Advanced Markers for Augmented Reality

Alexander Páldy*
Supervised by: Adam Herout†

Faculty of Information Technology
Brno University of Technology
Brno / Czech Republic

Abstract

Marker field is an image pattern that can be efficiently detected in a camera image, then exactly localized, even when heavily occluded, under strong motion blur, and under terrible lighting conditions. This work aims at synthesis of colored marker fields with the following goals in mind: the marker fields should be aesthetically appealing; they must be easily and reliably detectable; the marker fields must be the largest possible with given detection performance. This work includes design of new variants of marker fields, new algorithms for their synthesis, experimental work on the synthesized markers and a suggestion of sample application.

Keywords: uniform marker fields, camera localization, fiduciary markers, artificial markers, overlapping markers

1 Introduction

In augmented reality, one of the indispensable problems is the reliable determination of the camera position in the scene. Two approaches of the vision-based camera localization techniques can be differentiated: use of fiduciary markers or localization without them. Fiduciary markers are not readily available in the scene, methods working without them offer a more general approach. In recent years, methods using only the natural features of the scene have developed to a level that is operating in real-time and is very accurate, several of them are listed below:

- PTAM [8] and DTAM [10]
- homography-based [11]
- SLAM-based methods [6]
- local feature methods [12], etc...

However, these methods have limitations in particular environments, which are not textured or unique enough (such as an empty single color wall). As a consequence, artificial marker systems are still in use.

*xpaldy00@stud.fit.vutbr.cz
†herout@fit.vutbr.cz

Typically, fiduciary markers can be described as a black-and-white two-dimensional planar image, which is easily detectable when placed in the scene. To ensure a free camera movement several distinguishable markers have to be applied. To summarize, such systems have two crucial stages: detection and identification. For a successful localization, markers have to be present in every frame of the camera and a significant portion of them has to be visible. If all of these requirements are guaranteed, a homography is used for the precise computations. The prevalent procedure is the use of multiple disjoint markers, such as ARTag [4]. A more efficient approach was proposed by Szentandrási et al. [13] called Uniform Marker Fields (see Figure 1).

Uniform Marker Fields differ from the other markers by using mutually overlapping partial markers. The large size of the marker helps that its presence can be detectable within the scene from various views. In addition, fast localization is achieved by recognizing the sub-areas within the field. These marker fields are based on 4-orientable De Bruijn tori [2]. Szentandrási et al. [13] have designed a simple genetic algorithm for synthesis of such markers. Also, they have proposed a method for detection of checkerboard structures in a camera image.

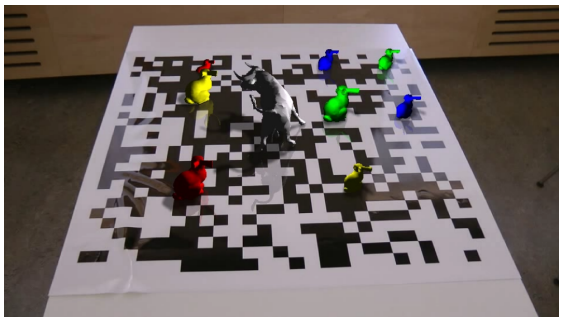

Figure 1: An example of a Uniform Marker Field from [13].

An improvement of these marker fields is described by Herout et al. [7], they have replaced the black-and-white squares in the grid by squares with shades of grey. This offers more information to code in a smaller window. To keep the algorithm working in different lighting conditions, the edge gradients between the modules are used

for the identification of the unique windows. This principle can be used with colored markers, using the gradient in each color channel. This kind of advanced marker fields are used by Dubská et al. for their automated matchmoving [3]. Colors offer a possibility of coding an existing image into the marker field. As a result a marker becomes more natural to the human vision, therefore it can be more acceptable.

The main contribution of this paper is a proposition of a genetic algorithm for synthesis of advanced marker fields such as those described before (Section 3). The genetic algorithm is based on the works of Szentandrási et al. [13]. However, the fitness of the generated maps is no longer defined by only the number of conflicts in it. The cost function is expanded by the quality of the edges between the modules, considering an effortless detection of such marker fields. The aspects of an effective detection and variations of the marker fields are also introduced in this work, such as a promising experiment with hexagonal markers (Section 2). Our algorithm is running on a supercomputer of around 1000 nodes.

The implemented algorithm is reviewed in several ways. The theoretical upper bounds are compared to the limits reached in practice. The runtime of the synthesis of a conflict-free map is measured with different starting options. Also the quality of miscellaneous results is tested by the detection algorithm. To provide an idea of usefulness of this kind of marker field design, we suggest several forms of application (Section 4, 5, 6).

2 Variations of Marker Fields

Uniform Marker Fields are defined by aperiodic 4-orientable binary n^2-window arrays. An aperiodic (m,n)-window array is a k-ary 2D array of size $h \times w$

$$A = (a_{i,j} \in \{0,\ldots,k-1\}; 0 \leq i < h; 0 \leq j < w), \quad (1)$$

where each sub-array $A_{r,c}$ of size $m \times n$ appears exactly once (also called De Bruijn torus [2]). Considering this definition, there is a possibility that multiple rotations of the same sub-array can occur. This example could lead to errors in the identification, it will be called *conflict* (as in Figure 2). The solution to this problem is brought by orientable window arrays. For the ordinary window arrays defined earlier the 1-orientable term is used. 2-orientable arrays are unique in respect to rotation by 180°, while 4-orientable arrays to rotation by 90°. An upper bound for the size of 4-orientable n^2-window arrays is described by eq. (2), where N is the window count.

$$N \leq \frac{k^{n^2} - k^{\lfloor \frac{n^2+1}{2} \rfloor}}{4} \quad (2)$$

For binary ($k = 2$) window arrays the upper bound for $n = 3,4,5$ values are $N = 120, 16320, 8386560$, which means the theoretical maximum of square maps will not be larger than 11×11, 127×127, 2895×2895. As an optimal value of $n = 4$ is used by Szentandrási et al. [13].

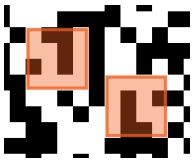

Figure 2: Two conflicting windows in a marker field.

However, an even smaller n (window size) can be used, if the k is increased (where k is the number of possible values for a module). This advancement also provides more edges in the marker field (i.e. improved detection), besides even less visible squares are needed for identifying the location. Herout et al. [7] implemented a detection algorithm handling greyscale or color k-ary marker fields. In contrast with the black-and-white checkerboard, where the absolute values were determining the unique window, the greyscale or color values cannot be treated the same way. The reason is that they cannot be accurately recognized in varying lighting conditions. Another problem provoked by various colors can be when the neighboring modules are in low contrast with each other. It cannot only cause loss of the grid lines, but also confusion with similar modules.

The solution – for the complications described before – can be the use of the gradient values in the edges. This is explained in the next section, along with the necessity of rating the quality of the edges (Section 2.1). Also an idea of preprocessing images to obtain characteristics of a marker field is described further. Other improvements can be achieved by benefiting from different windows or modules. In Section 2.3, a little note can be found about an experiment producing marker fields with regular hexagons as modules, which are more resistant against the deformations of the marker.

The detection and recognition of uniform marker fields is beyond the scope of this work. Papers [13] and [7] are focusing on these problems, both having similar approach. In a nutshell, their algorithms consist of the following steps:

- **Extraction of edgels**: An edgel is described by an image point and an edge orientation or by two endpoints. A small number of horizontal and vertical scanlines is helping in this.
- **Determining two dominant vanishing points**: The edgels are separated in to two groups based on their directions and two vanishing points are computed.
- **Finding the grid of marker field edges**: The two vanishing points determine the horizon, and the grid can be "reconstructed".
- **Localization within the field**: The unique features are extracted from the square modules and the localization can be easily accomplished.

RDM	ALVAR	UMF
164.4	30.1	8.8

Table 1: Speed of three tested algorithms in milliseconds for 1080p videos using a mid-range Intel(R) Core(TM) i5 CPU 661 (3.33GHz) CPU [3]

Table 1 shows the speed of three selected algorithms: RDM = Random Dot Markers [14], ALVAR [1], UMF = Uniform Marker Fields. The detection algorithm is 3× faster than ALVAR.

2.1 Expanding the color palette

As mentioned before the expansion of the color palette evokes some difficulties with the proper detection. The absolute values of the modules cannot be reliably extracted from the camera image. This is the reason why we use the edge gradients. An edge gradient e is the difference between two neighbor modules' values:

$$e_{ij}^{\rightarrow} = a_{i,j+1} - a_{i,j}, \tag{3}$$

$$e_{ij}^{\downarrow} = a_{i+1,j} - a_{i,j}, \tag{4}$$

However, this is an absolute value as well, which is still unstable for the detector. This is why we apply the signum function on e, so an edge gradient will be described with values from $\{-1,0,+1\}$. In Figure 3, the gradients are represented with arrows. The figure also demonstrates the difference between greyscale and a colored marker field. In a colored marker field, multiple gradients are defined, which correspond to each channel of the RGB model. Nevertheless, other color models can be used, too.

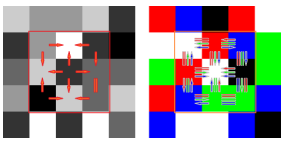

Figure 3: Edge gradients illustrated with arrows on a portion of the marker field. On the **left**, five shades of grey are used, while on the **right** five colors and multiple gradients are defined (e. g. in every channel of the RGB color model).

To achieve a valid detection, the quality of the edges must be considered. Edge gradients with higher absolute value are preferred. Figure 4 shows how a module is ranked by the quality of the surrounding edges.

2.2 Image as a targeted result

Use of multiple colors allow to generate marker fields resembling a selected image. This kind of marker field will be aesthetically appealing for a human viewer, furthermore

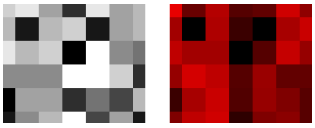

Figure 4: A marker field and a *cost field* belonging to it. **Left**: The marker field. **Right**: The grid symbolizing the cost of each module (the lighter red, the bigger cost it has).

it will be recognizable by the detection algorithm (Figure 5). When producing this kind of maps, we have to consider several aspects. It not only has to fulfill the features characterized in the previous section, but has to converge to a selected image, so the penalization of the modules becomes more complex. Also to keep a maximal similarity a dynamic color palette can be used.

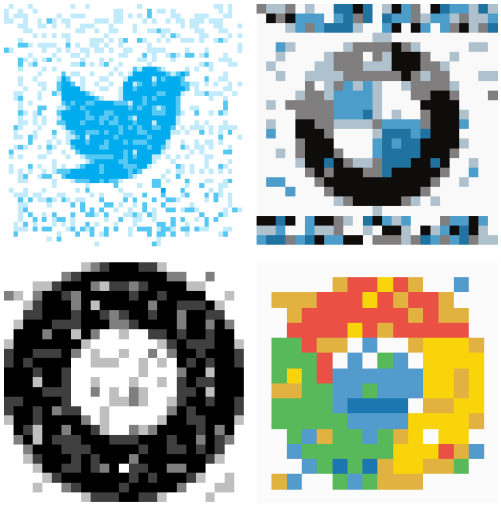

Figure 5: Some examples of images as marker fields.

2.3 Other improvements

Other improvements can be performed by changing the type of the modules or by selecting the edges to use in the identification. For a window size of k, the number of edges is $n_e = 2k^2 - 2k$. This results, that while the size of the window is increasing linearly, the number of edges used will grow exponentially. So the amount of the coded information cannot be comfortably controlled. The edge selection helps to solve this problem. Examples of different edge configurations can be seen in Figure 6.

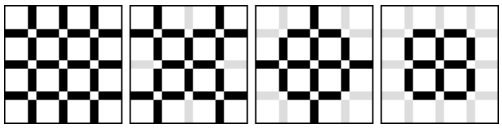

Figure 6: Different edge definitions for a 4×4 window.

An example of constructing a marker field from hexagonal components can be seen in Figure 7. This varia-

tion uses 6-orientable windows and a limitation is applied, where identical modules cannot lay next to each other. This limitation is necessary, because the detection algorithm for these kind of markers is relying on the Y-s constructed by three adjoining modules. Leading to a marker system more resistant against deformations, so the markers do not have to be planar anymore.

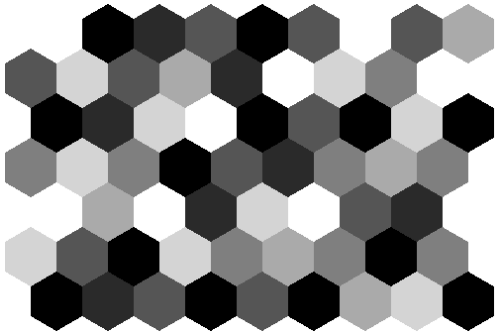

Figure 7: An experiment with hexagonal modules.

3 Algorithm for the synthesis

Synthesis of black-and-white marker fields was presented by Szentandrási et al. [13]. Our solution is based on their work, we are using a similar genetic algorithm and a client-server architecture. Our genetic algorithm is based on the following terms:

- The first population is initialized from a number of copies of the same (or various) randomly generated array(s).
- The fitness function is based on the number of conflicts and the quality of edges. Generally it looks like

$$f(A) = \sum_{i=0}^{w} \sum_{j=0}^{h} (c_{conf}(a_{i,j}) + c_{neighbor}(a_{i,j})), \quad (5)$$

  where w is the width, h is the height of the array, $c_{conf}(a)$ returns the cost for the conflicts module a is involved, while $c_{neighbor}(a)$ penalizes its edges. For the image markers, the similarity to the original image is included, too:

$$f(A) = \sum_{i=0}^{w} \sum_{j=0}^{h} (c_{conf}(a_{i,j}) + c_{neighbor}(a_{i,j}) + c_{similar}(a_{i,j})). \quad (6)$$

- No specific fitness threshold is defined. We try to improve the markers as much as we can, and we can manually decide if they are satisfactory enough to stop the algorithm.
- For selecting members for the next generation, rank selection is used.
- Mutation is performed by selecting a number of modules with the highest cost (see eq. 5) and replacing them with randomly generated values.

We are using a client-server model, where the server stores the active population, while the idle clients request for tasks from the server and apply the mutations on the received maps. This kind of architecture does not provide sufficient circumstances to keep the generations separated. Contrary to typical genetic algorithms [9], a single population is used through the life time of the algorithm.

On the server, we can start new populations by selecting their attributes, like size, type of modules, type of windows, etc. There is a *waiting room* for different maps, where we can choose which one should be improved. The chosen ones get into the queue, where from the clients obtain their tasks. Normally, if a client returns a conflict-free marker field, it is moved to the ready group. This can be manually overridden by selecting the *force continue* option, so the marker fields quality can be further increased.

While the server only stores the data about the marker fields, the clients' responsibility is to decrease the cost of the maps. The highest penalization is given for the modules, which take a share in a conflict. This ensures that the conflicts are solved in the first place (see Figure 8).

3.1 Estimating Marker Field Size Limits

For estimating upper bounds of the size of the generated maps is a complex problem. A very rough estimation can be described by the inequality from [2], also cited in Section 2. However, we must acknowledge the fact that in our case a window is characterized by the edge gradients, which are selected from three values (see Section 2.1). These edges are not arranged like a window array suitable for the referred inequality. To calculate a maximal amount of possible proper windows can be achieved by the following way: compute all the combinations from those three values for a window considering the fact that the windows must be 4-orientable, then reduce this number by the *paradoxes*. For an example a paradox can appear when we make a loop by starting and ending in the same module and using the same gradient value differing from *equal* as in Figure 9.

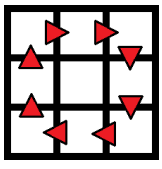

Figure 9: An example of a possible *paradox* defined only by the edges, the same as we say $a_1 < a_2 < a_3 < a_4 < a_5 < a_6 < a_7 < a_8 < a_1$ so $a_1 < a_1$, and this cannot be valid.

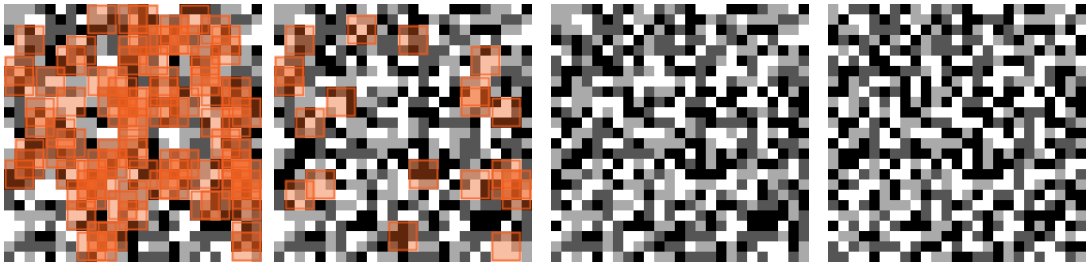

Figure 8: The process of improving the marker field, the first one containing 92 conflicts, the last one with the lowest cost.

Number of shades	Largest	Run-time (min)
2	10×10	2.44
3	33×33	31.05
4	52×52	212.38
5	61×61	193.08
6	61×61	67.67
7	68×68	222.39
8	69×69	377.06
9	69×69	358.73
10	71×71	159.56

Table 2: Largest maps generated with 3×3 windows and different number of shades of grey.

Edges	Largest	Run-time (min)
8 (✳)	19×19	47.33
12 (❋)	52×52	212.38
16 (❁)	144×144	260.91
20 (✼)	256×256	12.54
24 (❋)	256×256	1.93

Table 3: Largest maps with 4 shades of grey and different number of edges used. For 20 and 24 edges no larger maps were tested.

Size	Edges used				
	✳ 8	❋ 12	❁ 16	✼ 20	❋ 24
16×16	285.05	57.01	57.01	0.0	0.0
32×32	-	57.01	57.01	57.01	57.01
64×64	-	-	57.04	57.06	57.07
128×128	-	-	3036.53	57.15	57.25
256×256	-	-	-	752.87	115.85

Table 4: Run-time of the algorithm for 4 shades of gray in seconds. After 57 seconds, if the client have successfully decreased the number of conflicts, then it stops. Where 0.0 values were recorded, the marker field was randomly generated and it was highly probable that it would be without conflicts.

4 Results

As mentioned before, only rough limits exist for the theoretical limitations of the generated marker fields. However, a benchmark was created to illustrate the practical limits. We used a supercomputer with around 1000 nodes. The algorithm is still running, therefore the results will be updated in the future. Generally, a client has 20 minutes to improve the received map. However, if it makes significant reduction of the overall cost, it will send back the marker field after 57 seconds. This causes that a better map will be assigned sooner for the other clients. The measured run-time is the fastest route to a solution, it does not represents the total run-time of the clients working on a single population.

In our first test, we defined the first population by 3×3 windows and we have monitored how the maximum size of the marker field changes with different number of shades of grey (see Table 2). As we can see, this maximum size radically changes only at smaller numbers. This can be explained by the fact, that the edges with 3 possible values are used for the identification of a window, not the modules. If we have used the eq. (2) for $k = 3$ window size and different values of $n \in \{2,3,4,\ldots\}$, we would get the following limits: $11 \times 11, 69 \times 69, 255 \times 255, \ldots$

The other test shows the upper boundaries with 4 shades of grey and different window definitions (see Table 3). Marker fields bigger than 256×256 were not tested. The results show that the maximal size of the map can be controlled with the number of edges used for the identification of a window. This is useful, because for a detection algorithm the marker has to be as large as possible and the fewest edges have to identify a window, so it can easily deal with occlusions.

Finally, a test showing the speed of the algorithm for different sizes and edge definitions (see Table 4). As we can see, the marker fields with smaller sizes are freed from conflicts in one run of the client.

Also, a user study was performed with 38 respondents. They were asked about which marker would they prefer. Beside the standard binary fields and image marker fields, there was a usual image, too. The images used as markers gained the highest rating with an average of 7.42 on a scale from 1 to 10, where 10 was the best score. The average rating of the image marker fields was 5.61, while the black-and-white fields had 3.52. Most of the respondents knew what augmented reality is and half of them was already aware of what markers in augmented reality are.

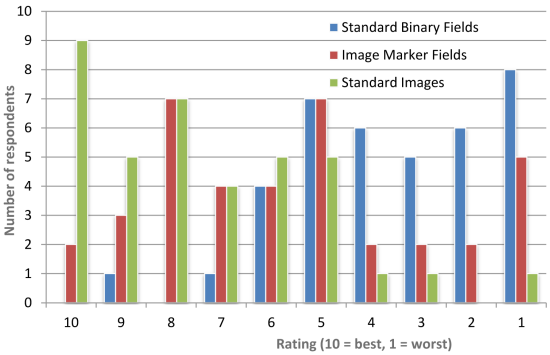

Figure 10: The results from the survey. The respondents were asked to rate the markers according to how pleasing they are visually.

5 Application

Camera localization technologies relying only on natural features are highly advanced, thus most of the applications do not require artificial markers anymore. Still, there exist areas, where the use of markers is necessary. A number of possible applications was listed in [7] and a working example can be found in [3].

The combination of computer graphics with living actors is frequently used in film and advertisement production. This is usually achieved by chroma keying, a technique which replaces a constant color (e.g. green screen) with another scene [5]. However, the matchmoving process is typically performed manually. Semi-automatic tools exist as well, but still a manual check-up is needed. An instant matchmoving solution was proposed by Dubská et al. [3]. They are using colored uniform marker fields, synthesized mainly from shades of green, but their method could work with any other colors (see Figure 11).

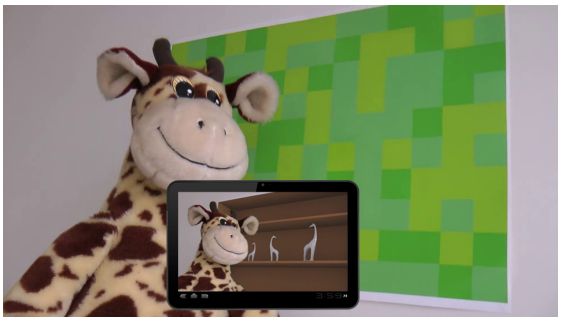

Figure 11: An automatic matchmoving application from [3].

See-through glasses for augmented reality are becoming generally available. This offers another use of marker fields: in tabletop scene interactions. Tracking of natural features can be used for camera localization, but a presence of a visually unobtrusive marker field can increase the reliability and offload some of the expensive computations (see Figure 12).

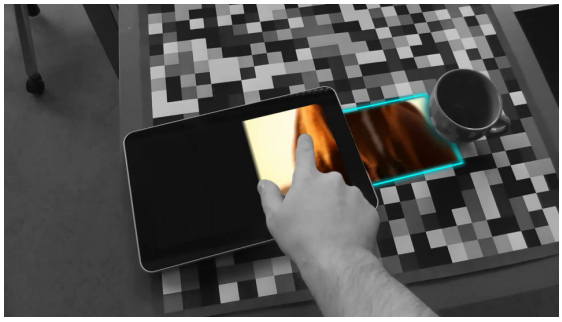

Figure 12: Tabletop screen interaction from [7].

Another utilization can be a direct visual interaction between desktop and ultra-mobile devices, like screen-to-screen task migration. The marker is mixed into the desktop screen image for short periods of time. This ensures the presence of enough unique keypoints for the localization, which is computed on the mobile devices.

In Section 2.3, non-planar markers were suggested with hexagonal modules. An example of an application of such a field can be on clothes. The marker does not have to stay planar, so it adapts to the movement of the one wearing it with ease. This can be advantageous in film and video game industry.

6 Conclusions

This paper presents an algorithm for synthesis of advanced markers. These markers are based on the uniform marker fields. Several possible improvements were considered and their utilization was discussed. The product of the algorithm can be used as a reliable marker for augmented reality systems. Some of the possible applications were mentioned.

We have shown, that our algorithm can produce different kind of marker fields depending on the application. It can generate markers with various colors, two kind of modules (square and hexagonal). It is able to deal with different edges selected for the identification. The creation of the maps can be influenced by a selected image. Also, some tests were discussed to present the capabilities of our method of synthesis.

We are collaborating with the authors of the following publications: [13], [7], [3]. The marker fields used in Figure 11 and Figure 12 were generated by our algorithm. The constant feedbacks from the developers of these applications help to improve the quality of the produced maps. We are also working on the improvement of the synthesis. There are situations when a more intelligent way would speed up the process. We would like to exploit these areas also.

7 Acknowledgments

I would like to thank Adam Herout for his constant support and constructive comments, also István Szentandrási for his useful suggestions.

References

[1] Alvar tracking subroutines library web page, 2012. http://virtual.vtt.fi/virtual/proj2/multimedia/alvar/.

[2] John Burns and Chris J. Mitchell. Coding Schemes for Two-dimensional Position Sensing. In *Institute of Mathematics and Its Applications Conference Series*, 1993.

[3] Markéta Dubská, István Szentandrási, Michal Zachariáš, and Adam Herout. Poor man's simulcam: Real-time and effortless matchmoving. submitted to SIGGRAPH 2013, 2013.

[4] Mark Fiala. Designing Highly Reliable Fiducial Markers. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, volume 32, pages 1317–1324, July 2010.

[5] Jeff Foster. *The Green Screen Handbook: Real-World Production Techniques*, volume 978. John Wiley & Sons, 2010.

[6] Steffen Gauglitz, Chris Sweeney, Jonathan Ventura, Matthew Turk, and Tobies Höllerer. Live tracking and mapping from both general and rotation-only camera motion. In *Mixed and Augmented Reality (ISMAR), 2012 IEEE International Symposium on*, pages 13 –22, nov. 2012.

[7] Adam Herout, István Szentandrási, Michal Zachariáš, Markéta Dubská, and Rudolf Kajan. Five shades of grey for fast and reliable camera pose estimation. In *CVPR 2013*, 2013.

[8] Georg Klein and David Murray. Parallel Tracking and Mapping for Small AR Workspaces. In *ISMAR '07*, pages 225–234, 2007.

[9] Melanie Mitchell. *An Introduction to Genetic Algorithms*, volume 5. MIT Press, Cambridge, Massachusetts, 1999.

[10] Richard A. Newcombe, Steven J. Lovegrove, and Andrew J. Davison. Dtam: Dense tracking and mapping in real-time. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2320 –2327, nov. 2011.

[11] Christian Pirchheim and Gerhard Reitmayr. Homography-based planar mapping and tracking for mobile phones. In *ISMAR*, 2011.

[12] Radu Bogdan Rusu, Nico Blodow, and Michael Beetz. Fast point feature histograms (fpfh) for 3d registration. In *Robotics and Automation, 2009. ICRA '09. IEEE International Conference on*, pages 3212–3217, 2009.

[13] István Szentandrási, Michal Zachariáš, Jiří Havel, Adam Herout, Markéta Dubská, and Rudolf Kajan. Uniform marker fields: Camera localization by orientable de bruijn tori. In *ISMAR 2012*. Institute of Electrical and Electronics Engineers, 2012.

[14] Hideaki Uchiyama and Hideo Saito. Random dot markers. In *IEEE Virtual Reality Conf. (VR)*, 2011.