Real-time Panorama Video Processing Using NVIDIA GPUs

Marius Tennøe, Espen Helgedagsrud, Mikkel Næss, Henrik Kjus Alstad, Håkon Kvale Stensland, Pål Halvorsen, Carsten Griwodz
Department of Informatics, University of Oslo, Norway
{mariuten, espenoh, mikkeln, henrials, haakonks, paalh, griff}@ifi.uio.no

IAD INFORMATION ACCESS DISRUPTIONS
[simula . research laboratory]

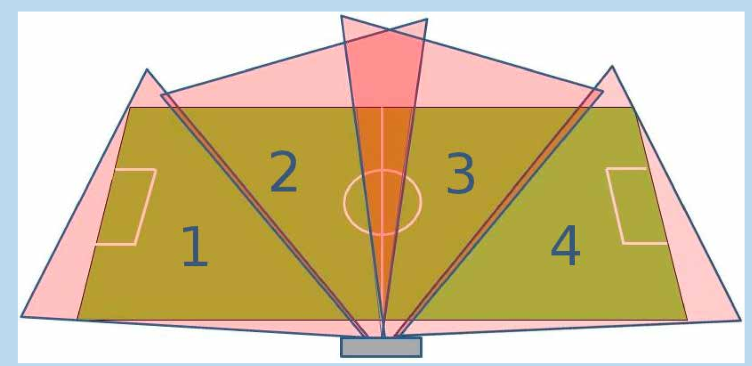
Abstract

Sports analytics is a growing area of interest, both from a computer system view to manage the technical challenges and from a sport performance view to aid the development of athletes. We have been working on Bagadus, a prototype of a sports analytics application using soccer as a case study. Bagadus integrates a sensor system, a soccer analytics annotations system and a video processing system using a video camera array. A prototype is currently installed at Alfheim Stadium in Norway. An important part of the system is playback of events from the games using stitched panorama video. This results in a lot of technical challenges to keep the creation of these panorama videos in real time. To be able to do this, we utilize the power of GPGPU by use of NVIDIA GPUs and CUDA.

System Components

The Bagadus system as a whole is based on three main subsystems.

Video subsystem:
The real-time video streaming is one of the main parts that will enable a fast, easy, and fully automated sports analysis solution. The system prototype currently works with four cameras that are positioned towards a football pitch. These are synchronized by an external trigger signal in order to enable a video stitching process that produces a panorama video picture. The cameras record the game at a rate of 30 fps, where the current cameras deliver frames of 1290 x 960 pixels. This camera setup lets us record videos from each individual camera stream, as well as create a 6742 x 960 pixels panoramic video that almost instantly can be used for game analysis.



Player tracking subsystem:
Another key feature in the system is the possibility to follow one or more players in a game. For this, we need some kind of player tracking solution. For stadium sports, one approach is to use sensors on players to capture the exact position. ZXY Sport Tracking provides such a solution where a sensor system submits position and orientation information at an accuracy of ±1 meter, at a frequency of 20 Hz. By retrieving this data from a database, we can combine it with the video being generated.

Annotation subsystem:
In order to mark important game events, being able to view the video almost instantly, and make this process as easy as possible, we use Muithu. Muithu is a novel notational analysis system that is noninvasive for the users, mobile and light-weight, and has already been developed. A cellular phone is used by head coaches during practice sessions or games for annotating important performance events. The system will enable you to select the players you want to follow, choose which video stream you wish to see, and playback events tagged by Muithu.

Together, these three main components create a fully automated sport analysis system. However, it can only be used in realtime if we manage to process the large amount of data in the limited time provided. In this respect, we use NVIDIA GPUs for large-volume processing, where CUDAs ability to parallelize tasks makes it perfect for many of the operations needed in the processing of the four video streams, including the creation of the panoramic video. One of our main goals is that the coaches can use the system in the break, or even during the game, and show events and gameplay to the players before they return to the pitch. CUDA greatly helps us in accomplishing this task.

The Pipeline

1) CamReader:
Retrieves frames from the four cameras each sample interval (30 fps). Frames retrieved as YUV 4:2:0.

2) Converter:
Responsible for converting from YUV 4:2:0 to RGBA. Due to the good performance of swscale in ffmpeg, this module consists of a single CPU thread.

3) Debarreler:
Wide angle lenses, results in images suffering from barrel distortion. This module removes this distortion. We currently use OpenCVs debarreling filter.

4) SingleCamWriter:
We want to store the original, single camera frames in addition to the stitched panorama. This module converts the frames from RGBA to YUV 4:2:0, then encodes the frames in lossless H264, and writes the result to disk.

5) Uploader:
Transfers the data to be processed to the GPU. Is also responsible for executing the CPU part of the BackgroundSubtractor module. Uses double buffering with asynchronous memory transfers.

11) Downloader:
Transfers the panorama image from the GPU to the CPU.

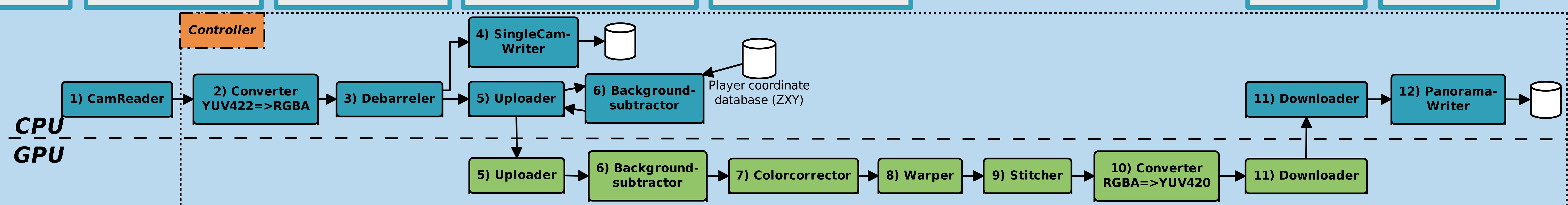
12) PanoramaWriter:
Encodes the panorama image to H264, and writes the result to disk.

Buffers:
Modules have two sets of buffers:
- One or more input buffers
- One or more output buffers

On the CPU-modules (blue color), these are located in ordinary RAM, while on the GPU (green color) they consist of global CUDA memory buffers.

Controller:
The controller loops with two steps.
1: For all modules, transfer data from output buffer of module N to input buffer of module N + 1. This is done by pointer swapping when possible, else memory copies.
2: Broadcast to all modules (except the reader) to make them process the data in the input buffer.

The end-to-end runtime for a frame is approximately 5.3



6) BackgroundSubtractor:
Background subtraction is the process of deciding what pixels of a video belong to the foreground or background. We can exploit the fact that we have the ZXY player coordinates to improve both performance and precision. By first retrieving player coordinates for a frame, we can then create a player lookup bytemap, where we only set the players pixels with a safety margin to 1. This is partially run and uploaded by the uploader on CPU. The BGS on GPU uses this lookup map to only process pixels close to a player, which approximately halves GPU processing times. When run in a pipelined fashion, the processing delay caused by the lookup map creation is eliminated.

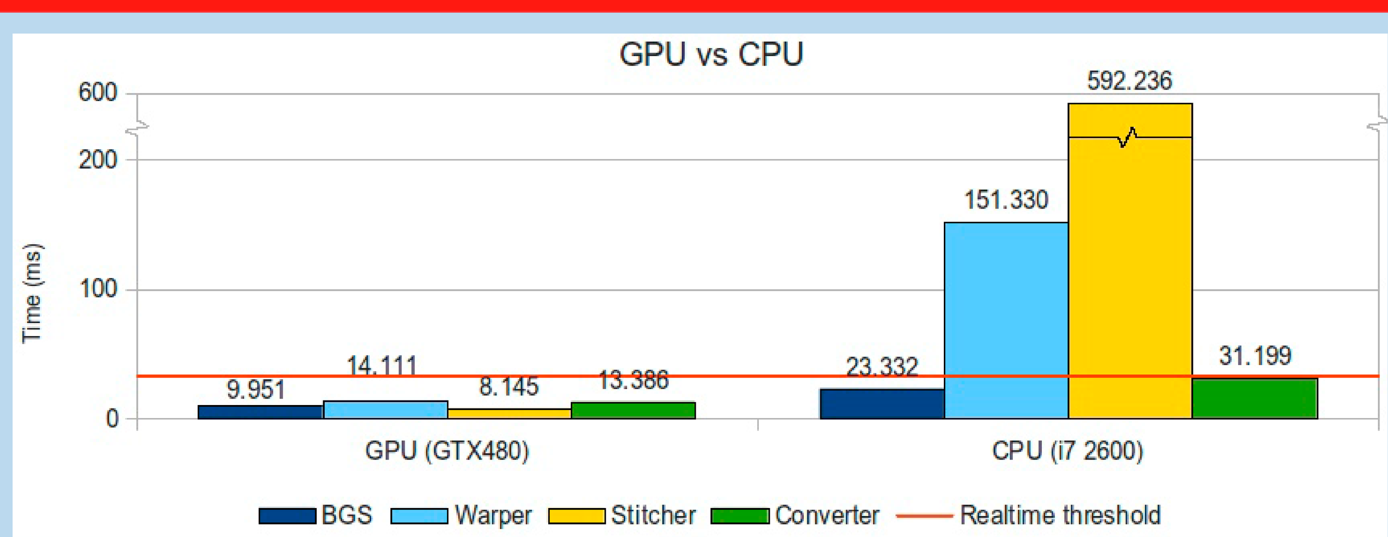
7) Color corrector:
When wanting to stitch frames from several cameras, we need to run a color correction to get the best result. The color corrector is currently not implemented, but is to be created in the nearest future.

8) Warper:
Warps the camera frames to fit the stitched panorama image, because the stitcher assumes that its input images are perfectly warped and aligned. Here we utilize the NVIDIA Performance Primitives library (NPP) for an optimised implementation.

9) Stitcher:
The stitcher is homography based, and simply copies data from the input frames to the larger panorama output frame. We currently use a stitching algorithm consisting of a static seam between the cameres, which results in visible artifacts. In the future, we want to calculate dynamic seams between the cameras by using the BGS masks, where we want to avoid a seam passing through the players.

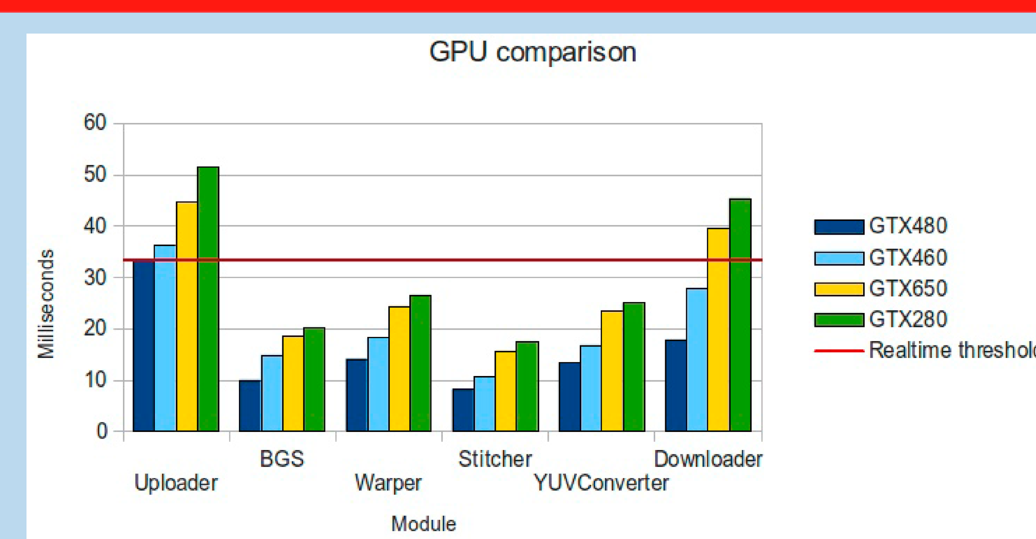
10) Converter:
To store the result on disk, we need to convert the panorama to YUV 4:2:0 to be used in the H264 encoder. Conversion of such a large frame is not fast enough on the CPU, so this therefore had to be implemented on the GPU.

Results



In the chart above, we can see the results of the video stitching modules running on the GPU, compared to their performance when running on the CPU. We can clearly see a massive performance boost with multipliers of up to 72x. It is clear that the possibilities delivered by CUDA allows us to outperform the CPU when running parallell execution tasks on big data sets.

The CPU implementations of the warper, the background subtractor and the stitcher used in these benchmarks were based on *OpenCVs* implementations. The converter on the CPU was based on *ffmpeg* and *swscale*.



From the measured timings of the pipeline, we can see that the system is now performing according to the realtime constraint. However, we need to use a high-end graphics card such as a NVIDIA Geforce GTX480 or better, to achieve this goal.

When analyzing the measured performance of the pipeline, we have found some clear bottlenecks. When ignoring the CPU as the main bottleneck of the whole pipeline, we can clearly see that the bandwidth of the GPU is the definite limiting factor on the GPU-side of the system. This is easy to see by observing the performance of the uploader and downloader modules. It would therefore be interesting to test the pipeline on a system with a higher GPU bandwidth.

Summary and future works

We show in this poster that we have been able to create a panoramic stitcher pipeline for generating a video stream in real-time by use of NVIDIA GPUs. We can now continue focusing on the next steps of the system, knowing that the performance of the system is adequate for now.

Future works include adding a color correction module to the pipeline, as well as a more advanced stitching module based on a dynamic seam algorithm. We also want to research further use for the background subtraction module. As of now we are using HD cameras, but we want to expand to 2K and later 4K cameras. Moreover, we see that more of the pipeline would fit well on the GPU, so a natural step would be to move more modules over to the graphics card. In addition, we have a development branch researching the possibility of a freeview application by use of a camera array.

References:

P. Halvorsen, S. Sægrov, A. Mortensen, D. K. C. Kristensen, A. Eichhorn, M. Stenhaug, S. Dahl, H. K. Stensland1, V. R. Gaddam, C. Griwodz, D. Johansen. Bagadus: An integrated system for arena sports analytics - a soccer case study -. In *Proceedings of the ACM Multimedia Systems Conference (MMSys)*, 2013.

S. Sægrov, A. Eichhorn, J. Emerslund, H. K. Stensland, C. Griwodz, and P. Halvorsen. BAGADUS: An Integrated System for Soccer Analysis (demo), In: *Proceedings of the International Conference on Distributed Smart Cameras (ICDSC)*, 2012.

Bagadus demo: BAGADUS - An Integrated System for Soccer Analysis.
http://www.youtube.com/watch?v=1zsgvjQkL1E