Optical Tracking and Video See-Through AR on Consumer Cell-Phones

Mathias Möhring, Christian Lessig, and Oliver Bimber
Bauhaus University
Bauhausstraße 11, 99423 Weimar, Germany,
E-Mail: {mathias.moehring, christian.lessig, oliver.bimber}@medien.uni-weimar.de

Abstract: We present a first running video see-through augmented reality system on a consumer cell-phone. It supports the detection and differentiation of different markers, and correct integration of rendered 3D graphics into the live video stream via a weak perspective projection camera model and an OpenGL rendering pipeline.

1 Introduction

Mobile devices such as head-mounted displays (HMDs) in combination with wearable computers or compact personal digital assistants (PDAs) have become popular platforms for video see-through augmented reality applications. While the HMD technology is still not widely spread outside the research community today, PDAs are much more prevalent. Yet, they are not nearly as common as cell phones. It has been estimated that by the end of the year 2005 more than 50% of all cell phones will be equipped with digital cameras [10]. Consequently, using cell phones as a platform for video see-through AR has the potential of addressing a brought group of end users and applications. Compared to high-end PDAs or HMDs together with powerful personal computers, the implementation of video see-through AR on the current fun- and smart-phone generation is a challenging task: Low video-stream resolutions, little graphics and memory capabilities, as well as slow processors set technological limitations. Although we are aware of the fact that future phone generations will set new boundaries and open more possibilities, we have realized a first prototype solution for video see-through AR on a present consumer cell phone.

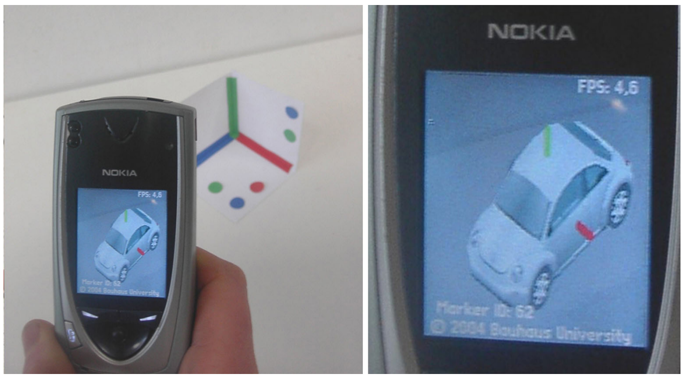


Figure 1: Video see-through example on a consumer cell-phone.

Technically, it supports optical detection of passive paper markers and the correct integration of 2D/3D graphics into the live video stream at interactive rates. Conceptually, it allows early experiments with such a technology for augmented reality applications. We believe that cell phones have the potential to bring AR to a mass marked. Our prototype was evaluated on a Nokia 7650 smart phone (cf. figure 1). It contains a 104MHz ARM processor without floating point unit, 4MB RAM, and runs Symbian OS 6.1.

2 Related Work

Video see–through AR is concerned with two main tasks: the detection of the camera's pose in the environment, and the rendering of synthetic models on top of a live video stream. Much previous work has been done on addressing these tasks throughout AR community. Marker-less tracking [2,7,19,20] is surely the preferred method, but it is computationally expensive, requires some information about the environment, and might apply several cameras or additional sensors. More often, artificial markers are used, to determine position and orientation of the camera within the environment [8,11,23]. A wide range of marker-based tracking and detection systems exists – each with its own advantages and disadvantages [24]. Many of them follow a pattern-matching approach and require many resources that prevent them from being used efficiently on low-end mobile devices, such as PDAs and cell-phones. Currently, most groups focus on wearable computers and head-attached displays for supporting mobile AR applications [3,14,15]. So far, only a few steps have been made towards hand-held devices. In [4,5,6,13] prototypes are described that use PDAs to augment a live video stream with synthetic information. However the necessary detection and registration computations are done on a powerful remote server to which the PDA is connected via WLAN or GSM. The first stand–alone AR application on a PDA was developed by TU Vienna using ARToolKit [22]. Their application can be run in two modes: One with a remote server carrying out most of the computations, and the other one with the PDA doing all computations itself. They achieve frame rates of around 5 fps in the remote mode, and 2.5 - 3.5 fps in the local mode, on a PocketPC, with a 400 MHz Intel XScale processor, 64MB RAM, an IEEE 802.11b wireless network connection, and a video resolution of 320x240 pixels. The markers used for their experiments had a 15x15cm base area and were tracked at a distance of up to 2 meters. Siemens has developed an AR game [18] for their camera phones that determines the pixel-flow within the video images for estimating the users' relative lateral (horizontal and vertical) motions. The video stream is then augmented with synthetic 2D objects. However, marker-less or marker-based tracking is not supported since the camera's pose is not recovered. Consequently graphics cannot be integrated into the video stream with a correct 3D perspective. Beside lateral motions, longitudinal (depth) motions cannot be detected. As mentioned above, the number of phones equipped with cameras and color displays is going to increase in the following years. Mainly driven by the game market, a lot of new technologies are emerging around these devices. OpenGL for embedded systems and mobile devices, as an example, was created by the Khronos group [9]. Mobile phone manufacturers, such as Nokia have now derived their own device specified versions. We apply a pre-release version of Nokia's OpenGL port (Nokia Graphics Library, NLG) for our prototype.

3 Marker Detection

The implementation of a pattern-matching algorithm on today's consumer phones would be constrained by several technological issues:

(1) The low resolution of live video streams provided by the phone's frame grabbing hardware (in our case: a resolution of 160x120 pixels, 12bits color depth provides a frame rate of 16fps; a resolution of 320x240 pixels drops the frame rate down to approximately 0.5fps). For pattern-matching, the camera-marker distance is mostly derived from the detected edge length of the marker. Doing this with a resolution of 160x120 pixels would either require inapplicably large markers, or would result in an unacceptable tracking quality.

(2) The high optical distortion of the phones' simple cameras. These cameras have been designed for taking low-quality pictures or videos. Optical distortion, such as barrel distortion is high. It varies strongly for each phone due to small variations in manufacturing – even if the same phone type is considered. To undistort these effects would require additional performance and memory capabilities, and –which is more critically for real-world applications– the end user to perform an individual camera calibration for his/her phone.

(3) The limited processing and memory capabilities of today's consumer phones (in our case: a 104MHz ARM processor without floating point unit, and a total of 4MB RAM for applications and data). For pattern-matching, a reasonable amount of memory is required to store the pattern information of different markers, as well as a decent amount of computational power for comparison and image transformation steps. Due to the lack of a floating point unit, all computations for image processing and rendering have to be performed on an integer level.

Although, with upcoming phone generations, most of these issues might be addressed to allow the implementation of a pattern-matching-based system in future, our goal was to realize a first running system on one of today's consumer phones.

The requirements for this are an acceptable marker detection quality and performance of normal-sized markers (e.g., with a 10x10cm base area) at a camera-marker distance that is suitable for phone-based applications (e.g., up to 1.5m – 1.8m) with a static and unchangeable camera focus. In addition, the need for calibration steps by a novice end-user has to be kept at a minimum. These requirements have to be met under consideration of the technical constraints described above. We decided to implement a simple, low-cost marker detection algorithm which is not based on pattern-matching, but is adapted more to the phones' capabilities. Our goal is to render three-dimensional graphical content perspectively correct into the video stream. If neither external nor internal camera parameters need to be known, the detection of four non-coplanar points on the marker is sufficient for a weak perspective projection camera model to estimate the camera-to-image plane transform [16]. Our non-pattern-matching-based approach is resource economical (performance and memory) and does not require a camera calibration.

3.1 Marker Design

For the reasons described above, we want to detect four non-coplanar points on the marker. This implicates a 3D marker design. Our layout has the shape of a common Cartesian coordinate system with three axes (cf. figure 2). The four basis points are then the end points of each axis and the origin where all axes intersect. The axes of the marker are defined by colored line features, which can be detected easily in the camera image. We decided to retain colors in the video frames instead of binarizing them to encode more information than possible with the pure feature geometry. In addition to line features, colored blob features are placed on the marker's faces. They are used as a barcode to encode the marker's ID. We have experimented with different marker designs. Our initial layout resembles a concave or convex (as in [5]) half-cube with three faces (cf. figure 2a). The concave marker is better suited if placed in corners than the convex version. A later design uses only the bottom face and free-standing vertical axis (cf. figure 2b). The advantage of the first layout is that a larger barcode can be encoded on two more faces. The advantage of the second design is that it supports a 360 degree surround view around the vertical axis. This is because all base points are visible simultaneously from most perspectives. The only impossible views are those, in which the vertical axis occludes one of the others.

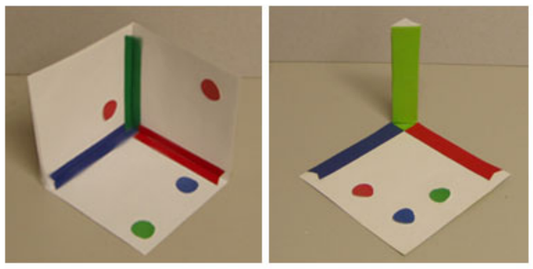

(a) (b)

Figure 2: Two different marker designs.

As for many other approaches, our markers are self-made and are built from paper. Thus, they are easy to create with a color printer.

3.2 Finding the Base Points

To speed up the image analysis, inter frame information is used whenever possible: As soon as the marker has been detected in one frame, a search window is defined that constrains the image processing in the following frame (cf. figure 3a). Initially, this window covers the entire image area. If the maker cannot be found in *n* consecutive frames, the search window has to be initialized. Searching for features within the search window is done by tracing horizontal or vertical scan lines in every *m*th pixel row or column[1] until two consecutive intersections with a feature are determined. The size of *m* is estimated heuristically, based on the maximum image resolution and the size of the search window which is proportional to the size of the projected marker. Due to our marker design, the detected feature must be the edge of the most horizontal marker axis. A feature is detected based on a relative increase in one of the R/G/B channels – also allowing to identify the specific axis. This is done with the following conditions:

$$\begin{aligned} R:&\quad (T \le r) \,\&\, (C_{rg} g \le r) \,\&\, (C_{rb} b \le r) \\ G:&\quad (T \le g) \,\&\, (C_{gr} r \le g) \,\&\, (C_{gb} b \le g) \\ B:&\quad (T \le b) \,\&\, (C_{bg} g \le b) \,\&\, (C_{br} r \le b) \end{aligned} \tag{1}$$

Each color channel [r,g,b] of a pixel is compared with the corresponding other two channels after scaling them with an individual color coefficient [$C_{rg}, C_{rb}, C_{gr}, C_{gb}, C_{bg}, C_{br}$]. These coefficients approximate the camera's relative color response and are estimated during the color calibration step (section 3.4). A general threshold T is used in addition for ensuring a high enough color response in each channel. Note, that wrong features might be detected (e.g., the colored blobs at the lower left in figure 3a). At this stage, we are not able to differentiate between features that belong to the marker, and those that do not. Thus, all of them will be processed in the same way. Wrong features, however, will be identified in the following steps, because they do not lead to a valid marker layout or do not satisfy pre-defined criteria (such as a relative minimum edge length, connectivity to already detected lined features, etc.). In most cases, wrong features are ignored and the scanning process is continued. Some cases, however, in which features are classified as wrong very late (e.g., because they resemble correct features), will make the marker detection fail for the particular frame. This causes the search window to be initialized.

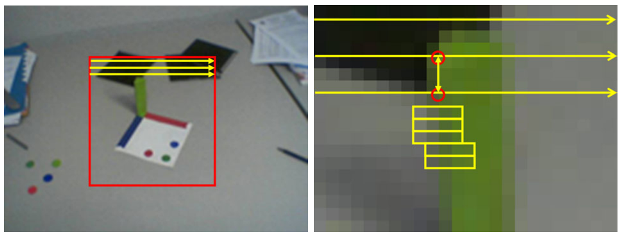
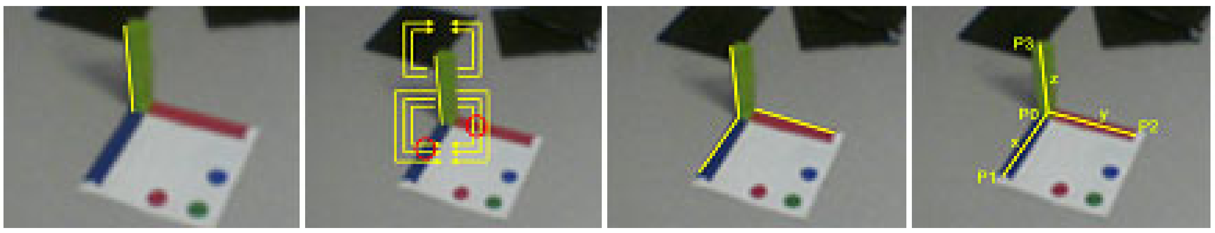

(a) (b)

Figure 3: (a) Straight scan lines in two-dimensional search window. (b) Feature intersections and one-dimensional search windows to trace edges.

The slope of the line that connects the two feature intersections is computed next. It gives a first estimate of the gradient of the line-feature's edge. The edge is traced in both directions using a one-dimensional search window that is normal to the gradient (cf. figure 3b) until the two endpoints are reached. The result is the edge and the endpoints of the most vertical marker axis in the image (cf. figure 4a).



(a) (b) (c) (d)

Figure 4: (a) Edge of most vertical axis. (b) Rectangular scan tracks to find attached axes. (c) Edges of all axes before correction. (d) Centers of all axes.

The other two axes have to emerge from one of the detected endpoints. To find them, we apply scan tracks that orbit the endpoints in rectangular paths with increasing radii until two consecutive intersections on two different line features are found (cf. figure 4b). This can only be the case for one endpoint. As for the first axis, the slopes between the intersections are computed to estimate the gradients of the edges which are used to trace them in both directions until their endpoints are reached. The results are the edges of the remaining axes and their four endpoints (cf. figure 4c). During the edges are traced, their average width is determined and continuously updated. This allows to estimate the center of the axes and to correct the position of the three line features (cf. figure 4d). Finally, the three endpoints which intersect are merged into a single one. The remaining four endpoints of the line features represent the base points P_0, P_1, P_2 and P_3 of the marker coordinate system.

3.3 Reading out the Barcode

Using the pixel projections of the base points on the image plane $[u,v]_{Pi}$, $0 \leq i \leq 3$ allows reading out the encoded marker ID. The colored blobs are placed at know positions on the faces of the marker. Each two axes span one face over three known points (a subset of the base points). These points allow to address each surface point on the corresponding face in a normalized two-dimensional coordinate system. Thus, the positions of the blobs can be described as Barycentric coordinates and linear interpolation on the projected face can be applied to estimate each blob's projected pixel positions in the image (cf. figure 5):

$$[u,v]_B = [u,v]_{P0} + \begin{bmatrix} [u,v]_{P1} - [u,v]_{P0} \\ [u,v]_{P2} - [u,v]_{P0} \\ [u,v]_{P3} - [u,v]_{P0} \end{bmatrix} \times \begin{bmatrix} u \\ v \\ w \end{bmatrix} \tag{2}$$

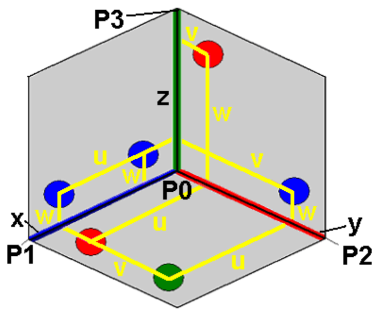


Figure 5: Estimating the blob positions analytically by linear interpolation within the projected face-space (one of the known parameters *u, v* and *w* has to be 0 to indicate the correct face).

To address lens distortion, a blob's position is computed analytically, and a small search region around the computed pixel position is scanned to look up its color. This is done for each blob – in a predefined face- and blob-order. The acquired colors represent states of single barcode elements whose ranks are defined by the order of the readout. In our current implementation, each blob can have one of four possible states (or colors): absent (white), red, green, or blue. For the three-face-marker design and a maximum of four blobs per face, as an example, a total number of 4^{3*4}= 16,777,216 different IDs can be encoded theoretically.

Reading out the barcode does not have to be done continuously, since it is unlikely that a different marker is visible in every consecutive frame. Thus, it is sufficient to look for a new marker in every *n*th frame.

3.4 Color Calibration

The detection of line and blob features is based on relative increases in the R/G/B channels of each pixel. The decision whether a pixel is red, green, blue, white, or anything else is done heuristically based on an approximated camera response function that is parameterized with the color coefficients for each color channel (equation 1).

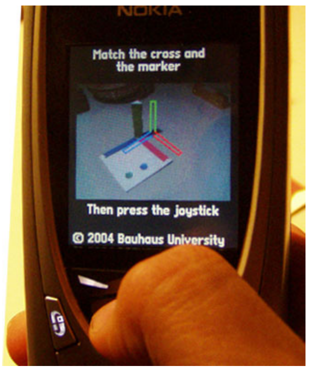


Figure 6: Interactive color calibration step.

These values (and consequently the success of the feature detection) depends on the absolute color of the features (which may vary if printed with different color printers), from extreme changes in the lighting situation, and from the camera response. To obtain good estimates, the color coefficients are initialized through an interactive color calibration routine. As illustrated in figure 6, the user has to match an outline of the marker coordinate system rendered on top of the video stream with the perspective on the real axes shown in each frame. Once aligned, a button has to be pressed to take color samples which are used to estimate the coefficient values as follows: Three sample pixels (one in the center and two at the outer extends) on each marker axis (red, green and blue) are taken out of n consecutive frames. The ratios between the sample pixels' color channel of the same frame leads to the corresponding color coefficients for this frame. As an example, the color coefficient that allows to compare red with green is computed with $C_{rg} = r/g$, and so on. This results in $3*n$ possibilities for each of the six coefficients. We select the highest value for each coefficient among all frames to make equation 1 less sensitive to noise.

4 Perspective Rendering

Our goal is to render three-dimensional graphics perspecively correct (i.e., relative to the marker, depending on the perspective of the camera) into the displayed live video stream. Since neither intrinsic nor extrinsic camera parameters are known, we apply an affine object representation presented by Vallino and Kutulakos [21]. They do not estimate the absolute position and orientation of the camera, but interpolate each pixel representation of 3D scene vertices with respect to the marker's projection on the image plane. This method allows being fully independent of the external and internal camera parameters – and consequently from underlying hardware and optics. Knowing the base point's world coordinates $[x,y,z]_{Pi}, 0 \le i \le 3$ and their projections on the image plane $[u,v]_{Pi}, 0 \le i \le 3$, two vectors χ, ϕ that span the image plane within the affine coordinate system can be determined as follows:

$$\begin{aligned} \chi &= \left[u_{P1} - u_{P0}, u_{P2} - u_{P0}, u_{P3} - u_{P0}\right] \\ \phi &= \left[v_{P1} - v_{P0}, v_{P2} - v_{P0}, v_{P3} - v_{P0}\right] \end{aligned} \tag{3}$$

The direction vector of the camera ς is normal to the image plane and can be computed through the cross-product of χ and ϕ:

$$\varsigma = \phi \times \chi \tag{4}$$

These three vectors are composed into a 4x4 matrix that maps normalized frame coordinates (i.e., normalized 3D vertices within the frame defined by the axes spanned through the base points) into window coordinates (i.e., 2D pixles) via interpolation on the image plane:

$$P = \begin{bmatrix} & \chi^T & & u_{P0} \\ & \phi^T & & v_{P0} \\ & \varsigma^T & & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix} \tag{5}$$

Essential for the applicability of this method is that the base points are non-coplanar. They define an affine coordinate frame into which the model has to be transformed first. This normalization step is realized with a simple scaling matrix:

$$NF = \begin{bmatrix} \frac{1}{P1-P0} & 0.0 & 0.0 & 0.0 \\ 0.0 & \frac{1}{P2-P0} & 0.0 & 0.0 \\ 0.0 & 0.0 & \frac{1}{P3-P0} & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix} \tag{6}$$

(with $P1 = [0,0,0]$, $P1 = [x,0,0]$, $P2 = [0,y,0]$, and $P3 = [0,0,z]$)

The multiplication of the scene vertices that are normalized to the affine coordinate frame with the χ and ϕ matrix-rows of P results in the interpolated pixel positions of their projections within the video stream. Due to the lack of the extrinsic camera parameters, the absolute distance from each vertex to the camera cannot be computed. These values are necessary for performing z-buffering. However, the multiplication of the normalized vertices with the ς row of P results in relative depth ratios that are sufficient for a depth test. Thus, they can be used for z-buffering. The affine object representation provides both, an acceptable accuracy for rendering 3D scenes via a weak perspective projection, and fast projection computations via simple interpolations. Moreover this method can be fully integrated into OpenGL's transformation pipeline. The multiplication of P results already in window coordinates (i.e., in pixel positions). However, it is the viewport transformation's task to map normalized device coordinates (i.e., normalized to a range of $[-1,1]$) into window coordinates. In standard OpenGL implementations, the viewport transformation cannot be disabled for such a special usage of the transformation pipeline. A solution to this problem is to apply its inverse which neutralizes the effect of the following viewport transformation. Thus, window coordinates are mapped to device coordinates, and back to window coordinates. The inverse of the viewport transformation consists of a scaling followed by a translation:

$$ND = \begin{bmatrix} \frac{2}{max\_u} & 0.0 & 0.0 & 0.0 \\ 0.0 & \frac{2}{max\_v} & 0.0 & 0.0 \\ 0.0 & 0.0 & \frac{1}{2 * obj\_size} & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix} * \begin{bmatrix} 1.0 & 0.0 & 0.0 & -1.0 \\ 0.0 & 1.0 & 0.0 & -1.0 \\ 0.0 & 0.0 & 1.0 & 0.5 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix} \quad (7)$$

The coefficient *obj_size* is reasonable for scaling depth values to the desired range. It depends on the maximal absolute z extend of the model in world coordinates. The composition of all three matrices can then be copied onto OpenGL's matrix stack, as illustrated in figure 7.

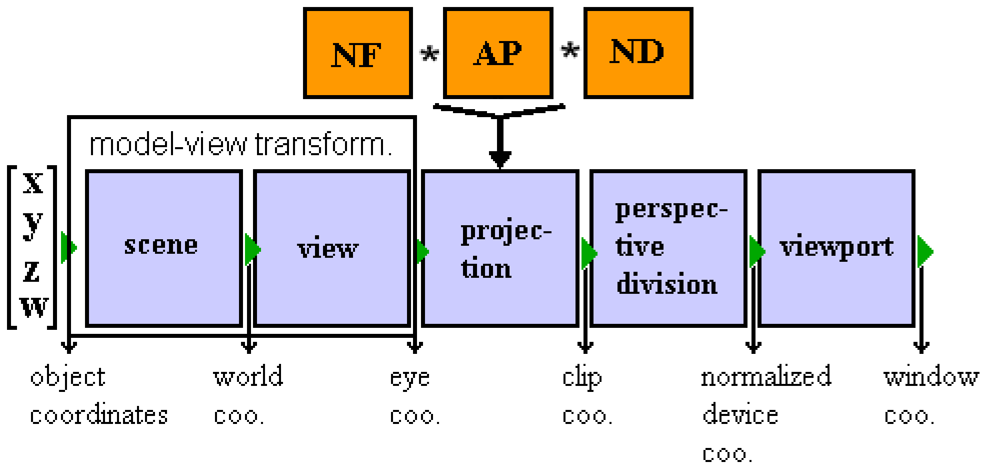


Figure 7: Integration of affine object mapping into the OpenGL transformation pipeline.

Note that the perspective division is not required and will not be performed in this case. Having the transformation pipeline set up as described above, the entire three dimensional scene (including geometry, light sources, clipping planes, etc.) will automatically be mapped correctly into the video stream, as shown in figure 1.

5 Results and Discussion

We have described a first running system that supports video see-through augmented reality on a current consumer cell-phone. It is adapted to the technological constraints of today's phones, such as low video resolution, limited computational performance and memory, and high optical distortion of integrated cameras. We aim at applications, such as interactive tour guiding for museums and tourism, as well as at mobile games. The choice of a weak perspective projection camera model and an affine object representation does not require to know the camera's intrinsic and extrinsic parameters in terms of rendering three-dimensional graphics perspectively correct into the video stream. This might be a disadvantage for some applications, because the position of the camera relative to the marker remains unknown. However, this approach can deal with the high optical distortion and the low image resolution without requiring the end user to perform any camera calibration step. The application of resource-intensive pattern-matching approaches is not necessary (and probably not possible on today's consumer-level phones) to support such tasks. Markers can be differentiated by reading out a barcode instead of matching them against –in some cases many– different sample patterns. As described earlier, our phone's frame grabber delivers 16 images per second with a resolution of 160x120 pixels and a color depth of 12 bits. Detecting the marker and reading the barcode requires approximately 25% performance - thus we achieve an average frame rate of about 12fps. The performance of the rendering step depends on the complexity of the scene. In the example shown in figure 1, the Beetle consists of 404 triangles, 6 textures and was Gouraud-shaded with a single point light source. The final frame rate which includes frame grabbing, marker and barcode detection, as well as rendering is 4-5fps. The marker size is 10x10x10cm and can be detected in a range of 30cm – 1.5m. However, the phone's camera and consequently the marker detection are sensitive to the environment lighting. The light radiated by phosphor tubes, for instance, cases a strong interference pattern (visible horizontal lines) in the video images, which in most cases causes the image processing to fail. Since we detect colored features instead of binarized ones, varying lighting situations can cause color shifts. If these shifts become too extreme our relative color increase heuristic does not hold. In such a case better results can be achieved by estimating new color coefficient parameters by performing the color calibration step. In our current implementation it is not possible to detect more than one marker at once, but we believe that our approach can be extended to support this. This belongs to our future work. he detection of two dimensional markers is possible, if the application is only interested in getting a rough idea about where the user is located. If no graphics has to be rendered perspectively correct, the barcode of the marker which is in the camera's viewing range is sufficient to provide a spatial awareness. This is similar to the application of infrared emitters that broadcast their IDs within a limited working range. Since in this case a much lower performance is required than for a 3D video see-through application, a higher image resolution can be used for marker detection. This results in a higher detection reliability. As it is the case for PCs, the graphics capabilities of cell-phones are clearly driven by the game industry. Vendors, such as ATI and nVidia, already offer the first 3D graphics acceleration chips for cell-phones [1,12]. Autostereoscopic displays are available for off-the-shelf phones [17] for viewing graphics in 3D. The processors of phones are becoming continuously faster and memory restrictions like today will become history. It is just a matter of time, until the integrated cameras of cell-phones are not only used for taking pictures of videos, but also as interaction devices. In this paper we demonstrated a first approach to a traditional augmented reality topic. We believe that many more applications will follow – also outside a pure AR scope.

Acknowledgements

A pre-release version of OpenGL ES / NGL and technical support were provided for this project by Nokia Research.

References

[1] ATI Imageon. Retrieved from WWW: www.ati.com/products/imageon2300/index.html, 2004.

[2] Comport, A.I., Marchand, É., Chaumette, F., A real – time tracker for marker-less augmented reality. In proceedings of IEEE and ACM International Symposium on Mixed and Augmented Reality, pp. 36-45, 2003.

[3] Feiner, D., MacIntyre, B., Höllerer, T., and Webster, T. A touring machine: prototyping 3D mobile augmented reality systems for exploring the urban environment. In proceedings of International Symposium on Wearable Computers, pp. 74-81, 1997.

[4] Fründ, J.; Geiger, C.; Grafe, M.; Kleinjohann, B.: The augmented reality personal digital assistant. In proceedings of International Symposium on Mixed Reality, 2001.

[5] Gausemeier, J., Fruend, J., Matysczok, C., Bruederlin, B., and Beier, D. Development of a real time image based object recognition method for mobile AR-devices. In proceedings of International Conference on Computer graphics, Virtual Reality, Visualisation and Interaction in Africa, pp. 133-139, 2003.

[6] Geiger, C., Kleinjohann, B., Reimann, C. Stichling, D. Mobile Ar4All In proceedings of IEEE and ACM International Symposium on Augmented Reality, pp. 181-182, 2001.

[7] Genc, Y. Riedel, S., Souvannacong, F., and Navab, N. Marker-less tracking for AR: A learning-based approach. In proceedings of IEEE and ACM International Symposium on Mixed and Augmented Reality, pp. 295-304, 2002.

[8] Kato, H. and Billinghurst, M. Marker tracking and HMD calibration for a video-based augmented reality conferencing system, In proceedings of IEEE International Workshop on Augmented Reality, pp. 125-133, 1999.

[9] Khronos Group. OpenGL|ES, Retrieved from WWW: www.khronos.org/, 2004.

[10] MIT's Technology Review. Markets and Trends, p. 16, February 2004.

[11] Naimark, L. and Foxlin, E. Circular data matrix fiducial system and robust image processing for wearable vision-inertial self-tracking. In proceedings of IEEE and ACM International Symposium on Mixed and Augmented Reality, pp. 27-36, 2002.

[12] nVIDIA. GoForce, Retrieved from WWW: www.nvidia.com/page/handheld.html, 2004.

[13] Pasman, W. and Woodward, C. Implementation of an augmented reality system on a PDA. In proceedings of IEEE and ACM International Symposium on Mixed and Augmented Reality, pp. 276-277, 2003.

[14] Piekarski, W., Gunther, B., and Thomas, B. Integrating virtual and augmented realities in an outdoor application. In proceedings of IEEE International Workshop on Augmented Reailty, pp. 45-54, 1999.

[15] Reitmayr, G. and Schmalstieg, D. A wearable 3D augmented reality workspace. In proceedings International Symposium on Wearable Computers, pp. 165-166, 2001.

[16] Shapiro, L., Zisserman, A., and Brady, M. 3D motion recovery via affine epipolar geometry. Int. Journal of Computer Vision, vol. 16, no.2, pp. 147-182, 1995.

[17] Sharp. Optical Imaging Technology, Retrieved from WWW: www.sle.sharp.co.uk/research/3d/, 2004.

[18] Siemens, Virtual mosquito hunt. Retrieved from WWW: w4.siemens.de/en2/html/press/newsdesk_archive/2003/foe03111.html, 2004

[19] Striker, D. and Kettenbach, T. Real-time and marker-less vision-based tracking for outdoor augmented reality applications. In proceedings of IEEE and ACM International Symposium on Augmented Reality, pp. 189-190, 2001.

[20] Vacchetti, L., Lepetit, V., and Fua, P. Fusing online anfd offline information for stable 3D tracking in real-time. In proceedings of Conference on Computer Vision and Pattern registration, vol. 2, pp. 241-248, 2003.

[21] Vallino, J., Kutulakos, K.N. 2001. Augmented reality using affine object representations. Fundamentals of Wearable Computers and Augmented Reality, pp. 157 – 182, ISBN: 0-8058-2902-4, Lawrence Erlbaum (publisher).

[22] Wagner, D., and Schmalstieg, D. First steps towards handheld augmented reality. In proceedings of International Conference on Wearable Computers, pp. 127-136, 2003.

[23] Zhang, X. and Navab, N. Tracking and pose estimation for computer assisted localization in industrial environments. In proceedings of IEEE Workshop on Applications of Computer Vision, pp. 214-221, 2000.

[24] Zang, X., Fronz, S., Navab, N. Visual marker detection and decoding in AR Systems: A comparative study. In proceedings of IEEE and ACM International Symposium on Mixed and Augmented Reality, pp. 97-106, 2002.