GPU-based View Interpolation for Smooth Camera Transitions in Soccer

Patrik Goorts, Sammy Rogmans, Philippe Bekaert

Hasselt University - tUL - iMinds, Expertise Centre for Digital Media, Wetenschapspark 2, 3590 Diepenbeek, Belgium.

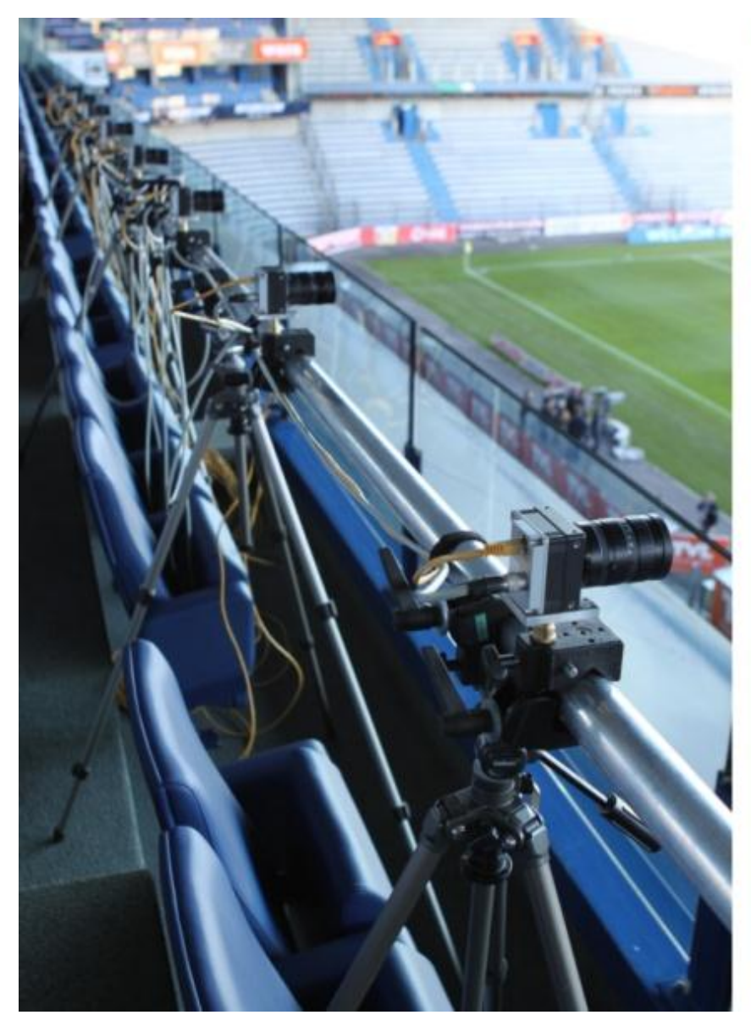
patrik.goorts@uhasselt.be, sammy.rogmans@uhasselt.be, philippe.bekaert@uhasselt.be

Abstract

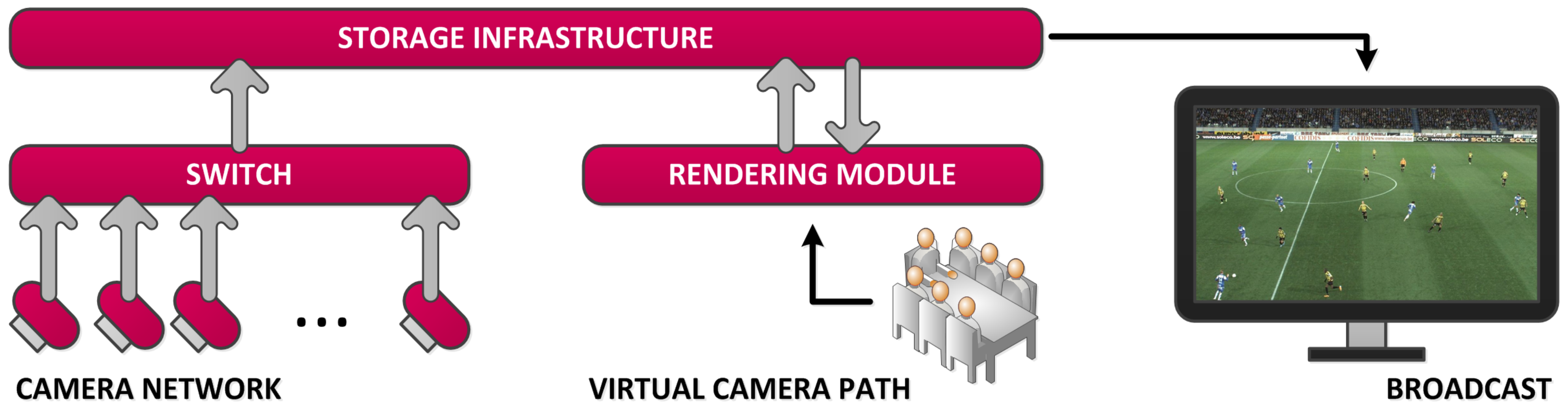
We present a system, capable of synthesizing free viewpoint video for smooth camera transitions in soccer scenes. The broadcaster can choose any camera viewpoint between the real, fixed cameras. This way, action can be followed across the field in a smooth manner, a frozen image or a replay can be viewed from multiple angles, and the broadcasted image can be transitioned from one to the other side of the field in a smooth manner to avoid orientation-related confusion of the viewers. We use a real-time image-based approach completely running on GPUs; no detailed geometry of the scene is required. Once running, the setup fully automatically generates the requested view; no other user intervention is required.

Setup

To build the proof-of-concept, we placed a number (typically 8 to 16) of computer vision cameras around the field. To cover all possible camera setups, we constructed two fundamental camera arrangements: a straight line and a quarter circle circumventing the field corner. The line setup can be used to follow players from one side to the field to the other, while the quarter arc can be used to look around frozen action scenes. Using 2 Mpixel (1600x1200) Basler and Prosilica cameras with Fujinon lenses that have a fixed focal distance of 12.5 mm, our tests indicate that the maximum inter-camera distance can be up to 10 meters, i.e. about 33 feet. Alternatively, higher resolution cameras with a more wide field-of-view can further increase this constraint. We performed two real-life on-site tests, capturing official soccer games in the respective national competitions; one in the stadium of Genk, Belgium and one in the mini stadium of Barcelona, Spain.



All cameras are connected using copper Gigabit Ethernet via a switch to a storage server that has a 10 Gigabit fiber optic backbone. The images are transferred in raw Bayer mosaic format to the experimental storage server, developed by EVS, Belgium, where they can be retrieved for real-time processing. Our rendering server, which is basically a strong workstation packing an NVIDIA GeForce GTX680 graphics card, will digest the original raw images in order to instantly render the virtual camera viewpoint that is requested by the director. All cameras are synchronized on-site using an external clock that is distributed over a separate coaxial wire.

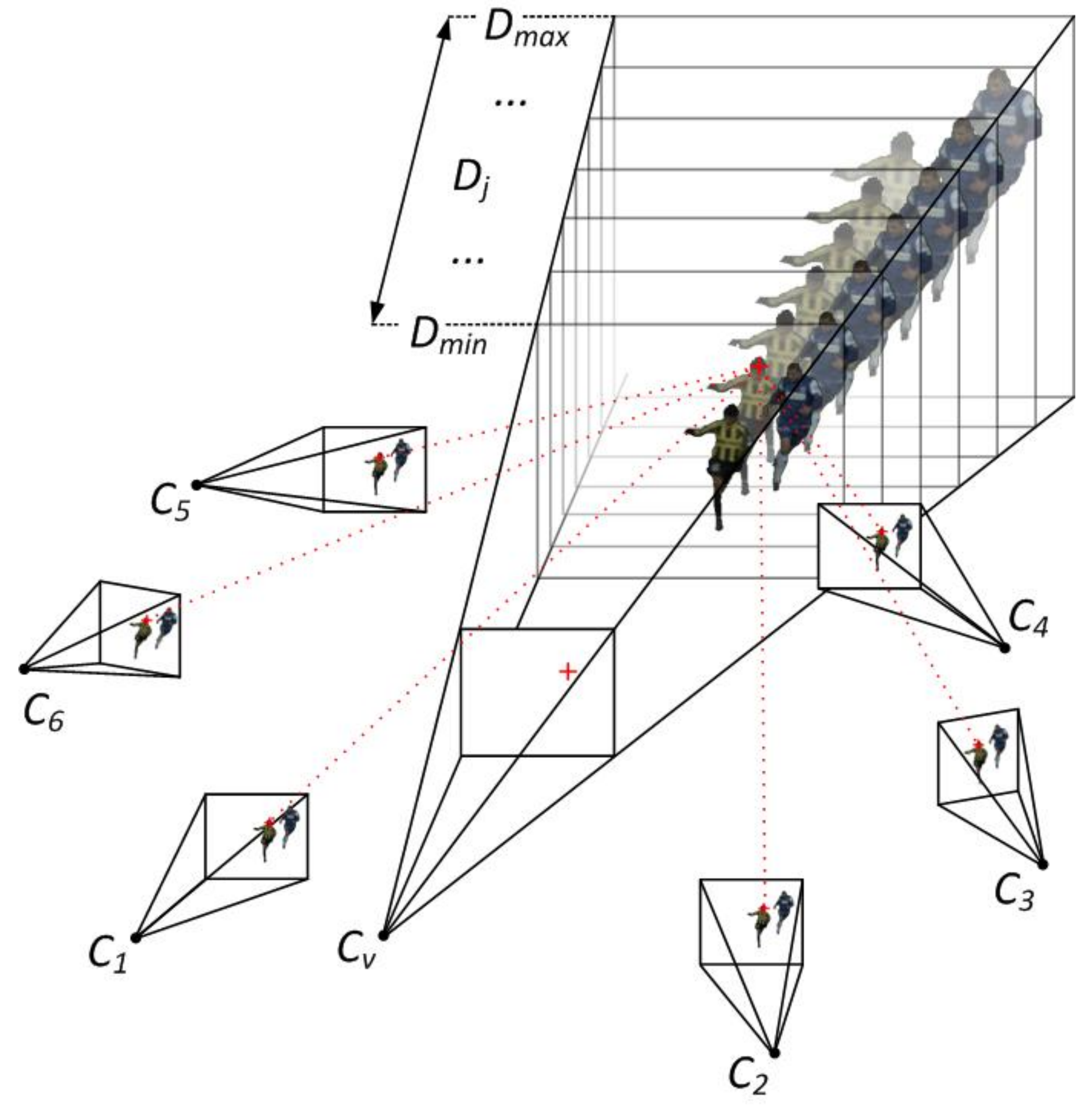


The rendering module

The render server runs a software processing module that is composed out of traditional Cg shaders and CUDA kernels, which perform the real-time rendering solely using the NVIDIA GTX680 graphics card. The processing steps can be divided in to the following summarized action points;

0. The director determines a spatiotemporal camera path for the transition and requests it.
1. Accept a virtual camera position and the according desired time.
2. Fetch the required input images from the EVS storage server.
3. Demosaic the raw Bayer images; the amount of data is tripled but kept locally on the high-speed GPU memory.
4. Segment the foreground (players) and background (playing field and spectators).
5. Render the virtual foreground and background independently; each optimized for their characteristics.
6. Merge the synthesized foreground and background to the resulting required spatiotemporal image.
7. Transfer this result back to the EVS storage server.
8. Repeat the procedure for the following spatiotemporal points of the requested path.

The storage and render server are connected with a high speed 10 Gigabit fiber optic connection, nevertheless the rendering module consistently ingests raw Bayer pattern camera images to minimize the data communication between the storage and render server. The demosaiced images are not buffered on neither the storage or render server, as the brute-force approach of reprocessing them – in the sporadic case of a revisit to a spatiotemporal point – proves to be more efficient than the implicit data communication that is required for a buffering mechanism. The interoperability between CUDA and Cg shaders in the traditional pipeline, is consequently exploited to maximize the performance of the image processing kernels in steps 3—6. Furthermore, the asynchronous DMA data transfer support from and to the system and GPU memory is used for extreme overall application throughput by performing simultaneous acquisition, download, readback and kernel processing.

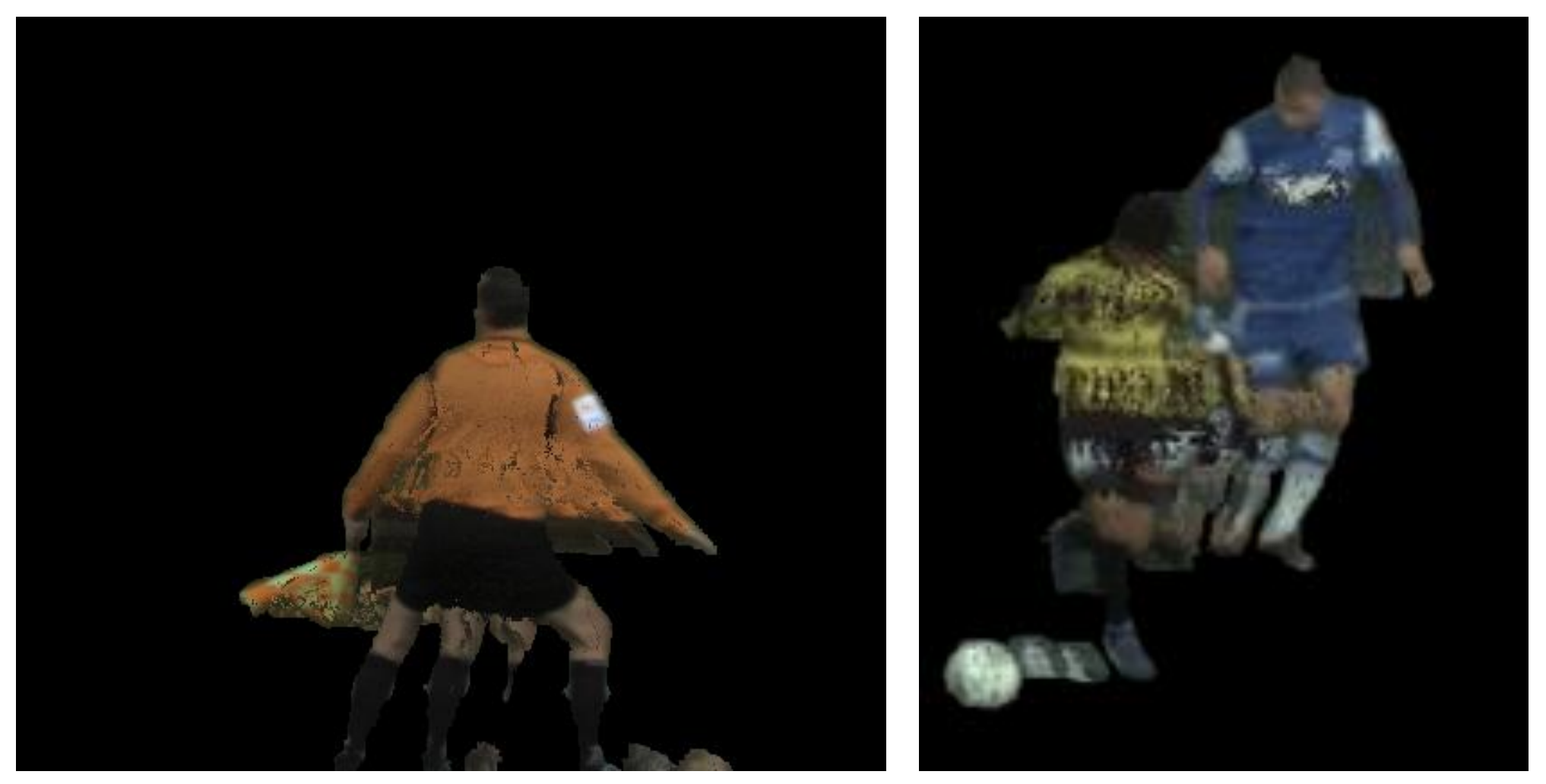


Foreground rendering

To acquire high quality results, we generate a depth map using a plane sweep approach. Next, we filter the depth map to reduce artifacts and use this filtered depth map to perform a second depth-aware plane sweep.

First plane sweep

We divide the space before the requested virtual camera in parallel planes with different depths. For every depth plane, we project the calibrated input images on the corresponding plane and calculate the color consistency for every pixel using the projective texturing capabilities of the GPU, driven by a winner-takes-all approach that is implemented by exploiting the depth test in the raster operators. The final result is a depth where the color consistency is highest for that pixel. This will however, yield mismatches between, for example, the left leg in one input image with the right leg in another input image, resulting in a third leg or other artifacts. Therefore, we will filter the depth map and perform a second plane sweep to effectively remove these artifacts and increase the confidence of the resulting depth information.



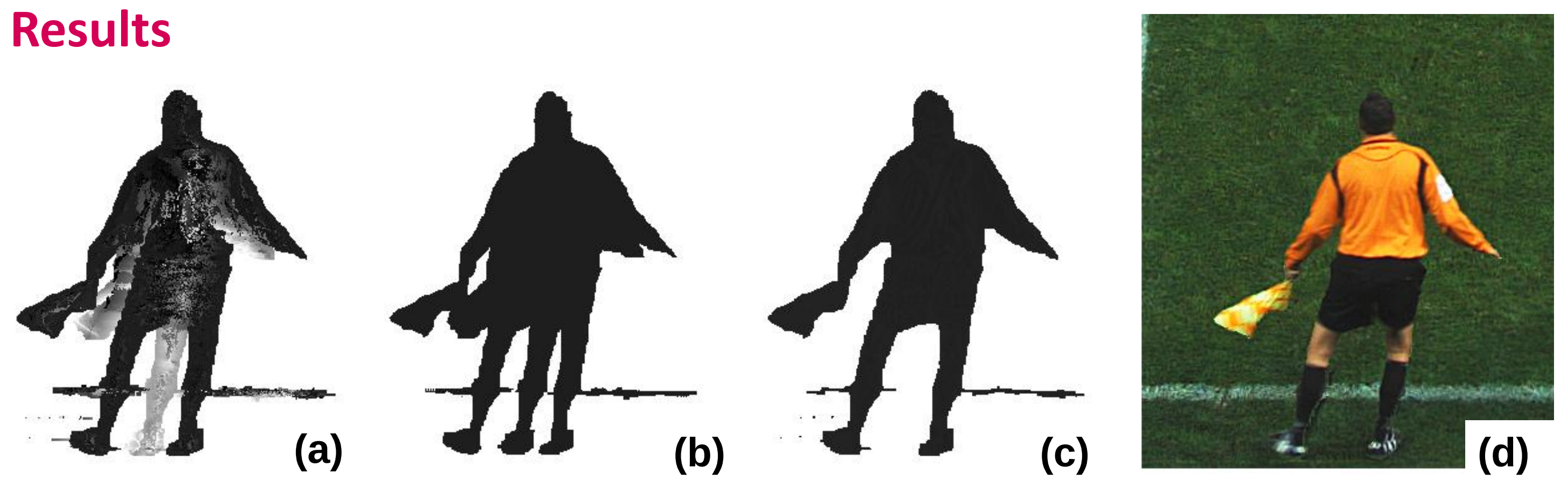
Depth selection

First, we divide all the foreground pixels of the virtual image in unconnected groups of pixels using a seed growing approach, implemented in CUDA. Next, we calculate for every group the median depth value using the Thrust library and pass these median depth values to the next plane sweep.

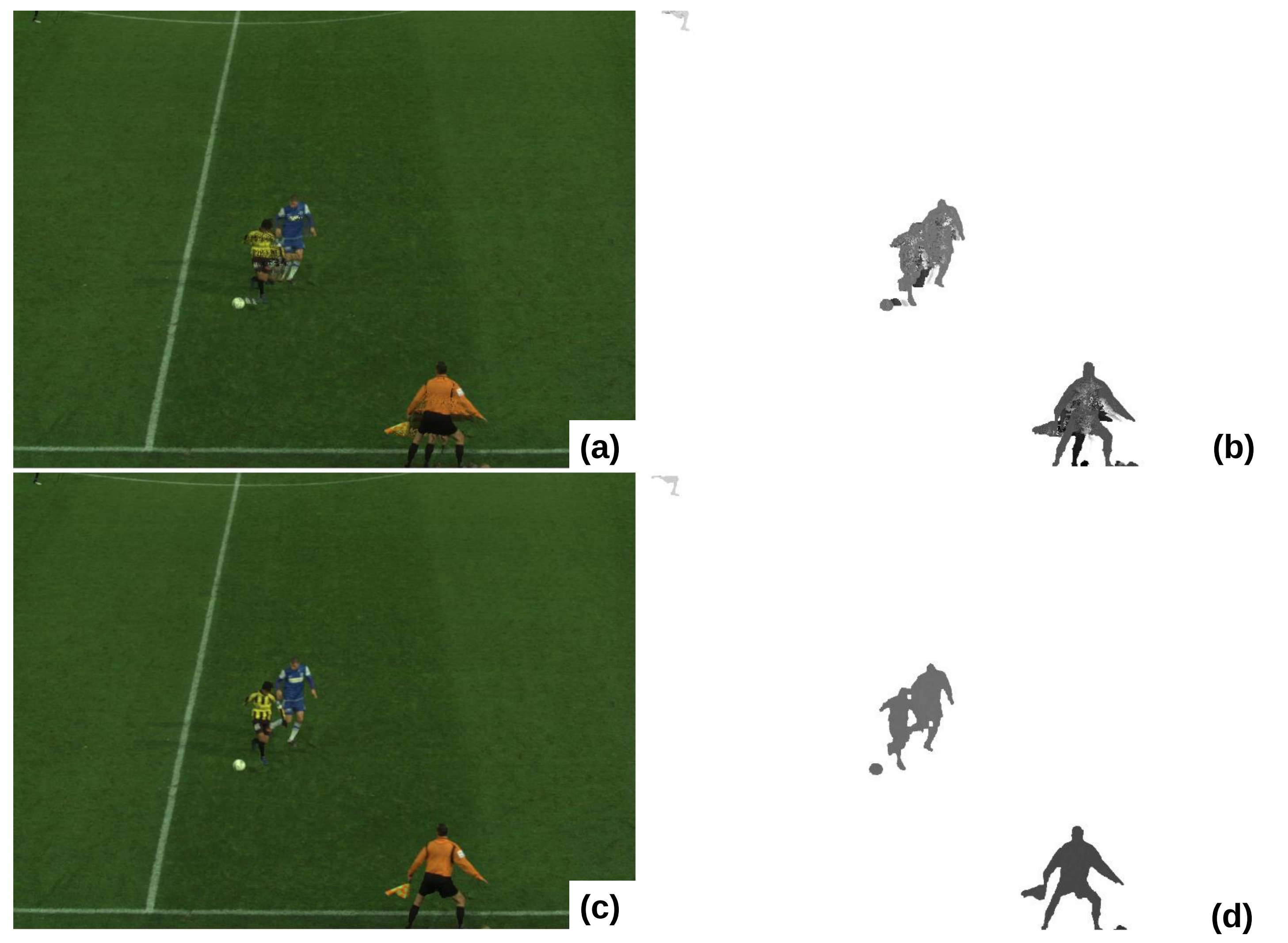
Depth-selective plane sweep

Finally, we perform a second plane sweep similar to the first one, but if the proposed depth for a pixel differs more than a specified threshold from the original depth value in the previous step, the depth is not considered and hence not processed. This will effectively remove the mentioned artifacts.

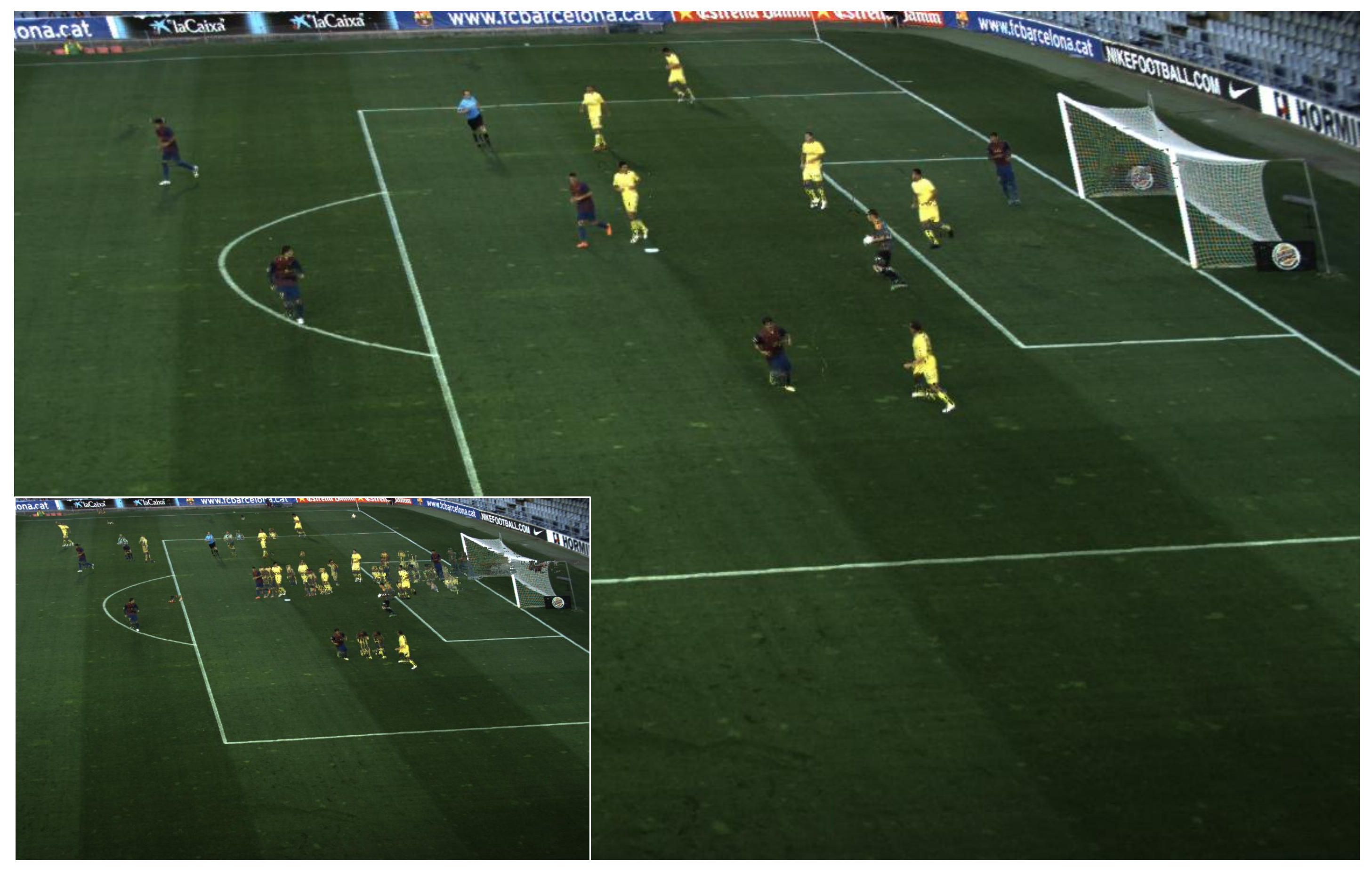
Results



(a) Depth map before filtering. **(b)** Median depth for this group of pixels. **(c)** Depth map after depth-selective plane sweeping. **(d)** Color result.



(a)(b) Result and depth map before filtering. **(c)(d)** Result and depth map after filtering.



Bottom left: The result without filtering, showing numbers of artifacts. The result after filtering is displayed enlarged; The artifacts and ghost players due to the single plane sweep are effectively removed. These results are achieved in real-time on a single graphics card.