VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

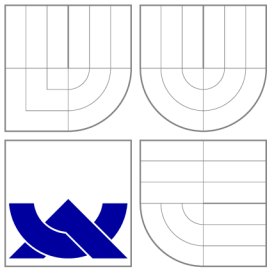
3D OBJECT RENDERING INTO REAL ENVIRONMENTS USING MOBILE DEVICES

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

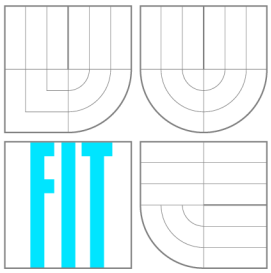
AUTOR PRÁCE
AUTHOR

JÁN ŠVEHLA

BRNO 2013



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

ZOBRAZOVÁNÍ 3D OBJEKTŮ DO OBRAZU REÁLNÉHO SVĚTA NA MOBILNÍCH ZAŘÍZENÍCH

3D OBJECT RENDERING INTO REAL ENVIRONMENTS USING MOBILE DEVICES

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

JÁN ŠVEHLA

VEDOUCÍ PRÁCE
SUPERVISOR

Prof. (FH) DI DR. REGINE BOLTER,

BRNO 2013

Abstrakt

Tato bakalářská práce je zaměřená na problémy vyskytující se při tvorbě aplikace pro mobilní zařízení využívající rozšířenou realitu. Jako vyvíjená aplikace byla zvolena jednoduchá strategická hra. Tato práce provede čtenáře základními tématami a problémy rozšířené reality, jejího využití a možnostech na mobilních zařízeních a samotným návrhem a implementací vyvíjené aplikace. Výsledek této práce je možno využít pro vývoj mobilních her nebo obecních aplikací využívajících rozšířenou realitu.

Abstract

This bachelor's thesis is aimed at the problems and issues which are encountered over development of application for mobile devices using augmented reality. As a developed application was chosen a simple tower-defense game. This thesis will guide the reader through general topics and issues of augmented reality, it's usage and possibilities with mobile devices and actual design and implementation of developed application. Results of this work can be used for development of mobile games or general purpose augmented reality applications.

Klíčová slova

rozšířená realita, 3D objekt, vykreslování, Android, Android NDK, tower-defense, ARLanding

Keywords

augmented reality, 3D object, rendering, Android, Android NDK, tower-defense, ARLanding

Citace

Ján Švehla: 3D Object Rendering into Real Environments Using Mobile Devices, bakalářská práce, Brno, FIT VUT v Brně, 2013

3D Object Rendering into Real Environments Using Mobile Devices

Declaration

I hereby declare that this thesis and the project reported herein is my original authorial work which I did on my own, and that all sources, references and literature used or cited in this thesis were properly acknowledged by complete reference to the source.

.....
Ján Švehla
May 22, 2013

Acknowledgement

I would like to thank my supervisor, Prof. (FH) Dl DR. Regine Bolter for taking on my thesis, inspiring me and overall help.

I also extend my thanks to Ing. Vítěslav Beran, Ph.D. for supervising my thesis in early stages and helping me to gain knowledge about augmented reality.

Last but not least I would like to thank my family and my friends for supporting me during work on this thesis.

Contents

Chapter 1

Introduction

Since the beginning of advanced computer graphics and virtual reality it was possible to display it only on monitor screens or similar equipment. However, people wanted to bring the technology closer. Augmentation of real world with virtual objects has always been very desired, but it was possible only in sci-fi.

With technological advancement came also the first mobile augmented reality devices. Attribute "mobile" might not be on the right place. To achieve enough processing power and required accessories these devices were sizeable and heavy. They used mainly head-mounted displays and the user had to carry the computer with him. The price was also unaffordable so it was used only in specific industry or academic circles.

Nowadays, almost everybody has a device capable of augmented reality. Smartphones provide all resources needed for it - sufficient performance, camera and also displaying unit. In addition they have sensors as GPS or accelerometer, which extends possibilities of usage.

This thesis will be aimed at possibilities of current tools for development of augmented reality application on Android platform and development itself. For this purpose a simple game using augmented reality will be developed. Principles of game are described in the third chapter.

The theoretical part will introduce issues of augmented reality and mobile devices. At first some history of augmented reality, basic principles and also examples of current usage will be described. The Second part will be about 3D models and how they work at Android platform. It will cover graphical subsystem on Android, most important graphics library - OpenGL ES and also general overview of 3D modelling. The end of the chapter will describe and analyze current frameworks for augmented reality applications.

The next chapter will be about design of developed application. It will state application goals and expected properties. It will also describe the integration with the chosen framework. Also, graphical user interface design and 3D models for application will be described.

There will be also described design of graphical user interface and 3D models used for the application.

In implementation you will be able to read about the main problems and the most important parts of implementation. It will cover processing of output received from framework, connection between Android SDK and native code.

Evaluation will cover fulfillment of expected attributes of application and used framework. It will discuss imperfections, deficiencies, and possible solutions.

Chapter 2

Theoretical Introduction

2.1 Augmented Reality

The term Augmented reality describes technologies allowing the user to see and interact with virtual computer-generated content overlapping the real world. It extends the user's perception of the world by mixing the view of the real world with virtual objects that contextually belong there. All this must work in real-time and in 3 dimensions. Augmented reality is not bond only to vision. It can be applied even to other senses - hearing, touch, smell.

Figure 2.1 shows reality-virtuality continuum from article [11]. On the one extreme there is the real environment, on the other the virtual environment. Everything in between belongs to mixed reality. Paul Milgram and colective [11] distinguishes between augmented reality and augmented virtuality. The first one is based on the real environment augmented on objects from the virtual environment. The other one does the exact opposite. The example could be a virtual landscape augmented with real persons.

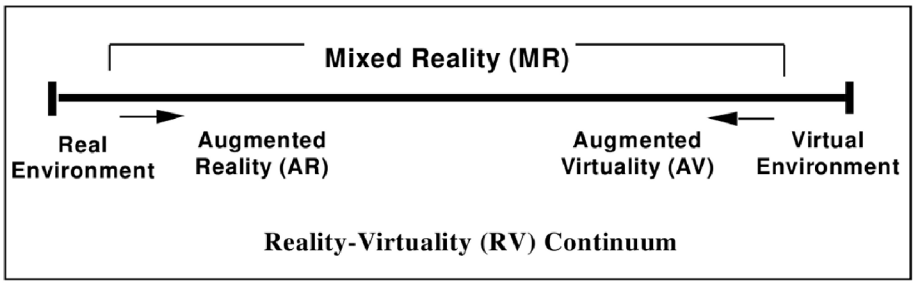


Figure 2.1: RV continuum Source: [11]

A few years ago augmented reality was known only in academic circles or in specific fields such as medicine, the automotive industry, etc. Very sizeable computers and specialized equipment were required for its operation. With the onset of smartphones, augmented reality quickly got in touch with the public.

Applications that provide augmented reality on smartphones use input data, which can be divided in 3 categories:

- Camera data
  Images from the camera are used as Real Environment from figure 2.1 in which augmented virtual objects will be blended. Additionally, they are used for recognition of

objects in the real environment.

- Location data
  These data are gained from GPS sensor or other location services as finding nearby WiFi access points. They are used mainly for positioning relevant objects. Recognition of objects from camera, especially in large areas, is not always adequate. If it is, it is very computationally expensive.

- Other sensor data
  They are required for specifying the orientation of virtual object in space. These data can be gained from gyroscope or compass. By using these data it is easier to correctly display an object at different angles.

Approaches to augmented reality

Devices which are capable of augmenting reality are nowadays accessible very easy [3]. The most common types are listed below.

- Head-mounted displays
  There are two different head-mounted display technologies to bring graphics onto the user's view of the real world. The first type is video see-through head-mounted displays that mix the virtual content with video background from attached camera device. Second approach is optical see-through head-mounted display that uses optical combiners as for example transparent LCD display. These two methods are the most comfortable but they also have many disadvantages. Optical see-through display requires difficult calibration and precise head-tracking to ensure correct graphical overlay. Good example of optical see-through display is Google Glass 2.2.

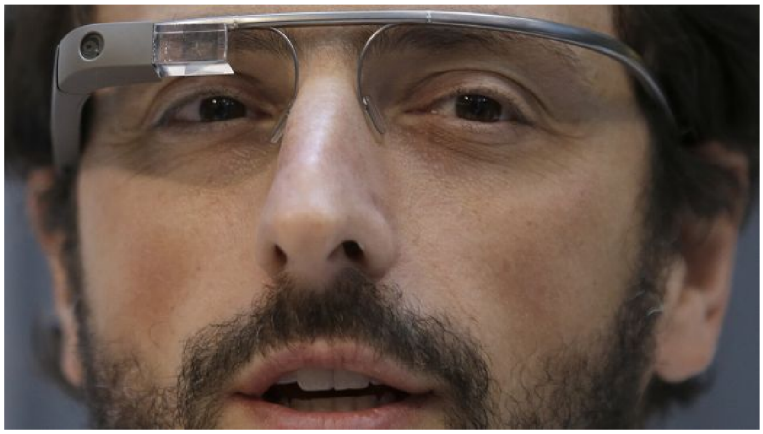


Figure 2.2: Optical see-through head-mounted augmented reality device Google Glass. Source:[1]

- Stationary, Screen-Based displays
  An augmented reality system can easily be set up with a stationary computer, a monitor and a webcam. These systems make use of video-mixing (video see-through) and display the merged images on a regular monitor. It can be used for magic mirror

[1] Source: http://www.foxnews.com/tech/2013/03/10/no-joke-guy-cant-walk-into-seattle-bar-wearing-google-glasses/

[5] metaphor which is a concept that instead of making the user see through the display, makes the user see him or herself in the display, as if it was a mirror. This can for example be achieved by putting the webcam on the top of the monitor and pointing it at the user. Example of this approach is again from Google - Google Hangout 2.3.

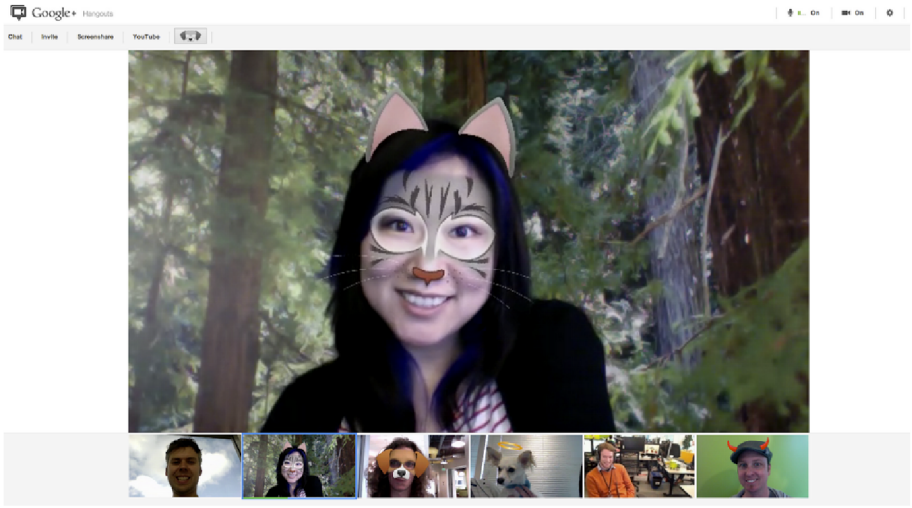


Figure 2.3: Example of magic mirror metaphor - Google Hangout. Source:[2]

- Handheld
  Nowadays almost every mobile device, e.g. smartphone or tablet, has built-in camera and rather large screen, which is exactly what is needed for an augmented reality system. The camera is mostly placed at the back of device, pointing away from the user. This makes it suitable for video see-through display technology. Device's camera captures the real environment, which is used as a background. Virtual content is added as an overlay and the result is displayed on device's screen. This approach is used also in this thesis.

Figure 2.4: Handheld device used for augmented reality. Source:[3]

[2]Source: https://plus.google.com/+googleplus/posts/NrGoAZKpyTx

Smartphones and tablets have standardly useful built-in equipment such as GPS units, digital compasses and six degrees of freedom (2.1) acelerometer–gyroscope. Another advantage is the portable nature of these devices. Disadvantage is the physical constraint of the user having to hold the handheld device out in front of him at all times. This limits the possibilities of interaction with real environment. On figure 2.4 you can see a smartphone which is used as handheld AR device.

Application examples

The first possibility of use for the public is use of interior design. Users use their AR device to see the real environment of their rooms augmented with their desired pieces of furniture. The result is almost authentic.

Alternative use of AR, for example is a tourist guide. The user aims his device at the street and the application adds labels of interesting places. By using location and sensor data virtual objects will be positioned right at the real world objects.

Essential possibilities are offered in entertainment. Adding game objects into the real environment have always been a dream. With the new possibility of easy and effective interactivity the gap between the real world and gaming will become minimal.

Mobile phones now have sufficient processing power for simple computer vision, video decoding and interactive 3D graphics. Many device manufacturers are also fitting graphics processing units (GPUs) into mobile phones, providing faster graphic and hardware floating-point support. This enables a very wide range of use.

Camera registration problem

Using location data to determine the location of the viewer (camera) is not precise enough. Thus, when using such sensors, it is necessary to employ a hybrid system that combines vision-based methods and sensor data. Vision-based registration relies on the identification of features in the images. In a simple example artificial markers are placed in the real world, model-based, and/ or natural features are used for the registration. Markers are designed to be easily detectable; however, arranging the markers takes extra efforts and it also limits the user's moving range.

Most of the previous work on natural feature tracking in AR has only attempted to track two-dimensional (2D) features across a sequence of images. This is considerably simpler than the full 3D problem and it can readily be achieved in real time. The recovered 2D motion field can be used to estimate the change in positions of labels for 2D geographic labeling applications: A possible approach is to measure the optical flow [12] between adjacent image frames. For the special case, in which the camera motion is pure rotation or the viewed scene is planar, the 2D positions of corresponding features in two different camera views are related by a homography.

Six Degrees of Freedom

It refers to the ability of the tracking system to maintain alignment of a real world object in three dimensional space. For a typical smartphone AR application, 6 degrees of freedom are possible. Location sensors are able to provide forward/back and left/right, up/down (GPS)

[3]Source: http://archive.picnicnetwork.org/page/22322/en

and yaw (compass) and the accelerometer can indicate pitch and roll of the device. Similarly image recognition techniques can calculate angles from a known reference orientation.

2.2 Android platform and 3D models

Android is an open source platform based on linux kernel. It is primary intended for devices with touch screen as smartphones or tablets. Android software stack includes operating system, middleware, user interface and user applications. Users can download thousands of applications through online market.

Brief history of Android

The first mention about Android platform comes from 2005 when Google bought a little start-up called Android. They were developing a simple and flexible operating system for wide range of devices. One of the primary goals was an easy and comfortable downloading of new actualizations. Three years later, in 2008 the first version of Android was released.

From the introduction of the first version, until the time of writing this thesis, Google released 10 major updates. Each of this releases carries a codename of a dessert.

Architecture of Android system

The Android software stack consists of five sections ordered in four layers as seen on figure 2.5.

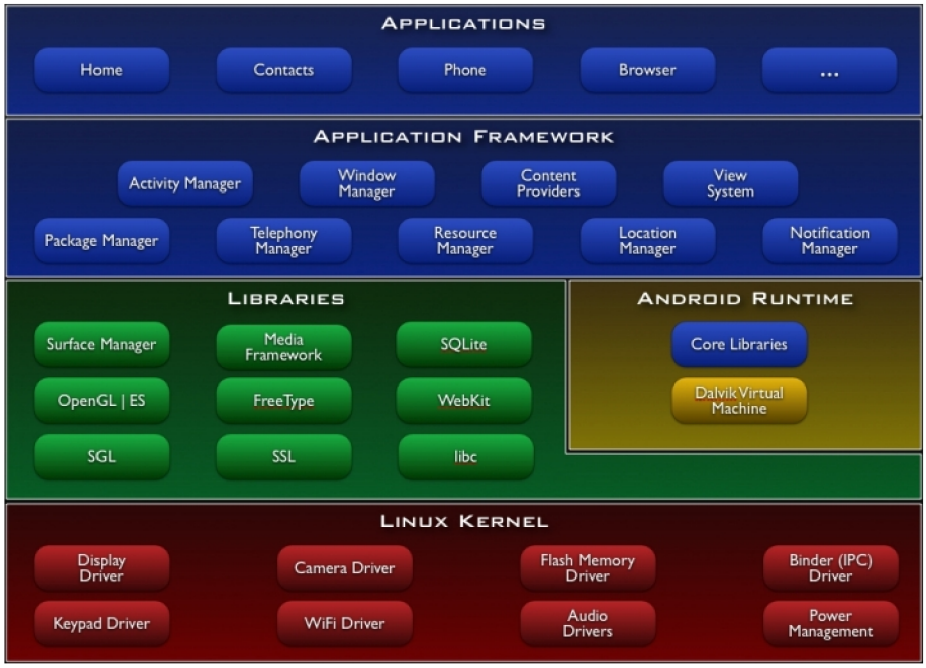


Figure 2.5: Android architecture.[4]

[4]Source: http://developer.android.com/images/system-architecture.jpg

Linux kernel - takes care about essential system services like security, management of processes, memory, networks and drivers. Kernel acts as abstract layer between hardware and software.

Libraries - Android contains set of libraries written in C and C++ language which access various components of system. They are accessible for developers through the Application framework.

Android runtime - every application for Android runs in its own process and own instance of virtual machine Dalvik. Virtual machine Dalvik transforms application from Java code to navite code.

Application framework - it is a layer containing other libraries, this time in Java, which form their own system API. It is a set of functions which enable the developer to work with parts of operating system.

Applications - build-in applications which provide the basic functionality for user. User can also download applications from online market.

2.2.1 Android Native Environment

The NDK (Native Development Kit) [13] is an adjunct to the Android's SDK (Software Development Kit). The Android VM allows application's source code to call methods implemented in native code through the JNI. Native code runs on the native device instead of in Java on the Dalvik virtual machine. Since communication between the virtual machine and the native machine is costly, normally it is best to develop applications completely in the SDK for the virtual machine.

The Android framework provides two ways of using native code:

- Write application using the Android framework and use JNI to access the APIs provided by the Android NDK. It covers declaring one or more methods with the `native` keyword to indicate that they are implemented through native code, providing a native shared library named according to Unix convention, which will be packaged into the .apk archive and statically loading the libraries at startup.

- Use native activity which will be implemented purely in native code. Android lifecycle events as onCreate(), onPause(), onResume(), will be forwarded to native environment where the developer can handle them. This method requires Android 2.3 or higher.

Very good example of usage is the Unreal Engine. It's a game engine - libraries for game physics, graphics libraries and other complex parts of computer video game. It's written with very complex commands and algorithms that just aren't possible or practical to code in Java and run under Android's Dalvik machine. The NDK allows developers to use the Unreal Engine, almost as written, to build intense 3D games.

In this thesis Android NDK will be used for major part of the implementation. It provides very good performance, even with older device. Development was though more difficult and not so comfortable that in Java. Application was developed using Eclipse[5] IDE and for Android NDK it missed many features that were available for Java delopment.

[5] See: http://www.eclipse.org/

2.2.2 Android Graphics Subsystem

Android provides a variety of powerful APIs for applying animation to UI elements and drawing custom 2D and 3D graphics. Article [2] describes the software components in the Android operating system responsible for graphics.

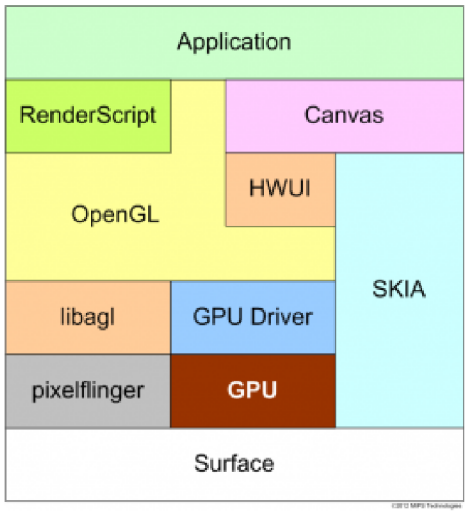


Figure 2.6: Approximate relationship between various graphics components in a typical Android application. Source: [2]

- GPU
  Graphical processing units (GPUs) are specialized processors with dedicated memory that conventionally perform floating point operations required for rendering graphics. Current generation of GPUs has evolved into many-core processors that are specifically designed to perform data-parallel computation.

- HWUI
  The HWUI library enables UI components to be accelerated using the GPU. HWUI was introduced in Honeycomb to make the user interface fast, responsive and smooth. Since tablets had very high resolutions, using Skia for effects like animation would have been too intensive and slow for CPU. HWUI requires an OpenGL ES 2.0 compliant GPU which cannot be emulated by software on Android.

- Canvas
  Canvas is the Android class that application developers would use to draw widgets, pictures etc. In Android versions Froyo and Gingerbread Canvas would do the drawing using Skia. In Android Honeycomb and onward, HWUI was added as an alternate GPU-accelerated option. From Android Ice Cream Sandwich (ICS) HWUI is the default.

- Skia
  Skia Graphics Engine is a compact open source 2D graphics library. For performance reasons Skia is slowly being replaced by HWUI.

- Renderscript
  Renderscript was a new API introduced in Honeycomb to address portability and

performance at the same time. The application developer writes the code in the Renderscript language (which is based on C99), and an LLVM cross compiler on the host converts it to machine-independent IR called bit code, the developer then packages the bitcode files into the Android application (APK). When the user downloads the APK, an on-device compiler (LLVM based, located in /system/lib/libbcc.so) compiles it from bit code to the actual machine instructions for the target platform.

- Surface
  A Surface in Android corresponds to an off screen buffer into which an application renders the contents. The application might be a game which uses OpenGL to draw 3D objects directly into a surface or a normal application which uses Skia to draw widgets, text, etc. It could even use HWUI library to enable a GPU accelerated user interface. From Android ICS, surfaces are implemented with a SurfaceTexture backend which means instead of a buffer a texture is used.

- OpenGL ES OpenGL is the most widely adopted standard for real-time computer graphics, covering most operating systems. It is used for visualizing 2D and 3D data. It is more described in the following text.

2.2.3 OpenGL ES and 3D models

OpenGL [6], the Open Graphics Library, is a standard specification defining a cross-language, multiplatform API for writing applications and simulating physics, that produce 2D and 3D computer graphics. ES in the name stands for Embedded Systems, and OpenGL ES[7] is a modified version of OpenGL for smaller, less powerful devices such as Android devices. The interface consists of over 250 different function calls, which can be used to draw complex three-dimensional scenes from simple primitives. OpenGL works around 3 basic concepts:

- Primitives
  OpenGL's primitives is limited to 3 types of objects:

  3D Point in space (x,y,z) - can be used as a particle in the space.

  3D Line in space (composed by two 3D Points) - can be used as a 3D vector.

  3D Triangle in space (composed by three 3D Points) - it is a type of polygon

- Buffers
  In simple words, the term buffer refers to a temporary optimized storage space. OpenGL works with 3 kind of buffers: Frame Buffer is represented as a special array on the video card. It has several components: `colour buffer` for RGBA, `depth buffer` for z-depth in space, `stencil buffer` for determining the visible part of objects and `accumulation buffer` for the intersection of objects.

  Render Buffer is a temporary storage of one single image. It has also a few components: `Color Render Buffer` stores the final colored image generated by OpenGL's render. Color Render Buffer is a colored (RGB) image. `Depth Render Buffer` stores the final Z depth information of the objects. It's a grey scale image about the Z position of the objects in 3D space. `Stencil Render Buffer` is aware about the visible part of the object. It is a black and white image.

[6] OpenGL, Khronos Group, http://www.opengl.org/
[7] OpenGL ES, Khronos Group, http://www.khronos.org/opengles/

Buffer Objects is the general term for unformatted linear memory allocated by the OpenGL context. These can be used to store information about 3D objects in an optimized format such as vertex data, pixel data retrieved from images or the framebuffer, and a variety of other things.

- Rasterize
  Rasterization is the process which a primitive is converted to a two-dimensional image. Rasterizing a primitive consists of two parts. The first is to determine which squares of an integer grid in window coordinates are occupied by the primitive. The second is assigning a color and a depth value to each such square.

3D modelling

A 3D Model [4] is a mathematical representation of any three-dimensional object (real or imagined) in a 3D software environment. Unlike a 2D image, 3D models can be viewed in specialized software suites from any angle, and can be scaled, rotated, or freely modified. The process of creating and shaping a 3D model is known as 3D modelling.

There are many concepts and representations of 3D models of which the most interesting are:

- NURBS models allows a curve to be created with the use of two control points. These points are generated in multiples to create the skeletal system of the model. This form of modelling is the best one for objects that are not going to be animated, as they require a lot of modifications to be suitable for the animation process. One of the main advantages of curve-based modelling in comparison with polygonal modelling is that it is resolution-independent. The software application interpolates the space between curves and creates a smooth mesh between them. NURBS surfaces have the highest level of mathematical precision, and therefore they are the most frequently used in modelling for engineering and automotive design.

- Polygonal modelling is the oldest type of 3D modelling. It is an approach for modelling objects by representing or approximating their surfaces using polygons. 3D polygonal models are composed of

  - Face stands for each element consisting of more than 2 vertices. Normally polygons are either four sided (character/organic modelling) or three sided (commonly in game modelling).
  - Edge is any point on the surface of a 3D model where two polygonal faces meet.
  - Vertes is a point of intersection between three or more edges in three dimensional space

- Constructive solid geometry uses simple objects called solids, constructed according to geometric rules. The special properties of CSG solids allow mathematical operations that are not possible with an arbitrary polygon mesh. The standard CSG primitives consist of the block, triangular prism, sphere, cylinder, cone and torus. The primitive may require transformations such as scaling, translation, and rotation to be positioned at the desired place. A CSG procedure can be represented by a tree structure. The root of the tree defines the object of interest, and the leaf nodes are geometric primitives. In between the root and the leaves lie operator nodes. Allowable operations are typically Boolean operations on sets: union, intersection and difference. Example of CSG tree is on figure 2.7.

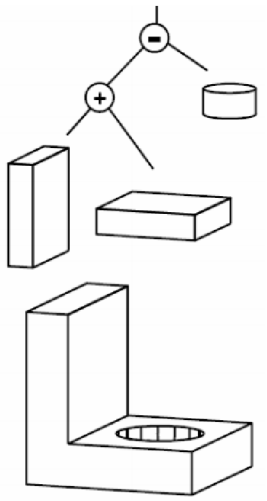


Figure 2.7: A simple CSG model as a result of boolean operations. Source:[8]

2.3 Frameworks for augmented reality

Thanks to general purpose and big appeal of users Augmented Reality attracted even many developers. Issues of augmented reality are perceived to be complex and difficult. This and the demand of developers for simple accessibility of augmented reality functionality allowed creation of many AR frameworks.

2.3.1 Overview of existing frameworks

In this subsection smaller projects are shortly described with interesting ideas. Some of them are standalone application or just libraries.

AndAR

AndAr [1] is simple marker-based augmented reality framework. Its architecture is shown at figure 2.8. Developer has to extend the abstract class `AndARActivity` which already handles everything Augmented Reality related, like opening the camera, detecting the markers and displaying the video stream. To detect markers developer has to register `ARObjects` to an instance of `ARToolkit`. This instance can be retrieved from the `AndARActivity`. The constructor requires filename of a pattern which can be created by a tool called `mk_patt`. The class `ARRenderer` is responsible for everything OpenGL related. In `OpenGLRenderer` interface there are three methods defined. One of them is `setupEnv` which is called once before the augmented objects are drawn. It can be used to issue OpenGL commands that shall effect all `ARObjects`, like initializing the lighting.

[8] Source: http://www.fit.vutbr.cz/study/courses/IZG/private/lecture/izg_slide_3d_objekty_present.pdf

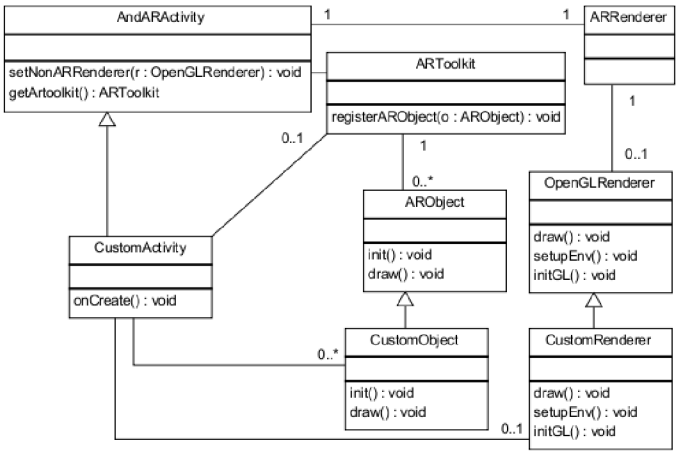


Figure 2.8: Simplified class diagram of an application that makes use of AndAR. Source: [14]

DroidAR

Droidar [6] is open-source Augmented Reality framework for Android developed by Simon Heinen, a computer science student from RWTH Aachen in Germany. It provides location-based or marker-based Augmented Reality functionality. DroidAR has great 3D model support using libgdx[9] framework. Application works with 3 display layers - image from device's camera, OpenGL layer with transparent background and default android overlay with UI.

The starting point is `Setup` class where developer configures the application. Developer can create GUI, add actions to events, create virtual world with objects, and camera. There is also method for updating virtual objects - providing animation and rotation of objects.

Mixare

Mixare [9] is a free open source augmented reality browser that works as a completely autonomous application and is available for the development of own implementations as well. Mixare is pure location-based. Application can read a list of markers in XML or JSON format, giving the latitude, longitude, and also altitude. For example, you can get an XML list from OSM (a map location provider) saying the nearby metro stations, loading it into Mixare apps. According to device's location from GPS it can show correct direction to targets or calculate the distance in AR view.

Target's position can be specified also with altitude. However, GPS altitude is rather inaccurate, so the position is calculated from the actual latitude+longitude by setting the altitude to a certain height. Targets will span only across +20 and +45 degrees above the horizon.

2.3.2 Junaio

Junaio [10] is an Augmented Reality platform for Android and iOS. It provides open web API for developers. Developer develops only content which will be delivered to user through

[9]libgdx - Android/HTML5/desktop game development framework, http://code.google.com/p/libgdx/

Junaio application in user's smartphone. User subscribes to developer's channel using application. There are 2 types of channels:

- Location-based channels
- GLUE channels (image recognition)

Junaio application sends sensor data (corresponding to channel's type) to Junaio servers. They will try to find appropriate POI (Point Of Interest) belonging to subscribed channel. If they are successful, they will send a request with POI ID, User ID, etc to developer's web server. There is a response with related content created - images, 3D models, videos and sent back to Junaio server. Junaio server checks the message and forwards to user's smartphone where it is displayed. This whole process is shown on figure 2.9.

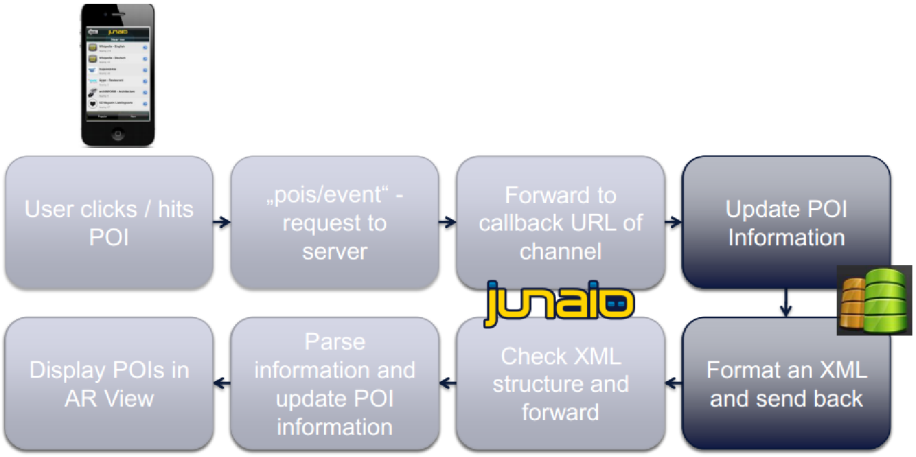


Figure 2.9: Communication cycle of Junaio AR browser. Source: [10]

Image recognition-based channels - Junaio GLUE

Junaio GLUE is service for image recognition with no markers are required. Besides classic 2D image recognition, it allows also 3D object recognition comparing to 3D model target. Recognition is processed at Junaio servers where the application sends the images captured from the device's camera. After successful recognition Junaio servers send request to channel owner's web server which returns relevant content. Junaio server forwards it along with camera registration data back to the user's device.

Junaio also offers a tool for creating 3D object targets. 3D object is placed on a flat surface. Next to it a special marker (printed on paper sheet) is placed. 3D object is captured by using device's camera and by application Creator Mobile. Application creates approximate 3D model which is sent to developer's email, ready for further editing.

Location-based Channels

Every smartphone that Junaio supports has a GPS sensor, gyroscope, and compass. These sensors must be enabled when using Junaio application. After successful localization of device, coordinates are sent to Junaio servers. They determine the nearest Points Of Interest

[10]Source: http://dev.junaio.com/publisherDownload/junaio_LocationBased.pdf

belonging to subscribed channel and send requests to developer's web server. Response with relevant content is created and forwarded to the user's device. Camera position is determined by using sensors data. POIs are displayed as virtual objects - signs with text, image or video.

Junaio also allows location-based content inside building where GPS signal is not available - using LLA markers. An LLA marker is similar to QR code but contains only geographical coordinates. They are used in the same way as if gained from GPS sensor.

2.3.3 Vuforia

Vuforia [7] is vision-based AR platform. It is available for Android, iOS and Unity. For Android it offers whole SDK with libraries for complete AR application. It includes computer vision algorithms, target creation tools, target management, cloud database tools, object rendering etc. Overview of SDK is on figure 2.10. Information about framework is available in developer guide [8].

This framework was chosen for this thesis. We used only marker targets which proved their functionality very well. Vuforia framework contained also sample applications from which one was used as a reference for our developed application. It contained good example of initialization of framework as well as processing data gained from framework's computer vision. Vuforia has also extensive forum for developers where a wide range of application development topics are discussed.

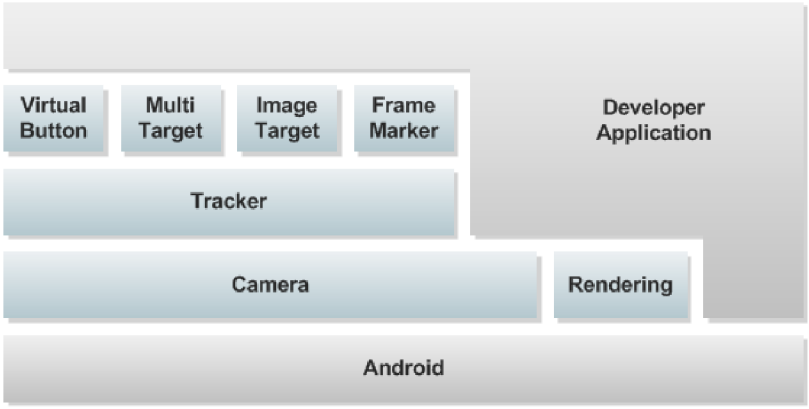


Figure 2.10: High-level system overview of Vuforia SDK Source: [7]

Targets

Targets are representations of real-world objects that can be detected and tracked. Vuforia works with 4 basic types of recognized targets:

- Image targets
  They do not require any black and white regions or codes to be recognized. Vuforia recognizes image target by comparing features (acquired from image analyzer) with a known target resource database. Once it is detected, SDK will track the image.

- Frame markers
  It is a special type of fiducial marker. Unique ID of a frame marker is encoded into

a binary pattern along the border of the marker image. When the ID is in range of [0..511]. Markers the Markers are manually added to the target image. Then they are distributed in Vuforia SDK and the developer has to scale them to an appropriate size. Frame and binary pattern must be entirely visible in camera image for recognition.

- Multi-targets
  A multi-target consists of multiple image targets that have a fixed spatial relationship. They are created by defining a relationship between multiple existing image targets using the Target Manager or by the direct manipulation of the Dataset Configuration XML file. The spatial relationship of the individual parts is stored in the XML file using simple transformations.

- Virtual buttons
  Virtual buttons are developer-defined rectangular regions on image targets which, when touched or occluded in the camera view, can trigger an event. Evaluation of virtual buttons is possible if the button area is in the camera view and the camera is not moving quickly.

Vuforia SDK uses right-handed coordinate systems. Each image target and frame marker defines a local coordinate system with (0,0,0) in the center (middle) of the target.

Image rating

Vuforia offers the possibility of creating user-defined targets during runtime. After analysis of the captured image it is rated. The rating depends on the count and distribution of shared edges. Even if image contains enough sharp edges and good distribution, repetitive patterns hinder detection performance. Images must be in JPG or PNG format with maximal size of 2MB and in grayscale or RGB.

Database of targets

Vuforia offers an online GUI - Target Manager that supports creating device databases and cloud databases, adding targets to these databases, and managing targets. Targets can be stored in:

- Device database Due to mobile device performance it is limited to 100 targets per database.

- Cloud database It is possible to use only clouds hosted by Vuforia. Besides Target Manager it can be managed through Vuforia Web Services API. After creating database, server access key is generated. A server access key is required for use of the Vuforia Web Services API and for accessing targets from application.

Chapter 3

Application Design

The goal of this thesis was to create an experimental application which uses augmented reality and interactive 3D models. For this purpose a simple tower-defense game named ARLanding was chosen.

The game will use markers printed on paper. These markers will be used for camera registration and for game objects positioning. The user will be able to interact with these game object by touching their virtual objects on the device's screen.

For augmented reality functionality the Vuforia AR framework will be used. The code will be based on a sample application from Vuforia - Dominno for easier integration with framework.

3.1 Game principles

The rules of the game are as in the most of other tower-defense games. The user needs to defend his base against enemy units coming from their base. There are 3 types of enemies - light vehicle, tank and flying vehicle. Every enemy has health points which are reduced by every hit received. When it drops to zero, the enemy is terminated. Enemies are coming in waves according to the level of game. When the enemy reaches the user's base, it is terminated and their base loses health points. The user can use two sentry towers for defense by placing them into the battlefield. They are bound to their markers and will shoot at enemy units which are passing by. Towers also have health points and enemies will shoot at them as well. If a tower's health is below zero, the tower is deactivated. It can be revived by being repaired in exchange for game currency. The tower's attributes can be upgraded for game money. Money can be gained by killing enemies.

The main element of the game for the user is a big virtual tower placed at the user's base. It has a transparent top with sight of the battlefield. The user is able to shoot at the battlefield only if the camera is located in the top of the tower. Shooting is done by touching the enemy object on screen. Weapons and other attributes of the tower can also be upgraded for game currency.

Every active object has limited health points. For enemies it depends on the enemy type, for user objects on the level of upgrade. With every hit the object loses health points according to attacker's damage value. Every active object has a health-bar located above its virtual model.

Attributes which can be upgraded are: maximal health of object, damage done to enemies, speed by which object can shoot enemies, range where the shots of the object can

reach and angle which determines the segment in front of the object where it can shoot. The user can upgrade attributes of main tower and two sentry towers for game currency. There are 5 levels of each upgrade. Every level improves the value of the attribute so it is more effective. With every new upgrade, the price for the next one rises. Prices for individual levels of upgrade are the same for main tower and sentry towers. Values of attributes are enhanced for the main tower.

Game objects behavior

The behavior of active game objects is simple. Enemy units need to move from their base to user's base. For simulation of authentic movement, units move in curved trajectory. With each step units do, vector direction is set to enemy base. Then, it is turned by deviation angle which is modified by 5 degrees every step. When it reaches the maximum limit, a new maximum in opposite positiveness is randomly generated and the deviation angle will proceed to it.

Enemy units (A) also need to avoid obstacles (B). It is done by checking for the intersection of two lines. The first line is the vector direction of A multiplied by constant - distance for which this is to be tested. The second line is each side of a square created around obstacle object B. The single direction line leads from the center of the object A. Therefore, the bounding square around the object B is bigger than its real size so it also covers the size of object A. If the intersection occurs, it keeps creating new deviation angles for object A until a clear path is found. If no deviation angle is found after 100 tries, the direction vector is turned by 180° and it makes a step back. In the next step this procedure repeats.

The logic of the sentry towers is even simpler. They try to find the nearest target in their range and shoot at it. If no suitable object is found, no action is done. This repeats for every shot.

3.2 Integration with augmented reality framework

In standard tower-defense games the main occupation for user is to build new towers or to upgrade them using only device's screen (or desktop's monitor and mouse). In ARLanding we want to use augmented reality to enhance the user experience by mixing the game world with real world. Users will be able to interact with game world not only by touching the screen or keys, but also by moving the camera in real world as it was a window through which they can look at the game world. To bring users even deeper into the game world, the important part of the game is positioning the camera at the top of a virtual tower. Users will be able to look from there to the battlefield as if they were standing there themselves.

Another enhancement of interaction will be positioning of game world elements using markers. Normally users would have to use only game-generated worlds or to use some kind of editor to position the game objects. In ARLanding this will be done by moving the markers in real world. Main

To mix the game world with the real world we will use Vuforia framework. Framework takes a frame from the device's camera and finds the positions of markers as well as the position of the camera itself. With this we can orientate in the game world and draw it as if it was set in the real world. Coordinates of markers gained from the framework are processed by the game logic to provide gameplay as well as by graphics to correctly position and draw game objects.

3.2.1 User testing

The main user's occupation in ARLanding's gameplay is positioning of the camera into the top of the main tower and shooting at enemies from there. To provide proper use for this action, it is required to have adequate strength and number of enemies. These three issues are very important and will be tested with users and adjusted to improve the gameplay and overall user experience.

Testing will be done directly during playing the game. Positioning of the camera will be tested by measuring the ratio of time when camera is positioned inside the top of the tower to overall gameplay time. User can shoot from the main tower only if the camera is positioned there so reliable registration of camera is very important. From the result we will be able to adjust the size and position of a bounding box around the top of the tower so it is not difficult to position it correctly and not too easy so user could shoot from anywhere around the tower.

For shooting at enemies user needs to touch the virtual object on device's screen. We will test this action by counting the attempts when touch was handled properly and attempts when the invoked action was incorrect or wasn't invoked at all. Result of this test will help us to adjust the bounding boxes around enemies and towers.

Strength and number of enemies needs to be set properly so the sentry towers can take care of significal part. Weaken or a small amount of enemies need to pass by sentry towers so user can shoot at them from the main tower. To destroy the rest of the enemies should be challenging for user and also not very difficult. This complex issue will be tested by playing and counting the rate of successfully finished games and games when enemies destroyed the user's base. With result we will be able to adjust the strength so average users can successfully finish the game in a few attempts.

3.2.2 Markers

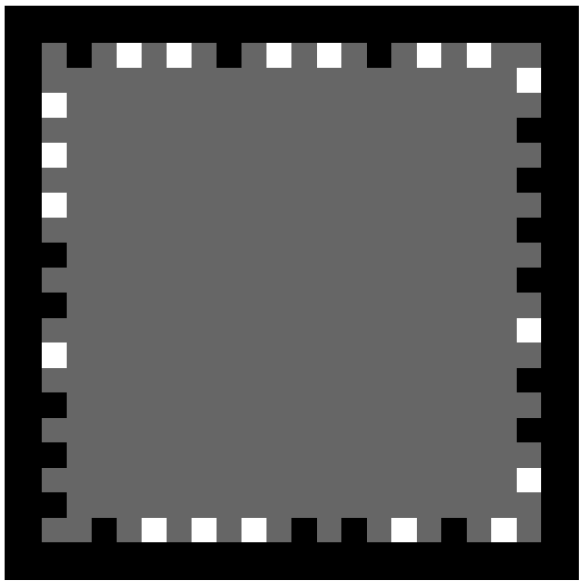


Figure 3.1: Image of frame marker used by Vuforia.

The vuforia framework primarily works with image targets. Recognition works very well, but with a larger amount of targets the performance lags behind. ARLanding uses

5 targets - enemy base, user base and two sentry towers. The last marker is used as an obstacle object but it is also the center of the game world and coordinate system. Because of performance reason, ARLanding uses frame markers 3.1. Vuforia comes with 256 pre-made frame markers. Recognized patterns are located only at the edges of a square marker (OR square markers), therefore there is a free space in the middle. It can be used for easier distinguishing of markers by the user as well as determining the direction that the markers are heading.

3.3 Graphical user interface

ARLanding is situated in a sci-fi environment, therefore the menu and other non-game screens are based on a background appearing as metallic material.

Main menu

The main menu 3.2 contains only 4 buttons - New Game, How To, Credits and Exit. Buttons are positioned in the middle with no disturbing elements.

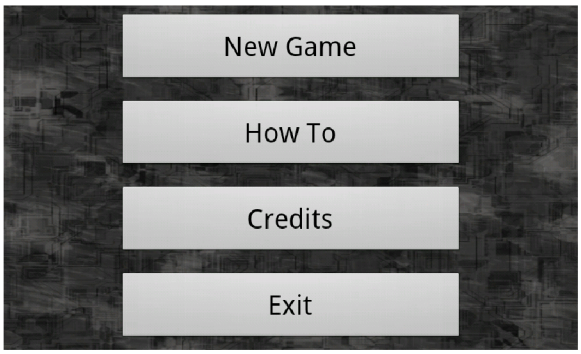


Figure 3.2: Main menu of application.

Game screen

The game screen 3.3 is very simple. It can be used only in landscape mode. The major part of the screen is taken by the camera view. This part is used for all of the user's interaction with the game world and the objects. At the right edge of the screen there is a quick access panel. It contains buttons for pausing the game and returning to main menu. In the left top corner there is textual information about the game state - current health of main tower, available money and current wave. This text changes to "Pause" if the game is paused or to "Markers not visible" when markers are not visible.

In the middle of the screen there is an information box which appears if there is a problem with the location of markers. Central marker is the crucial and game objects are rendered using its coordinates. Therefore if it is not recognized in captured frame, the game is paused immediately and an information box with a warning message appears on the screen. The same also happens if some of the other markers are not visible for more than 100 frames. If the marker is not found for 50 frames, a toast message appears in the bottom of screen.

Figure 3.3: Gameplay screen with warning displayed.

Upgrades screen

Upgrades screen 3.4 serves to upgrade the user's towers. It is invoked when the user touches the game object in paused mode. Before showing this screen, it is required to bring the current attributes of building from native environment to Java, as mentioned in implementation (4.1.1). The screen contains labels with current values of attributes and buttons to upgrade. If the user does not have enough game currency to buy the upgrade or the upgrade is at maximum level, the corresponding button changes to non-clickable. After the user clicks the back button, the game screen is shown again. Then the game stays in paused mode.

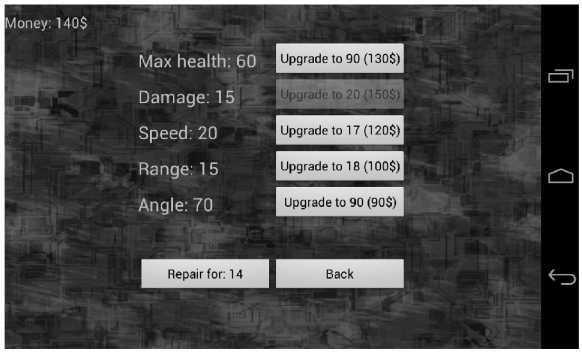


Figure 3.4: Screen where user can upgrade attributes of towers.

3D models

ARLanding's environment has a sci-fi theme. This theme allows relatively simple-shaped models with narrow edges. Most of the models were downloaded from the internet and slightly edited. Models were highly detailed and contained many vertices, which were not needed for this application. For the reduction of details we used `Decimate` modifier in Blender [4]. OpenGL ES uses a specific format for loading 3D models. It is a C header file with arrays of vertices, normals and texture coordinates. To convert exported 3D models in .obj model to these arrays the `obj2opengl`[1] utility was used. Textures are loaded separately as a .png images and are later associated with the models. ARLanding uses only 2 general

[1] obj2opengl, Heiko Behrens, http://github.com/HBehrens/obj2opengl

textures which are applied for multiple objects. You can see examples of models used in ARLanding at figure 3.5

Figure 3.5: Models of game towers. On the left is the sentry tower, on the right is the main tower.

Chapter 4

Realization

4.1 Implementation details

Implementation is based on a sample application which came with the Vuforia framework - Dominoes. It contains a very good example of multi-threaded initialization of framework with respect to Android platform lifecycle events. The application uses only one activity. Screens are changed dynamically using `setContentProvider` method.

The Vuforia API for the Android platform is written in Android native code - C++. This lets Android developers build performance-critical parts of the applications in native code. The SDK and NDK communicate over the Java Native Interface (JNI). In ARLanding native code is used for the most of the features - including framework data processing, graphics rendering, and the whole game logics. Android SDK is used only for the graphical user interface, handling application lifecycle events and the main application workflow.

To make the behavior of application clearer, the game operates in 4 states - `running`, `paused`, `no marker` and `finished`. The game is in state `paused` when the user pauses the game by touching the pause button or when the user is not on game screen - for example, the upgrades screen. `no marker` state occurs when one of the markers is not positioned. The game switches to `finished` state when the level is ended (both win or lose) and the application is waiting for the user's action.

On figure 4.1 you can see a simplified structure of ARLanding. The first operation of the application loop is capturing the camera frame. The image is drawn in OpenGL as a background and then processed by framework to recognize markers. If the marker is found, its pose matrix is used to compute the new game world coordinates. Matrix is also stored, for positioning of the drawing content for OpenGL. If the game is in state `running`, these new coordinates are passed to game logic which invokes every active game object to make a new step. When this is done, game objects which are still active are drawn at their new positions.

When a graphical user interface action is done and needs to be handled in native environment, a message is sent from the Main Activity. The same but in reversed way happens when a GUI action is required from native code. User touches are served by their own method. GUI Manager takes care of switching screens and layouts. For displaying a game itself, it uses `GLSurfaceView`, which is used from native environment.

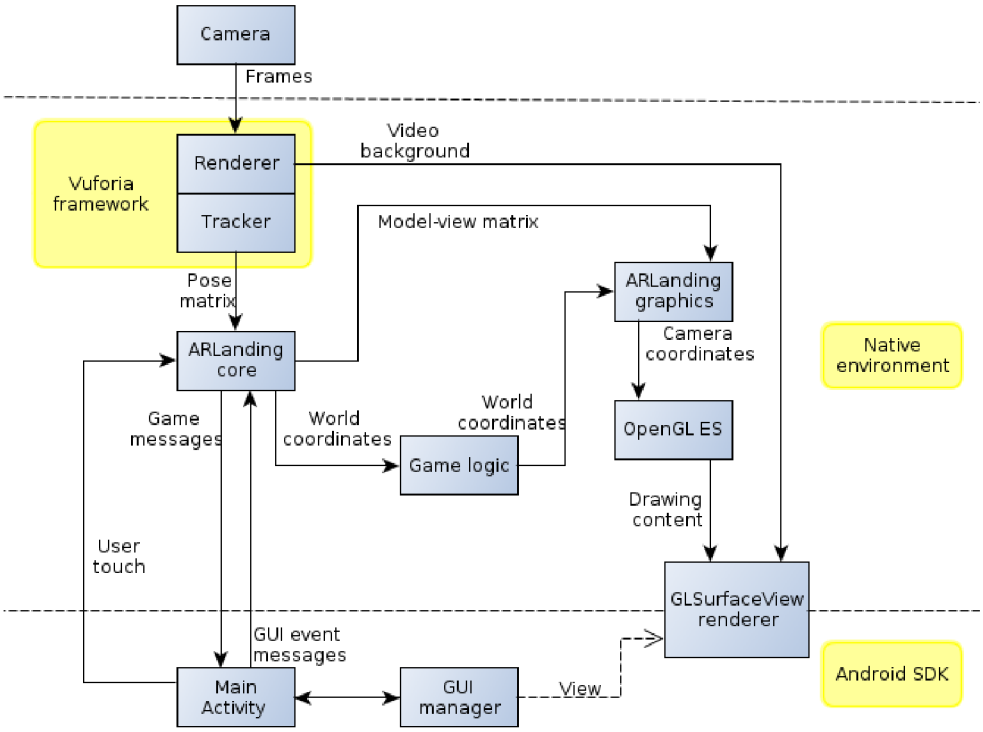


Figure 4.1: Structure of ARLanding implementation.

4.1.1 Android NDK, JNI

As mentioned, Vuforia framework uses Android native code - C++ because of performance efficiency. The main application loop is called by `onDrawFrame` method inherited from `GLSurfaceView.Renderer`. In ARLanding's Java code this method is implemented only to call the native rendering function.

During the initialization process the main activity calls several native methods. These methods uses the Java environment variable to obtain the references to Java methods from it. References will be used and stored for the communication. JNI offers a possibility to share the variables between the environments. This was found performance-inefficient so ARLanding uses mechanism of two methods for sending messages with integer or string argument.

4.1.2 Markers and global coordinate system

The Vuforia framework returns for every visible marker the 3D pose of the target as seen from the camera. It is a 3x4 row-major matrix. The 3x3 sub-matrix of the left three columns is a pure rotation matrix (ortho-normal), whereas the rightmost column is the translation vector. The difference between coordinate systems is shown at figure 4.2.

Right after this pose matrix is obtained from framework, it is converted converted from Vuforia's 3x4 row-major pose matrix to OpenGL's 4x4 col-major model-view matrix. It is done by a method provided by the toolset in the framework. These coordinates in model-view matrix are exactly what is needed for augmentation, but are not suitable for game

[1]Frédéric Ntawiniga, Université Laval, http://archimede.bibl.ulaval.ca/archimede/fichiers/25229/ch05.html

world orientation.

One marker is chosen to act as the game world center. For this purpose, a central marker is devoted. In gameplay it acts only as an obstacle so the user does not have to interact with it and therefore worsen the conditions for computer vision. Coordinates of the other markers, as well as game objects with no marker, are relative to this center. We can get these coordinates by multiplying the inverse pose of the world center target (A) by the pose of other targets (B). This creates an offset matrix that can be used to bring points on B into A's coordinate system.

4.1.3 Drawing content

Drawing of augmented content is done by OpenGL ES 2.0. There are two types of positioning game objects.

- Bond with a marker
  Game objects of the main tower, sentry towers and the enemy base are bound to corresponding markers. The best way for positioning their 3D models is by directly using model-view matrix of the marker provided by the Vuforia framework. However, this matrix is not in the OpenGL coordinate system, so the last step is to multiply this matrix with the projection matrix, which switches the coordinate system around the x-axis.

  With every frame captured and new pose matrix gained, a new game world position is computed. It is used in case the marker is not recognized by Vuforia. A 3D model is then drawn in relation to the central marker whose visibility is conditional.

- Relative to the world center
  Game objects can be positioned by coordinates relative to central marker. This is the case of game objects of enemies and objects with currently non-visible markers. Positioning of these objects is based on the modelview matrix of central marker. At first, it is required to compute the transformation matrix for the current frame. It can include translation (in game world coordinates), rotating and scaling. Now we can apply this transformation to modelview matrix of the central marker, by multiplication. The result is that the modelview matrix is ready for OpenGL.

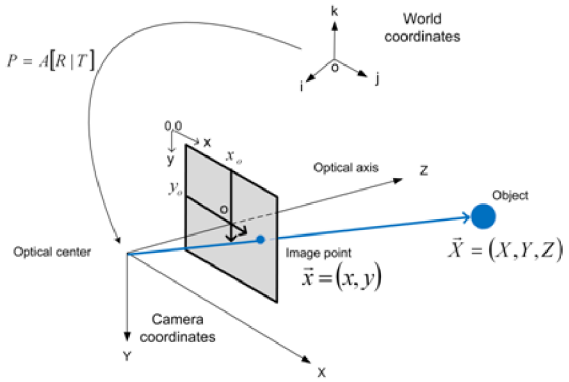


Figure 4.2: Camera and world coordinates. [Source: [1]]

Health-bars 4.3 are drawn for every game object with health points. Position of health-bar is relative to the corresponding object. Two 2D rectangles next to each other in a ratio of maxHealth/health are created. The Health-bar is drawn using the same model-view matrix as it's game object. It uses coordinates of corners of the health bar instead of 3D model's vertices. Central model-view matrix is transformed by the transformation matrix of its game object so it is heading in the direction of object.

Shots are drawn separately from game objects, as a single line between objects. Every type of 3D model has it's own offset of the shot origin. This 3D offset is added to game object's position (in game world coordinates). Line end is directly in the second objects' origin. The line is drawn using central model-view matrix. Therefore it is required to rotate offset vector by the angle which the game object heading to. Every shot has a duration which is measured in frames. Standardly it is set to 5 frames.

The user's base and sentry towers have a limited range and angle of where they can shoot. To determine it we need to know which way the corresponding marker is heading. The direction of the marker is extracted from the model-view matrix. This vector is used with the game world coordinates to determine points of triangle which specifies the area where the object can shoot. When the game is paused, these points are used for drawing lines around borders of the triangle so the user can adjust the position of sentry towers.

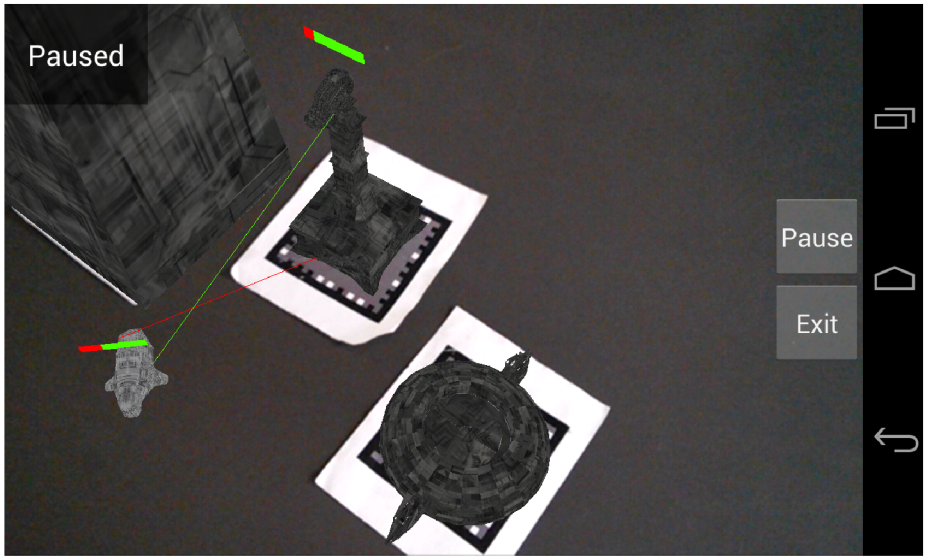


Figure 4.3: Detailed view at health bars and shots.

4.1.4 User touch handling

The only input of user's touch are two-dimensional coordinates in the coordinate system of the device's screen. It is captured by the `onTouchEvent` method of the main activity. Here it is pre-processed and sent to the native environment.

Bounding boxes around the game objects are registred in game world coordinates, even though the first step is to convert received window coordinates to normalized device coordinates. The next step is to project this point into a line in camera view. We create near and far planes and transfer this point to them. To get game world's coordinates the last step would be to multiply camera coordinates with inversed modelview matrix of central target.

Now we can check for intersections with bounding boxes around active objects. If they intersect, the action depends on the selected object and current state of the application.

4.2 Evaluation and assessments

ARLanding was successfully implemented. For debugging and testing during the development we used new device Google Nexus 4 with Android 4.2.2 and older device HTC HD2 with Android 2.3.7.

4.2.1 Startup time

Startup time was tested on Google Nexus 4 and HTC HD2. Measuring was done by comparing time values gained from method `SystemClock.elapsedRealtime()`. Time of activity displayed was gained from time of debugging messages in LogCat because this event does not come from the application. With the debugger attached the overall performance is slightly worse, but for this test it wasn't crucial. Before the measurements, the process was properly killed.

	Google Nexus 4				HTC HD2			
	1.	2.	3.	Average	1.	2.	3.	Average
Start	0	0	0	0	0	0	0	0
Native libraries loaded	210	150	100	153	170	150	150	156
onCreate finished	1101	1161	981	1081	1750	1690	1653	1698
QCAR initialized	1111	1171	1031	1104	2060	1940	1970	2323
Activity displayed	1311	1446	1272	1343	1780	1720	1860	1787
Initialization finished	1422	1561	1292	1425	2920	2739	2396	2685

Table 4.1: Measured startup times on two different devices. Times are in milliseconds.

First measured time is the loading of native libraries. It is executed by the Java Runtime before the class is loaded so there is no difference between devices. As mentioned in design, the initialization process is executed in multiple threads. Therefore the onCreate method of Android application lifecycle is finished in the middle of initialization. QCAR - Vuforia framework is loading in asynchronous task and on older device it finishes after the main activity is displayed.

The last value is time when all initialization stages are done and application is ready to use. On Nexus 4 this time is about 1.5 seconds. If compared with other similar application, this time is short and from the user's view ordinary.

The sample application on which ARLanding is based, by default displayed a splash screen while it was loading. Minimum time for the splash screen was set to 2 seconds. Because of faster startup time on current devices we reduced this limit to 1 second. Shorter start-up time may be due to using frame marker targets. It is not required to load image targets into Vuforia tracker.

4.2.2 User experience testing

User testing began right when the implementation was done. Development continued also after it and major defects were corrected. In testing, 6 people representing general audience were involved. To improve user experience we performed all tests described in 3.2.1. The

results helped us to adjust attributes of tested aspects so the gameplay is more interesting and challenging. In every test the results of multiple users were averaged for every attempt.

Positioning of camera in top of main tower

In standard tower-defense games the main occupation is to build new towers or to upgrade them using only devices' screens (or desktops' monitors and mouses). ARLanding adds to this an opportunity to position the device into the top of the virtual model of the main tower. This action, together with interacting with the game world, is challenging and requires good concentration.

This test was to measure the time during gameplay when the camera was positioned at the top of the tower. Time was measured only when the game was running and user was interacting with the game world. The resulting value was the ratio of time spent in the top of the tower to the total time of the game running. Measurement was done in the application's code by counting "positioned" frames.

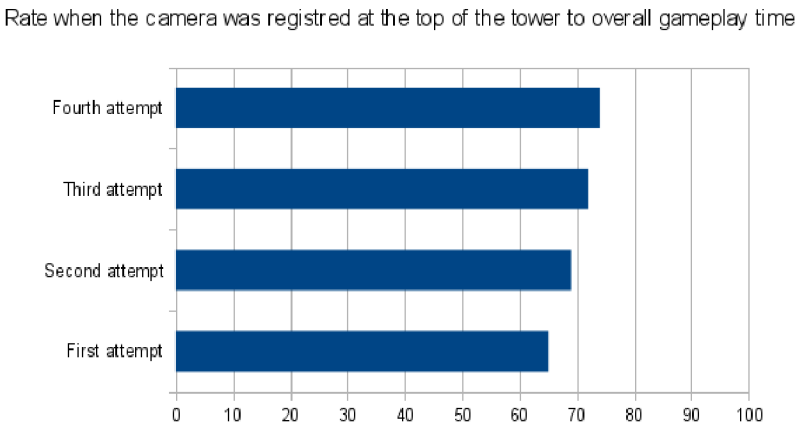


Figure 4.4: Average ratio of time when camera was registered at the top of the tower to overall gameplay time.

On figure 4.4 you can see that the resulting values are not satisfying. Average time when camera was registered in the top of the main tower is about 70%. This was caused mainly because of the unstable position and rotation of the main tower. When the user touches the screen, he unwillingly moves or rotates the device which can result in losing the position. This imperfection was corrected by extending the bounding box around the top of tower and adding countdown of 10 frames before camera is claimed as not positioned. You can also see that the times improve with new attempts.

Interactivity with game world

Interactivity with game world is realized by the simplest way - touching the game screen. A touch is handled and its coordinates are processed to game world coordinates to find out which object was selected. During this test we measured the successful rate if a touch event brought the desired action. Touches were targeted at enemies (shooting) or at buildings (upgrading). We measured 100 touches and asked if reaction was as expected.

Figure 4.5 shows that touches mostly brought the desired action and does not represent an obstruction in gameplay. After this test we adjusted the size of bounding boxes to better

enclose the game objects.

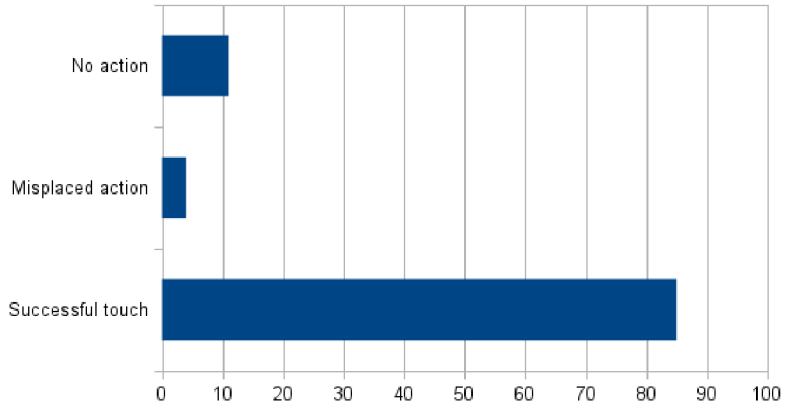


Figure 4.5: Average successful rate of selecting game object by touching the game screen.

Difficulty of gameplay

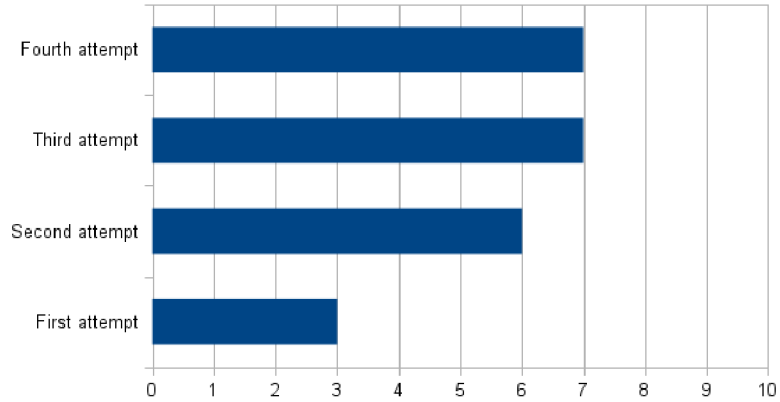


Figure 4.6: Levels in which users finished the game - both win or loss.

As mentioned, the gameplay requires significant concentration to correctly position the camera and interact with game world at the same time. Enemy units try to get to the main tower and sentry towers can't beat them by themselves so the user has to also shoot them. In this test we measured the rate of finishing the game successfully. With initial attributes of enemies and towers we achieved these values. Values represent the level where user finished the game (successfully or unsuccessfully).

On figure 4.6 you can see that initial attributes were set too difficult and users could very rarely finish the game. To optimize the game difficulty, we adjusted the quantity and spawning rate of enemies and key attributes of game objects like health points, speed, shooting speed, etc. As a result, gameplay is easier and the game can be finished with less problems.

Gameplay, performance

Gameplay was tested on both devices and on both of them it was very smooth. Registration of markers of course works better with the newer device, which has better camera. Vuforia framework has minor problems with initial recognition of patterns under low angle. This, however, gets better when once it recognizes the marker. It remembers its properties and later recognizes it in even worse conditions.

From time to time it occurs that the marker is not recognized for a few frames. This has been solved by tolerated limit for missing markers. In this case, the application remembers its last position and renders it there. When the loss of marker occurs, the user can register a small hop in position of the marker. It happens because of switching from using corresponding marker's pose matrix to using translated pose matrix of the central marker.

Overall assessment

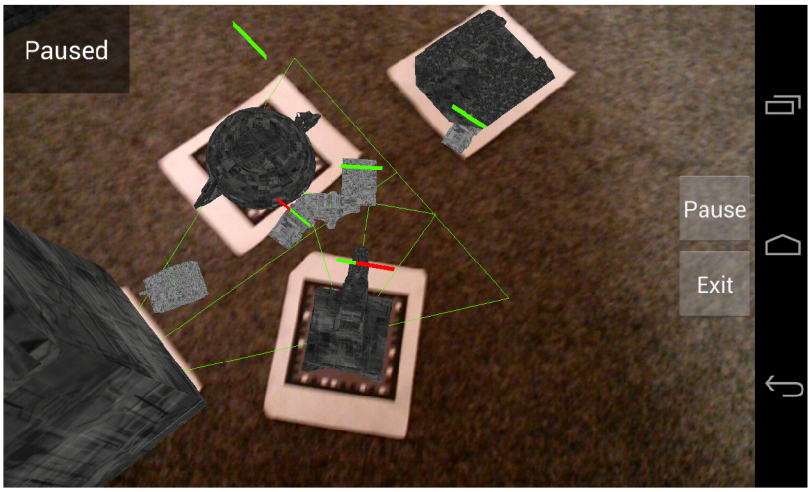


Figure 4.7: Final state of application.

All tests brought interesting results and findings which were used to improve user experience and gameplay. Before the testing, the first versions of game were not very nice to play and user had to make a big effort to actually play the game. After applying the improvements the game became much more smooth and attractive to play although there are many aspects to optimize and refine. The game was developed using only basic tools and framework for computer vision functionality. These resources were sufficient for creating application with credible positioning and rendering of virtual objects into real environment. The final application showed interesting approaches how to use augmented reality for improving human-computer interaction. Technological advance in mobile devices will open new possibilities in this area and findings from this application could be used to bring virtual objects even closer to real world.

4.2.3 Possibilities of future development

The main problem of comfortable gameplay is the unstable position of virtual objects, especially the main tower. ARLanding uses new pose matrix for every camera frame. Vuforia framework finds the targets very satisfactory but the rotation coordinates are slightly different. With normal-sized model this is no problem and the positioning looks stable. The

model of the main tower is bigger though (in real world measurement about 60cm high with standard-sized markers). For drawing this model we used the model-view matrix of the central marker so that the marker of the main tower does not have to be visible. Transformation matrix of the main tower stays the same while the marker is out of captured region. After applying it to central model-view matrix, the resultant position and rotation of the main tower is variable. With regards to tower size, the differences in position with every frame are significant and disturbing.

A possible solution would be to store previous model-view projections and use them to average big deviations. This can however lead to slower reactions if the bigger deviation was intentional. It could be solved by specifying limit of deviation for averaging.

Another possible solution is to use model-view matrices from other visible markers and average them. Comparing deviations between multiple markers could estimate the correct translation.

Another inconvenience in gameplay is requisite of visible central marker. When the Vuforia framework can't find the target for the central marker, the game pauses and waits until it is found again. This is not necessary because drawing of content which is not bond to its marker can also be done in relation to other markers. For easier conversion to the new relative coordinates a game object map can be created. This would also open possibilities for path-finding algorithms for moving game objects.

Chapter 5

Conclusion

The goal of this thesis was to design and develop an application for mobile devices which renders 3D virtual models into a real world environment, using augmented reality. For this purpose we chose a simple tower-defense game.

Standard tower-defense games use only the device's screen or keyboard and mouse to interact with the game world. We wanted to enhance it by placing the game world into the real world. Users could interact with it not only by pressing keys but also by moving the camera around and look at game objects as if they were really placed in real world. To register the position of the camera in this virtual world we used several markers. Users could move these markers to position the game objects and change the gameplay. The position of the camera was also used for enhancing the gameplay. Users could only properly play the game if the camera was positioned at the top of a virtual tower, so they could look at the battlefield like they themselves were standing there.

For the realization we chose Vuforia framework which is provides computer vision functionality. From the framework we gained coordinates of markers and camera in every captured camera frame. Coordinates were used for game logic to create gameplay and also for graphics to position and render the game object correctly. Framework's API is written in Android native code so the whole game logic and graphics is implemented in C++. Android SDK is used only for interaction with user.

During the development and for the evaluation of results we used two Android devices. New Google Nexus 4 running on Android 4.2.2 and older device HTC HD2 with installed Android 2.2.3. The application works on both devices without problems. The application was also tested by users. We tested registration camera position at the top of a virtual tower, user touch handling and difficulty of game. The results helped us to adjust attributes of game to improve the user experience even more. Tests also showed that user's progress improves in time. The final game is interesting, challenging and shows a new ways and concepts how can be augmented reality used in this area.

All goals were fulfilled and the game is working. During the development we also focused on expandability of game features so it is ready for future improvements and new features. The application will be published in the Google Play store and will also be improved by users demands.

Researching for this thesis gave me a very detailed overview about augmented reality and Android programming. I found these to be interesting and also perspective for future. Advances in mobile technologies will extend possibilities of augmented reality applications and thus I'll be able to use my newly gained knowledge very well.

Bibliography

[1] AndAR. Andar - android augmented reality - google project hosting. https://code.google.com/p/andar/, 2012.

[2] MIPS Technologies Bhanu Chetlapalli, Software Engineer. Learning about android graphics subsystem — mips developers. http://developer.mips.com/2012/04/11/learning-about-android-graphics-subsystem/, 2012.

[3] Oliver Bimber. *Spatial augmented reality : merging real and virtual worlds*. A K Peters, Wellesley, Mass, 2005.

[4] John Blain. *The complete guide to Blender graphics : computer modeling and animation*. CRC Press, Boca Raton, Fla. London, 2012.

[5] Stephen Cawood. *Augmented reality : a practical guide*. Pragmatic Bookshelf, Raleigh, N.C, 2007.

[6] DroidAR. Droidar. http://droidar.blogspot.co.at/, 2013.

[7] Qualcomm Austria Research Center GmbH. Vuforia — augmented reality unleashed. https://www.vuforia.com/, 2011.

[8] Qualcomm Austria Research Center GmbH. Developing with vuforia — vuforia developer portal. http://developer.vuforia.com/resources/dev-guide/getting-started, 2013.

[9] Peer Srl (Peer internet solutions). mixare — free open source augmented reality engine. http://www.mixare.org/, 2013.

[10] Inc. Metaio. Home — augmented reality - junaio... your mobile companion. http://www.junaio.com/, 2013.

[11] Paul Milgram, Haruo Takemura, Akira Utsumi, and Fumio Kishino. Augmented reality: A class of displays on the reality-virtuality continuum. pages 282–292, 1994.

[12] J. Mooser, S. You, and U. Neumann. Real-time object tracking for augmented reality combining graph cuts and optical flow. In *Mixed and Augmented Reality, 2007. ISMAR 2007. 6th IEEE and ACM International Symposium on*, pages 145–152, 2007.

[13] Sylvain Ratabouil. *Android NDK beginner's guide discover the native side of Android and inject the power of C/C++ in your applications*. Packt Pub, Birmingham, U.K, 2012.

[14] tdomhan. Howtobuildapplicationsbasedonandar - andar. http://code.google.com/p/andar/wiki/HowToBuildApplicationsBasedOnAndAR, 2013.

Appendix A

Contents of the provided CD

- The source codes of developed application
- The .apk archive of developed application
- Vuforia Framework required for compiling the application
- The LaTeXsource files for this document
- Poster
- Video