AUGMENTED REALITY IMPLEMENTATION METHODS IN MAINSTREAM APPLICATIONS

David Procházka, Tomáš Koubek

Abstract

PROCHÁZKA, D., KOUBEK, T.: *Augmented Reality Implementation Methods in Mainstream Applications*

Augmented reality has became an useful tool in many areas from space exploration to military applications. Although used theoretical principles are well known for almost a decade, the augmented reality is almost exclusively used in high budget solutions with a special hardware. However, in last few years we could see rising popularity of many projects focused on deployment of the augmented reality on different mobile devices. Our article is aimed on developers who consider development of an augmented reality application for the mainstream market. Such developers will be forced to keep the application price, therefore also the development price, at reasonable level. Usage of existing image processing software library could bring a significant cut-down of the development costs. In the theoretical part of the article is presented an overview of the augmented reality application structure. Further, an approach for selection appropriate library as well as the review of the existing software libraries focused in this area is described. The last part of the article outlines our implementation of key parts of the augmented reality application using the OpenCV library.

augmented reality, OpenCV, template matching, ARToolkit, computer vision, image processing.

1 Introduction

The general principle of the augmented reality (AR) is embedding digital information into the real world scene. Thus it is a step between virtual reality and the real world. The embedded information is usually based on the content of the scene. A selected real object could be augmented by a virtual object or completely replaced. Well-known examples are the presentation of the fighter status report on the head-up display before a pilot or a navigation information projected on the wind shield of a car.

Applications based on the augmented reality have been used for decision making process support for many years. Military solutions for field operations made the pioneering work in this area – from mentioned fighter head-up displays to tactical suits for troopers. Moreover there is a number of applications for construction and maintenance of complex systems. Especially space devices, planes and helicopters. A number of these solutions is outlined in Ong – Nee (2004). The discussed applications are usually developed for a single specific purpose. Especially from this reason, they are rather expensive. Therefore, they are used just for saving lives or speed-up production or maintenance of very expensive devices.

However, in last few years AR applications have been emerging into other areas – design, medicine and even consumer electronics. Development of AR applications for this field is discussed in this article. It is obvious that one of the most important prerequisites for the success in this area is the price. This price is given by the price of the hardware (which is nowadays usually quite cheap) and the development costs.

The development costs could be significantly cut-off using different image processing software libraries. AR applications are working on similar well-know principles. For a number of complex algorithms is therefore possible to use an existing implementation (image preprocessing, edge detection, etc.). The goal of this article is to present these libraries potential for development of such mainstream AR applications and clearly outline the general AR application structure including the

implementation in a selected library.

The section 2 shortly describes the augmented reality hardware and examples of its usage. In the section 3 is outlined the key problem – detection of an object in an image. The section 4 is a short review of the existing software libraries for image processing. In section 5 is presented our implementation of key parts of an augmented reality application.

2 Hardware used for augmented reality solutions

Our reality could be augmented in many ways. Widely spread are for example audio navigation tools for visually impaired people. However, in the following review we will focus especially on a visual augmentation of the reality. This visual augmentation could be divided into three main categories. The first one is based on usage of the head mounted displays. The other group is based on projectors. This kind of augmented reality is called *spatial augmented reality*. The last category is based on common displays (tablets, cell phones, etc.).

Currently used head mounted displays are based on the optical composition of the scene or on the video composition. The optical composition is a projection of artificial objects on a semi-transparent screen before user eyes. The video composition combines an image from a camera with artificial objects and the result presents on a small LCD screen in a virtual helmet. These semi-transparent screens are suitable especially for applications when a camera signal blackout could be critical (fighters, troopers, etc.). On the same principle there are in fact based head-up displays in cars with status and navigation information. The most important problem of this solution is the exact overlaying of a real and digital object.

The solution is in the usage of the video composition based device. The image from the front-side camera is analysed, position of the real object is found and this object could be seamlessly replaced. Drawbacks of this solution are usually higher head mounted display size and weight, limited field-of-view and the price.

Applications based on them are usually from the category discussed in the beginning of the article – special high budget solutions. However, these solutions are not suitable for a common customer.

The other group of products – a solution based on projectors – is significantly growing in last years. A computer is analysing the scene using a camera and searching for predefined objects. If the found, the attached projector is able to augment directly the real world object surface by the given digital information. A quite common example of this *spatial augmented reality* is the adaptive projector used for projection on heterogeneous surfaces. The projected image is controlled by the software and adjusted to compensate the differences between the anticipated and the real image. A thorough description of this technology could be found in Bimber – Raskar (2005).

The last group of devices are solutions based on different screens. This area has been growing in the fastest way in last years. It is given especially by a huge emerge of advanced cellphones and tablets. These devices have all necessary components: a suitable display, a high resolution camera, a processor fast enough to make the real-time image analysis and also a GPS accompanied by compass. One of the most popular AR application of this kind is project *Layar*[1]. From the technical point-of-view, it is a video composition based application merging the camera image with additional information from map layers stored inside the device.

The cellphones and other portable devices present a platform with a significant economical potential. The number of users is in comparison to previously mentioned categories incomparable. According to the research done by the Garther agency, 62 million *smartphones* with ability to run such AR applications was sold only in second quarter of the year 2010. The rapidly growing market of tablets founded by the Apple's *iPad*[2] is also important.

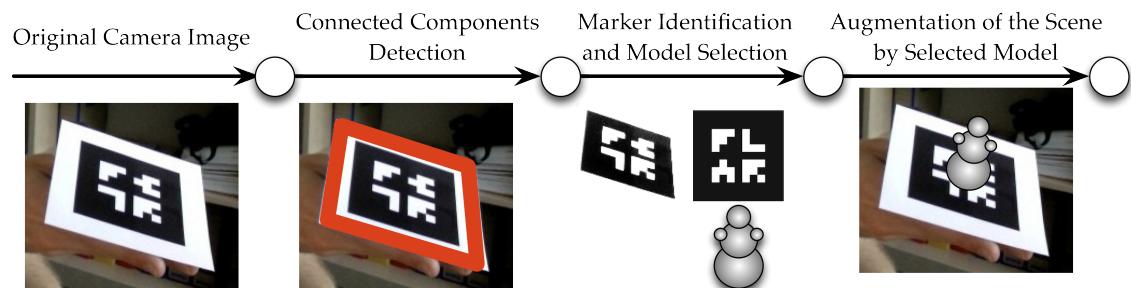[1] http://www.layar.com/
[2] http://www.apple.com/ipad/

2.1 Comparion of AR implementations

Although the presented output devices are completely different, there is a number of common principles. The first key problem is the identification of the screen before the user. For this purpose could be used solely image processing (searching for known objects) or there could be used other position techniques. There is frequently used the triangulation from a cellphone network, Wi-Fi hotspots, the GPS or some inertial sensors. At the moment the scene before the user is identified, it is necessary just to insert appropriate information. These first two steps are based on well-known common principles described later.

A significant difference is in the method of presentation to the user. However it is just a question of the used hardware. Software architecture of the systems is usually very similar.

3   Object detection methods

The general structure of any application based on image processing is following: We will acquire an image from a camera and store it into an inner representation (a kind of RGB color matrix). Further we will make an image analysis and identify a possible wanted object, its position and orientation. This potentially wanted object is compared with a predefined pattern or patterns. In the case of success, the last step is insertion of the artificial object. This process is illustrated on the fig. 1. This process is quite common for all AR applications.

1: Scheme outlining the basic functionality of the augmented reality application.

Another significant difference is especially in the step of comparison of the possible desired object with the patterns. It depends whether searched object is a face, a natural object (building), a simple shape (window) or an artificial marker (usually black and white square with a predefined pattern). In the following part of the article we will focus on the artificial marker. This case is quite simple and could be the first step in building of a robust AR application.

3.1 Artificial marker detection

An example of the marker detection process could be described by following steps. The whole process is also outlined on Fig. 2. Our presented method is not the only possible solution, however it is widely used by many well tested applications.

As been already mentioned, from an input device a color image is taken and stored into the inner representation of the image processing library. Further, this image is transformed into the gray scale. It is possible to make a standard conversion of all color channels or prefer a specified color channel (e.g. green). The grey value now presents the brightness of the pixel (hence an object).

Such image contains a number of objects – markers, persons, furniture, etc. For performance improvement it is necessary to remove most of these objects from the image. This step is usually done via thresholding. Simple thresholding could be generally described using following formula:

$$g(i,j) = \begin{cases} 0, & f(i,j) \leq P \\ 1, & f(i,j) > P \end{cases} \tag{1}$$

where function $f(i,j)$ is the source image (brightness of the pixel), value P is the threshold and $g(i,j)$ is the result image. Value of the threshold is usually determined according to the scene content.

The gist of the thresholding is transformation to a bitmap in such way that allows to remove most of unnecessary objects. A well chosen thresholding method

could significantly improve the performance of the application. Each object that is not filtered out is a potential marker and therefore must be tested as described further. If its possible, it is recommended to prepare also an appropriate testing environment. Suitable lighting and high contrast markers could significantly simplify the preprocessing phase. Generally, a homogeneous controlled environment allows to wipe out most of the unwanted objects (se e.g. Dutta – Chaudhuri (2009)). In case of complex lighting conditions, different adaptive thresholding techniques are usually used.

The result of the previous step is an image with number of vertices and edges. The other step is detection of connected components with the required shape. This could be done using an image morphology algorithm. This algorithm produces a list or tree of image components. For identification of potential markers it is sufficient to browse this list of objects and test whether the provided entity fulfils the given criterion. In case of a common white square marker with a black inner square with a pattern, it is an entity with four vertices with an inner entity again with four vertices.

As soon as we have potential marker vertices, they must be compared with predefined patterns. For this comparison there must be done marker pixels transformation between a marker plane and a camera plane (in other words: perspective distortion must be eliminated). Equation 2 describes this transformation. If defined transformation matrix is applied on point $[x_m, y_m, z_m]$ in marker plane, we will receive the position of this vertex in the camera plane. By inversion of this process we could receive the original vertex position before perspective distortion (generally we will receive the original object shape). Elements $T_1 - T_3$ represent a translation vector. Elements $R_{11} - R_{33}$ represent well known 3×3 rotation matrix (see Neider et al. (2007), p. 806). Calculation of the transformation matrix elements is described in Šťastný et al. (2011). By this step we have restored the original shape of the object and it is possible to make comparison with the marker patterns.

$$\begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix} = \begin{bmatrix} R_{11} & R_{12} & R_{13} & T_1 \\ R_{21} & R_{22} & R_{23} & T_2 \\ R_{31} & R_{32} & R_{33} & T_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_m \\ y_m \\ z_m \\ 1 \end{bmatrix} \tag{2}$$
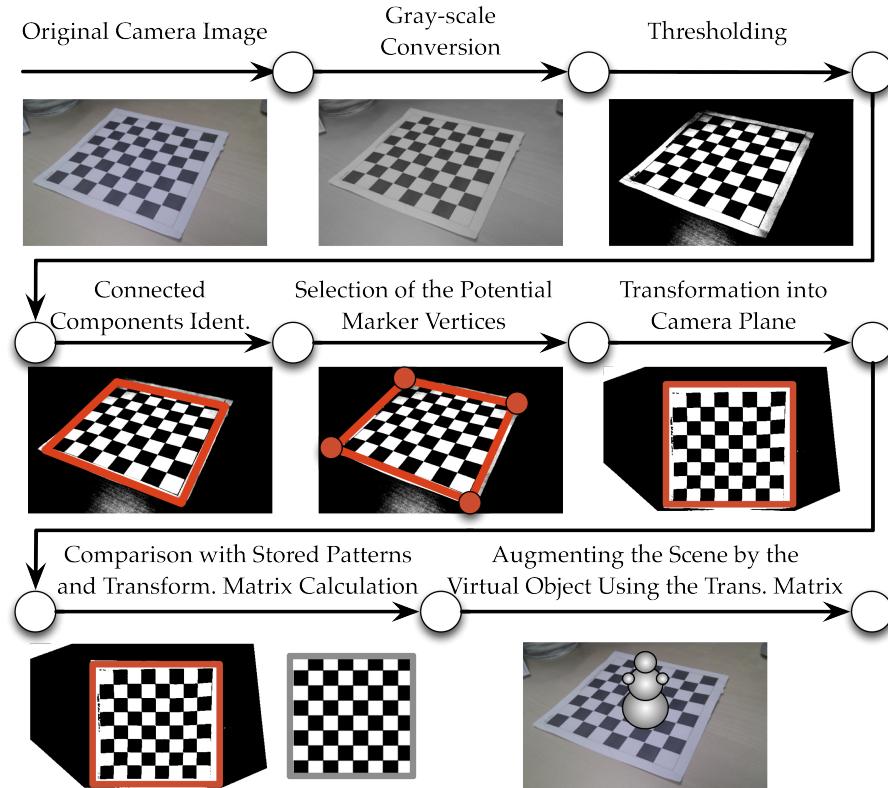
A precondition of a successful transformation is a camera calibration step. The camera calibration matrix describes optical properties of a given device and compensates also possible optical errors. This calibration matrix is calculated for a given camera only once. Therefore, the performance of the application is not affected. The application of calibration matrix is described in 3. Its structure and calculation is outlined in Kato – Billinghurst (1999).

$$\begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix} = \begin{bmatrix} C_{11} & C_{12} & C_{13} & C_{14} \\ C_{21} & C_{22} & C_{23} & C_{24} \\ C_{31} & C_{32} & C_{33} & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} R_{11} & R_{12} & R_{13} & T_1 \\ R_{21} & R_{22} & R_{23} & T_2 \\ R_{31} & R_{32} & R_{33} & T_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_m \\ y_m \\ z_m \\ 1 \end{bmatrix} \tag{3}$$

The last step – marker identification – is the most time consuming operation. It is a correlation calculation between the potential marker image and the pattern. In case matching is above the given limit, images are taken as corresponding. For the correlation calculation could be used a number of methods. From neural networks to least squares algorithm. Selected approaches are described in Kato et al. (2003). Methods implemented directly in *OpenCV* library could be found in Bradski – Kaehler (2008) on page 214. Each potential marker must be tested against all patterns using a selected method until all patterns are tested or the appropriate pattern is found. It is obvious that computation complexity grows linearly with the number of patterns. That is the reason why in many applications there are used special markers with patterns given by algorithms such as the well known Golay

error code.

In case there is a corresponding pattern, the application will store the transformation matrix that defines the orientation of the marker in the scene. This matrix is the homomorphy matrix described before.

2: Stages of object detection and augmenting of the scene.

3.2 Conclusion of marker identification process

Basic principles of the AR applications are quite similar, as is obvious from the description outlined above. The difference is mostly in particular algorithms for image comparison. That is the reason why it is not effective to implement the application from the scratch, but to use an existing library that supports the mentioned well-known algorithms. The following part of the article describes an appropriate library selection.

4   Methodics of image processing library selection

Before library selection for our AR application it is necessary to specify exactly the functionality which is required. For a huge number of applications, following criteria are important:

1. Required programming language support: Fulfilment of this criterion could be complicated. Most of the libraries support just C/C++. An exception is the *OpenCV* library that supports also the Python language and *NyARToolKit* supporting many mainstream languages.

2. Required platform and architecture support: This problem is obviously the most limiting criterion. Especially in case application is targeted on mobile devices. On personal computers there is limited only support of 64 bit architecture that is necessary for complex applications.

3. Project is under active development: There is a huge number of already unsupported projects or projects with very limited numbers of active developers and users. Usage of such library is not recommended. Support of new architectures, input devices, etc. is an essential feature of any library.

4. Documentation: This aspect is in the beginning of the projects frequently underestimated. However, developers will be facing a number of situations where is utmost important to understand thoroughly the implemented method. It is not enough to know that this method or function if proving some correlation coefficient, it is necessary to know exactly how the algorithm works.

5. Provided functions number: Again, it is quite an obvious requirement. It is recommended to consider the application future development. To base the project on a library with limited functionality could be in long term point-of-view expensive.

Further section briefly reviews selected frequently used image processing toolkits.

4.1 OpenCV

OpenCV[3] (*Open Computer Vision*) is an open source toolkit for the real-time image processing. *OpenCV* is the most robust solution among all frameworks in comparison. It supports C++, C and Python languages. It consists of tools for image analysis, image transformations, camera calibration, stereo vision and also tools for simple graphical user interface and more.

OpenCV is associated with *Intel* company, which implements the support for *OpenCV* into hardware. This library uses *Intel Integrated Performance Primitives* that provides high performance for low-level routines for sound, video, speech recognition, coding, decoding, cryptography etc. *Intel Threading Building Blocks* is used for parallel processing.

The library is a cross-platform, there are versions for GNU/Linux, Microsoft Windows and Mac OS X, 32b and 64b systems. A big advantage of this project is that it is still in development. In December 2010 there was released version 2.2. The development of a new version is in progress. The community of *OpenCV* users is huge, there are a lot of manuals, tutorials and discussion forums. Basics of *OpenCV*, mathematical principles of image processing and their implementation are described by Bradski – Kaehler (2008).

Easy usage of cameras is very useful for augmented reality applications development. This framework uses resources of operating systems and communication with hardware drivers is fully provided as well. If a camera driver is installed in an operating system, it can be initialized and used immediately. Disadvantage of this library is that there are not implemented any direct methods for marker registration. As shown below, for this purpose it is necessary to link several different *OpenCV* functions.

---

[3]More information on: `http://opencv.willowgarage.com`

4.2 ARToolKit

ARToolKit[4] is a software library for development of augmented reality applications. It is a cross-platform. There are versions for GNU/Linux, Microsoft Windows, Mac OS X and SGI, but officially only for 32b versions of the systems. This framework implements some basic tools for marker registration. This is more straightforward in comparison with the *OpenCV*. Implemented methods are robust. The library supports C language. The community of *ARToolKit* users is not as big as *OpenCV* community, but still quite large.

Working with cameras is more complicated than in the case of *OpenCV*. However, a more substantial problem is that development of a free version of this framework was stopped and it continues only for the paid version called *ARToolKit Professional*[5]. The last free version was released in February 2007 and since then no further update has been released. The development is stopped and probably no new version will be released in the future. Compatibility with new versions of the operating systems or other features is not guaranteed. These can be found in the paid version, which is very costly. The development license costs 4995$ a year, use of the framework in third party applications costs 995$ a year. The development of a commercial version is currently uncertain, because the last major changes of *ARToolKit Professional* were made, according to the documentation, in 2007. The library is still maintained.

4.3 NyARToolKit and derivates

NyARToolKit[6] was derived from *ARToolKit* version release 2.72.1 (the last version of free *ARToolKit*). Nowadays is *NyARToolKit* developed by Japanese author. In fact, it is derivated from *ARToolKit* for other platforms and languages (*ARToolKit* is for C language only). There is a version for Java, Flash, Android, Silverlight

---

[4]http://www.hitl.washington.edu/artoolkit
[5]http://www.artoolworks.com
[6]http://nyatla.jp/nyartoolkit

(*SLARToolKit*), Actionscript (*FLARToolKit*), or C#. C++ version is in beta stage (no support for 3D or cameras). There are still some restrictions, i.e. documentation is in Japanese but project is active and other shortages can be removed in next versions. The community will be crucial.

4.4 Other AR libraries

Other frameworks for working with the AR are *Morgan*, *DART*, *Goblin XNA* or *Studierstube*. Mostly these are university projects, but they are not so recent or their support is only marginal, as well as their user community is only limited. *Studierstube* is worth of noticing, because of the best documentation (among university projects). It is developed by Institute for Computer Graphics and Vision at Graz University of Technology. At first *Studierstube* has been used for collaborative AR, later was focused on mobile applications. This library uses other frameworks for tracking, video and registration.

It is advisable to check the development of *Goblin XNA*. It is a Columbian University project, derived from the project *Goblin* with added support for Microsoft XNA framework. *ARTag* is used for marker tracking. Connection with XNA helps to bring augmented reality applications on the Microsoft Xbox platform. However, it is also available for GNU/Linux and MacOS X.

4.5 Choice of appropriate library

OpenCV, *ARToolKit (Profesional)*, *NyARToolKit* and *Studierstube* have some pros and cons, but they belong to better ones. But none of these fulfilled completely specified criteria for the framework choice. All frameworks are cross-platform. Support for 64b is provided for *OpenCV*, *NyARToolKit* and *ARToolKit Professional*. *OpenCV*, *ARToolKit (Professinal)* and *Studierstube* have a lot of documentation (*OpenCV* has both official documentation and literature).

OpenCV has the best stability of development among all frameworks. New

versions are released often and a lot of mistakes are corrected and a new features are added. For other projects, continuation of the development is uncertain. *NyARToolKit* publishes updates approximately in six-month intervals, the last in April 2011. *Studierstube* released the latest version of the framework two years ago.

The choice of a library is not clear at all, because it depends on particular needs of the project. However, for production deployment *NyARToolkit*, *ARToolKit* and *Studierstube* are not appropriate. The big disadvantage is the uncertain or stopped development. These libraries have big potential, but there should be some issues in future, especially new architecture or operating system compatibility etc. The development of the *ARToolKit Professional* is also questionable. The project website was updated at 2011, but only version 2.72.1 (from 2007) is avalaible for free.

The choice of an appropriate library is a very import decision at this point. From our point-of-view, there are two options – *ARToolKit Professional* and *OpenCV*. The first one is ready for development augmented reality applications, but it is very expensive. *OpenCV* has a good documentation, user support and it is free, but not ready for AR applications development. The user has to implement a method for marker tracking or recognition. However, the *OpenCV* providea a lot of image processing functions which can be useful later. We decided to use the *OpenCV* for development of our AR application.

5 Detection of artificial marker in OpenCV environment

As we described earlier, the *OpenCV* library does not implement any direct methods for identifying and registering artificial markers in space. But it provides a lot of methods for image analysing and image processing, which can be used for implementation of marker recognition and registration. For testing of the AR applications *OpenCV* offers functionality for finding of the special type of marker – chessboard. The use of chessboard marker makes development of simple AR applications much easier. Next section describes main ideas for finding of a artificial

marker. The method details can be found in Bradski – Kaehler (2008), in *OpenCV* reference manual on the project homepage and in code examples distributed within the installation package.

5.1 Finding of artificial marker vertices

As mentioned in section 3, firstly the image has to be converted to gray-scale and thresholded. Gray-scaling is done by function `cvCvtColor` which converts *OpenCV* internal format to different color spaces. This conversion is possible to color spaces RGB, CIE Luv, CIE Lab, HLS, HSV, YCrCb and CIE XYZ. One parameter of `cvCvtColor` specifies between which two color spaces the image is converted. To threshold image function `cvAdaptiveThreshold` can be used. Its output is a binary image. The threshold value, threshold methods and their settings are specified in the function parameters.

Next step of our analysis is finding of edges and vertices. *OpenCV* provides several functions which implement different algorithms for these actions. For edge detection is available i. e. Canny edge detector or Hough transformation. Mentioned methods are described at Wang – Fan (2009), respectively Duda – Hart (1972). Another option is *OpenCV* function `cvFindCountour`. The process described below works with well-known artificial marker – a black rectangle within a white field, with a unique picture (can be compared with template).

Our application uses the `cvFindContour` function to find contours in the thresholded image. There are two kinds of contours – an inner and outer contour. All contours are represented by an *OpenCV* structure called `cvSeq` and is stored into the special storage. Some parameters of `cvFindContour` are important for the whole algorithm. Parameter `mode`, in function `cvFindContour`, means, how the found contours will be organized. If `mode` is set to `CV_RETR_CCOMP`, all contours are organized into two level hierarchy – parents and children (it is possible to obtain also a list or tree of contours).

All contours are stored in the internal storage, and we must decide, which contours belong to a particular marker. At first, we can approximate polygon from contours by function `ApproxPoly`. It is an important step in finding a marker, bacause polygons have some additional attributes, i.e. attribute `total`, that expresses amount of lines of a polygon.

Choice of right contours is crucial. We can use attributes of `cvSeq` and attributes of polygons. We declare, that a contour is a marker, if it consists of 4 lines (being the attribute of polygon – determining outer contours of the black rectangle), if has a child (being the attribute of `cvSeq` – determining inner contours of black rectangle) and the child consists of 4 lines. These conditions determine the edges of the marker. Each contour is the input for structure `CvSeqReader`. This is used for finding vertices of markers. For this purpose there is called macro `CV_READ_SEQ_ELEM`. This algorithm output is a marker vertices quaternion. It is used for transformation of the coordinate system (see 2, step 6). Now we have a quadruple of points that describe the vertices of a potential marker.

5.2   Transformation of vertices coordinates into camera plane and image matching

For matching of the found object with an image template it is necessary to compensate its perspective transformation. For computing of the transformation matrix, which describes this marker pose and position, function `cvFindHomography` can be used. Its input is the matrix of points which represents the marker template and matrix of points which represents the marker in the image (this is just a square of given size). Their transformation is described by the matrix in equation 2. It is also used for compensation of marker rotation. For this `cvWarpPerspective` is used. This function makes inverse operation for transformation which is described by the matrix (for inverse transformation is necessary to use flag `CV_WARP_INVERSE_MAP`). The output of this function is an image without perspective transformation. This
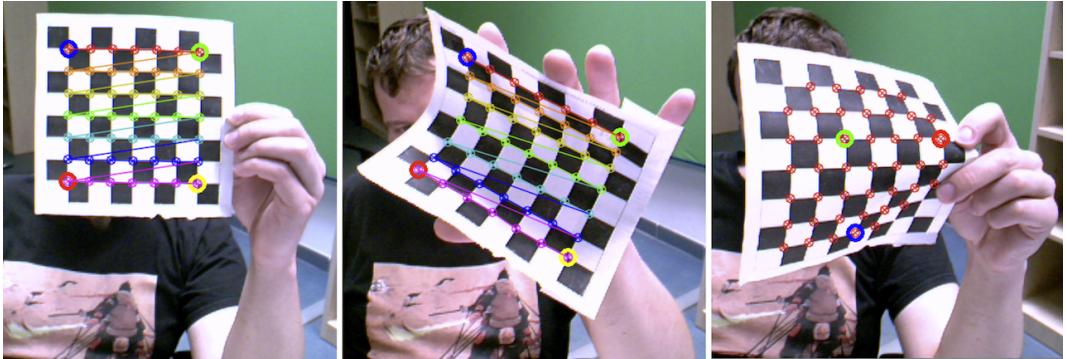
image can be matched with the template.

For comparison of two images *OpenCV* provides own implementation of a template matching method, described by Brunelli (2009). It is implemented in function `matchTemplate`. The template slides through the image, compares the overlapped patches against the template using a specified method and stores the comparison results to a matrix (result value expresses probability of the template presence in the image and depends on the matching method). For finding the template position in the image function `minMaxLoc` must be called. This function finds the position and value of global minimum and maximum in the result matrix.

Surely the match `matchTemplate` method is not the only solution for pattern recognition. There could be used a number of methods including different clustering methods (see e. g. Fejfar et al. (2010)) or neural networks (image processing applications outlined e. g. in Prochazka et al. (2011)).

5.3 Finding chessboard marker

In this section there is implemented detection of chessboard vertices. This is a special case of the issue described in section 5.1. For this chessboard vertices detection *OpenCV* includes function `cvFindChessboardCorners` which finds all inner corners of the chessboard. The input of this algorithm is the chessboard size – amount of the inner corners (height and width of the chessboard). There are few methods for finding the corners. Function `cvFindCornerSubPix` can be used for a more precise determination of the corner position. After this, the position of all inner corners of the chessboard is stored into an array of *cvPoint*. Vertices, which represent the four edge vertices of the marker are on these array coordinates: `0`, `boardWidth-1`, `(boardHeight-1)*boardWidth` and `(boardHeight-1)*boardWidth+boardWidth-1`, number `boardWidth` represents the amount of inner chessboard corners in rows and `boardHeight` represents the amount of corners in columns. This algorithm output is a marker vertices quaternion. These are the corners coordinates of the chessboard

marker in the image. These corners are highlighted in fig. 3 by big circles. Other found corners are highlighted by smaller circles. There is an apparent robustness for rotation and bending of the chessboard. A loss of inner corners occurs at big deformation of the marker.

3: Example of chessboard marker detection.

Discussion and conclusions

The implementation clearly shows that the *OpenCV* library is able to implement same functionality as the well-know *ARToolkit* and other libraries based on this project. This is a key issue for many developers considering the AR application development. According to our experience, the *OpenCV* is more feasible solution than the *ARToolkit*, despite more complicated beginnings.

The augmented reality applications for common users are emerging area. Low user-friendliness is a crucial problem of the development, although there is a significant research in this area for almost two decades. For instance, there are no standard approaches for user interface design, even despite the fact these are commonly used for a desktop and mobile applications design – see Saffer (2010), Kryštof (2009) and many others. One of the reasons is that many metrics and design patterns are not applicable to AR applications development. Assimilation of new metrics and patterns is a significant challenge. In future work we want to focus on this area.

Summary

As been outlined in our article, all augmented reality applications work on similar theoretical principles. The crucial difference is in a position/orientation detection and in a template matching approach.

Only satellite navigation systems, compasses and motion sensors are used in a part of applications. However, these applications usually do not allow representation of complex graphics objects. The other group of applications use the image analysis to specify the information about orientation of the user. The base of these applications lies in composition of a signal from a camera and a digital information. In our article we deal with this area. We chose the problem of an identification of an artificial marker for illustration.

The functionality of the application based on the image analysis was defined in the section 3.1 as the ability of: image reading from an input device, its preprocessing (gray scale transformation, thresholding), segmentation on continuous objects, vertex detection, compensation of geometric distortions and the comparison with given patterns. The list of these processes could be supplemented by an ability to insert the model/information to the image. Its implementation depends on the architecture of the application (if OpenGL, Microsoft DirectX or other API is used). The aim of this section is to explain comprehensibly basic principles of the image processing in AR applications.

The basic implementation scheme, which is briefly described in section 5, presents clearly the implementation of discussed problems using the *OpenCV* library. The usage of presented solutions can reduce development time. Discussed projects (and especially the *OpenCV* project) are open-source. Therefore, it is possible to modify or extend the unsatisfactory existing implementations of the methods. On the basis of our experience with discussed libraries, we recommend to use the *OpenCV* library. The significance of the project, open source codes, high-quality implementation, available documentation and also the wide community and support of Intel

company are indicators of project quality and to some measure guarantee of further development.

Acknowledgements

References

BIMBER, O. – RASKAR, R., 2005: *Spatial Augmented Reality: Meeting Real and Virtual Worlds*. Wellesley, Massachusetts, USA : A K Peters. ISBN 1-56881-230-2.

BRADSKI, G. – KAEHLER, A., 2008: *Learning OpenCV: Computer Vision with the OpenCV Library*. USA : O'Reilly Media. ISBN 0-596-51613-4.

BRUNELLI, R., 2009: *Template Matching Techniques in Computer Vision: Theory and Practice*. USA : Wiley. ISBN 978-0-470-51706-2.

DUDA, R. O. – HART, P. E., 1972: Use of the Hough transformation to detect lines and curves in pictures. *Commun. ACM*. 15, 1, pp. 11–15. ISSN 0001-0782.

DUTTA, S. – CHAUDHURI, B. B., 2009: Proceedings of the World Congress on Engineering and Computer Science 2009. In *Proceedings of the World Congress on Engineering and Computer Science*, San Francisco, CA. ISBN 978-988-18210-2-7.

FEJFAR, J. – WEINLICHOVÁ, J. – ŠŤASTNÝ, J., 2010: Musical Form Retrieval. In *MENDEL 2010, 16th International Conference on Soft Computing*. Brno University of Technology.

KATO, H. – BILLINGHURST, M., 1999: Marker Tracking and HMD Calibration for a Video-based Augmented Reality Conferencing System. In *Proceedings of*

International Workshop on Augmented Reality, pp. 85–94, San Francisco, CA. IEEE. ISBN 0-7695-0359-4.

Kato, H. et al., 2003: A Registration Method based on Texture Tracking using ARToolKit. In *Proceedings of IEEE International Augmented Reality Toolkit Workshop*, pp. 77–85. ISBN 0-7803-7680-3.

Kryštof, J., 2009: Towards an MDA-based approach for development of a structural scope of the presentation layer. *Acta univ. agric. et silvic. Mendel. Brun.*, LVII, 6, pp. 123–133. ISSN 1211-8516.

Neider, J. – Davis, T. – Woo, M., 2007: *The OpenGL Programming Guide: The Official Guide to Learning OpenGL Version 3.0 and 3.1.* USA : Addison-Wesley Publishing Company. ISBN 0-3214-8100-3.

Ong, S. – Nee, A., 2004: *Virtual and Augmented Reality Applications in Manufacturing.* London : Springer. ISBN 978-1-85233-796-4.

Prochazka, D. et al., 2011: Mobile Augmented Reality Applications. In *MENDEL 2011, 17th International Conference on Soft Computing.* Brno University of Technology. ISSN 1803-3814.

Saffer, D., 2010: *Designing for Interaction: Creating Innovative Applications and Devices.* Berkeley, CA : New Riders. ISBN 0-321-64339-9.

Wang, B. – Fan, S., 2009: An Improved CANNY Edge Detection Algorithm. In *IWCSE '09: Proceedings of the 2009 Second International Workshop on Computer Science and Engineering*, pp. 497–500, Washington, DC, USA. IEEE Computer Society. ISBN 978-0-7695-3881-5.

Šťastný, J. et al., 2011: Augmented reality usage for prototyping speed up. *Acta univ. agric. et silvic. Mendel. Brun.*, LIX, 2, pp. 353–360. ISSN 1211-8516.

Address

Ing. David Procházka, Ph.D., Department of Informatics, Mendel University in Brno, 61300 Brno, Czech Republic, email: david.prochazka@mendelu.cz