Panoramic Mapping on a Mobile Phone GPU

Georg Reinisch* Clemens Arth† Dieter Schmalstieg‡

Institute of Computer Graphics and Vision
Graz University of Technology (Austria)

ABSTRACT

Creating panoramic images in real-time is an expensive operation for mobile devices. Mapping of individual pixels into the panoramic image is the main focus of this paper, since it is one of the most time consuming parts. The pixel-mapping process is transferred from the Central Processing Unit (CPU) to the Graphics Processing Unit (GPU). The independence of pixels being projected allows OpenGL shaders to perform this operation very efficiently. We propose a shader-based mapping approach and confront it with an existing solution. The application is implemented for Android phones and works fluently on current generation devices.

Index Terms: H.5.1 [Information Interfaces and Presentation]: Multimedia Information Systems —Artificial, augmented, and virtual realities; Evaluation/methodology

1 INTRODUCTION

For AR purposes, Wagner *et al*. created a method that captures an image with the camera of a mobile phone and maps it onto the panoramic image in real-time [5]. The approach takes the camera feed as input and continuously extends the panoramic image, while the rotational parameters of the camera motion are estimated.

In this work we complement the original CPU-based rendering approach with a GPU-based implementation to transfer computational costs from the CPU to the GPU. The striking advantages of our approach are the parallel processing of pixels and the efficient way of improving the image quality.

2 PANORAMIC MAPPING AND TRACKING

The original approach used as a baseline for this work was proposed by Wagner *et al*. [5]. It combines the panoramic mapping and orientation tracking on live camera images, performing in real-time on current mobile phones. In the past it was used for various applications, such as the creation of panoramic images and outdoor Augmented Reality [1, 3]. In the tracking process, the FAST corner detector [4] for feature point extraction is used, ranking the found points by strength. A motion model estimates the orientation of the camera in a new frame and projects the extents of the current frame into the existing map, Features projected outside of this frame are eliminated. For a valid tracking result, the number of remaining features successfully matched against the existing map must exceed a given threshold, since the final rotation matrix is acquired from these matches.

The rotation matrix is used to map the current frame into the panoramic image. The corner pixel coordinates of the frame are projected forward to define the area occupied by the current frame in the panoramic image. Finally all pixels occupied are mapped from the current frame via back-projection into the image and retrieving the corresponding pixel values and entering it into the panoramic map.

*e-mail: georg.reinisch@student.tugraz.at
†e-mail: arth@icg.tugraz.at
‡e-mail: dieter@icg.tugraz.at

Figure 1: CPU-mapped image by the application of Wagner *et al*. [5] (above), GPU-mapped image with image refinements proposed (below).

3 GPU SHADER IMPLEMENTATION

The mapping process is completely independent for each individual pixel, thus it can be parallelized heavily by using shader-programs on the GPU. The GPU can perform operations that are extremely costly on a CPU otherwise. In our context, this refers primarily to (i) clearing certain areas from a panoramic image, (ii) image refinement requiring pixel blending, and (iii) enlarging the amount of pixels, *i.e*. the image size, to be rendered.

Unfortunately on the GPU it is not possible to perform write operation on an other texture than the one the shader is applied to. Therefore we use a render-to-texture approach based on two framebuffers for the panoramic map, applying a common method also known as ”ping-pong technique”. For each frame, the role of the texture is swapped such that one serves as an input and the other one as an output texture. The shaders are always applied on all pixels of the output texture. Every pixel is treated in its own fragment shader program run, whether it lies in the area where the current frame is projected or not. If the pixel lies in this area, the color of the respective pixel of the camera image is stored at this location, otherwise the pixel of the input texture is copied.

An optional optimization step is to only pass the area covered by the actual frame to the shader, since the panoramic image will only be updated in this area anyway. This reduces the maximal number of shader runs of an *e.g*. 2048x512 pixels image from about 1 Mio. to about 75,000 (320x240 pixels), which is equivalent to a reduction in computational complexity to about 7.5 % over a naive implementation.

(i) Wiping: To remove unwanted areas, such as pedestrians or cars, the panoramic image can be edited in real-time by wiping over the panoramic image preview displayed on the mobile phone’s screen. By specifying an area in a preview image of the panoramic map, the coordinates are passed to the shader and the region around that coordinate is cleared and marked as unmapped. A new frame arriving can cover those cleared areas and fill the empty spots with color information again (see Figure 2).

N **radius**
$\vec{w}$ **coordinate to wipe**
$\vec{t}$ **actual pixel coordinate**
$(\vec{t}-\vec{w})\cdot(\vec{t}-\vec{w}) < (N^2)$

Figure 2: Exemplary wiping of all pixels $\vec{t}$ falling into a circular area of radius N around the wiping coordinate $\vec{w}$.

Table 1: Comparison of the average number of keypoints and matches for the original approach [5] and different image refinement approaches proposed. Note that the tracker stops to search for more matches as the number of 80 is reached.

Approach	⌀ # Matches	⌀ # Key Points
CPU		
Standard Mapping	80.00	1000.30
GPU		
a) No Refinements	80.00	1050.18
b) Brightness Corr. [2] only	80.00	1053.41
c) Frame Blending only	73.95	1028.54
d) Frame Blending + Brightness Corr. [2]	78.03	1030.62

(ii) Image Refinement: The exposure time of smartphone cameras usually varies automatically and cannot easily be fixed. Artifacts arising thereby are sharp edges between earlier mapped regions and newly mapped areas. Degendorfer [2] calculates the brightness offset for tracked feature points and adjusts the newly mapped pixels according to the average brightness difference estimated. This solution is not ideal, as the best areas for comparing brightnesses are homogenous regions rather than corners. Still the approach can be performed at almost no additional computational overhead, since the tracker inherently provides the matches.

To achieve even smoother transitions between different brightness values, we propose a combination of the work of [2] and a frame-based blending approach. The color values of newly mapped pixels in the first frame are directly drawn from the camera image. For all consecutive frames, pixels within an area represented by an inner and an outer frame of the camera image are blended. Pixels at the image border (outer frame) are directly taken from the panoramic map (input texture), the region inside the inner frame is directly mapped from the camera image. A linear blending operation is used in the area between the frames along the direction of the normal to the frame boundaries.

(iii) Large panoramic images: The GPU-based mapping approach allows to handle larger panoramic images as the CPU at a negligible loss in render speed. By reducing the area passed to the fragment shader in the optional optimization step mentioned above, the size of the panoramic map does not have any influence on the real-time frame rates at all. Likewise, the influence of the chosen input image size of usually 640x480 on current mobile phones is effectively negligible. A clear limitation for the GPU-mapping is the limited texture size of a mobile phone's GPU, this problem is circumvented by splitting the panoramic texture into several parts at the cost of additional program logic, however.

4 Experimental Results

The image quality is tested by means of the image refinement approach, such as brightness offset correction and blending. The visual appeal is judged from a perceptual point of view for achieving continuous results without seams and artifacts. The robustness of the tracking process of each refinement approach and the render speed performed for every approach is tested.

Panoramic image refinement: Figures 1 and 3 show the results created by the original application [5] and our approach. In the original approach, brightness differences and seams are visible. In our approach, the seams as well as general differences in brightness are smoothed by blending combined with the brightness offset correction [2]. As a result of the smoothing, the image gets a bit blurry. However it emphasizes the impression of one artifact-free image.

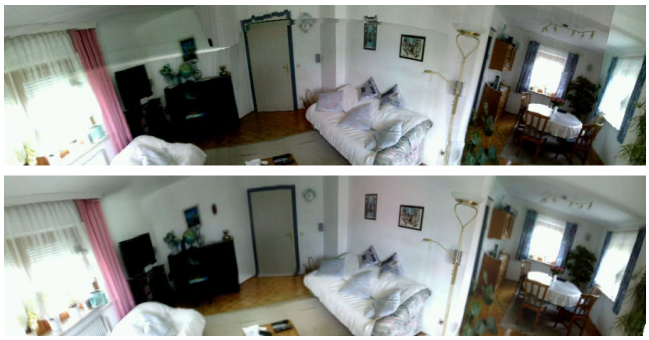

Figure 3: CPU-mapped image by the application of Wagner *et al*. [5] (above), GPU-mapped image with image refinements proposed (below).

Robustness: The quality of the panoramic image has a direct impact on the quality of the tracker estimating the rotational motion, *i.e*. for an increase in image quality, a concurrent increase in robustness is expected. We forward the GPU-mapped images to the tracker and count the number of matching points for the current camera frame. Subsequently this number is compared to the number of key points found using the CPU-mapped image (see Table 1). For the FAST corner detection algorithm, the sharpness of corners is important, thus blending has an adverse influence. Compensating the brightness offset only [2] achieves a better result than those approaches that employ frame blending. On the contrary, strong differences in brightness may force the tracker to loose its orientation and cause the need to relocalize, however. In summary a combination of brightness correction and frame blending brings achieves the most visually appealing results at a negligible loss in matching performance.

Render Speed: We measured the averagely rendered frames per second for each image refinement approach and for different panoramic mapping sizes (2048x512 pixels and 4096x1024 pixels). For the standard resolution of 2048x512 pixels, all image refinement approaches run fluently with a frame rate higher than 20 FPS on a *Samsung Galaxy S2* smartphone, while the frame rate does not significantly degrade for higher resolution panoramic images (*e.g*. 4096x1024 pixels).

5 Conclusion

In this work we proposed a GPU-based approach for mapping panoramic images on mobile phones. Artifacts and brightness seams are eliminated or strongly reduced. Our approach allows for the creation of larger panoramic images in real-time and additional functionality, such as the removal of undesired objects depicted.

References

[1] C. Arth, M. Klopschitz, G. Reitmayr, and D. Schmalstieg. Real-Time Self-Localization from Panoramic Images on Mobile Devices. In *ISMAR*, pages 37–46, 2011.

[2] C. Degendorfer. Mobile augmented reality campus guide. Master's thesis, Graz University of Technology, 2010.

[3] T. Langlotz, M. Zingerle, R. Grasset, H. Kaufmann, and G. Reitmayr. AR Record & Replay: Situated Compositing of Video Content in Mobile Augmented Reality. In *OzCHI*, pages 318–326. ACM, 2012.

[4] E. Rosten and T. Drummond. Machine learning for high-speed corner detection. In *ECCV*, volume 1, pages 430–443, 2006.

[5] D. Wagner, A. Mulloni, T. Langlotz, and D. Schmalstieg. Real-Time Panoramic Mapping and Tracking on Mobile Phones. In *VR*, pages 211–218, 2010.