Scarlet – Real Time Mobile Augmented Reality Library

Csaba Bolyós *
Supervised by: Zuzana Haladová†

Faculty of Mathematics, Physics and Informatics
Comenius University
Bratislava, Slovakia

Abstract

Since the beginning of the new century, augmented reality (AR) applications became very popular among smart phone users. There is also an emerging trend of users becoming the producers of the visual content in the AR environments.

In this paper, we present Scarlet – a novel library for Android developers of AR applications. The key problem of AR is the registration of a virtual and the real world. Scarlet can recognize different visual markers (images, logos, black-and-white markers) very fast using the state-of-the-art local feature detection methods. Different local features were tested and are presented in the paper. Scarlet library also provides several functions for designing AR application. The main advantage in comparison to OpenCV4Android is that it is faster and the creation of AR application is easier for the user. Scarlet is nearly 11.5 times faster then OpenCV4Android 2.3.1. Also, the number of the native calls is minimized.

Keywords: Local features, Visual marks, SIFT, SURF, ORB, BRIEF, FAST, Augmented reality

1 Introduction

AR becomes more and more popular in a great variety of different applications, including augmentations in sports tournaments, AR games, cultural heritage AR applications, medical AR applications and many others. It has been popularized in the last few years. Apart from AR applications, there is also a strong trend of user becoming the producer of the visual content on the internet and in the AR environments.

Our aim was to create a user friendly AR library that could help developers to create their AR application faster and easier. Our solution is called Scarlet and it is based on the mobile operation system Android and focuses mainly on local features. It is able to recognize objects in the frames of the mobile camera at interactive rates. Our library is of the small size and should offer the developers both: the possibility of customization of the library and the ability to produce results by single function calling.

This paper is organized as follows. The first section introduces the reader to the selected previous work. Next, properties of local feature *descriptors* are briefly stated. The next section describes a simple sample application utilizing our library. In the last two sections achievements are summed up and future work is presented.

2 Previous work

We have examined three libraries for Android AR developers: *openCV4Android* [9], *Vuforia* [10] and *Layar Vision* [6]. The *Layar Vision* is a commercial library which can be purchased for more than two thousand dollars. It works with geodata and provides image *descriptor* functions, but the set up of these functions is not free either. *Vuforia* is a free AR library developed by Qualcomm. It works best on devices powered by Qualcomm processor or graphic card. It works well with 2D markers and also supports texture recognition. The third solution is the *openCV4Android*, which is the port of openCV. *OpenCV4Android* is basically the wrapper of the OpenCV, and can be used similarly to OpenCV. It implements many AR methods and offers good and reliable results. The main problem of this library is, that it does not operate in real time and includes wide range of redundant Computer vision, Image and Video processing, Pattern recognition and other algorithms - not only those needed for AR. It also has too many native calls, resulting in slowing down the programs.

3 Local features

When performing object recognition (and later object registration), we usually want to correctly classify also in situations when the object is scaled, translated, rotated, occluded or only partially presented in the image. To achieve this we need to describe the object using only features invariant under these transformations. The efficient solution to this problem are local features. They mainly consists of two methods, the *detector* and the *descriptor*. The *detector* seeks the interesting points in the image area and the

*bladeszasza@gmail.com
†zhaladova@gmail.com

descriptor describes the neighbourhood of the interesting point via the feature vector.

We distinguish two types of the feature vectors created via local feature *descriptors*: the integer and the binary vectors. The integer vectors are usually compared using the *Euclidean distance*:

$$d(p,q) = d(q,p) = \sqrt{(q_1 - p_1)^2 + \ldots + (q_n - p_n)^2} = \\ = \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}.$$

On the other hand, the binary vectors can be compared, using the *Hamming distance* whose computation is very efficient. For example, the distance between the string "0110" and "0000" is equal to two, and can be calculated using following equation:

$$d_h(p,q) = \sum_{i=1}^{n} \delta(p_i, q_i)/n$$

where

$$\delta(p,q) = \begin{cases} 0 \text{ if } = (x < 0 \wedge y < 0) \vee (x = 0 \wedge y = 0) \\ 1 \text{ if } = otherwise \end{cases}.$$

In this paper we examined and tested six local feature *descriptors*. The results of comparison of different local feature methods can be seen in Table 3.

There not only the methods which are used as *detector* and *descriptor* are mentioned, but also the type, size and invariance under rotation, scale and noise of their result vector.

3.1 Detectors

The traditional computer vision feature methods compute the features uniformly from all the points in the image. Local features however use only points which are somehow interesting to us. An interesting point is the point of the image in whose neighbourhood intensity varies. Computation of the features only for these points can speed up the process considerably. *Harris corner detector* [5] is one of the first *detectors* for interesting points (IP). The detection is based on the responses of the Hessian matrix. The responses are calculated over a small region surrounding the point.

The FAST (Features from the Accelerated Segment test) is one of the algorithms implemented in our library. It examines every pixelś 9 point radius and compares the intensity of the pixel with the pixels in the radius. If the difference of at least 3 pixels in the radius with the central pixel is over a threshold, the central pixel is considered the key point. The FAST algorithm can work very fast, produces a lot of key-points but it is not invariant under scale and rotation.

The *detector* of the classical local feature method SIFT (Scale Invariant Feature Transform) [8] produces more robust points, but is remarkably slower. SIFT's [8] *detector* works in the scale space which ensures the invariance under scaling (The example of scale space can be seen in Figure 1). To remove the noise it filters images using the Gaussian filter. The key-points are detected as the maxima and minima in the 27-neighbourhood in the scale space.

A speed improvement of the SIFT's [8] *detector* was introduced in the SURF (Speeded Up Robust Features) [2]. It not only detects the key-points faster than SIFT, but also maintains the invariance. As *detector* it uses scale space as SIFT space, but it is created using an approximation of second order derivatives of Gaussians in Y and XY direction with box filters on integral images. An example, of the SURF's [2] Box filters see Figure 2.

Integral images allow us to calculate rapidly the sum over subregion of the image. Value of a pixel equal to the sum of all values of the pixels above and on the left of it. Second step is increasing the resolution of filters from 9*9 up to 27*27. Finally only the remaining points after a non-maximum suppression in 3*3*3 scale space are selected.

All of the above methods give good results, but they are a bit costly to be real-time even on high performance mobile devices.

The ORB (Oriented FAST and Rotated BRIEF) method[11] adds the rotation invariance to FAST. It can be computed in real time even on mid-end smart phones. ORB's *detector* is called oFAST. It works similarly to FAST, but extracts rotation using intensity of centroid. It creates a vector with direction from IP to centroid. Vector orientation could be easily gained.

The recent papers show that recent *descriptors* like BRISK 3 and FREAK 4 calculate their *detectors* not on the whole image but only on its part. BRISK calculates the saliency first and then the key-points are extracted from the region. FREAK [1] samples the image similarly to human eye, using a technique called retinal sampling. It uses more samples at the center of the view focus and less samples at the edge of the focus. For our library, all of the previously mentioned methods matter, as we want to offer a large variety of algorithms.

3.2 Descriptors

Having the key-points extracted, we can sample information from the region around them. We recognize two types of *descriptor*, one is the n-dimensional numerical vector and the other is the binary string. Numerical vectors are created by SIFT [8] and SURF [2], giving a robust description of the region around a key point. They are both invariant under noise, rotation and scale. But although SURF methid is calculated faster, they are both too slow on mobile devices. The SIFT *descriptor* calculates the magnitude and the orientation of the gradient, which are then weighted by a Gaussian window. The SURF method is faster. To maintain fast filtering it uses integral images. First it creates square regions around the key-point aligned to it. Than each of these regions is subdivided into 4x4 smaller axial subregions. On all of the subregions, the

	detector	*descriptor*	vector type	n dimensional vector	invariance (R,S,N)
SIFT [8]	scale space of DoG	orientation of gradients	integer	128	R,S,N
SURF [2]	box filters	responses from Haar wavelet	integer	64	R,S,N
BRIEF [3]	—	binary tests	binary	based on patch size	S(partial),N
ORB [11]	oFAST	rotation aware BRIEF	binary	based on patch size	R,N
FREAK [1]	retina sampling	lighting intensity tests	binary	based on patch size	R,S,N
BRISK [7]	scale space FAST	binary tests	binary	based on patch size	R,S,N

Table 1: The comparison of local feature vectors. For each method their *detector* and *descriptor* are given and the type of the resulting vector is presented. Also, the robustness of the results is given. In the *invariance (R,S,N)* column the "R" means the rotation, the "S" the scale and the "N" the noise

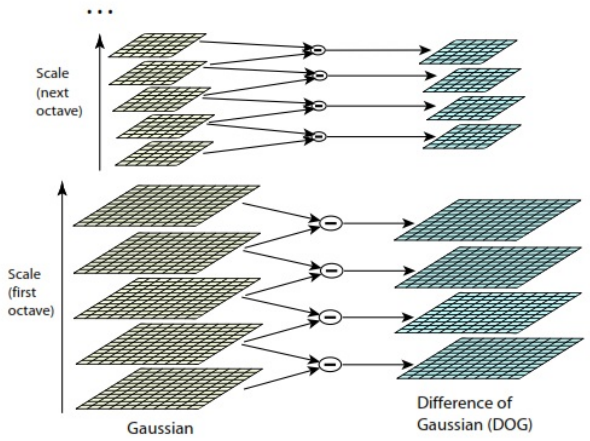

Figure 1: Scale space created and the difference of Gaussian calculated [8].

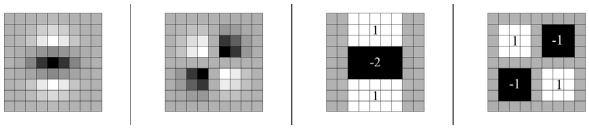

Figure 2: The box filters which are used by SURF [2]

responses from Haar wavelets are calculated. To achieve even more robustness towards geometric deformations, the authors weighted these responses with Gaussian.

The other group of *descriptors* that utilize binary string are BRIEF (Binary Robust Independent Elementary Feature) [3], ORB [11], BRISK (Binary Robust Invariant Scalable Keypoints) [7] and FREAK (Fast Retina Keypoint) [1]. As mentioned before the distance between two binary strings can be compared fast using the *Hamming distance*. It speeds up to the final process of comparison. The main advantage of these methods is that they can be computed real-time, but lot of them lack of the robustness of SIFT [8] or SURF [2].

BRIEF [3] lacks the rotation invariance, but it can be computed in real-time. BRIEF calculates its *descriptors* by sampling the image with binary tests. The responses of these tests are stored in the binary string. Rotation invariance can be achieved, for example using a bigger dataset for one image. If at that dataset the chosen pattern is slightly rotated and the *descriptor* is calculated from it, we achieve similar *descriptors* and the pattern can be found even if it is rotated. But our goal is to make development faster and smoother, and therefore we do not want to use big datasets.

ORB's [11] description algorithm is based on BRIEF [3] and it is called rBRIEF. The rBRIEF works similarly to BRIEF but it has stored rotation which was gained from oFAST at the key-point detection phase.

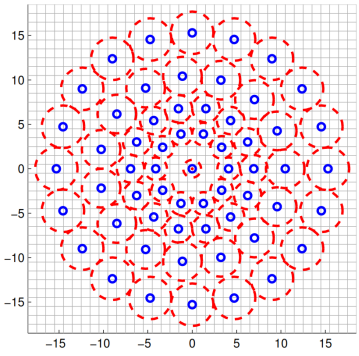

Figure 3: A sampling pattern used by BRISK. The red circles are regions smoothed with Gaussian kernel, the small blue points are the sampling locations [7].

The BRISK *descriptor* produces binary strings that are joined results of simple brightness comparison tests. The algorithm works as follows: First, the characteristic direction is identified to achieve rotation invariance, next the brightness comparisons are carefully chosen. These chosen brightness comparison tests are evaluated at sampling pattern which can be seen in Figure 3.

The last presented *descriptor* is FREAK [1]. FREAK similarly to BRISK [7], ORB [11] and BRIEF [3] creates a binary string. This binary string is achieved by concatenating results from binary intensity tests. Like BRISK, FREAK uses the sampling pattern inspired by the human vision system. It is called retinal sampling pattern (retinal pattern can be seen in Figure 4).

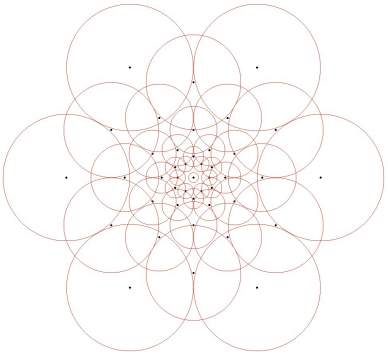

Figure 4: The retinal sampling pattern. Every circle represent a receptive region, smoothed by corresponding Gaussian kernel. The circles are placed similarly like the retinal ganglion cells at human eye [1].

4 Application

This section describes the simple production of an AR application utilizing Scarlet. First, the work flow of image recognition is presented in Figure 5. Our library supports multiple local *descriptors*, but ORB seems to be the best solution for our purpose. The recognition is acquired in less than a tenth of a second. In preprocessing, we create a database of precalculated *descriptors*. Therefore, at runtime we do not need to calculate the *descriptor* repeatedly, which saves us lots of computational time. Next we want provide the user with the best experience possible. The application runs on multiple threads, as can be seen in Figure 5. The main thread starts at least one thread, providing the video output for the screen. The processing of the image or the pattern recognition is done in separate threads. In the Figure 5, we can see this "computing" threads don't need to run long, they can be put to sleep after they return the result.

One iteration consists of capturing the frame from the video thread. After that the computing thread is started. It calculates the key-points on that frame and runs the *descriptor* algorithm. When the *descriptor* is acquired it is compared to our database of precomputed *descriptors*. Some threshold is needed to tell that the distance between two *descriptors* is within the similarity tolerance, if the distance is small, the two *descriptors* are probably describing the same region. If it is big the two *descriptor* are not similar. If we detected which *descriptors* from database are nearest we can classify the pattern on that frame.

If we need to draw virtual object fast on the screen, we can measure the median of the inlier points. If we want to determinate the precise position and square region of the pattern in the image, we could use algorithms for computation of the homography matrix between the points. Using this matrix, we can give every point on the image the exact location on the captured video frame.

One of the algorithms which can be used to find the homography matrix is called RANSAC (Random sampling consensus) [4] it suffices to use 4 points to create the homography matrix, but the homography is more stable if we use more points. In the best case, RANSAC determinates the homography only from the inlier points, the outlier points have no influence on the result, as far as we use enough iteration and inliers. In most of the cases it is suffices to have just the axial boundary of the found pattern, and on that square our virtual content can be drawn.

The Scarlet library is written natively in C++, to gain speed of the native language supported by Android. The functions are designed for easy and quick creation of an AR application. We also propose functions in Java which aggregate some background processing and background calculation. As an example, *GiveSquareRegion*(database, image) returns the axial region where the closest pattern is detected. Our library aggregates functionalities which are supposed to be near each other, so it reduces the unnecessary native calls. Those in comparison to *openCV4Android* we gain increase in speed.

5 Future work

In the next phase of our development, Scarlet will be extended with more local *descriptors*. We plan to implement GPU version of some algorithms, which will require high computational power, especially for *descriptors*. We plan to utilize another pattern recognition method, for example the contour recognition. It will fasten up the development of AR applications.

Currently our library enables adding only 2D images as objects, but in the near future it should provide a way to draw out simple 3D objects. Later it will provide functionality extended to video stream, where any kind of 3D objects stored in .obj format can be incorporated. For this purpose we want to use some 3D libraries which provides us with some of the mentioned functionalities.

6 Validation

The Scarlet was tested on high and mid-end mobile device's(HTC Sensation XE, Samsung Galaxy Nexus). The results were compared to *openCV4Android* (ver.: 2.3.1). The results show us a significant gain of speed. Scarlet is nearly 11.5 times faster than *openCV4Android* (ver.: 2.3.1). Table 4 gives also results from processing a higher resolution image. We see that the computational speed drops as the resolution of the image grows. Also it shows that even with that lost of computational speed more than five frames are processed and the matrix of homography is computed in one second. With this computational speed, the user can still produce quality AR application, with only in-betweens interpolated smoothly between two results.

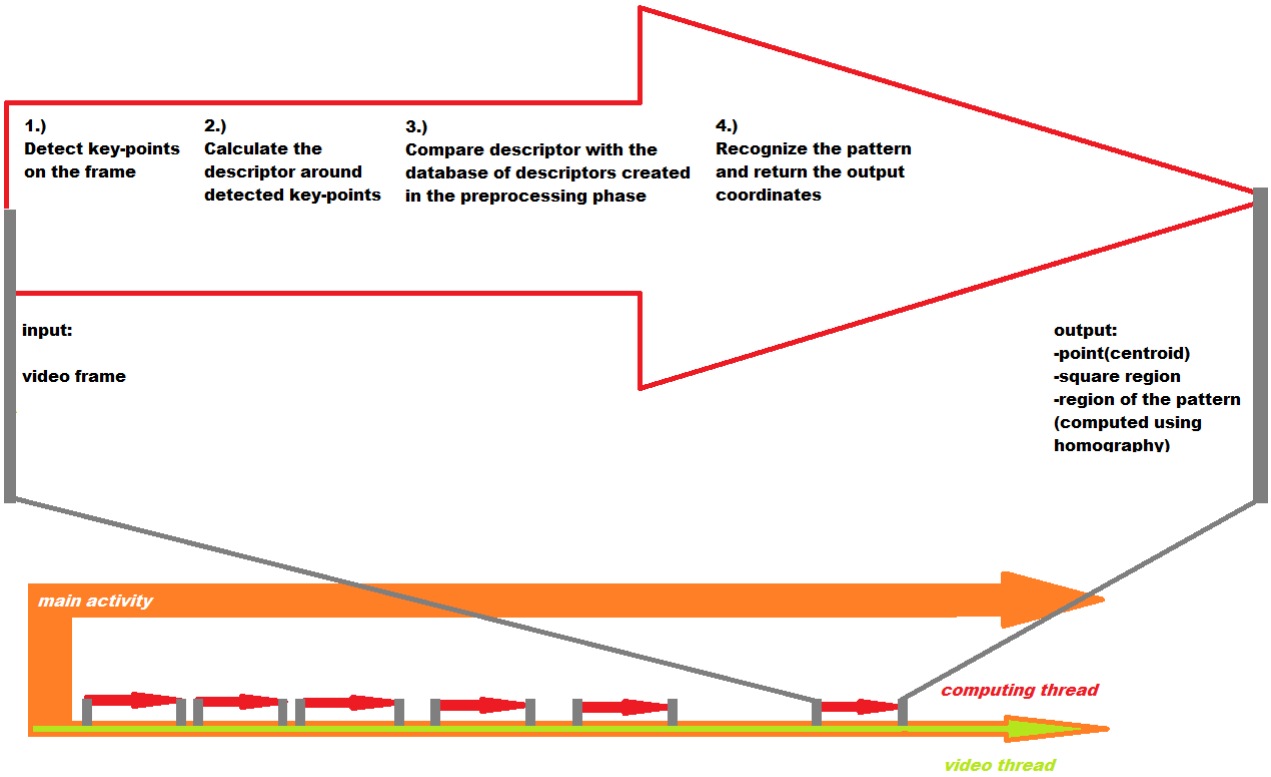

Figure 5: The work-flow of the application, demonstrating how multi-threading can be used in AR application. The top image shows how the image is processed in one iteration. The result of this iteration is either a point or the whole region defined by the pattern. On the bottom image, the work-flow of the whole application is showed. The computing thread is not running all the time, it is put asleep as soon as it returns the result

	800*480 frame with RANSAC	800*480 frame without RANSAC	480*320 frame
SCARLET	5,3 frame/sec	7,4 frame/sec	13,8 frame/sec
openCV4Android 2.3.1	—	—	1,2 frame/sec

Table 2: The frame rate comparison of Scarlet and *openCV4Android* (ver.: 2.3.1) on video 480x320 frame.It also shows that, the Scarlet computes real time not only the local features but the matrix of homography as well.

Conclusion

The presented solution is fast, reliable and light weight library suitable to produce Augmented reality application. It is faster than *openCV4Android* (ver.: 2.3.1) nearly 11.5 times when using ORB for detection and description. As default settings of the library, the ORB *detector* and *descriptor* is used. Under these circumstances it provides real time object recognition and gives good results. The Table 4 shows how fast the ORB *detector* and *descriptor* is computed and match with our database. We achieved our goal to speed up the recognition time.

7 Acknowledgement

This work was partially funded by Slovenská Sporitelňa, a.s. as a project of their own Augmented reality application. The author also wishes to thank to RNDr. Zuzana Haladová, RNDr. Martina Bátorová and doc. RNDr. Andrej Ferko, PhD. for their supervision and help.

References

[1] A. Alahi, R. Ortiz, and P. Vandergheynst. Freak: Fast retina keypoint. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 510 –517, June 2012.

[2] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. SURF: Speeded up robust features. In Ale Leonardis, Horst Bischof, and Axel Pinz, editors, *Computer Vision ECCV 2006*, volume 3951 of *Lecture Notes in Computer Science*, pages 404–417. Springer Berlin / Heidelberg, 2006. 10.1007/11744023_32.

[3] Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal Fua. BRIEF: Binary robust independent elementary features. In Kostas Daniilidis, Petros Maragos, and Nikos Paragios, editors, *Computer Vision ECCV 2010*, volume 6314 of *Lecture Notes in Computer Science*, pages 778–792. Springer Berlin / Heidelberg, 2010. 10.1007/978-3-642-15561-1_56.

[4] Martin A. Fischler and Robert C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, June 1981.

[5] Chris Harris and Mike Stephens. A combined corner and edge detector. In *Alvey vision conference*, volume 15, page 50. Manchester, UK, 1988.

[6] Layar. Layar vision @ONLINE. `http://www.layar.com/documentation/browser/howtos/layar-vision-doc/`, 2011.

[7] S. Leutenegger, M. Chli, and R.Y. Siegwart. BRISK: Binary robust invariant scalable keypoints. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2548 –2555, nov. 2011.

[8] D.G. Lowe. Object recognition from local scale-invariant features. In *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, volume 2, pages 1150 –1157 vol.2, 1999.

[9] OpenCV. Opencv for android @ONLINE. `http://opencv.org/android`, 2013.

[10] Inc. Qualcomm Technologies. Augmented reality (vuforia) @ONLINE. `https://developer.qualcomm.com/mobile-development/mobile-technologies/augmented-reality`, 2012.

[11] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski. Orb: An efficient alternative to sift or surf. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2564 –2571, nov. 2011.