# HTML5 Marker-less Tracking Algorithm

Eduardo A Lundgren Melo
Informatics Center (CIn)
Federal University of Pernambuco (UFPE)
ealm@cin.ufpe.br

May 20, 2012

## Abstract

This work proposes a marker-less tracking algorithm for native web targeted platforms leveraging simple development of Augmented Reality (AR) applications based on natural features of the real scene. The following technique uses the scene texture to extract keypoints through Features from Accelerated Segment Test (FAST) [1]. Binary Robust Independent Elementary Features (BRIEF) [2] is the main focus of this article and it's used as an efficient feature point descriptor. Both techniques combined results in a performant marker-less tracking algorithm which fits the native web needs due to the existing browsers performance limitations.

## 1 Introduction

In the beginning of the 2D web plain text and images were the most advanced features available on the browsers, it has changed with the introduction of new browser features such as CSS3 [3], HTML5 [4] and WebGL [5]. Image processing and AR applications started to become possible without the installation of third-party plug-ins such as Flash or Silverlight though the Canvas element defined in the HTML5 [4] specification. Plug-ins are problematic as they require end-user installation which is not transparent, placing a burden on the end-user, and are frequently forbidden by companies security guidelines [6]. Client-side native web applications have intrinsic performance limitations, although this premise is changing due WebGL that expose a low-level JavaScript Application Programming Interface (API) that enables web applications to take advantage of 3D graphics hardware acceleration in a standard way, opening the way for enhancing the web experience. As result, this work proposes a marker-less tracking algorithm able to run natively on any modern browser without requiring any plug-in installation. The algorithm proposed in this work uses BRIEF [2]

that utilizes binary strings as an efficient feature point descriptor. This technique is highly discriminative even when utilizing few bits to store the descriptor. The descriptor can be evaluated using the Hamming distance, which is very efficient and could be calculated with few binary operations and yields a similar or better recognition performance when compared against SURF and U-SURF [7], while takes in a fraction of the time spent by either.

**Outline** The remainder of this article is organized as follows. Section 2 gives account of BRIEF [2] algorithm. The improvements proposed by this article are described in Section 3. Finally, Section 4 gives the conclusions.

### 1.1 Motivation

Feature point descriptors are currently the core of many Computer Vision technologies, such as object recognition, image retrieval, and camera localization. Having the real scene augmented with virtual objects, without requiring any artificial elements such as markers, running natively on web browsers capable to run in desktops and handled devices such as smart-phones and tablets with limited computational resources is very attractive. Currently there aren't any similar commercial or open source algorithm with the same characteristics available for native web. As results of the proposed technique applications could simultaneously tracks and detects an unknown object in a video stream making minimal assumptions about the object, the scene or the camera's motion requiring only initialization by a bounding box.

### 1.2 Objectives

The approach is therefore to build an algorithm using BRIEF [2] as feature point descriptor that runs on native web using new browser features such as HTML5 [4] Canvas element. JavaScript language was used as main language to develop the proposed algorithm.

## 2 Metodology

The first steps were research and analyze Binary Robust Independent Elementary Features (BRIEF) [2] and its current implementation in C++. Since web and handled devices have limited computational power having local descriptors that are fast to compute, to match and being memory efficient are important aspects, thus using binary strings reduces the size of the descriptor and provides an interesting data structure that is fast to operate with whose similarity can be measured by the Hamming distance which, on desktop implementations, the computation time could be driven almost to zero using the POPCNT instruction from SSE4.2 [8]. Only the latest Intel Core i7 CPUs support this instruction.

To generate the binary string for each keypoint found in the smoothed frame, the individual bits are obtained by comparing the intensities of pairs of points along the ker-

nel window centered on each key-point without requiring a training phase. Empirically, this technique shows that 256 bits or even 128 bits, often suffice to obtain very good matching results and the best spatial arrangement of the tested ($\mathbf{x}$, $\mathbf{y}$)-pairs of points have better results when selected based on an isotropic Gaussian distribution, which could be simply replaced by a random function due to its random characteristics.

To generate the binary strings is defined test $\tau$ on patch $\mathbf{p}$ of size $\mathbf{S} \times \mathbf{S}$ as

$$\tau(\mathbf{p}; x, y) := \begin{cases} 1 & \text{if} \quad \mathbf{p(x)} < \mathbf{p(y)}, \\ 0 & \text{otherwise} \end{cases}$$

where $\mathbf{p(x)}$ is the pixel intensity. The set of binary tests is defined by the $n_d$ ($\mathbf{x}$, $\mathbf{y}$)-location pairs uniquely chosen during the initialization. The $n_d$-dimensional bit-string is our BRIEF descriptor for each key-point

$$f_{n_d}(\mathbf{p}) := \sum_{1 \le i \le n_d} 2^{i-1} \tau(\mathbf{p}; x, y).$$

In [2], $n_d$ = 128, 256, and 512 were used in the tests and any of those values yield good compromises between speed, storage efficiency, and recognition rate. The number of bytes required to store the descriptor can be calculated by $k = n_d/8$, proving that BRIEF is also a memory-efficient method.

## 2.1 Analyses

The recognition rate could be associated as a function of the descriptor size, so for $n_d$ = 128, 256, and 512 tests, is denoted as BRIEF-$k$, BRIEF-16, BRIEF-32, and BRIEF-64. Where $k = n_d/8$ is the number of bytes required to store the descriptor. Increasing $n_d$ the recognition rates drops for all descriptors, therefore in this work $n_d$ = 128 showed to have a good recognition rate keeping a small memory consumption, only 16 bytes per key-point, thus it was the preferred $n_d$ value on the web implementation.

BRIEF is also not designed to be rotationally invariant, it tolerates small amounts of rotation, up to 10 to 15 degrees.

# 3 Improvements

The BRIEF technique performs very well compared to other feature descriptor algorithms, such as SURF and U-SURF [7], although to achieve the goal of having a tracking algorithm requires more than only the descriptor phase. Matching a number of points between two images typically involves four steps:

1. Detecting the feature points.
2. Computing the description vectors.
3. Matching, which means finding the best match in descriptor space.
4. Homography computation.

For steps (1) and (4) a technique needed to be chosen to work together with item (2) and (3) handled by BRIEF, the details for BRIEF is explained in Section 2. To

achieve the best results on the tracking algorithm was selected FAST [1] to detect feature points, and RANdom SAmple Consensus (RANSAC) [9] to calculate the homography (4), both techniques are very efficient combined with BRIEF [2].

## 3.1 Detecting the Feature Points

Corner detection is used as the first step of many vision tasks such as tracking and localization. In this article we call "feature" or "key"-point to refer to a point of interest in two dimensions. The method to extract feature points suggested in Features from Accelerated Segment Test (FAST) [1] was the Direct Grey-level Detectors, therefore is also used in this work.

The proposed Direct Grey-level Detectors works by testing a small patch of an image to see if it could be a corner. The detector is evaluated using a circle surrounding the candidate pixel, the test is based on whether the concentric contiguous arcs around the pixel are significantly different from the central pixel $p$.

To classify $p$ as a corner should exists a set of $n$ contiguous pixels in the circle which are all brighter than the intensity of the candidate pixel $I_p + t$ (threshold), or all darker than $I_p - t$. See Figure 1.

The repeatability of found feature points is also important because determines whether the technique is useful in a real-world application. The number of contiguous tested pixels could vary accordingly [1], being more common to be FAST-12 or FAST-9. Empirically, FAST-9 showed to have a good repeatability and a better efficiency on the web.

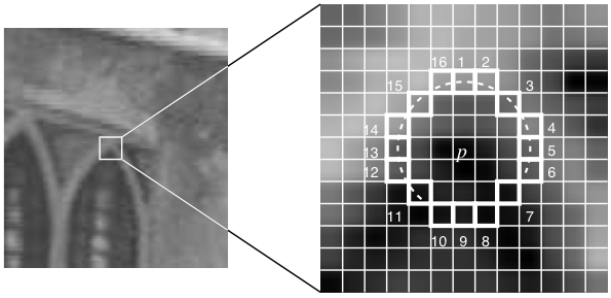


Figure 1: The highlighted squares are the pixels used in the corner detection. The pixel at $p$ is the central pixel. The arc is indicating that the dashed line passes through 12 contiguous pixels which are brighter or darker than $p$.

## 3.2 Homography Computation

Each feature point is extracted with FAST [1], described and matched with BRIEF [2] as results we have a set of matching points between the frames to test, there is a high chance that the matched points are correct (inliers), although the matched points could be wrong (outliers), thus they need to be scored to encounter the best homography. Since four correspondences determine a homography each pair of points is tested against the other pairs.

This robust estimation is defined by the following steps:

1. Randomly select two pair of points.

2. Compute the homography $H$ for those points.

3. Test if the computed homography attend the *inlier* condition. The calculated point should be within a distance threshold condition $c = (\sqrt{d_x^2 + d_y^2} < d_{min})$, where $d_x$ and $d_y$ is the delta between the original and the calculated point values and $d_{min}$ is the tolerated distance error, $d_{min} = 4$ for this article. The first pair $P_1$ is multiplied by $H$ to find the respective point $P_2$. Thus, $P_2 = H \times P_1$.

4. Repeat that process $N$ times, $N = 100$ empirically defined for this article.

5. The best homography is the one that generates more *inliers*.

## 3.3 Experiments

BRIEF [2] Section 2, FAST [1] Section 3.1 and RANSAC [9] Section 3.2 were implemented using JavaScript language as specified in ECMA-262, 5th edition. The tests were executed on the most popular browsers such as Firefox 4+, Chrome 15.0+, Safari 5+ and Internet Explorer 10+ and was tested on a Mac OS X 10.7.2, 3.0.6 GHz Intel Core 2 Duo 8GB RAM. The Canvas element defined in the HTML5 [4] specification was used to read the pixel matrix from the images and videos. Canvas was initially introduced by Apple for their own usage, later, it was adopted by Gecko browsers and Opera and standardized by the Web Hypertext Application Technology Working Group (WHATWG) on new proposed specifications for next generation web technologies, its specification describes an immediate-mode API and associated utility methods for drawing two-dimensional vector graphics to a raster rendering area.

## 3.4 Analysys

The results of the proposed algorithm running on HTML5 and JavaScript are presented in the following sub sections.

### 3.4.1 Detecting the Feature Points

Figure 2 and 3 represents the plotted key-points found with the method described in Section 3.1 to detect feature points using a threshold $t = 35$. Note that regardless the difference in the number of found key-points most part of them are invariant to the camera rotation.

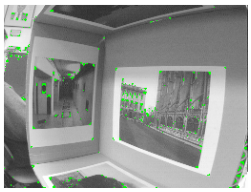

Figure 2: HTML5 Canvas image. 432 key-points found.

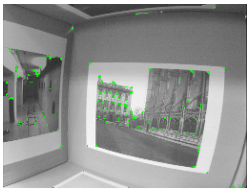

Figure 3: HTML5 Canvas rotated image. 634 key-points found.

### 3.4.2 Feature Point Descriptor

Figure 4 represents the descriptors matched by the closest Hamming distance calculated

using BRIEF [2] using threshold $t = 80$. Note that there are a big number of *outliers*.

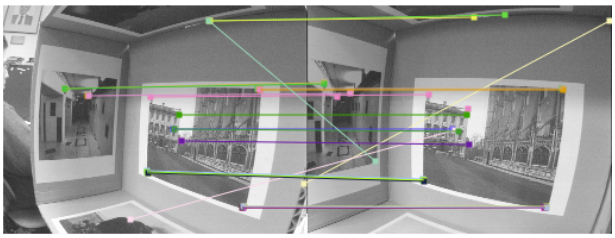

Figure 4: BRIEF descriptors matched using the Hamming distance, 27 in total.

### 3.4.3 Homography Computation

Figure 5 represents the points matched by the homography estimated using RANSAC [9] using threshold $t = 80$. Note that the number of *outliers* decreased when compared to Subsection 3.4.2 due to the robust estimation.

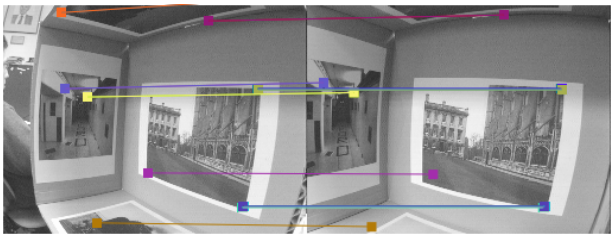

Figure 5: Points matched by homography estimated by RANSAC, 19 in total.

# 4 Conclusions

This work proposes a marker-less tracking algorithm for native web targeted platforms using HTML5 and JavaScript taking advantage of Features from Accelerated Segment Test (FAST) [1] to detect features points, Binary Robust Independent Elementary Features (BRIEF) [2] as a feature point descriptor and RANdom SAmple Consensus (RANSAC) [9] to do a robust estimation to compute the homography covering the four steps described in Section 3, resulting in an efficient marker-less tracking algorithm for the web.

# References


[1] E. Rosten, R. Porter, and T. Drummond, "Faster and better: A machine learning approach to corner detection," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 32, no. 1, pp. 105–119, 2010.

[2] M. Calonder, V. Lepetit, C. Strecha, and P. Fua, "Brief: Binary robust independent elementary features," *Computer Vision–ECCV 2010*, pp. 778–792, 2010.

[3] E. J. Etemad, *Cascading Style Sheets (CSS)*. W3C, http://www.w3.org/TR/css-2010/, 2010 ed., 5 2011.

[4] I. Hickson, *HTML5*. Google Inc., http://www.w3.org/TR/html5/, 2011 ed., 11 2011.

[5] C. Marrin, *WebGL Specification*. Apple Inc., 1.0 ed., February 2010.

[6] J. Behr, P. Eschler, Y. Jung, and M. Zöllner, "X3dom: a dom-based html5/x3d integration model," in *Proceedings of the 14th International Conference on 3D Web Technology*, pp. 127–135, ACM, 2009.

[7] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, "Speeded-up robust features (surf)," *Computer Vision and Image Understanding*, vol. 110, no. 3, pp. 346–359, 2008.

[8] Intel Corporation, *SSE4 Programming Reference*, d91561-003 ed., 07 2007.

[9] R. Hartley and A. Zisserman, *Multiple view geometry in computer vision*, vol. 2. Cambridge Univ Press, 2000.