Graphics hardware accelerated panorama builder for mobile phones

Miguel Bordallo López\textsuperscript{a}, Jari Hannuksela\textsuperscript{a}, Olli Silvén\textsuperscript{a} and Markku Vehviläinen\textsuperscript{b}

\textsuperscript{a}Machine Vision Group, University of Oulu, Oulu, Finland
\textsuperscript{b}Nokia Research Center, Tampere, Finland

ABSTRACT

Modern mobile communication devices frequently contain built-in cameras allowing users to capture high-resolution still images, but at the same time the imaging applications are facing both usability and throughput bottlenecks. The difficulties in taking \textit{ad hoc} pictures of printed paper documents with multi-megapixel cellular phone cameras on a common business use case, illustrate these problems for anyone. The result can be examined only after several seconds, and is often blurry, so a new picture is needed, although the view-finder image had looked good. The process can be a frustrating one with waits and the user not being able to predict the quality beforehand. The problems can be traced to the processor speed and camera resolution mismatch, and application interactivity demands. In this context we analyze building mosaic images of printed documents from frames selected from VGA resolution (640x480 pixel) video. High interactivity is achieved by providing real-time feedback on the quality, while simultaneously guiding the user actions. The graphics processing unit of the mobile device can be used to speed up the reconstruction computations. To demonstrate the viability of the concept, we present an interactive document scanning application implemented on a Nokia N95 mobile phone.

Keywords: Computer vision, image mosaicing, graphics processing unit, mobile device

1. INTRODUCTION

Mobile communication devices have increasingly been equipped with built-in cameras that allow the users to capture high resolution still images as well as lower resolution video sequences. While the cellular phone cameras can be used for any scenes, a typical business application is capturing images of documents ranging from white-board contents and business cards to maps and printed A4/Letter sized sheets. This is typically done in an ad hoc unprofessional manner with the users expecting the capture process being an easy and straightforward one.

However, document imaging using a hand held multi-megapixel camera tends to turn out into a cumbersome and frustrating process. First, the result often lacks desired sharpness due to insufficient focusing or motion blur that are difficult to assess from at most QVGA or VGA resolution view-finder images. Together with shading effects and text line alignment errors, not to speak about the problems caused by the flash, these are fundamental limitations that are unlikely to go away with improving hardware, and will irritate the users even in the future. Second, the interactivity of the process is poor as it may take several seconds after the image capture to view and evaluate the result. Clearly, the usability leaves space for improvement, encouraging to re-consider the application thinking from the mobile device point of view.

The origins of the usability problem can be traced to the mismatches between processor speed, display and camera resolution, and in particular, application interactivity demands. We argue, that as a high degree of interactivity is expected from handheld devices, even document imaging uses should exhibit this characteristic. The speed of processing plays an obvious role, but only after the features of the application fit together with the mobile usage context. For the starters, capturing a high-quality image of a document with a single snapshot is difficult, so an alternative approach is needed.

One approach for acquiring large document images is to capture and stitch together several partial images, providing a possibility to eliminate imaging artefacts such as blurred or excessively shaded regions. In a conventional solution, the captured images are uploaded to a desktop computer for stitching, but during this off-line
process the user often observes imaging problems that are very difficult or impossible to correct afterwards. Obviously, in this area a mobile device based solution can excel, provided that proper means for interaction are identified.

In this paper we present a solution for building mosaic images of printed documents from frames selected from VGA resolution (640x480 pixel) video. The idea is to guide the user through the capture process, while the quality and registration of each frame are automatically assessed before being selected for stitching. In essence, the original single-shot multiple-attempt document imaging approach has been transformed into a highly interactive single-take video application. The proposed system applies on-line camera motion estimation, and allows the user to see real-time preview of already captured areas, enabling to observe stitching problems. Unlike in many conventional solutions, the camera movements are not restricted in any manner, so even a zig-zag style capturing path is possible. The final document panorama image is reconstructed automatically with the help of estimated device motion.

The use of video rather than still image resolution reduces the processor speed and data transfer rate requirements set by interactivity, but they remain at a very high level for low-power mobile devices. Consequently, we need to consider both computationally efficient algorithms and hardware platform solutions. Our preferred option is the utilization of the graphics processing unit (GPU) to perform the most demanding computations needed to construct the image mosaic.

On desktop computers GPUs have been used to accelerate computer vision algorithms and some of the recent mobile phones such as Nokia N95 include a graphics accelerator accessible via the OpenGL ES application programming interface (API). Although the current OpenGL ES 1.1 API supports only fixed function pipelines originally designed to render 3-D graphics, it can be used even in implementing image processing functions. Furthermore, future OpenGL ES versions are likely to provide for more flexibility, which help in implementing and mapping algorithms to the GPU. For the time being, the most obvious operations to be accelerated using OpenGL ES are warps and interpolations.

To demonstrate the viability of the concept, we have implemented an interactive document scanning application on a Nokia N95 mobile phone. The current design employs the OpenGL ES 1.1 interface to stitch captured images together. The approach has been applied to two application scenarios: capturing panorama images and building a document image mosaics. In the following, we highlight the application development challenges and trade-offs that need to be dealt with battery powered mobile devices. We also discuss the observed platform dependent limitations and features in future GPU interfaces that would help in implementing vision based algorithms and solutions.

2. DOCUMENT PANORAMA BUILDER

While not a replacement for flat-bed scanners, the document panorama builder application running on a cellular phone platform is essentially an interactive camera based scanner that can be used in less constrained situations. Compared to flat-bed scanners, the usage concept of the handheld solution differs significantly as the users can’t realistically be expected to capture single shot high-quality images of complete document pages. Instead, our approach relies on real-time user interaction and capturing of VGA resolution images that are registered and stitched together.

In addition to interactivity benefits, the use of low resolution video imaging can be defended from purely technical aspects. In low resolution mode the sensitivity of the camera can be better as the effective size of the pixels is larger, reducing the illumination requirements and improving the tolerance against motion blur. On the other hand, while a single-shot high resolution document image, if captured properly, could be used directly after acquisition, the low resolution video capture approach requires advanced algorithms and significant post-processing effort.

2.1 Interactive user interface

The key usability challenge of a handheld camera based document scanner is enabling and exploiting interactivity. As a solution we have developed a method where the device interactively guides the user to move the device over, for example, a newspaper page in a manner that a high quality image can be assembled from individual
The data flow of the mosaicing application. The interactive capture interface sends images to the frame evaluation subsystem and gives feedback to the user. The best frames are selected. The images are corrected, unwarped and interpolated. The final stage constructs the mosaic.

video frames. The user starts the scanning by taking an initial image of some part of the document. Then, the application instructs him to move the device to the next location. The scanning direction is not restricted in any manner, and a zig-zag style path can be used.

The user guidance can employ a number of techniques in each of which the camera motion is estimated from sequences of image frames using methodology described in. Based on shutter time and illumination dependent motion blur the user can be informed to slow down when the suitable overlap between images has been achieved, say 25 %, and a new image for stitching is selected from among the image frames based on quality assessment. The user can also be asked to back-up or he can return to lower quality regions later in the scanning process. As a result, good images of the document page can be captured for the final stitching stage.

The coarse frame registration information based on motion estimates computed during interactive scanning is employed as the starting point in constructing the mosaic image. The strategy in scanning is to keep sufficient overlaps between stored images to provision for frame re-registration using a highly accurate feature based method during the final processing step.

The panorama capturing procedure is illustrated in Fig. 1. In order to get a final panorama image, the user focuses the camera on the desired starting point document mosaic. To demonstrate the generality of the implementation, the user starts moving the camera while capturing a sequence of images. Rotating the camera may be necessary to avoid and eliminate shadows from the target, and is a practically useful degree of freedom. Each image is individually processed to estimate motion. The blurriness of each picture is measured and eventual moving objects are detected. Based on the quality of each individual frame, a selection process takes place. The idea of selection is to consider only good quality frames for creating the best possible output. Each frame is either accepted or discarded.

The treatment of moving objects depends on the type of the scene that is processed. For every selected frame with natural scenes, if a moving object is present and it fits the sub-image, the image is blended, drawing a seam that is outside the boundaries the object. If only a partial object is present, the part of the frame without the object is the one that is blended. In document scanning the regions with moving objects, typically shadows, are simply discarded.

Fig. 2 a) illustrate the interactive capture interface. The allowed free scanning path is a very nice feature from the document imaging point of view, however, it sets significant computational and memory demands for the implementation and prevents building final mosaics in the fly. We have also developed a panorama application that limits the scanning path into a uni-directional one. With this approach, Nokia N95 phone can be used
to stitch images in real time, the results growing with the frame acquisitions. The memory requirements are smaller as not all selected frames need to be stored till the end of the panorama process.

### 2.2 Registration

After on-line image capturing the registration errors between the regions to be stitched can be on the order of pixels that would been seen as unacceptable artefacts. In principle, it would be possible to perform accurate registration during image capture, but building final document images will in any case require post-processing to adjust the alignments and scales. Because a handheld camera is used, it is very difficult for the user to maintain a constant viewing angle and distance, so the user interaction scheme just targets at capturing the document using free scanning path.

The fine-registration employed for automatic mosaicing of document images is based on a robust estimator (RANSAC\(^4\)) with a feature point detector (SIFT\(^5\)). In addition, graph based global alignment and bundle adjustment steps are performed in order to minimize the registration errors and to further improve quality. Finally, warped images are blended to the mosaic using simple Gaussian weighting. Memory needs are an obvious implementation bottleneck of fine-registration with current mobile devices, limiting the size of the final mosaics and the number of input frames.

Fig. 2 b) illustrates an actual mosaic (1397x1099 pixels, in uncompressed RGB format close to 5MB) constructed for an A4 document page from eight VGA images to show the practical capabilities of a current platform. The used Nokia N95 cellular phone provides only 16 MB of memory for applications so using high resolution 5 Megapixel images instead of VGA frames would not make sense, unless the single snapshot strategy was selected. It should be noticed that with the lower resolution frames the registration and blending errors are easy to see and reveal the eventual shortcomings of the methodology.

![Figure 2](image)

**Figure 2.** An example is illustrated for building a large document page image. (a) An interactive user interface helps to acquire good quality initial images. One possible scan style is zig-zag scanning. (b) A final mosaic obtained.

### 2.3 Quality determination and frame selection

To ensure the quality of the final stitched document image and to conserve memory, the best individual frames are selected based on motion and blur measures\(^6\) computed in-real time to enable feedback to the user. The motion blur in the overlapping part of each frame is evaluated with summed derivatives of each row and column.\(^7\) The blur calculation produces a single number that measures the amount of high-frequency detail in the image, and makes sense if it is used to compare images that in our case are the regions in overlap. If the summed derivatives for image \(I_a\) are larger than for image \(I_b\), both depicting approximately the same scene, \(I_a\) is sharper.
As the differences in the image content may distort the results, the accuracy of motion estimates used for preliminary registration needs to be a reasonable one. In practice, this is a computational cost, interactivity, and quality trade-off. The better the accuracy, the less overlap is needed between frames, with reduced computing requirements. However, the eventual shadows add a complication that need to be taken into account.

Taking pictures of documents with a hand-held camera is often hampered by the self-shadow of the device, present as a moving region in the sequence of frames. To detect these moving objects, the difference between the current frame and the previous frame is calculated via subtraction. The result is a two-dimensional matrix that covers the overlapping area of the frames, that is then low-pass filtered to remove noise, and thresholded to produce a binary motion map. If the map contains a sufficient amount of pixels classified as motion, the dimensions of the moving object is estimated to guide in frame selection.

In practice, the regions with moving objects, whether they are shadows or anything else, are not desirable when stitching the final document image. Instead of developing advanced methods for coping with these phenomena, we mostly count on the user interaction to avoid the problems from harming the reconstruction result. During document scanning the frame selection is performed based on a score from blur measurements and motion detection, always trying to select the best possible frame while discarding the others to conserve memory.

### 2.4 Computing power requirements

The computing requirements of the document imaging are quite significant for battery powered mobile devices, although the application can be broken down into the interactive, real-time frame capture part, and the non-interactive final mosaic stitching post-processor. In the order of decreasing computational cost, the most expensive parts of the VGA frame based document scanning application are:

1. warping the frames with interpolation before final stitching (approx. 2 frames/s in software)
2. Registration with SIFT feature based RANSAC to find the warping and stitching parameters (approx. 6 frames/minute in software)
3. Quality determination and frame selection during the capture stage (approx. 10 frames/s in software)
4. Motion estimation during capture stage (approx. 30 frames/s in software)

The application has been implemented using only fixed point arithmetic to achieve good performance on most devices. For the capture stage the speed is about 8 frames/second on Nokia N95 (ARM11 processor) in VGA resolution mode. The speed of moving the camera is actually limited by the motion blur. The post processing stage operates at about 4-5 frames per minute, and depends both on the available memory resources and processor speed.

In the remaining sections of this paper we consider speeding up the application using the graphics processing resources. On PC platforms more than an order of magnitude speedups have been achieved so this is an attractive direction for investigation.

### 3. GRAPHICS PROCESSOR AS A COMPUTING ENGINE

Computing using Graphics Processing Units (GPU) to perform computationally intensive tasks has become popular in scientific applications. As GPU computing is very suited for parallel processing. It is also a very interesting candidate for accelerating the document mosaicing application. Fig. 3.3. depicts a typical graphics pipeline. In our case, the GPU is useful as a co-processor to execute certain functions, while employing its resources is most conveniently and portably done with a standard graphics API. On a mobile device platform the choice is essentially limited to OpenGL ES, while the emerging OpenCL Embedded Profile is likely to offer flexibility similar to vendor specific solutions, such as CUDA of Nvidia.
3.1 OpenGL ES

OpenGL (Open Graphics Library) is a multiplatform standard defining a cross-language cross-platform API used for producing 2D and 3D scenes from simple graphic primitives such as points, lines and triangles. OpenGL ES (OpenGL for Embedded Systems) is in turn a subset of the OpenGL 3D graphics API designed for embedded devices such as mobile phones, PDAs, and video game consoles. Currently, there are several versions of the OpenGL ES specification. OpenGL ES 1.0 is drawn up against the OpenGL 1.3 specification, while OpenGL ES 1.1 and OpenGL ES 2.0 are defined relative to OpenGL 1.5 and OpenGL 2.0 specification, respectively. OpenGL ES 2.0 is not backwards compatible with OpenGL ES 1.1.

Currently, OpenGL ES 1.1 has been implemented in many mobile phones, some of which, such as Nokia N95, include GPU hardware. Newer phones on the market have support for OpenGL ES 2.0, increasing the capabilities of the GPU as an image processing unit.

Several OpenGL ES profiles exist on each specification with support for fixed-point or floating-point data types. The fixed point types are supported due to the lack of hardware floating-point instruction sets on many embedded processors. Many functionalities have been reduced on OpenGL ES 1.0 from the original OpenGL API, although some minor functionalities have been also added. In comparison to OpenGL 1.0, the 1.1 adds support for multitexture with combiners and dot product texture operations, automatic mipmap generation, vertex buffer objects, state queries, user clip planes, and greater control over point rendering. The rendering pipeline is of fixed-function type.

In practice, these features of OpenGL ES 1.1 provide for possibilities of using the graphics accelerator as a co-processing engine. General purpose image processing capabilities are available via texture rendering. The image data can be copied a texture to the graphics memory, then allowing the application of several matrix transformations and performing bilinear interpolations for the rendered texture.

On the other hand, OpenGL ES 1.1 has several limitations. The most important one is that the GPU is forced to work in single buffer mode to allow the readback of the rendered textures. Furthermore, the overheads of copying images as textures to graphics memory result in significant slowdowns. Other limitations include the need to use power of two textures or the restricted types of pixel data.

OpenGL ES 2.0, eliminates most of the fixed-function rendering pipeline API in favor of a programmable one, and a shading language allows programming most of the rendering features of the transform and lighting pipelines. However, some limitations are remaining. The images must still be copied the GPU memory in a matching format and the lack of shared video memory causes multiple accesses to the GPU memory to retrieve the data for the processing engine.

3.2 CUDA

CUDA is a parallel computing architecture for NVIDIA graphics processing units. Although CUDA supports several higher level application programming interfaces, including OpenGL and DX11, it can be used as such.
To code algorithms for execution on the GPU, C language programs, with some extensions for CUDA, can be written and then compiled on a PathScale Open64 C compiler. The main advantage of the CUDA model lies on the use of a shared video memory among all the processing cores and threads that avoid multiple accesses to the video memory to retrieve the same data. These features have been borrowed to the emerging OpenCL API, that in time will improve the efficiency of mobile platforms. Fig. 4 depicts the differences of the computational models for parallel computing.

4. GPU ACCELERATED PANORAMA BUILDER

Traditional approaches to mosaic building algorithms use to follow a sequential path with multiple accesses to the memory from the processing unit. GPU approaches require an understanding of the data nature and the careful tailoring of the algorithms. Every overhead included by the several needed copies of the data from the input camera to the processing engines have to be taken into account while mapping the operations in the most appropriate way.

Current mobile phones on the market have not yet taken into account the use of GPU as general purpose capable processors. Image processing algorithms that use the camera as the main source for data lack on fast ways of data transferring between processing units and capturing or saving devices. Each step of the mosaicing algorithm has then to be evaluated separately in order to know the best way of organizing the data and reduce the overheads to obtain the best possible performance.

4.1 Image warping with interpolation

Mosaic stitching requires the correction of each selected frame with a warping function, that must interpolate the pixel data to the coordinates of the new frame. This costly process can be done in a straightforward manner on several steps using any fixed or programmable graphics pipeline. The warping process can be mapped onto a GPU, offering a remarkable improvement in processing time. Our application includes the implementation of a warping and interpolation subsystem based on the OpenGL ES 1.1 API.

To map the warping algorithm properly on the GPU, the camera frame data must be reordered to suit the specific model of OpenGL textures. After it, the data should be copied to the video memory. A two triangles quad matching the desired rendering surface that will be copied to the destiny image must be created. The vertices can then be transformed applying rotations and warpings. While the quad is textured, bilinear interpolations...
are for each pixel can be calculated in parallel on the GPU. The rendering surface should be copied back to the main memory as a native bitmap.

A HW accelerated implementation of VGA frame warping offers processing times of less than 100ms. per frame on a N95 phone. This figure includes all the overheads. The software implementation on the device's CPU is about four times slower, while its overheads are much smaller.

The implementation using OpenGL ES 2.0 could simplify the process avoiding the rendering of the quad and applying the transformations directly on the texture through a programmable shader. Overheads due to the reorder of the data could then be avoided causing some minor speedups.

The CUDA and OpenCL models allow the access to several pixels in parallel while performing the warpings and the interpolations. The use of shared memory among threads, reduces the overhead caused by the readbacks from the GPU memory. CUDA also notably simplifies the concurrent execution of GPU and CPU code, leaving the main processor free to perform the tasks that cannot be mapped onto the GPU.

4.2 Pixel blending
Pixel blending is an operation that has been traditionally mapped well on a GPU due to the highly parallelization that can be achieved by mixing two textures. The process of pixel blending can be performed on an OpenGL 1.1 capable GPU through the rendering of several quads with different values of transparency. Linear interpolations for several pixels can be performed in parallel. Unfortunately, the overheads included as data copying and vertex calculations make the process quite inefficient.

OpenGL 2.0, is able to apply transformations to the pictures with no need for quad rendering. The values of the texture pixels and a mask can be combined on a single stage. The overheads are then reduced considerably. The CUDA and OpenCL models present some advantages if the algorithm is carefully tailored to perform several operations such as warping and blending at the same stage. The pixel data does not need to be accessed several times from the graphics memory and the improvements on the speed could be very significant.

4.3 Image registration
OpenGL ES 1.1 fixed pipeline interface does not offer many possibilities to accelerate the image registration process. On the other hand, the implementation of this system might be accelerated using OpenGL ES 2.0 API and a shader programming language capable of executing on a HW accelerated platform.

Part of the computations of a feature machine based registration should be moved to the GPU through the use of programmable thread shaders, present already in OpenGL 2.0. SIFT feature extraction is suitable to be highly parallelized and therefore accelerated on the GPU. To complete the registration process, RANSAC estimator should be executed on the CPU in parallel with the feature extraction. Desktop implementations of the system suggest that the CPU load can be reduced more than a 50%, and that feature extraction times on VGA frames can be reduced about 10 times.

The CUDA and OpenCL models can outperform the approaches that can be taken on OpenGL stream-based processing. A RANSAC implementation on CUDA could be used on image registration together with a feature extractor on a GPU remarkably reducing the CPU load.

4.4 Frame quality evaluation
Frame quality evaluation is a multiple-step process that involves blur detection, motion estimation, and a scoring system. The computational requirements of this stage are not high if a simple approach is taken. However, OpenGL ES 1.1 API could transfer part of the blur detection process to the GPU through the manipulation of data as textures and the combination of them through blending functions. Unfortunately, no speedup is achieved due to overheads from readbacks. The programmable pipeline of OpenGL ES 2.0 enables using shaders programming in implementing blur detection in a similar way as the feature extraction method. Similar speedups can be expected.
5. FUTURE DEVELOPMENTS

Emerging mobile phone graphics platforms are intended to use the graphics processing unit as a general purpose computation device. They will offer the flexibility required to speed up computer vision algorithms. OpenCL (Open Computing Language) is a language for programming heterogeneous data and task parallel computing across GPUs and CPUs. It was conceived by Apple Inc. (which holds trade mark rights), and established as standard by Khronos Group in cooperation with others, and is based on C99. The purpose is to recall OpenGL and OpenAL, which are open industry standards for 3D graphics and computer audio respectively, to extend the power of the GPU beyond graphics (General Purpose computation on Graphics Processing Units, GPGPU).

OpenCL defines a set of functions and extensions that should be implemented by hardware vendors. Vendors should provide the compiler and other tools to allow the execution of OpenCL code on their specific hardware. OpenCL implemented on an embedded system will allow the distribution of tasks with highly parallel programming through all the processing units present on a chipset (CPU, GPU, DSP, ...)

A document panorama application will greatly benefit from a standard like OpenCL that enables to distribute the different tasks on each step onto available processing units. Each specific part of the code could be compiled to be executed concurrently. This implies that a good understanding of the data and possible parallelizations becomes the crucial step for successful designs.

6. SUMMARY

We have presented a system for building document mosaic images from selected video frames on mobile phones. High interactivity is achieved by providing a real-time feedback of motion and quality, while simultaneously guiding the user. The captured images are automatically stitched together with good quality and high resolution. The graphics processing unit of the mobile device is used to speed up the computations. Based on our experiments, the use of the GPU clearly accelerates performance and helps in implementing user interface and imaging applications. However, the GPUs on mobile device platforms are a rather recent add-on, primarily intended for displaying graphics. In the future, emerging graphics platforms such as OpenCL will offer more flexibility for image processing.

REFERENCES