, the client is still able to perform close to the smooth playout. Of course, this varies depending on the bandwidth available to the client. Since the client side processing is pipelined, only one segment buffer is required to make this run in real-time – if the segments can be downloaded in real-time. By applying adaptive HTTP streaming (we expect that this is feasible when the GPU port of the new server pipeline is complete), we will be able to overcome this bottleneck at the price of reduced quality.

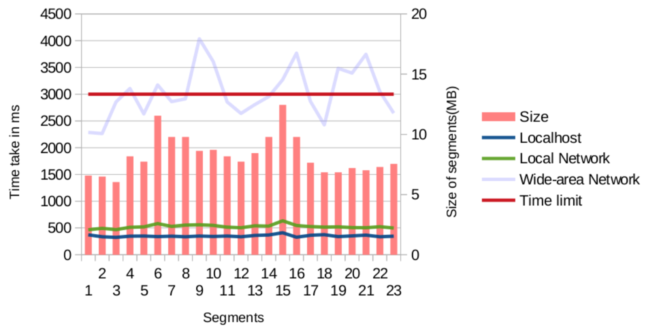


Figure 7: A plot showing various times taken

4.2.2 *Comparison of implementations*

In section 3.2.3, we described three different implementations of the computation and rendering of the virtual camera. The average execution time for each of these approaches is presented in table 1. The experiments were performed on a machine with an Intel i7-2600 CPU and an Nvidia GeForce GTX 460 GPU. The time is observed for the same video segment in all three cases and for a virtual camera with Full HD resolution (1920 × 1080). The time also includes the decoding time of the panorama frame and all transfers between host and GPU.

Approach	Average	variance
CPU	255.7	35.8
GPU1	13.3	2.6
GPU2	10.1	3.2

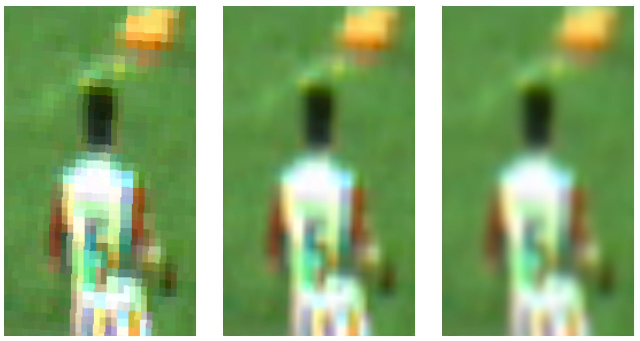
Table 1: Execution time per frame (ms).

The unoptimized (no multi-threading or SIMD instructions) CPU implementation is not fast enough to provide the full frame rate of 25 fps. A straight-forward GPU implementation reduces the computation time drastically.

4.2.3 *Interpolation comparison*

Since the panorama is rendered on a cylinder, pixel interpolation is required for every virtual camera independent of zoom or pan setting. Interpolation trades computing time for smoothness and sharpness, and we tested three approaches: nearest neighbour, bilinear and bicubic. Figure 8 presents a highly zoomed frame in the three different modes with the kernel execution time for each of these approaches.

As it can be seen in the figure, bicubic interpolation seems to provide the best visual quality at the cost of a higher execution time. However, we pay less than 3.5ms and choose the higher image quality. Furthermore, CUDA supports an optimized linear interpolation on some hardware, which could be used as an alternative.



(a) Nearest neighbour (2916 us). (b) Bilinear (2840 us). (c) Bicubic (3242 us).

Figure 8: Execution time for the interpolation algorithms

4.2.4 *Size of virtual view*

Most of the performance measures provided in this section are for a Full HD resolution, but the resolution of the virtual camera varies with the viewing device. Figure 9 therefore demonstrates the effect of the size on the kernel execution time of the final generation of the zoomed image (note the *micro*second scale). As can be seen in the plot, the resolution has negligible impact on performance (as the whole panorama video has to be moved to the GPU anyway).

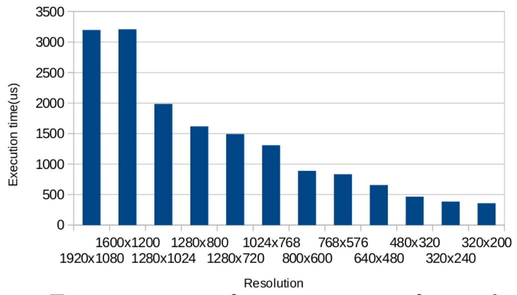


Figure 9: Execution times for various sizes of virtual camera

5. DISCUSSIONS

5.1 Panoramic stitching

In the current system, panoramic stitching is performed using a static lookup table. Each pixel in the output panorama is mapped to a pixel in one of the source images. At runtime, the pixels are merely interpolated, both from the cylindrical warp and the down-sampled chroma channels in the YUV 4:2:2 pixel format.

We intend to move this module to GPU, and re-introduce dynamic seam-finding [11]. The dynamic seam can be determined at runtime, and pixel values dynamically chosen from one of two lookup tables in the overlapping region of the source images.

5.2 Scalability

Even though the generation of each frame is fast, the scalability of the system depends on where the video for the virtual camera is generated. By running everything on the sender side, we have the advantage of supporting light client devices and reducing the bandwidth required of the client's access network. This option, however, requires large amounts of processing power, as well as network bandwidth on the server side, i.e., the scalability is limited on the server side. On the other hand, by transferring the panorama video to the client device, our servers can easily support a large number of clients at the cost of client processing and a distribution infrastructure that can deliver the panorama stream (broadcast, multicast, caches, proxies, P2P, ...). Since the current trend is to equip even mobile phones with powerful CPUs and GPUs, we consider the second option most promising. The benefit of this is particularly visible in our chosen scenario of delivering personalized panning and zooming to the spectators in the stadium, where broadcast support is feasible, while the limited wireless spectrum prevents the delivery of individual streams.

5.3 Ongoing work

Such a system, even though it provides an interactive way to control the virtual camera, is not practical if the user is really meant to perform pan and zoom operations manually during the entire game. So, we are currently working on providing a virtual camera based on a request from the user. For example, the user will have the ability to select the main feature of interest, such as the ball, a specific player or a group of players. The challenge then is to create an aesthetic and smooth virtual camera on the client based on meta-information such as player and ball positions. Naturally, this must be done on the client in real-time to maintain our scalability goals.

We are currently experimenting with different approaches for generating a virtual view that follows a feature point aesthetically and smoothly. Our preliminary prototype seems to follow the ball well[2] (note that finding the position of the ball is out of the scope of this paper).

6. CONCLUSION

In this paper, we have presented a system for real-time interactive zoom and panning of panorama video used in a soccer stadium scenario. Based on video streams from five stationary 2K cameras, processed and stitched into a high-resolution panorama video, we are able to support any free view angle from the position of the camera array, i.e., a virtual camera. The unoptimized running prototype is able to generate one frame of the virtual camera from the panorama image in less than 10 ms on a commodity computer with a standard GPU. Thus, it can easily be scaled to support many concurrent users. If the processing is performed on the client device, at the cost of transferring the panorama video, any number of virtual cameras can be supported, i.e., all users may have their own virtual view.

There are a number of open challenges that are currently under investigation. We aim for more efficient implementations to reduce the resource consumption further, and we investigate more efficient algorithms for following a player or the ball smoothly.

[2]See http://www.youtube.com/watch?v=554RjEEtw3o

Acknowledgments

This work has been performed in the context of the *iAD* center for Research-based Innovation (project number 174867) funded by the Norwegian Research Council. Furthermore, the authors also acknowledge the support and hardware given by Dolphin Interconnect Solutions.

7. REFERENCES

[1] Azure-0814m5m. http://www.azurephotonicsus.com/-products/azure-0814M5M.html.

[2] Basler aca2000-50gc. http://www.baslerweb.com/products/ace.html?model=173.

[3] Y. Ariki, S. Kubota, and M. Kumano. Automatic production system of soccer sports video by digital camera work based on situation recognition. In *Proc. of ISM*, pages 851–860, 2006.

[4] P. Carr and R. Hartley. Portable multi-megapixel camera with real-time recording and playback. In *Proc. of DICTA*, pages 74–80, 2009.

[5] P. Carr, M. Mistry, and I. Matthews. Hybrid robotic/virtual pan-tilt-zom cameras for autonomous event recording. In *Proc. of ACM MM*, pages 193–202, 2013.

[6] R. Guntur and W. T. Ooi. On tile assignment for region-of-interest video streaming in a wireless lan. In *Proc. of NOSSDAV*, pages 59–64, 2012.

[7] P. Halvorsen, S. Sægrov, A. Mortensen, D. K. Kristensen, A. Eichhorn, M. Stenhaug, S. Dahl, H. K. Stensland, V. R. Gaddam, C. Griwodz, and D. Johansen. Bagadus: An integrated system for arena sports analytics – a soccer case study. In *Proc. of ACM MMSys*, pages 48–59, Mar. 2013.

[8] A. Mavlankar and B. Girod. Video streaming with interactive pan/tilt/zoom. In M. Mrak, M. Grgic, and M. Kunt, editors, *High-Quality Visual Experience*, Signals and Communication Technology, pages 431–455. Springer Berlin Heidelberg, 2010.

[9] S. Sægrov, A. Eichhorn, J. Emerslund, H. K. Stensland, C. Griwodz, D. Johansen, and P. Halvorsen. Bagadus: An integrated system for soccer analysis (demo). In *Proc. of ICDSC*, Oct. 2012.

[10] X. Sun, J. Foote, D. Kimber, and B. Manjunath. Region of interest extraction and virtual camera control based on panoramic video capturing. *IEEE Transactions on Multimedia*, 7(5):981–990, 2005.

[11] M. Tennøe, E. Helgedagsrud, M. Næss, H. K. Alstad, H. K. Stensland, V. R. Gaddam, D. Johansen, C. Griwodz, and P. Halvorsen. Efficient implementation and processing of a real-time panorama video pipeline. In *Proc. of IEEE ISM*, Dec. 2013.

[12] J. Wang, C. Xu, E. Chng, K. Wah, and Q. Tian. Automatic replay generation for soccer video broadcasting. In *Proc. of ACM MM*, pages 32–39, 2004.

[13] M. Wieland, R. Steinmetz, and P. Sander. Remote camera control in a distributed multimedia system. In B. Wolfinger, editor, *Innovationen bei Rechen- und Kommunikationssystemen*, Informatik aktuell, pages 174–181. Springer Berlin Heidelberg, 1994.