Markerless Tracking Solutions for Augmented Reality on the Web

by

Eduardo A. Lundgren Melo

Submitted to the Center for Informatics
in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science

at the

FEDERAL UNIVERSITY OF PERNAMBUCO

February 2013

Author .....
Center for Informatics
Mar 20, 2013

Certified by .....
Silvio de Barros Melo
Associate Professor
Thesis Supervisor

Accepted by .....
Arthur C. Smith
Chairman, Department Committee on Master Theses

Markerless Tracking Solutions for Augmented Reality on the Web

by

Eduardo A. Lundgren Melo

Submitted to the Center for Informatics
on Mar 20, 2013, in partial fulfillment of the
requirements for the degree of
Master of Science in Computer Science

Abstract

In this thesis, I designed and implemented an Augmented Reality (AR) framework aiming to provide a common infrastructure to develop applications and to accelerate the use of those techniques on the web in commercial products. It runs on native web browsers without requiring third-party plugins installation. This involves the use of several modern browser specifications as well as implementation of different computer vision algorithms and techniques into the browser environment. These algorithms can be used to detect and recognize faces, identify objects, track moving objects, etc. The source language of the framework is JavaScript that is the language interpreted by all modern browsers. The majority of interpreted languages have limited computational power when compared to compiled languages, such as C. The computational complexity involved in AR requires highly optimized implementations. Some optimizations are discussed and implemented on this work in order to achieve good results when compared with similar implementations in compiled languages. A series of evaluation tests were made, to determine how effective these techniques were on the web.

Thesis Supervisor: Silvio de Barros Melo
Title: Associate Professor

Acknowledgments

This is the acknowledgements section. You should replace this with your own acknowledgements [7] foo [7].

This master thesis has been examined by a Committee as follows:

Professor Silvio de Barros Melo .....
Thesis Supervisor
Associate Professor

Professor Veronica Teichrieb .....
Chairman, Thesis Committee
Associate Professor

Professor Alvo Dumbledore .....
Member, Thesis Committee
Hogwarts School of Witchcraft and Wizardry

Contents

List of Figures

List of Tables

List of Acronyms

AJAX	Asynchronous JavaScript and XML
BAST	Bug Report Analysis and Search Tool
BTT	Bug Report Tracker Tool
BRN	Bug Report Network
CCB	Change Control Board

Listings

Chapter 1

Introduction

Lorem ipsum dolor sit amet, consectetur adipisicing elit.

1.1 Motivation

Lorem ipsum dolor sit amet, consectetur adipisicing elit.

1.2 Problem Definition

Lorem ipsum dolor sit amet, consectetur adipisicing elit.

1.3 Objectives

Lorem ipsum dolor sit amet, consectetur adipisicing elit.

1.4 Thesis Structure

Lorem ipsum dolor sit amet, consectetur adipisicing elit.

Chapter 2

Basic Concepts

2.1 Web

Using concepts from existing hypertext systems, Tim Berners-Lee, computer scientist and at that time employee of CERN, wrote a proposal in March 1989 for what would eventually become the World Wide Web (WWW) [30].

The World Wide Web is a shared information system operating on top of the Internet. Web browsers retrieve content and display from remote web servers using a stateless and anonymous protocol called HyperText Transfer Protocol (HTTP). Web pages are written using a simple language called HyperText Markup Language (HTML). They may be augmented with other technologies such as Cascading Style Sheets (CSS), which adds additional layout and style information to the page, and JavaScript (JS) language, which allows client-side computation. Browsers typically provide other useful features such as bookmarking, history, password management, and accessibility features to accommodate users with disabilities [7].

In the beginning of the web, plain text and images were the most advanced features available on the browsers. In 1994, the World Wide Web Consortium (W3C) was founded to promote interoperability among web technologies. Companies behind web browser development together with the web community, were able to contribute to the W3C specifications [30]. Today's web is a result of the ongoing efforts of an open web community that helps define these technologies and ensure that they're supported

in all web browsers. Those contributions transformed the web in a growing universe of interlinked pages and applications, with videos, photos, interactive content, 3D graphics processed by the Graphics Processing Unit (GPU), and other varieties of features without requiring any third-party plugins installation [10]. The significant reuse of open source components among different browsers and the emergence of extensive web standards have caused the browsers to exhibit "convergent evolution" [7].

The browser main functionality is to present a web resource, by requesting it from the server and displaying it on the browser window. There are four major browsers used today: Internet Explorer, Firefox, Safari and Chrome. Currently, the usage share of Firefox, Safari and Chrome together is nearly 60% [31].

Three mature browser implementations were selected and, for each browser, a conceptual architecture was described based on domain knowledge and available documentation. Firefox version 16.0, Safari version 6.0.4 and Chrome version 25.0.1364 were used to derive the reference architecture because they are mature systems, have reasonably large developer communities and user bases, provide good support for web standards, and are entirely open source.

The reference architecture for web browsers based on three well known open source implementations architecture, is shown in Figure 2-1; it comprises eight major subsystems plus the dependencies between them: (1) the User Interface, this includes the address bar, back and forward buttons, bookmarking menu etc. Every part of the browser display except the main window where you see the requested resource; (2) the Browser Engine, an embeddable component that provides a high-level interface for querying and manipulating the Rendering Engine; (3) the Rendering Engine, which performs parsing and layout for HTML documents, optionally styled with CSS; (4) the Networking subsystem, used for network calls, like HTTP requests. It has platform independent interface and underneath implementations for each platform; (5) the JavaScript Parser, used to parse and execute the JavaScript code; (6) the XML Parser; (7) the UI Backend, which provides drawing and windowing primitives, user interface widgets, and fonts. Underneath it uses the operating system user interface

methods; and (8) the Data Persistence subsystem, which stores various data associated with the browsing session on disk, including bookmarks, cookies, and cache [7].

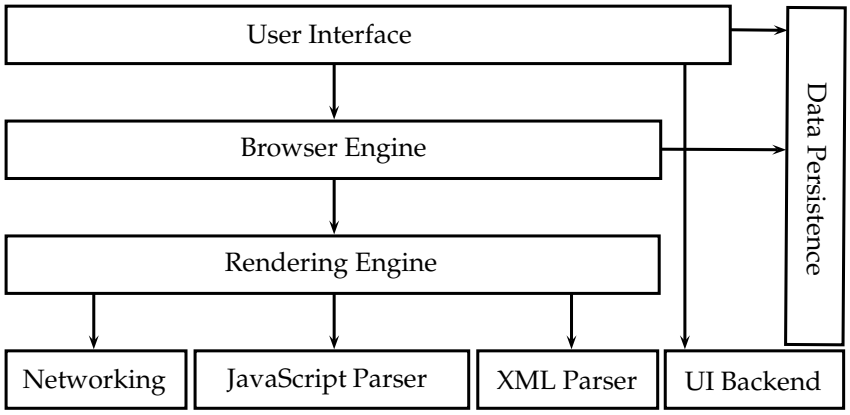

Figure 2-1: Reference architecture for web browsers

Browser subsystem are swappable and could vary for browser vendor, platform or operational system. The browsers mostly differ between different vendors in subsystems (2) the Browser Engine, (3) the Rendering Engine, and (5) the JavaScript Parser. Firefox subsystems (2) and (3) is known as Gecko [27] [25], Safari as WebKit [12] [13] and Chrome uses a fork of WebKit project called Blink [15] [16]. Those browsers subsystems, often called Browser Engines, are shown on Figure 2-2.

Another common swappable subsystem is (5) the JavaScript Parser. JavaScript is a lightweight, interpreted, object-oriented language with first-class functions, most known as the scripting language for Web pages [25]. The JavaScript standard is ECMAScript. As of 2013, all modern browsers fully support ECMAScript 5.1. Older browsers support at least ECMAScript 3 [25] [20].

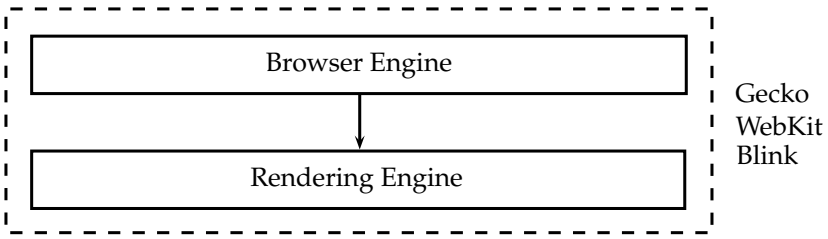

Figure 2-2: Reference architecture for browsers engines

The JavaScript language is intended to be used within some larger environment, be it a browser, server-side scripts, or similar. For a basic example of the language syntax a *println* might have been defined in Listing 2.1.

```
1 function println(string) {
2   window.alert(string);
3 }
```

Listing 2.1: Basic example of JavaScript syntax

JavaScript core language features comprises few major features: (1) Functions and function scope, function is a "subprogram" that can be called by code external, functions have a scope it references for execution; (2) Global Objects, refer to objects in the global scope, such as general-purpose constructors (*Array, Boolean, Date* etc.), Typed array constructors (*Float32Array, Int32Array, Uint32Array* etc.), Error constructors etc.; (3) Statements, consist of keywords used with the appropriate syntax (*function, if...else, block, break, const, continue, debugger etc.*); (4) Operators and keywords, arithmetic operators, bitwise operators, assignment operators, comparison operators, logical operators, string operators, member operators, conditional operator etc. [26].

As web applications become more and more powerful, adding features such as audio and video manipulation, access to raw data using WebSockets [26], and so forth, it has become clear that there are times when it would be helpful for JavaScript code to be able to quickly and easily manipulate raw binary data. In the past, this had to be simulated by treating the raw data as a string and using the *charCodeAt()* method to read the bytes from the data buffer [26] [9].

However, this is slow and error-prone, due to the need for multiple conversions, especially if the binary data is not actually byte-format data, but, for example, 32-bit integers or floats. Superior, and typed data structures were added to JavaScript specification, such as JavaScript typed arrays [20].

JavaScript typed arrays provide a mechanism for accessing raw binary data much more efficiently. This thesis takes advantage of typed arrays in order to achieve

acceptable performance on the web of complex algorithms implementations.

A performance benchmark comparing regular *vs* typed arrays were executed on the three well known open-source browsers, Firefox, Safari and Chrome. The comparison was executed on Mac OS X 10.8.3, 2.6 GHz Intel Core i7 16 GB 1600 MHz RAM. The array types selected were the not strongly typed *Array*; *Float32Array*, represents an array of 32-bit floating point numbers; *Uint8Array*, represents an array of 8-bit unsigned integers.

For each array type a read and a write operation were executed $100,000$ times. In order to not compromise benchmark results caused by run-time type conversion [20], the write value used for each array type were proper selected, e.g. *Number* 1.0 was used for regular arrays *Array*, *Number* 1.0 was used for *Float32Array*, and unsigned *Number* 1 for *Uint8Array*. Regular *vs* typed arrays performance benchmark is shown in Figure 2-3 [1].

As conclusion, typed arrays provides faster read and write operations than regular arrays in JavaScript, i.e. 7872 *ops/sec* for unsigned array *vs* 4437 *ops/sec* for regular arrays in Firefox browser, similar behavior is noticeable on Safari and Chrome, thereby float and unsigned arrays are vastly used on complex algorithms implementations on the web.

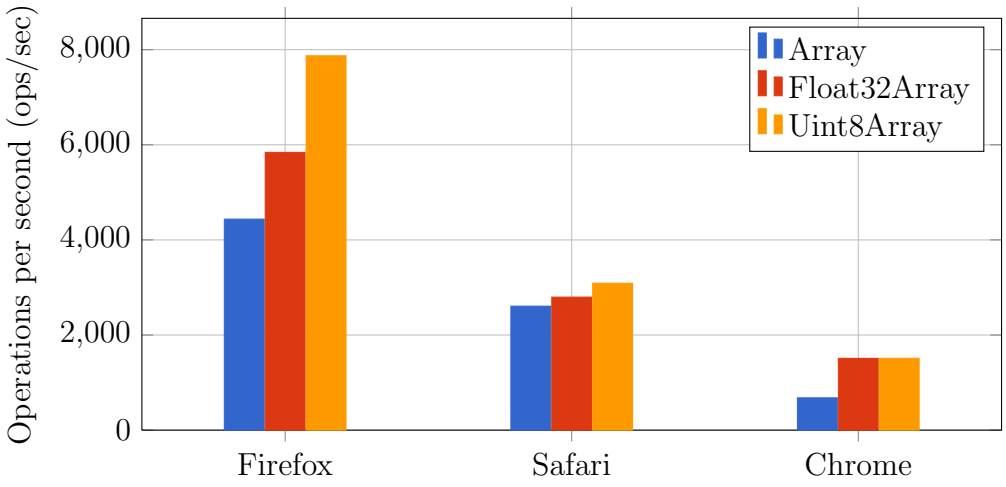

Figure 2-3: Regular *vs* typed arrays performance benchmark

Nevertheless, reading and writing raw binary data using typed arrays only solves

part of the problem of manipulating video and images data. The other missing feature was solved by HTML5 [10] *canvas* element, which one important feature is to provide access to the pixel matrix of those medias. The raw binary data is used by Augmented Reality (AR) algorithms.

The *canvas* element provides scripts with a resolution-dependent bitmap canvas, which can be used for rendering graphs, game graphics, art, or other visual images on the fly [4]. Authors should not use the *canvas* element in a document when a more suitable element is available, e.g. it is inappropriate to use a *canvas* element to render a page heading. The usage of *canvas* conveys essentially the same function or purpose as the canvas' bitmap. Listing 2.2 shows an example of a basic *canvas* element HTML markup given a width and height in pixels.

```
1 <canvas width="200" height="200"></canvas>
```

Listing 2.2: The HTML canvas element markup

For each *canvas* element a "context" is available, then, from that, the drawing context can be accessed and JavaScript commands can be invoked to draw or read data. Browsers can implement multiple canvas contexts and the different APIs provide the drawing functionality. Most of the major browsers include the 2D canvas context capabilities. Individual vendors have experimented with their own three-dimensional canvas APIs, but none of them have been standardized. The HTML5 [10] specification notes, "A future version of this specification will probably define a 3D context" [4]. Even though 3D context is not available in most part of the major browsers, three-dimensional applications are already being developed based on the 2D canvas context.

Is mandatory the use of the *canvas* element to develop AR applications on the web. Canvas provides APIs to: Draw basic shapes, images, videos frames, Bezier and quadratic curves [8]; Apply transformations, translate, rotate and scale; Read raw pixel data etc.

The canvas is a two-dimensional grid that could be described as a simple computer graphics coordinate system [8]. Normally 1 unit in the grid corresponds to 1 pixel

on the canvas. The origin of this grid is positioned in the top left corner coordinate $(0, 0)$. All elements are placed relative to this origin. So the position of the top left corner of the blue square becomes x pixels from the left and y pixels from the top coordinate (x, y). The canvas coordinate space is shown on Figure 2-4 [26].

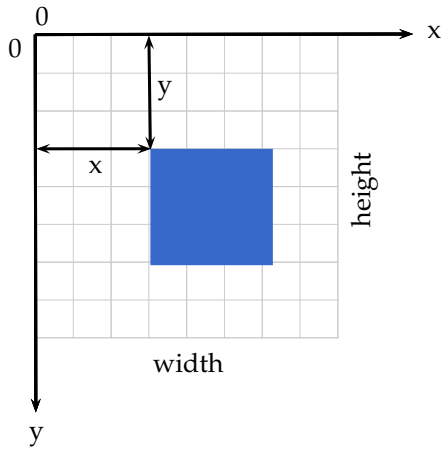

Figure 2-4: The canvas coordinate space

Videos and images can also be "hosted" on a canvas bitmap. Canvas raw binary data can be accessed from the canvas JavaScript API as an object of type *ImageData* [4]. Each object has three properties: width, height and data. The data property is of type *Uint8ClampedArray* that is a one-dimensional array containing the data in RGBA order, as integers in the range 0 to 255. The *Uint8ClampedArray* interface type is specifically used in the definition of the canvas element's 2D API and its structure is similar to the previous shown typed array *Uint8Array* [4] [9].

The *ImageData* data property, or array of pixels, is in row-major order, a multidimensional array in linear memory. For example, consider the 2×3 array $\begin{bmatrix} 1 & 2 & 3 \\ 5 & 5 & 6 \end{bmatrix}$, in row-major order it is laid out contiguously in linear memory as $\begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 \end{bmatrix}$. Each array value is represented as integers between 0 and 255, where each four-integer group represents the four color channels of one pixel: red, green, blue and alpha (RGBA). While RGBA is sometimes described as a color space, it is actually simply a use of the RGB color model [6]. An example of the canvas image data array

of pixels is shown on Figure 2-5.

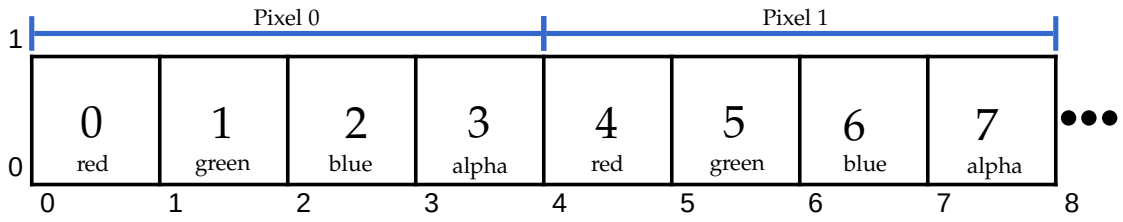

Figure 2-5: The canvas image data array of pixels

Audio and video capture has been a limitation of web browsers for a long time. For many years the authors had to rely on browser plugins, such as Flash [11] or Silverlight [23] [17]. HTML5 [10] has brought a surge of access to device hardware, Geolocation (GPS), the Orientation API (accelerometer), WebGL [14] and the Real-time Communication Between Browsers specification (WebRTC) [19] in conjunction with Media Capture and Streams specification, a set of JavaScript APIs that allow local media, including audio and video, to be requested from a platform trough *getUserMedia* API [30].

With *getUserMedia*, the browser can access the camera and microphone input without a plugin. It's available directly into the browser. The access camera and microphone hardware access can be used in combination with the HTML5 [10] *audio* and *video* elements. The access flow of raw binary data captured from videos on modern browsers is shown on Figure 2-6. It comprises five major steps: (1) Hardware access, camera and microphone hardware is accessed by the browser; (2) Streaming, using *getUserMedia* API the hardware streams audio and video to the browser UI Elements; (3) UI Elements, HTML elements that displays the data stream; (4) Raw Binary Data, HTML *canvas* element that "hosts" video frames providing access to the array of pixels;

The *audio* and *video* tag is one of those HTML5 [10] features that gets a lot of attention. Often presented as an alternative to Flash [11] in the media, the video tag has advantages due to its natural integration with the other layers of the web development stack such as CSS and JavaScript as well as the other HTML tags. The

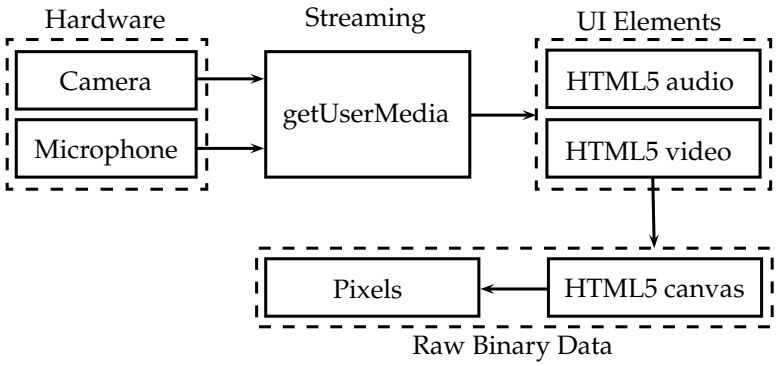

Figure 2-6: Access flow of raw binary data captured from videos on modern browsers

three video formats supported by the three well known browsers cited in this thesis are, webm (VP8 Vorbis), mp4 (H.264 AAC) and ogv (Theora Vorbis) [30] [17]. The audio formats available are ogg (Theora Vorbis) and mp4 (H.264 AAC). An example of *getUserMedia* API being used to capture the camera and microphone using a *video* tag to display is shown on Listing 2.3.

```
1  <video autoplay></video>
2  <script>
3    var video = document.querySelector('video');
4    navigator.getUserMedia(
5      {video: true, audio: true},
6      function(localMediaStream) {
7          video.src = window.URL.createObjectURL(localMediaStream);
8          video.onloadedmetadata = function(e) {
9            // Ready to go.
10         };
11     },
12     onFail);
13 </script>
```

Listing 2.3: Capture and display microphone and camera

2.2 Augmented Reality

Imagine a technology in which you could see more than others see, hear more than others hear, and even touch things that others cannot. A technology able to perceive computational elements and objects within our real world experience that help us in our daily activities, while interacting almost unconsciously through mere gestures and speech.

With such technology, mechanics could see instructions what to do next when repairing an unknown piece of equipment, surgeons could take advantage of augmented reality while performing surgery on them and we could read reviews for each restaurant in the street we're walking in on the way to work [21].

Augmented reality (AR) is this technology to create a "next generation, reality-based interface" [21] and is moving from laboratories around the world into various industries and consumer markets. AR supplements the real world with virtual (computer-generated) objects that appear to coexist in the same space as the real world. AR was recognized as an emerging technology of 2007 [21], and with today's smart-phones and AR browsers we are starting to embrace this very new and exciting kind of human-computer interaction.

On the reality-virtuality *continuum* by Milgram and Ki [24], Augmented Reality (AR) is one part of the general area of mixed reality. Both virtual environments (or virtual reality) and augmented virtuality, in which real objects are added to virtual ones, replace the surrounding environment by a virtual one. In contrast, AR provides local virtuality. The reality-virtuality *continuum* is shown on Figure 2-7. Three important aspects of an AR system: (1) combines real and virtual objects in a real environment; (2) registers (aligns) real and virtual objects with each other and (3) runs interactively, in three dimensions, and in real time [2].

The first AR prototypes, shown in Figure 2-8, created by computer graphics pioneer Ivan Sutherland and his students at Harvard University and the University of Utah, appeared in the 1960s and used a see-through to present 3D graphics [2]. It took until the early 1990s before the term "augmented reality" was proposed by Caudell

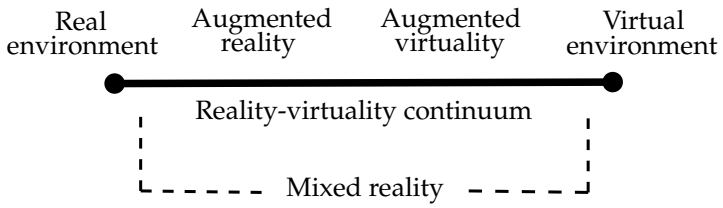

Figure 2-7: Reality-virtuality continuum

and Mizell [2], scientists at Boeing Corporation who were developing an experimental AR system to help workers put together wiring harnesses.

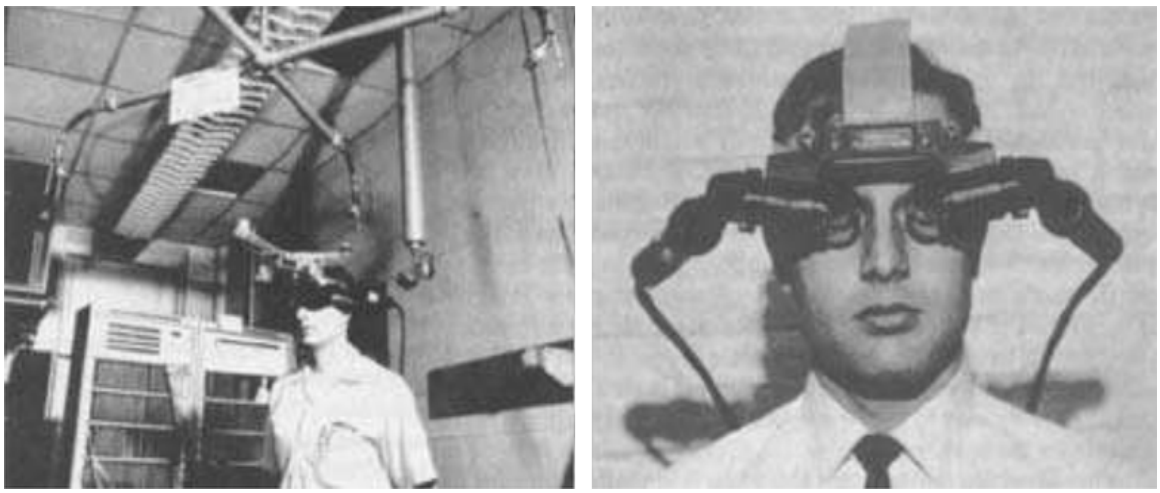

Figure 2-8: The world's first head-mounted display with the "Sword of Damocles"

By the late 1990s, as AR became a distinct field of research. Displays, trackers, graphics computers and software remain essential in many AR experiences. There are several displays devices that could be used on AR applications, the major ones could be divided as follows: (1) Visual display: Video see-through, optical see-through and projective; (2) Display positioning: Head-worn, hand-held and spatial [2].

More details for each sub-section from section (1) Visual displays and (2) Display positioning, are presented in the following paragraphs.

Section (1) Visual display: Video see-through is the closest to virtual reality, where the virtual environment is replaced by a video feed of reality and the AR is placed on the video frames. The optical see-through systems combine computer-generated imagery with holographic optical elements (HOEs), providing a "through the glasses"

image of the real world. The last sub-category in section (1) is the projective, these displays have the advantage that they do not require special eye-wear, it projects the information into a surface, wall or even the human palm [2]. The use of a regular computer monitor has become popular as a visual display since AR techniques are now available on the Internet.

Section (2) Display positioning: Head-worn is a visual displays attached to the head include the video or optical see-through head-mounted display (HMD), virtual retinal display (VRD), and head-mounted projective display (HMPD). Google Inc. released a first-class optical HDM called Project Glass shown on Figure 2-9 [18]. The hand-held sub-category includes hand-held video or optical see-through displays as well as hand-held projectors, it is currently the best choice to introduce AR to a mass market due to low production costs and ease of use since it may be based on existing consumer products, such as mobile phones. The last sub-category in section (2) is the spatial, those are displays that could placed statically within the environment and include screen-based video see-through displays [2].

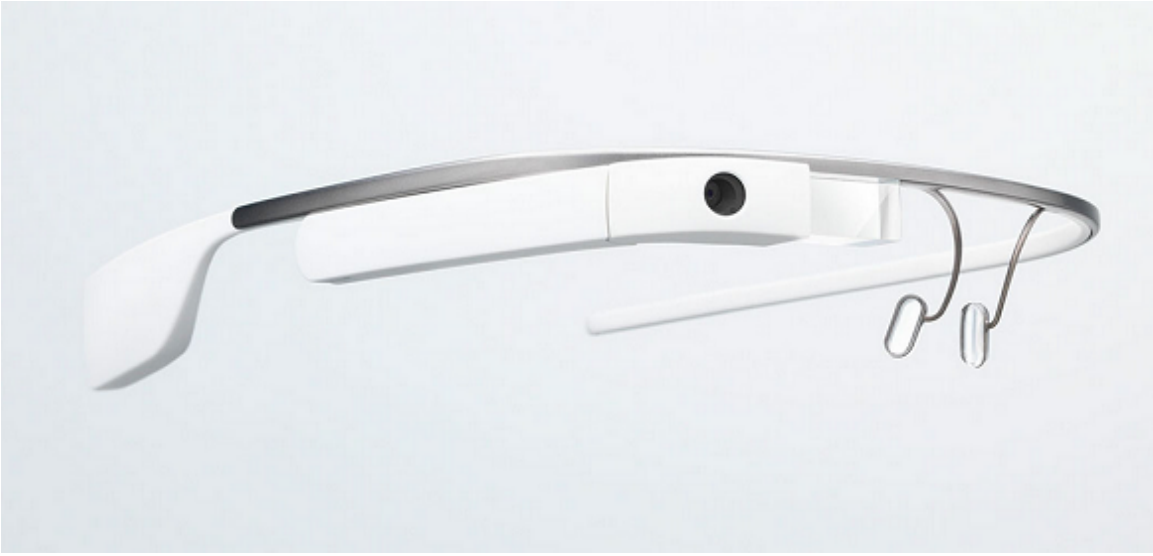

Figure 2-9: Optical head-mounted display Google Glass [18]

In this thesis, it was designed and implemented an Augmented Reality (AR) framework aiming to provide a common infrastructure to develop applications and to accelerate the use of those techniques on the web in commercial products. It runs

on native web browsers without requiring third-party plugins installation, therefore any browser ready device can eventually use the proposed cross-platform code base to develop AR applications. In order to develop for different devices different skills are required, since APIs and programming languages may differ between them. In the current day, the most popular devices, such as smart phones, tablets, computers, notebooks and HDM (i.e. Google Glass) are browser ready. They all could benefit from a reusable, cross-platform framework for AR applications.

2.3 Augmented Reality on the Web

Client-side native web applications use to have intrinsic performance limitations, although this premise is changing. Browsers are evolving very fast when compared to the the previous decade. JavaScript language wasn't prepared to handle typed data structures able to manipulate raw binary data safely, all the computational complexity required by AR algorithms was too much for that growing environment. Browsers weren't able to capture audio and video natively, without plugin installation, an essential feature for AR applications. This reality has changed, this involves the use of several modern browser specifications as well as implementation of different computer vision algorithms and techniques into the browser environment taking advantage of all those modern APIs. Some optimizations are discussed and implemented on this work in order to achieve good results when compared with similar implementations in compiled languages.

2.4 Tracking

Tracking an object in a video sequence means continuously identifying its location when either the object or the camera are moving. There are a variety of approaches, depending on the type of object, the degrees of freedom of the object and the camera, and the target application. 2D tracking typically aims at following the image projection of objects or parts of objects whose 3D displacement results in a motion

that can be modeled as a 2D transformation [22].

One of the major responsibility of a tracking algorithms is to extract information that allows superposing computer generated images on real scenes. Many potential Augmented Reality (AR) applications have been explored, such as medical visualization, maintenance and repair, annotation, entertainment, aircraft navigation and targeting. The objects in the real and virtual worlds must be properly aligned and the system latency should also be low, lest the illusion that the two worlds coexist be compromised. [22] Due to the importance of tracking in AR, this area owns the biggest number of publications and challenge requests on the International Symposium on Mixed and Augmented Reality (ISMAR) [32].

There are a variety of ways to track a real environment: e.g. Global Positioning System (GPS), mechanical trackers, magnetic trackers, ultrasonic trackers etc. [21], but they all have their weaknesses. Mechanical trackers are accurate enough, although they tether the user to a limited working. Magnetic trackers are vulnerable to distortions by metal in the environment. Ultrasonic trackers suffer from noise and tend to be inaccurate at long ranges because of variations in the ambient temperature [22].

By contrast, vision has the potential to yield non-invasive, accurate and low-cost solutions to this problem, provided that one is willing to invest the effort required to develop sufficiently robust algorithms. Bringing video tracking based techniques on the web are the focus of this work [22].

In some cases, it is acceptable to add fiducials, such as special markers, to the scene or target object. The addition of fiducials in the scene, also called landmarks or markers, greatly helps since they constitute image features easy to extract, and they provide reliable, easy to exploit measurements for the pose estimation.

It is therefore much more desirable to rely on naturally present features, such as edges, corners, or texture. The latter, makes tracking far more difficult because finding and following feature points or edges can be difficult. There are too few of them on many typical objects. Section 2.5 details markerless tracking techniques, main technique implemented on the web by this thesis.

2.5 Markerless Tracking Techniques

Markerless Augmented Reality (MAR) systems integrate virtual objects into a real environment in real-time, enhancing user's perception of and interaction with the world. This approach is not based on the use of traditional artificial markers that need to be placed in the world to be tracked by the system in order to calculate their position and orientation [29], instead, MAR relies on naturally present features in order to position virtual objects. Tracking techniques become more complex in MAR systems when compared to marker techniques.

MAR can be classified in two major types: Model based and Structure from Motion (SfM) based. The online monocular MAR taxonomy is shown on Figure 2-10.

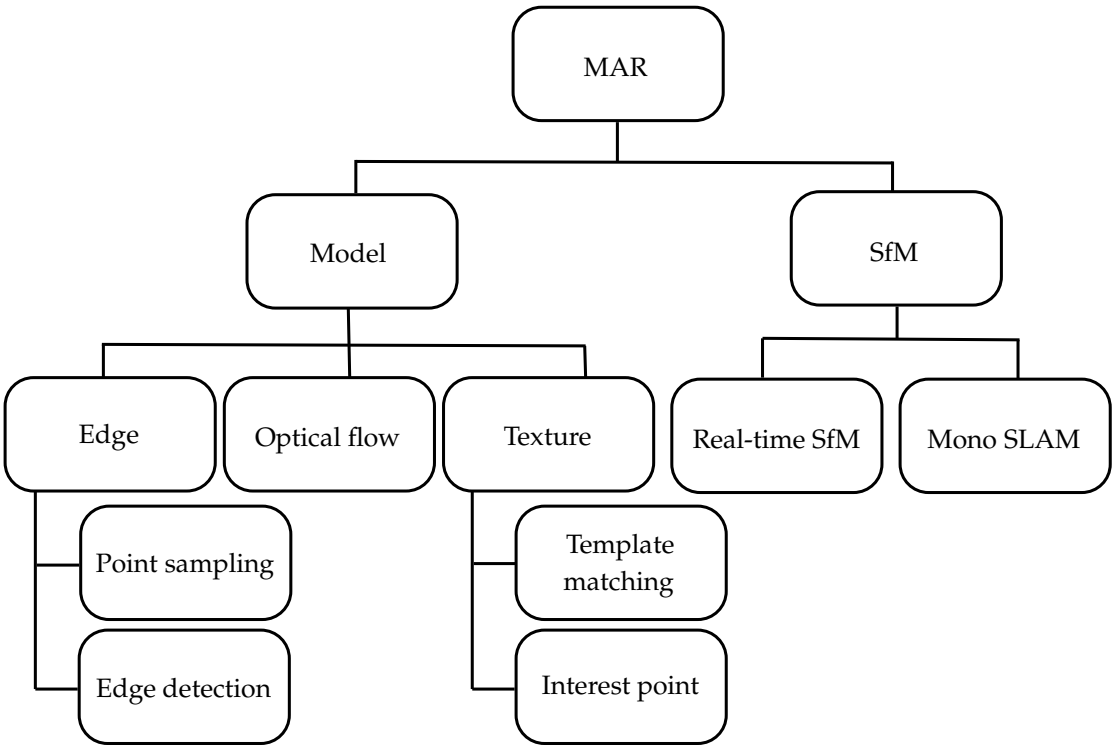

Figure 2-10: Online monocular Markerless Augmented Reality taxonomy

2.5.1 Structure from Motion (SfM)

SfM based systems have advantages, since they are capable of continuously tracking the camera in unknown scenes, although it's not detailed in this thesis. Model based

systems usually requires lower computational complexity, ideal to the web environment.

2.5.2 Model Based

The advantage of using a model based approach is the possibility of interaction between real and virtual worlds, like occlusion and collision. In order to accomplish such types of interaction, the application exploits the fact that the real object pose is known and its structure is described by the 3D model. Model based approaches require an a priori knowledge about the real scene. Due to the offline model acquiring process, these techniques are, in general, not totally online and cannot be used in unprepared environments. Furthermore, tracker initialization is usually done manually or requires a prior training in order to be automatic [29].

Model based techniques can be classified in three categories, (1) Edge Based, The camera pose is estimated by matching a wireframe 3D model of an object with the real world image edge information; (2) Optical Flow Based, rely on spatial information obtained by image-model matching extracted from the relative movement of the object projection onto the image; and (3) Texture Based, takes into account texture information presented in images.

Category (3) Texture Based is the tracking technique used in this thesis, therefore a much detailed description is presented in the following lines. Texture Based could be divided into two sub-categories: template-matching and interest point.

The template matching approach is based on global information, this subcategory lies in its ability to treat complex patterns that would be difficult to model by local features. These techniques are also called sum-of-square-difference (SSD), as they consist in minimizing the difference between a region of the image and a reference template.

The subcategory of interest point based (or feature points) techniques takes into account localized features. As a result, this subcategory is less computer-intensive than template matching approach, this characteristic makes interest point technique a good choice for AR applications on the web.

The tracking solution proposed, uses Features from Accelerated Segment Test (FAST) [28] to extracts feature points using corner detection. This technique is detailed on Section 3.3. Binary Robust Independent Elementary Features (BRIEF) [3] is used as an efficient feature point descriptor (also known as feature matching), detailed on Section 3.4. Both techniques combined results in a high performance marker-less tracking solution which fits the native web needs due to the existing browsers performance limitations.

Chapter 3

Augmented Reality Library for the Web (tracking.js)

3.1 Introduction

Lorem ipsum dolor sit amet, consectetur adipisicing elit.

3.2 Marker-less Tracking Algorithm (Keypoints)

Lorem ipsum dolor sit amet, consectetur adipisicing elit.

3.3 Interest Point Detection

This technique rely on matching individual features across images and are therefore easy to robustify against partial occlusions or matching errors. Illumination invariance is also simple to achieve. Feature Points detection is used as the first step of many vision tasks such as tracking, localization, image matching and recognition. In this article we call "feature" or "keypoint" to refer to a point of interest in two dimensions.

The algorithms perform a simple loo and for each frame, the object features are matched by localizing feature templates in search windows around hypothesized locations [22]. The method to extract feature points suggested in Features from Ac-

celerated Segment Test (FAST) [28]. FAST hypothesizes the matches using corner detection. A large number of corner detectors exist in the literature. However, we have a strong interest in realtime frame rate applications which computational resources are required requisites. The approach proposed by FAST allows the detector to produce a suite of high-speed detectors which we currently use for real-time tracking and AR label placement [3]. In particular, it is still true that when processing live video streams at full frame rate, existing feature detectors leave little if any time for further processing, even despite the consequences of Moore's Law [28].

To show that speed can been obtained without necessarily sacrificing the quality of the feature detector, in Chapter 4, we compare our detector, to a variety of well-known detectors. A number of the detectors described below compute a corner response, (1) Edge based corner detectors, corresponds to the boundary between two regions; (2) Greylevel derivative based detectors, the assumption that corners exist along edges is an inadequate model for patches of texture and point like features, and is difficult to use at junctions. Therefore a large number of detectors operate directly on greylevel images without requiring edge detection; and (3) Direct greylevel detectors, Another major class of corner detectors work by examining a small patch of an image to see if it "looks" like a corner [28].

The thesis choice was (3) Direct greylevel detectors. It 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 [28]. 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 $I_p - t$ [28].

The number of contiguous tested pixels could vary accordingly [28], 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. The repeatability of found feature points is also important because determines whether the technique is useful in a real-world application.

This detector in itself exhibits high performance, but there are several weaknesses: (1) This high-speed test does not reject as many candidates; (2) The efficiency of the detector will depend on the ordering of the questions and the distribution of corner appearances; and (3) Multiple features are detected adjacent to one another [28].

On Figure 3-1, 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 FAST-n, let n be 9 or 12 contiguous pixels which are brighter or darker than p [28].

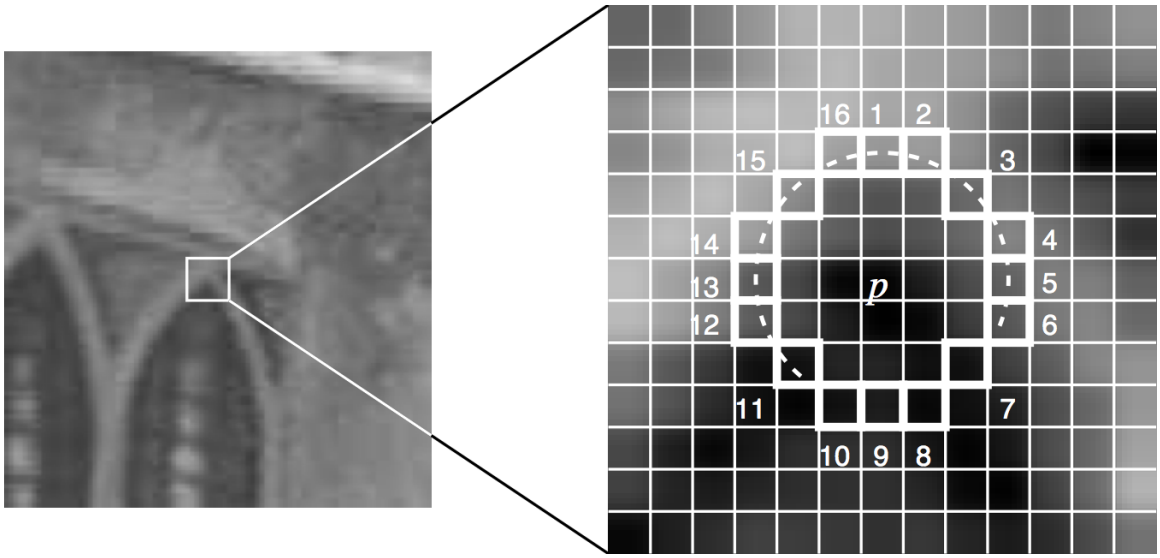

Figure 3-1: Point segment test corner detection in an image patch [18]

3.4 Interest Point Matching

To estimate motion, one can then match sets of interest points $\{m_i\}$ and $\{m'_j\}$ extracted from two images taken from similar, and often successive, viewpoints. A classical procedure [3] runs as follows. For each point $\{m_i\}$ in the first image, search in a region of the second image around location $\{m_i\}$ for point $\{m'_j\}$. The search is based on the similarity of the local image windows centered on the points, which strongly characterizes the points when the images are sufficiently close [22].

The feature matching used in the case studies performed in this work search for

correspondent points in the current frame. Only points that are highly descriptive invariant features, called keypoints, are tested. Those keypoints were detected using FAST [28] in Section 3.3.

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, therefore an efficient method Binary Robust Independent Elementary Features (BRIEF) [3] was used.

Distance metric is critical to the performance of intrusion detection systems. 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 [5]. Only the latest Intel Core i7 CPUs support this instruction.

The Hamming distance is an important step on interest point matching, it provides a fast and memory-efficient way to calculate distance between binary strings. Given two image patches x and y, denote their binary descriptors as $b(x) \in \{0,1\}^n$ and $b(y) \in \{0,1\}^n$ respectively. Then their Hamming distance is computed by:

$$Ham(x,y) = \sum_{i=1}^{n} b_i(x) \otimes b_i(y)$$

In which n is the dimension of binary descriptor and stands for bitwise XOR operation. According to the definition of Hamming distance, all the elements of a binary descriptor contribute equally to the distance. From the hamming distance, the Hamming weight can be calculated. It is used to find the best feature point match. Here, is generalized the Hamming distance to the weighted Hamming:

$$WHam(x,y) = \sum_{i=1}^{n} w_i(b_i(x) \otimes b_i(y))$$

Where w_i is the weight of the ith element. The goal is to learn $w_i, i = 1, 2 \cdots, n$ for the binary descriptor (BRIEF) based on a set of feature points. By assigning different weights to binary codes, what we expect is to obtain a distance space in

which the distances of matching patches are less than those of non-matching patches.

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

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

In [3], $n_d = 128$, 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. Detailed results can be found in Chapter 4.

3.5 Pose Estimation

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo.

Chapter 4

Evaluation

4.1 Tools

OpenCV, JSFlartoolkit, Others. Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

4.2 Scenario Description

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

4.3 Evaluation Methodology

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

4.3.1 Matching Robustness

4.3.2 Oclusion Robustness

4.3.3 FPS

4.4 Results

Graphics, Analysys. Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Chapter 5

Conclusion

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

5.1 Contributions

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

5.2 Future Work

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud

exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Bibliography

[1] Eduardo A Lundgren Melo. Typed Arrays Performance, 2013.

[2] Steve Benford, Chris Greenhalgh, Gail Reynard, Chris Brown, and Boriana Koleva. Understanding and constructing shared spaces with mixed-reality boundaries. *ACM Transactions on Computer-Human Interaction*, 5(3):185–223, 1998.

[3] Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal Fua. BRIEF : Binary Robust Independent Elementary Features. *Computer*, 6314(3):778–792, 2010.

[4] WHATWG Community. The canvas element, 2013.

[5] Intel Corporation. Intel® SSE4 Programming Reference, 2007.

[6] Rafael C Gonzalez and Richard E Woods. *Digital Image Processing (3rd Edition)*. Prentice Hall, 2007.

[7] Alan Grosskurth and M.W. Michael W Godfrey. A reference architecture for web browsers. In *Software Maintenance, 2005. ICSM'05. Proceedings of the 21st IEEE International Conference on*, pages 661–664. IEEE, IEEE, 2005.

[8] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*, volume 2. Cambridge University Press, 2004.

[9] David Herman and Kenneth Russell. Typed Array Specification, 2013.

[10] Ian Hickson. HTML 5 Nightly Specification (W3C), 2013.

[11] Adobe Inc. Adobe Flash, 2013.

[12] Apple Inc. Safari Browser, 2013.

[13] Apple Inc. The WebKit Open Source Project, 2013.

[14] Apple Inc. WebGL Specification, 2013.

[15] Google Inc. Google Chrome Browser, 2010.

[16] Google Inc. Blink, 2013.

[17] Google Inc. HTML5 Rocks Tutorials. 2013.

[18] Google Inc. Project Glass, 2013.

[19] Google Inc., Mozilla Inc., and Opera Inc. WebRTC, 2013.

[20] Ecma International. ECMA-262 ECMAScript Language Specification. *JavaScript Specification*, 16(December):1–252, 2009.

[21] D W F Van Krevelen and R Poelman. A Survey of Augmented Reality Technologies , Applications and Limitations. *International Journal*, 9(2):1–20, 2010.

[22] Vincent Lepetit and Pascal Fua. Monocular Model-Based 3D Tracking of Rigid Objects. *Foundations and Trends® in Computer Graphics and Vision*, 1(1):1–89, 2005.

[23] Microsoft. Silverlight, 2013.

[24] Pranav Mistry, Pattie Maes, and Liyan Chang. WUW - Wear Ur World - A Wearable Gestural Interface. *Proceedings of the 27th international conference extended abstracts on Human factors in computing systems*, 68(3):4111–4116, 2009.

[25] Inc. Mozilla. Gecko, 2013.

[26] Inc. Mozilla. Mozilla Developer Network, 2013.

[27] Inc. Mozilla. Mozilla Firefox Browser, 2013.

[28] Edward Rosten, Reid Porter, and Tom Drummond. Machine learning for high-speed corner detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(1):1–14, 2010.

[29] Veronica Teichrieb, Monte Lima, Eduardo Lourenc, Silva Bueno, Judith Kelner, and Ismael H F Santos. A Survey of Online Monocular Markerless Augmented Reality. *International Journal of Modeling and Simulation for the Petroleum Industry*, 1(1):1–7, 2007.

[30] W C. The World Wide Web Consortium (W3C), 2006.

[31] Wikimedia. Wikimedia Traffic Analysis Report - Browsers, 2013.

[32] Feng Zhou, Henry Been-lirn Duh, and Mark Billinghurst. Trends in augmented reality tracking, interaction and display: A review of ten years of ISMAR. *2008 7th IEEEACM International Symposium on Mixed and Augmented Reality*, 2(4):193–202, 2008.