Augmented Reality on Android Smartphones

Studienarbeit

des Studiengangs Informationstechnik
an der Dualen Hochschule Baden-Württemberg Stuttgart

von

Tobias Domhan

8. Juni 2010

Matrikelnummer	136114
Kurs	TIT07INA
Gutachter der Dualen Hochschule	Prof. Dr. Rudolf Messer

Zusammenfassung

Diese Studienarbeit beschäftigt sich mit Augmented Reality (AR) auf der Android-Plattform. Bei Android handelt es sich um ein Betriebssystem für Smartphones. Augmented Reality ist eine Technologie mit der sich virtuelle, dreidimensionale Objekte in Echtzeit mit der Realität verknüpfen lassen. In dieser Arbeit wurde gezeigt, dass sich Smartphones, trotz der Einschränkungen bzgl. ihrer Leistung, für AR Anwendungen eignen. Dabei entstand ein, auf der ARToolkit Bibliothek basierendes, Open Source Framework, welches als Basis für AR Projekte auf Android dienen kann. Dieses wurde anschließend benutzt, um beispielhaft eine Anwendung zu erstellen, die in der Lage ist beliebige 3D Modelle auf AR Markern darzustellen.

Abstract

This paper is about Augmented Reality (AR) on the Android platform. Augmented Reality (AR) combines the real world with virtual, three-dimensional objects in real-time. This paper showed, that todays Android smartphones can successfully be used for AR applications, although they have only limited processing power available. In the context of this paper an Open Source AR framework, based on the ARToolkit library, was developed. This framework might be used as a foundation for AR projects on Android. Furthermore an AR application making use of this framework has been developed. This application is capable of displaying three-dimensional models on AR markers.

Contents

List of Figures

Acronyms

API	Application Programming Interface.
AR	Augmented Reality.
ELF	Executable and Linkable Format.
FPU	Floating-Point Unit.
GPL	General Public License.
GUI	Graphical User Interface.
HMD	Head-Mounted Display.
JIT	Just-In-Time Compilation.
JNI	Java Native Interface.
JVM	Java Virtual Machine.
OHA	Open Handset Alliance.
SDK	Software Development Kit.
VM	Virtual Machine.

1 Introduction

1.1 Augmented Reality (AR)

1.1.1 Definition

In a nutshell Augmented Reality (AR) blends virtual objects into the real world. However, there is no official definition of the term Augmented Reality. It is common in this field[1] to use the definition made by [1]. According to it an AR system must have the following characteristics:

1. Combines real and virtual
2. Interactive in real time
3. Registered in 3-D

The first point says, that the end result must contain both, parts from the virtual reality and the reality itself. Because of the second point, movies like Avatar, that contain virtual objects blended into real scenes, can not be regarded as AR, as they are not interactive. Additionally, as stated in the third point, chroma keying, which is often used in weather reports on the TV, is not AR either, because it blends in 2D maps and not 3D objects.

Figure 1.1 shows the Reality-Virtuality (RV) Continuum proposed by [12]. On the one extreme there is the Real Environment, on the other the Virtual Environment. Everything in between, e.g. AR, belongs to the Mixed Reality. [12] distinguishes between AR and Augmented Virtuality (AV). The first is based on the Real Environment mixing in objects from the Virtual Environment. Whereas the latter does the

[1] cf. [21, 19, 7]

exact opposite. An example for this would be a virtual room in which real persons are blended in.

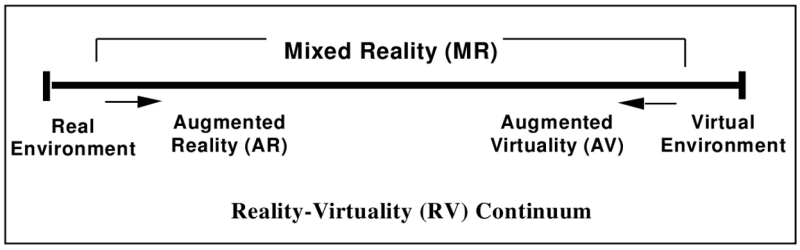


Figure 1.1: Reality-Virtuality (RV) Continuum[12]

1.1.2 Applications

Interior design

One application of AR is in interior design. It allows you to see how furniture fits into a room before buying it. Another use case would be the planning of a rearrangement of furniture. Instead of moving the furniture around, you can place markers at the desired destinations and see how that looks, immediately. Figure 1.2 shows for example a 3D model of table drawn onto a marker in a room using AR techniques.

Figure 1.2: A virtual table blended into a room [9]

Product information

Lego started to equip some of their stores with a AR terminal. It allows the customers to see what is inside the box. In order to do so, they have to present the box to the webcam. The terminal will then determine whats inside the box by analyzing the marker on it. A complete 3D model of the assembled vehicle on top of the box itself will be show in a display. (see figure 1.3)

Figure 1.3: AR used to display a 3D model of what is inside a Lego box [2]

Medicine

There are many fields in which AR can be applied to medicine. For example Computed Tomography (CT) is widespread today. During this process, a 3D model of body parts is created. AR makes it possible to display those models directly where they belong, as seen in figure 1.4.

[2] image from http://www.metaio.com/fileadmin/homepage/Presse/Dokumente/Pressemitteilungen/English/E_2009_01_15_PR_LEGO.pdf

Figure 1.4: Image of a foot overlayed by a 3D bone model[15]

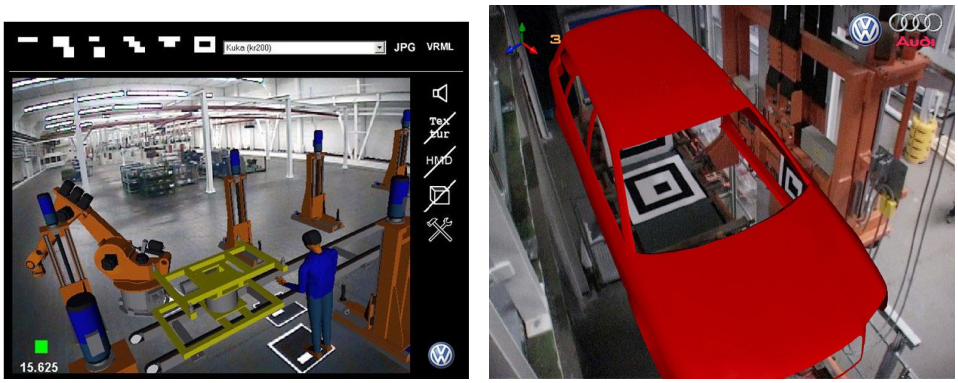


Figure 1.5: AR for factory design[18]

Industry

In [18] a application for factory planning and design developed for Volkswagen is described. The application was already used in the planning process of different industrial environments. Figure 1.5 shows the software in use.

Another imaginable application would be the annotation of parts for a repairman.

Entertainment

AR can be found in the entertainment sector, too. There are various applications that were developed during research processes, like [13, 8, 21]. Those include a racing game, a tennis game and a train game. The latter one can be seen in figure 1.6. The game displays a virtual train on trackage made of wood. The application itself runs

on a PDA.

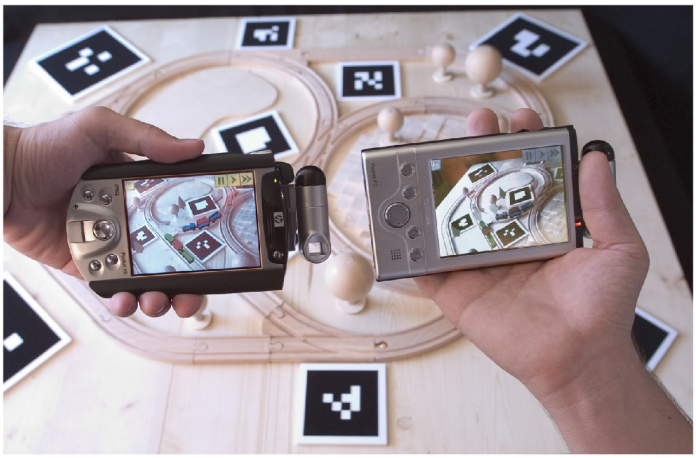


Figure 1.6: The invisible train game[21]

1.1.3 Main types of AR

All AR applications have a common problem to solve. Namely the pose of the camera with respect to the virtual objects has to be determined. There are two possibilities to do this. The easier and less computational intensive method is to use markers placed at specified locations, as seen in figure 1.6, 1.6 and 1.2. Those markers might be disturbing in some situations. The other way to achieve the same is to use natural features in the determination process. Methods for markerless tracking are not part of this paper, but can be found in [6, 17, 22].

1.1.4 Hardware for AR

Basically there only three things needed for AR. A camera to capture the the Real Environment, a display to show the end result and lastly a device providing some computational power. There is a wide range of devices that fulfill those requirements. Figure 1.7 shows the different form factors those have. The devices are from left to right: a backpack PC with a HMD, a tablet PC, a PDA and a smartphone. They vary not only in size but also in the available CPU power. This paper concentrates on AR on smartphones.

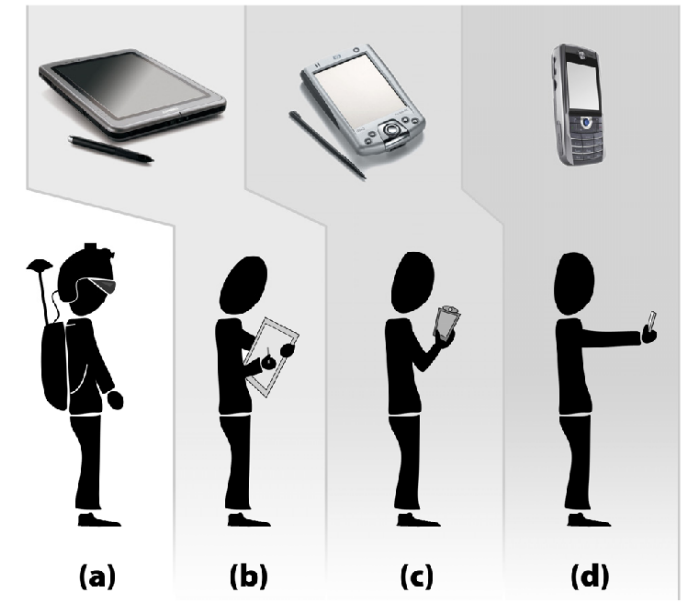


Figure 1.7: Common devices used for AR applications, (a): PC + HMD (b): Tablet PC (c): PDA (d): Smartphone.[21]

1.1.5 General workflow of AR applications

There is a basic workflow that all marker based AR applications have in common. It can be seen in figure 1.8. After acquiring an image from the camera, the marker has to be separated from the rest of the image. After that the contour of the marker is extracted. From the four corner points (or the contour) you can calculate the translation matrix for the virtual object. At last you apply that matrix and display the object above the image acquired in the first step.

1.2 Development Environment

1.2.1 Android

Android is a open source project initiated by Google Inc. In a nutshell it is a platform for mobile devices, including the operating system, a Software Development Kit (SDK), an application framework and key applications. Android is developed by

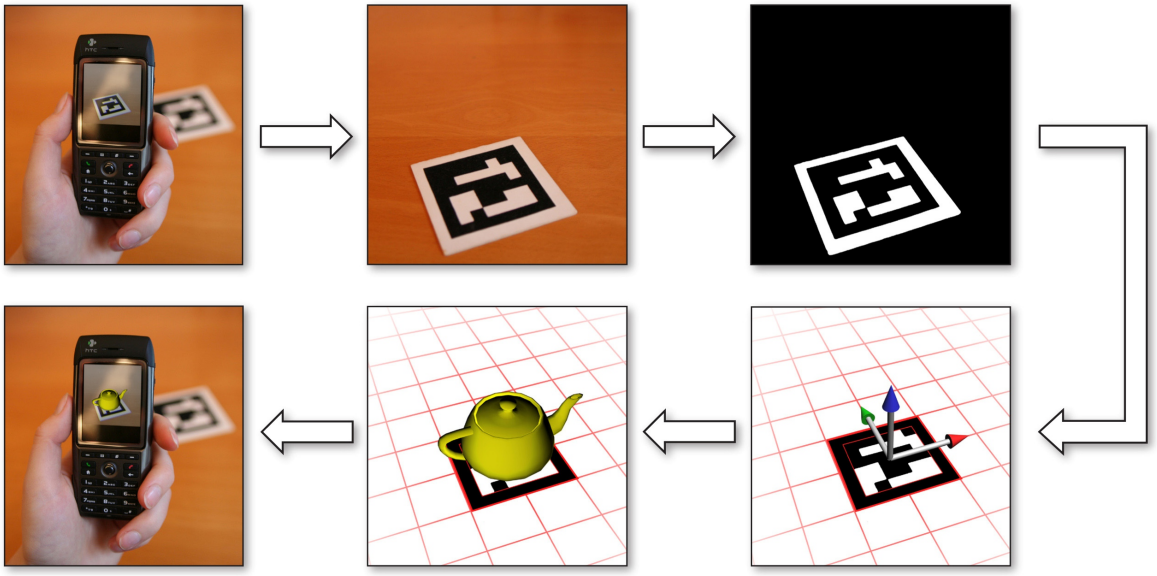


Figure 1.8: AR workflow.[22]

the Open Handset Alliance (OHA)[3], a consortium consisting of 65 technology and mobile companies. Among those are Google Inc., T-Mobile, HTC, Qualcomm and Motorola.

Most parts of the Android project are release under the Apache 2.0 open source license. This allows anyone to build a customized version of Android. One of those modified versions is called CyanogenMod. It is based on Android 1.6, but includes many features that were backported from newer versions. Furthermore it contains modifications and applications created by the community. It is specifically targeted to the HTC Dream smartphone[4]. For that phone the maximum CPU frequency is increased from 384 MHz to 528 MHz, when CyanogenMod runs on it. CyanogenMod is the firmware running on the phone used for this paper.

Figure 1.9 shows the architecture of Android. It is based on the Linux kernel version 2.6, which provides memory management, a driver model, process management, networking, a security model and power management to the layers above. Although it is based on the Linux kernel, it very much differs from traditional Linux distributions, unlike Maemo developed by Nokia. Pretty much everything above the kernel has been replaced, e.g. udev, glibc and X11 are missing. Therefor you can't run standard Linux

[3] `http://www.openhandsetalliance.com/press_110507.html`
[4] see section 1.2.3

applications on an Android smartphone.

An attempt to push the Android kernel into the Linux code base failed recently.[5] Future drivers that will be written for the Android kernel can't be used with the standard Linux kernel, as some new interfaces inside the kernel were introduced.

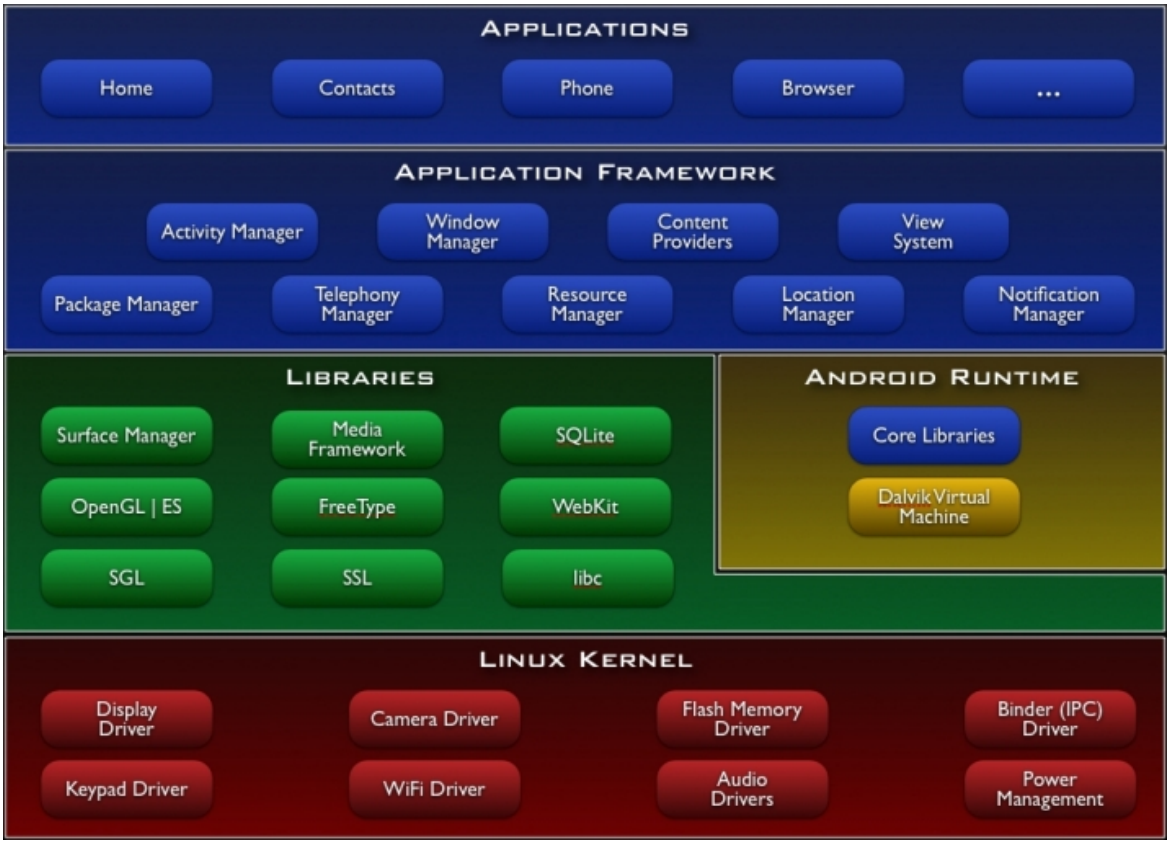


Figure 1.9: Android architecture[4]

Above the kernel there are the native libraries, represented by the green block in figure 1.9. Android is leveraging many already existing open source projects for those. For example WebKit for HTML rendering, which is also used in the Safari web browser. Other libraries provide things like media playback and data storage. They can not be accessed directly, but through Java APIs.

[5] `http://www.kroah.com/log/Linux/android-kernel-problems.html`

1.2.2 Android SDK

Dalvik VM (DVM)

The programming language used for all Android applications is Java. Java applications are generally not considered being conservative regarding memory consumption. On a workstation having multiple gigabytes of RAM this will seldomly be a problem. But smartphones usually have not very much RAM available. Furthermore the CPU power is not the greatest either. The standard Java Virtual Machine (JVM) doesn't address those problems, because it is not designed for smartphones. Therefor the Android team developed a custom Virtual Machine (VM) called Dalvik, specifically designed to run on battery powered devices with a slow CPU and little RAM.[3, p. 6]

It is based on Apache Harmony.[2, p. 17] Normally Java source code is compiled into `.class` files, one file for each class definition. The DVM is not able to execute those files directly. Instead it uses a custom file format called `dex`. A `dex` file will contain multiple classes. Those files will not be created directly, but the `.class` files are actually converted and combined into a single `dex` file. The resulting file will be smaller than the aggregated size of all `.class` files. This is because constant values (including method signatures) will be defined only once and then pointed to inside the file. For example if there different methods having the same signature, you will find one signature definition and multiple references to it in the resulting file. See figure 1.10 on how references are used in `dex` files. This will reduce redundancy and therefore save memory. According to [3, p. 22] this will reduce the size to half.

Instead of being stack based, like JVM is, DVM is register based. Because of this the resulting `dex` file will not contain any Java byte code, but instead a custom byte code format.

The standard JVM does Just-In-Time Compilation (JIT) of byte code. This means the Java byte code is compiled to machine code at runtime, which speeds up the whole application. DVM doesn't provide such a feature. According to [4] the reasons for this are:

- JIT would increase the overall memory consumption.

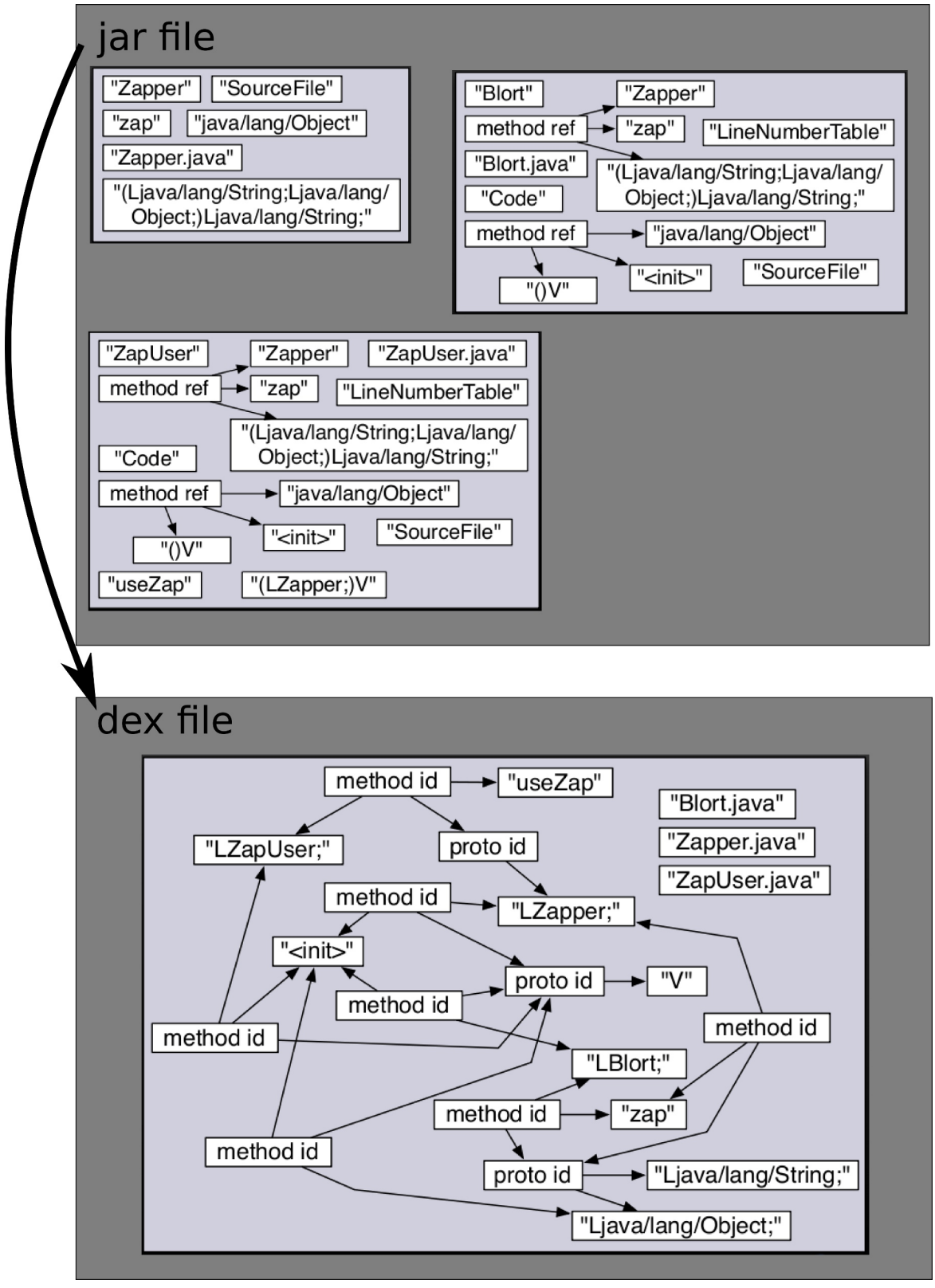


Figure 1.10: Conversion of a jar file to a dex file (modified version of [4, p. 17,20])

- Much of the functionality is implemented in native code anyway.
- Many things are even done in hardware, like graphics and audio.

There also exists an equivalent to *jar* files called *apk*. It is an archive containing the `dex` file, resources, native libraries and meta information. This is the file that is transfered to the phone during the installation process.

Java API

Android doesn't officially support either Java SE or Java ME. Basically this means you have to rewrite respectively modify your existing applications. Nevertheless there are parts of those APIs that are supported by Android. Among those are for example the classes *Vector* and *HashMap*. Android introduces a custom API for the Graphical User Interface (GUI). Because of that all *Swing* and *awt* classes are missing. More important is the fact that Java beans are not supported, which are used by many existing open source projects. [2, p. 327] gives a rough overview of what other packages are missing, too.

Graphics API

Android supports OpenGL ES in version 1.0, a special version of OpenGL designed for embedded devices. An API is available for both Java and C/C++. Some features of the 1.1 version of OpenGL ES are already support, though the support can not be regarded complete.[6]

Eclipse

The Eclipse SDK is used for developing applications. A screenshot of it can be seen in Figure 1.11. The Android specific functionality is enabled through a plugin. It allows the creation of an user interface via an visual editor. Debugging can be done

[6] cf. `http://developer.android.com/guide/topics/graphics/opengl.html`

inside an QEMU[7] based emulator, or directly on a smartphone. In order to deploy your applications you can create signed *apk* archives.

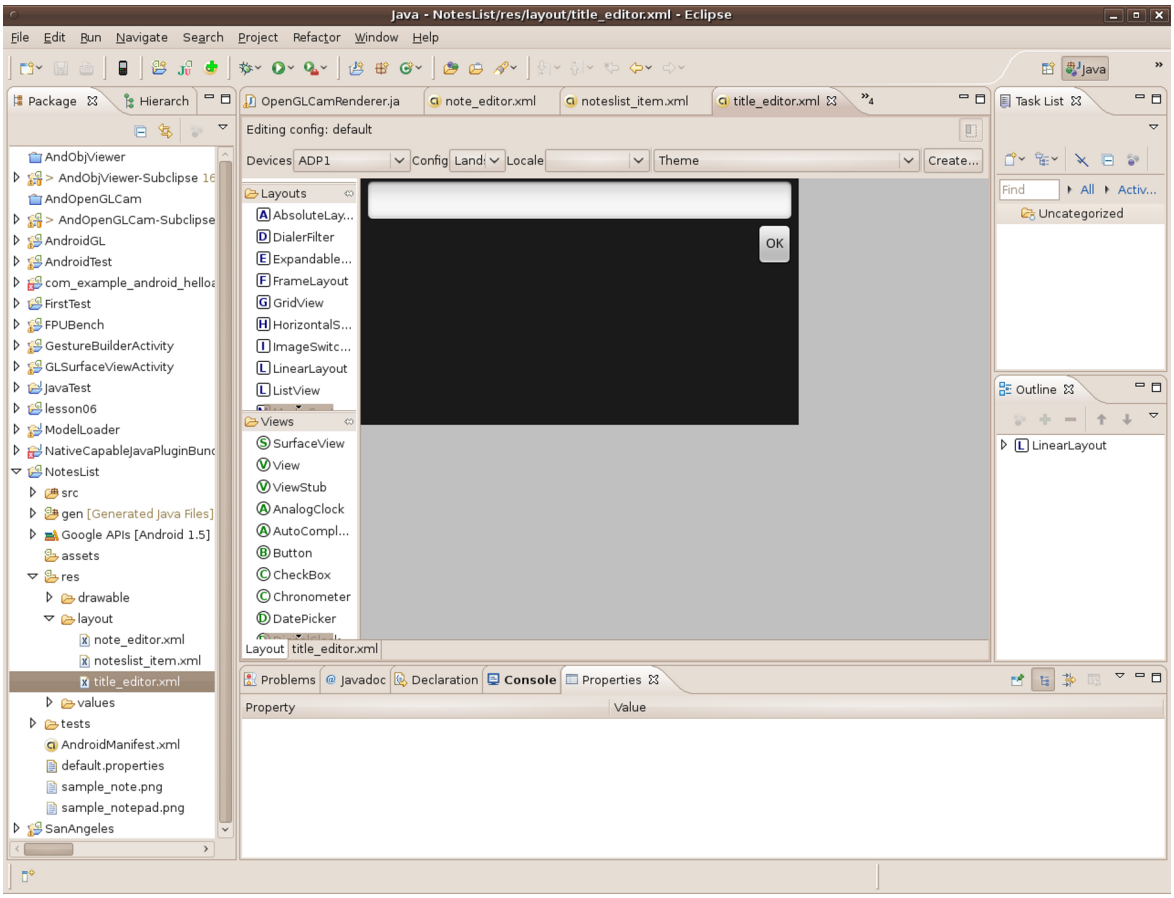


Figure 1.11: The Eclipse SDK

Native Development Kit (NDK)

Java is the only supported programming language for creating applications. However, it is possible to combine Java with C/C++ through JNI. The provided NDK contains the completed toolchain for cross compilation. It is based on the GNU Compiler Collection[8] and GNU make. With those tools you are able to create shared libraries in the Executable and Linkable Format (ELF) format used by Linux. There are only a few libraries that are officially supported. Among those are libc, libm, libz, liblog

[7] www.qemu.org
[8] http://gcc.gnu.org/

and the OpenGL ES 1.1 libraries.[9]

1.2.3 Hardware

The hardware used for this paper was a T-Mobile G1 also known as HTC Dream. It was the first Android powered smartphone. Figure 1.12 show a picture of it. It got a ARM based 528 MHz processor, 192 MB of RAM, a 3.2 megapixel color camera, a GPS sensor, a digital compass and an accelerometer.[10]

Figure 1.12: HTC Dream[11]

The second smartphone that was used during this paper is a Nexus One. It is a smartphone produced by HTC and marketed directly by Google Inc.. It is powered by a 1 GHz ARM processor, has 512 MB of RAM, a 5.0 megapixel color camera, a GPS sensor, a digital compass and an accelerometer. [12]

[9] see: `http://developer.android.com/sdk/ndk/1.6_r1/index.html`
[10] for further information see `http://www.htc.com/www/product/g1/specification.html`
[11] image from `http://www.htc.com/www/product/g1/gallery.html`
[12] for further information see `http://www.google.com/phone/static/en_US-nexusone_tech_specs.html`

Figure 1.13: Nexus One[13]

1.3 Object and Layout of this Paper

The object of this paper is to examine Augmented Reality on Android smartphones. In the context of this paper it shall be evaluated to which degree todays smartphones can be used for Augmented Reality. In section 1.1 the term Augmented Reality was defined and applications of this technology were shown. Moreover, section 1.2 described the Android platform and the development environment of this paper. While AR in general was discussed, chapter 2 will be about AR particularly on mobile phones. Chapter 3.1 will describe the AR framework that was developed and in the last chapter (4) the API offered by this framework will be illustrated. Additionally to that an application making use of the framework will be presented.

[13]image from `http://www.google.com/phone/?hl=en&s7e=`

2 Augmented Reality on Mobile Devices

2.1 Restrictions of mobile devices

2.1.1 Hardware

Android is designed for having 64 MB of RAM at a bare minimum.[3] If that's the case, the most of the RAM will be used by the system and there will only be around 20 MB left, which all other applications have to share which each other. The HTC Dream has got 192 MB of RAM. Nevertheless memory is a scarce resource on this phone, too. Not only the amount, but also the bandwidth of the memory is not comparable to PC hardware. It's common that CPU, GPU and the camera share the same memory, which will limit the available bandwidth even more.[24]

The CPUs on embedded processors don't allow parallel execution in most cases. Still AR can make use of threads to speed up the applications, as there are many IO bound parts in them.[23]

Floating-Point Units (FPU) have been common on standard PCs for a long time now. A FPU is a coprocessor that does nothing but performing floating point operations. When those opertions are carried out directly on the hardware they are way faster. If there is no FPU present, the operations have to be emulated by software instructions. Either those software instructions are inserted into the binaries at compile time or the exceptions, thrown when the floating point operations are used, are caught inside the kernel. According to [22] software emulated floating point operations are roughly 50 times slower than integer operations.

Due to restrictions in size and power consumptions, most of todays embedded processors found in smartphones lack a FPU. So does the HTC Dream, used for this paper. As stated in [23] applications will run 5-10 times slower on mobile phones than on a

average PC due to hardware limitations.

2.1.2 OpenGL ES

Android supports OpenGL ES currently in version 1.0. OpenGL ES is a 3D API specifically targeted to embedded devices. Only those features of OpenGL that are really needed are included. Often there are many ways to achieve the same in standard OpenGL, whereas for the embedded edition the Khronos group decided to reduce the redundancy by only supporting the most used way. Features that were used rarely were left out completely.[14]

As stated in section 2.1.1 FPUs are not common on embedded hardware. Same applies to the OpenGL pipeline. Even though OpenGL ES 1.x supports both fixed-point and floating-point at the API level, the pipeline itself is defined to be fixed-point based.[24] This is subject to change in version 2.0 of OpenGL ES. From then on only floating-point will be supported.[25, p. 746]

The most obvious difference to the standard OpenGL is that the `glBegin/glEnd` entry points are completely missing. Instead of specifying each vertex one by one, you have to provide references to arrays of vertexes.[25, p. 739]

All textures must be square, with the size being a power of two. This restriction not only applies to the embedded version of OpenGL, but also to the standard edition until version 1.3, as stated by the API documentation[25, p. 1103]. However this applies only to the function `glTexImage2D`, which is used to initially transfer the texture image data to the OpenGL driver. When updating the texture through the function `glTexSubImage2D`[1] you may provide images of any size after all.

Other characteristics of the embedded edition of OpenGL were not relevant for this paper. They can be found in [25, p. 739 ff.] and [14].

[1] which is what you should do for performance reasons according to [25, p. 307]

2.2 Related work

2.2.1 ARToolkit

ARToolkit is a open source library for marker based AR developed by the University of Washington. It is distributed under the General Public License (GPL). This is a reason why it is used in many AR related projects. There is also a commercial license available for purchase, though. Although it is written in portable C code, it was designed for running on PCs not on smartphones. Therefor it makes heavy use of floating point arithmetic. The library provides the following features:[2]

- Single camera position/orientation tracking.
- Tracking code that uses simple black squares.
- The ability to use any square marker patterns.
- Easy camera calibration code.
- Fast enough for real time AR applications.
- SGI IRIX, Linux, MacOS and Windows OS distributions.
- Distributed with complete source code.

It can be downloaded from `http://www.hitl.washington.edu/artoolkit/`. ARToolkit is the basis for the framework developed during the work for this paper. The porting of the library is described in chapter 3.

2.2.2 ARToolkitPlus

ARToolkitPlus is a library based on ARToolkit. It was developed until 2006 as part of the Handheld AR project at the Technische Universität Graz. Sadly the project will not experience any further updates. The library was ported from C to

[2] cf. `http://www.hitl.washington.edu/artoolkit/`

C++, has now an object-oriented API, was optimized for mobile devices[22] and uses a new algorithms for pose estimation. Additionally it will not bother with image acquisition and OpenGL rendering, contrary to what the original ARToolkit did. As it is derived work from ARToolkit it is also released under the GPL. The source code can be downloaded from: `http://studierstube.icg.tu-graz.ac.at/handheld_ar/artoolkitplus.php`

Although Android supports both C and C++, it lacks support for the Standard Template Library (STL). Because of that the ARToolkitPlus can currently not be used on Android devices. Though this might change in future Android versions, making this toolkit an alternative to the ARToolkit.

2.2.3 Studierstube Tracker

Studierstube Tracker is also a library for AR. It was developed by Daniel Wagner during his PhD thesis[21]. It is a complete rewrite from scratch, not based on ARToolkit. It is not released as open source and also not publicly available. It was designed not only for PCs, but also for mobile devices. However not for Android, but for windows mobile smartphones. According to [20] processing is about twice as fast compared to ARToolkitPlus on mobile phones.

3 Design and Implementation of AndAR

3.1 Architecture of the ARToolkit

The ARToolkit is the foundation of the AR framework developed in the context of this paper, AndAR. It does not only bother with tracking markers and calculating translation matrices, but also with image acquisition and OpenGL rendering. Table 3.1 shows the steps a application using ARToolkit is going through. Steps 2 to 5 are repeated until the application terminates. During initialization the video device is opened, the OpenGL display initialized and ARToolkit specific parameters are read, including camera characteristics and pattern data. Steps 2, 3 and 4 in the table each correspond to a library function call. In step 5 the transformation matrix created by the toolkit will be loaded and afterwards the object will be drawn using the OpenGL API.

Initialization	1. Initialize the video capture and read in the marker pattern files and camera parameters.
Main Loop	2. Grab a video input frame. 3. Detect the markers and recognized patterns in the video input frame. 4. Calculate the camera transformation relative to the detected patterns. 5. Draw the virtual objects on the detected patterns.
Shutdown	6. Close the video capture down.

Table 3.1: Steps inside a application using ARToolkit[16]

The basic architecture of the ARToolkit can be seen in figure 3.1. It relies on OpenGL for rendering, GLUT for creating the OpenGL window, a hardware-dependent video library and a standard API(which represents the platform dependent parts). The toolkit itself consists of three different modules[16]:

- AR module: core module with marker tracking routines, calibration and parameter collection.

- Video module: a collection of video routines for capturing the video input frames. It is a wrapper around the standard platform SDK video capture routines.

- Gsub module: a collection of graphic routines based on the OpenGL and OpenGL Utility Toolkit(GLUT) libraries.

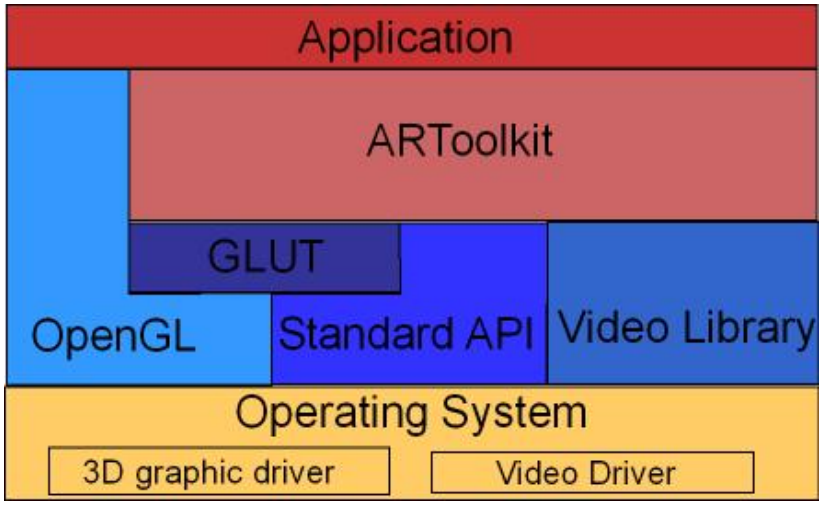


Figure 3.1: Architecture of the ARToolkit[16]

3.2 Interfacing the ARToolkit library

Section 3.1 described the basic modules of which the ARToolkit consists. For maintainability reasons, as little of the underlying library as possible should be changed, when using it. Only that way new releases of the toolkit may be used without major modifications.

Just one of the mentioned modules, the AR module, can be used without any modifications. That's not that much of a drawback as it contains the most important parts anyway. The Video module may not be used, as the video API of Android is only available from Java code. The Gsub module can't be used without modifications, as it relies on the GLUT library, which is not available on Android. For this reason most of the OpenGL parts had to be rewritten, as described in section 3.4. The only function used of this module is one that creates a transformation matrix that can be loaded directly into OpenGL. In OpenGL ES you may only load float arrays. Where as the

function of the module produces a double array. Hence this function had to be slightly modified.

On Android you may not write applications that entirely consist of native code. Furthermore great parts of the Android API are only accessible through Java code. There always has to be an Java skeleton, invoking the native functions. This is done by the Java Native Interface (JNI). The JNI allows you to load shared C/C++ libraries into Java applications at runtime. Inside the Java application you just have to specify the signature of the methods without their body. Each method that is not implemented in Java, but in native code, has to be marked with the `native` keyword.

The library is loaded once the application was started the first time. However it will not get unloaded after the user exits the application. The toolkit assigns an ID to each supported pattern, as soon as you load the pattern from a special pattern file. This ID is used in order to distinguish different markers. Multiple invocations of the function assigning the ID will result in different IDs, even though you use the same pattern file. The returned ID will be incremented by one with each invocation. However the detection function will only return the first of those IDs. For this reason you should not call the function multiple times for the same pattern file. This may be regarded a bug of the toolkit.

The current ID is stored by the toolkit inside a static variable. It will not be reset on application startup, as the shared library will only be loaded on the first invocation of the application. Because of that the marker will only be detected the first time you start the application. To avoid this issue a global list is used to cache the IDs of the pattern files.

3.3 Architecture of AndAR

AndAR is designed to be a framework for Augmented Reality applications on Android. It offers a object oriented, pure Java API, hiding all the native library calls.

Figure 3.2 shows the rough architecture of AndAR. It uses the Java API to access the camera and retrieve a video stream from it. The images are then in turn handed

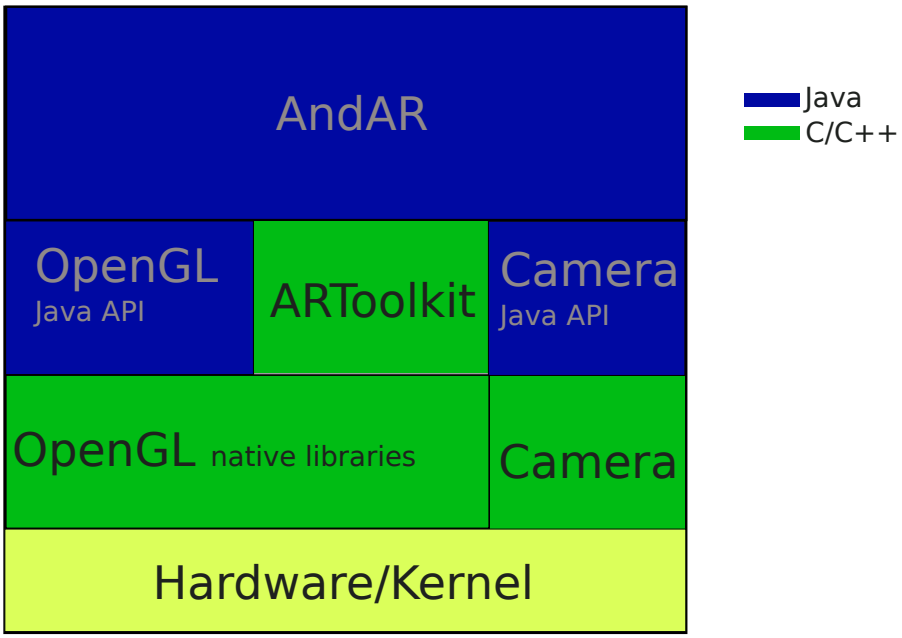


Figure 3.2: AndAR architecture

over to the ARToolkit library. Parallel to that the images will be converted to an appropriate colorspace, as described in 3.4. The multithreading and synchronization will be described in section 3.5. Inside the library a transformation matrix for the 3D object is calculated. Through the JNI the matrix is passed back to the Java code. The converted image will be loaded as a OpenGL texture, the transformation matrix applied and the 3D object drawn.

3.4 Graphics and Rendering

3.4.1 Overlaying 2D and 3D surfaces

An Augmented Reality application has to combine 2D (video stream) and 3D graphics. Android has an API for both use cases. However there is no official way to combine them. The only way to go is drawing the images on a rectangle as a OpenGL texture with the size of the screen. This means just using nothing but the 3D API.

The camera API is not designed to just provide a raw stream of byte arrays. It always

has to be connected to a surface on which it can directly draw the video stream. Some smartphones don't care if they won't get a surface provided, e.g. the Nexus One and the T-Mobile G1. On those you may start the preview right away, without setting a preview surface. Nevertheless there are Android smartphones that will not provide a video stream if no preview surface was set. One of them is the Motorola Milestone. However you may not provide the OpenGL surface as a preview surface to the camera. This would cause conflicts, as both the camera and the OpenGL thread would try to access this surface at the same time. Furthermore this surface must be visible on the screen, otherwise the preview callback will not be invoked, either.

The solution for this problem is to layer a OpenGL surface on top of the preview surface. This way the preview surface is regarded to be visible, even tough it is not. This method works on all Android smartphones. The video stream is drawn on a surface that is not visible to the user and additionally on a OpenGL texture. Compatibility is more important, then avoiding this overhead. There is no other way to circumvent this API design decision.

3.4.2 Color space conversion

In order to get a video stream on a Android smartphone you have to register a callback method. The method will be called every time a new frame arrives, providing the image as a byte array. There are quite a number of formats that might be supported by the camera hardware. The default format, YCbCr 4:2:0 SP, is defined by the Android compatibility program[11, p. 21][10, p. 18]. Every device must support this format by default. However the emulator uses YCbCr 4:2:2 SP by default.

The YCbCr 4:2:0 SP format is illustrated in figure 3.3. The first width times height bytes contain the grayscale image. Those are followed by the color information. For every block of 4 bytes there are two bytes of color information, one for the blue-difference and one for the red-difference. Those are alternating byte by byte.

As the images from the camera are in the YCbCr 4:2:2 SP format and OpenGL only supports RGB textures, an extra conversion step is needed. The conversion can be

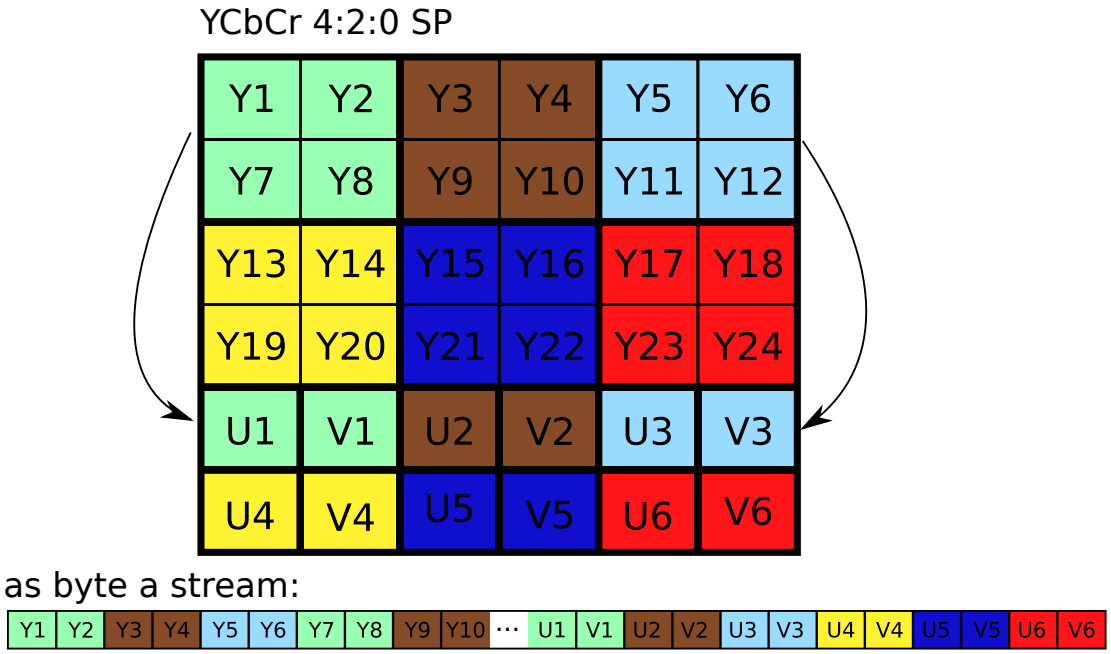


Figure 3.3: YCbCr 4:2:0 SP color space

done with the following equations:

$$B = 1.164(Y - 16) + 2.018(U - 128) \tag{3.1}$$

$$G = 1.164(Y - 16) - 0.813(V - 128) - 0.391(U - 128) \tag{3.2}$$

$$R = 1.164(Y - 16) + 1.596(V - 128) \tag{3.3}$$

3.4.3 Performance improvements

For performance reasons the conversion is done in native code. The array storing the result is allocated only once. The data of it will be overwritten with every new image arriving. This makes synchronization necessary, as described in the next section. In order to reduce the used bandwidth of the memory bus the preview images of the camera are set to 240 pixels in width and 160 pixels in height. On newer Android versions(2.0 and above) the available preview sizes can be queried. The application will choose the smallest one that still has the same aspect ratio as the display.

The marker detection function needs a grayscale image. As the image is in the YCbCr

format, there is no need for any conversion. The first width times height bytes of the array form the grayscale image.

With every preview frame an array containing the image data will be newly allocated. This causes the garbage collector to become active quite frequently. Causing a small lag every time it frees unused memory. There is no way to reuse the same array multiple times. However there is a method that allows you to register a callback, that will only be called as soon as the very next preview frame arrives. By leveraging this method, an appropriate frame rate can be achieved. Only requesting a new frame every time the color conversion has been done. Which minimizes the amount of memory being allocated. Even though it is not documented, you may not register this kind of callback while a callback function is currently being executed. Thus synchronization is necessary.

OpenGL ES version 1.x supports only square textures with the size being a power of two (see section 2.1.2). The texture is first initialized with an empty byte array that meets those conditions and is big enough to encompass a image of the video stream. The function `glTexSubImage2D` is not affected by the mentioned restriction. As a result the images are transfered to the OpenGL driver just as an update of a portion of the texture. This will speed up the whole process. As stated by [25, p. 307], updating a texture with `glTexSubImage2D` is faster than using `glTexImage2D` anyway. Only the part of the texture that contains real data is mapped to the rectangle that fills the screen. This is done by calculating the corresponding texture coordinates.

3.5 Multithreading

A callback method will be called every time a new preview frame arrives from the camera, providing the image as a byte array. There are basically two types of work that have to be done after a new frame arrived. On the one side the color space of the array has to be converted in order to display it as a OpenGL texture. On the other side the markers inside that image have to be detected by the ARToolkit library. Doing those two things is very time consuming, especially detecting the markers. First tests revealed that, if you do them inside the callback method in a sequential manner, the queue of preview frames waiting to be processes fills up until you phone is out of

memory. Camera panning will be noticed by the user with a lag of several seconds, at best.

In order to mitigate those problems two worker classes have been created. This is basically a thread running a endless loop, waiting for being interrupted at the end of it. The thread is in either of two states: waiting or working. The state transitions can be seen in figure 3.4. If it is waiting and a new frame arrives, it will start processing it. This is either converting the color space of the image, or detecting the markers, depending on the implementation of the worker. All new frames that arrive while the worker is processing an image will be ignored. From now on the only thing done in the callback method is setting the next frame of the worker threads, as seen in the sequence diagram in figure 3.5. Those methods will return immediately and the queue will not fill up. The conversion and the detection will be handled both at their very own frame rate, depending on how fast each processing can be. If there are constantly new frames arriving, the worker threads will be working to capacity.

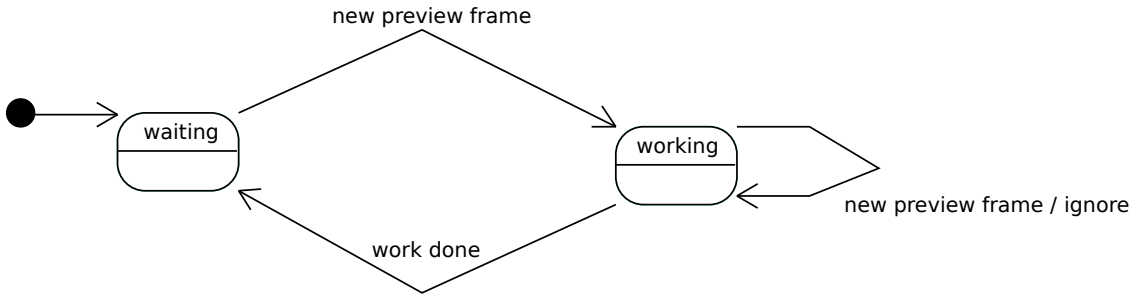


Figure 3.4: State diagram: States a worker object can be in.

There are three main threads inside AndAR doing all the work. In order for this to properly work, same synchronization work has to be done. The color space of the video images will be converted in one thread. The converted image will be accessed from the OpenGL thread later on. For performance reasons there will only be one array that is used for this. In order to access the image a `ReentrantLock` has to be locked first. Figure 3.6 shows the states the three threads are in for synchronization. During the yellow states the application will execute native code. The transformation matrix for each 3D object is accessed both from the marker detection thread and from the OpenGL render thread. The synchronization is achieved through Java monitors. For the OpenGL rendering part this will be done directly from Java. The monitor is

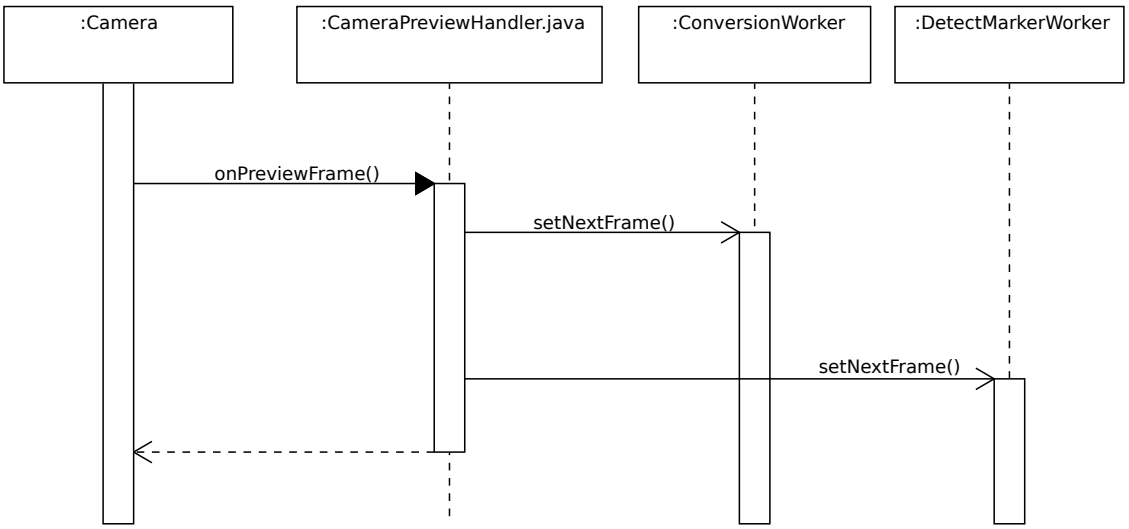


Figure 3.5: Sequence diagram of work being delegated to worker threads.

handed over to the detection function, as the matrix will not be written to before the end of the function. The monitor is used directly in the native function in order to minimize the lock time. Figure 3.6 shows only what is relevant for synchronization, things like OpenGL initialization were left out for the sake of clarity.

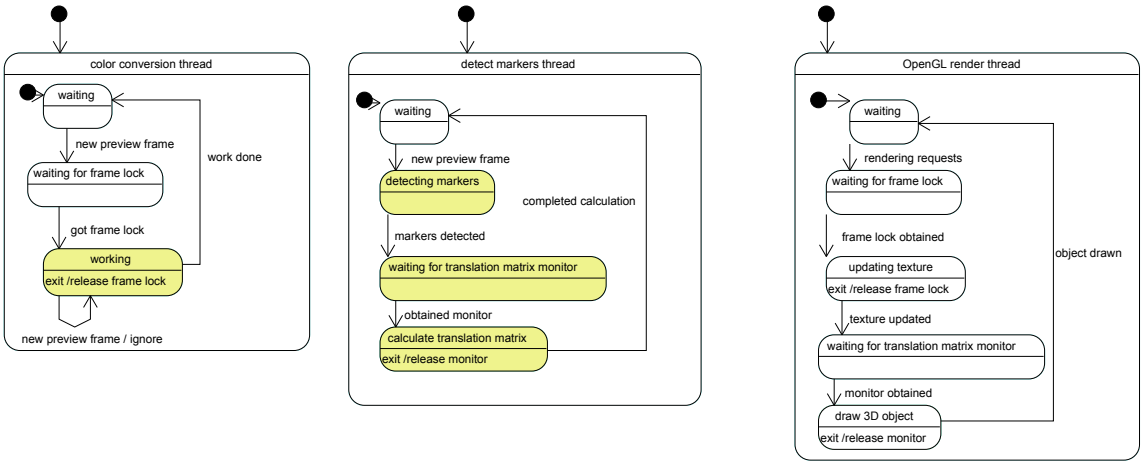


Figure 3.6: State diagram: AndAR synchronization.

3.6 Achieving backward compatibility

Android is a rather young operating system, released by the end of 2008. Nevertheless there already 60 different Android smartphones, of which 100,000 units are sold on a daily basis[5]. Furthermore there have already been seven major releases of the Android operating system. Each introduction new features to the developers. In Figure 3.7 the market share of each of those releases can be seen. The diagram is missing the newest release, codenamed Froyo, which has just been released.

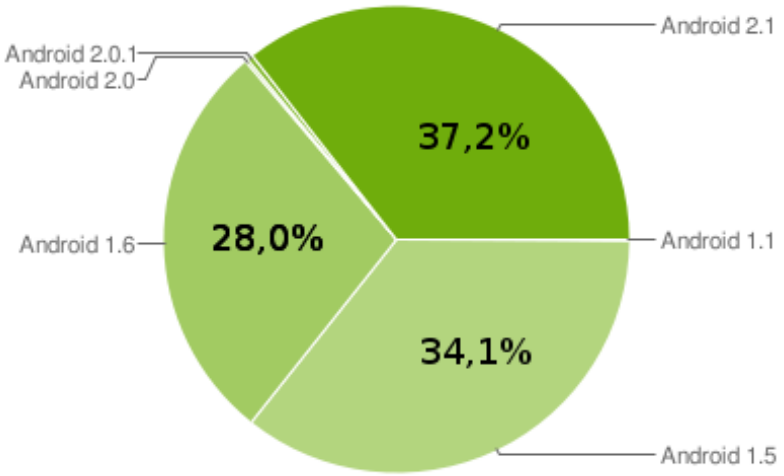


Figure 3.7: Android versions available today, and their corresponding market share.[1]

Every application targeted to a specific Android version will run on any newer version that was released. However you may not use any of the new features, if you want to maintain backward compatibility with older releases. Otherwise a `VerifyError` will be thrown upon application startup. This exception will be thrown if a reference to an unkown class or method is found. The key here is to test the software on different Android versions on a regular basis. There are basically two kinds of a new features:

- A completely new class was introduced.
- A class was extended by a new method.

Each requires a different approach. The first kind can be solved through a wrapper class. This class will store an instance of the newly introduced class. Furthermore it

[1] see `http://developer.android.com/resources/dashboard/platform-versions.html`

offers the same methods. In those it will just call the corresponding methods of the wrapped class. The trick lies in a static initailization block. In there the the VM will be forced to load the class by invoking: `Class.forName("NewClass")`. If it fails an exception will be thrown. This tells the client code that this class is not available. Hence it will not be used. If no such exception is thrown, the class may be used the way it is intented. Wrapping all methods is still necessary so that the client code will not contain a direct reference to the new class.

A new method may be used without causing a `VerifyError` through the use of reflection. This means the application will request a method at runtime. It has to provide the method name and the argument types this method has. If this succeeds, an object, representing this method, will be returned. If it fails, an exception will be thrown here, too. Again this tells the client code, that this method is not available. As the method is queried through a string, there will not be a direct reference to it. This avoids the `VerfiyError`.

For example the size of the preview images of the video stream has to be reduced, so that the application will run smoothly. Before Android 2.0 there was no possibility to query the available sizes. By leveraging reflection, this feature will be used by AndAR on devices supporting it.

4 Building applications based on AndAR

4.1 Programming Interface

AndAR is an Augmented Reality Framework for Android. It not only offers a pure Java API but is also object oriented. Figure 4.1 shows a simplified class diagram of an application that makes use of AndAR.

Every Android application consists of one or more *Activities*. An *Activity* is a visual user interface, targeted to a single purpose. Only one may be active at a time. In order to write an Augmented Reality application, one has to extend the abstract class *AndARActivity*. This class already handles everything Augmented Reality related, like opening the camera, detecting the markers and displaying the video stream. The application would run already, by just doing that. However it would not detect any markers.

In order to do so, you have to register *ARObjects* to an instance of *ARToolkit*. This instance can be retrieved from the *AndARActivity*. The *ARObject* class itself is abstract. This means, it has to be extended, too. It expects the file name of a pattern file in it's constructor. This file must be located in the assets folder of the Eclipse project. Pattern files can be created by a tool called *mk_patt*, which is part of the ARToolkit. They are used to distinguish different markers. In order to draw a custom object, the method *draw* has to be overridden. Before this method is invoked a transformation matrix will already have been applied. This means the object will be alligned to the marker, without any further steps. This method will not be invoked, if the marker belonging to this object is not visible.

The class *ARRenderer* is reponsible for everything OpenGL related. If you want to mix augmented with non augmented 3D objects you may provide a class implementing the *OpenGLRenderer* interface. There are three methods defined by this interface. *initGL*

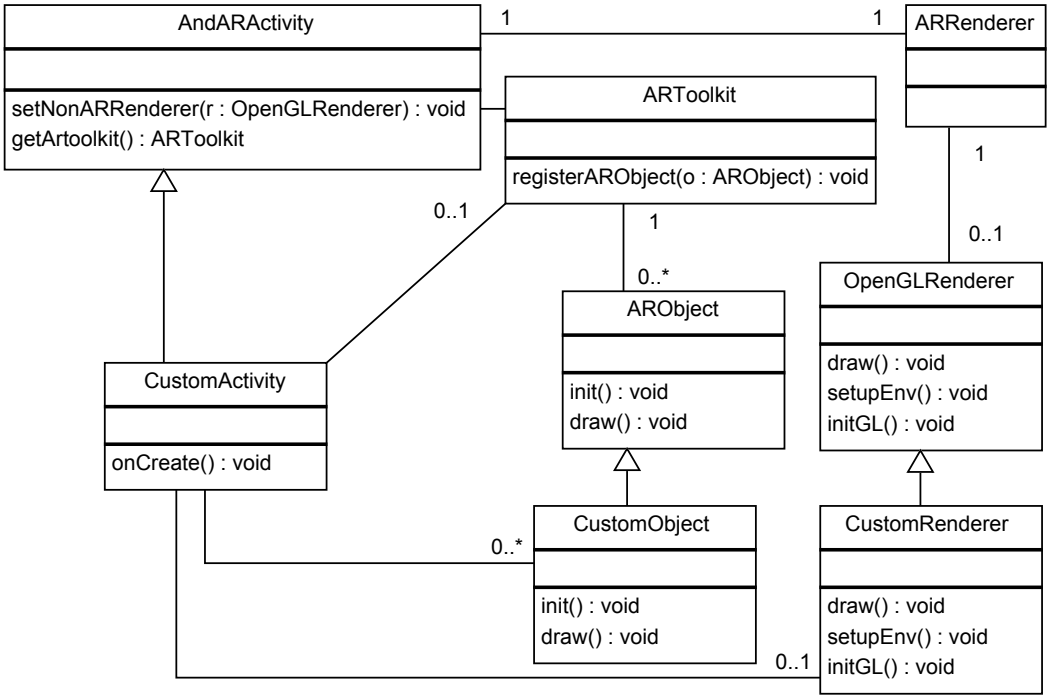


Figure 4.1: Simplified class diagram of an application based on AndAR.

being called only once, when the OpenGL surface is initialized. Whereas *setupEnv* 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. In the *draw* method you may draw any non augmented 3D objects. It will be called once for every frame. Specifying such the described renderer is optional.

The *AndARActivity* furthermore offers a method that allows the application to take screenshots. Examples of those can be seen in figure 4.2.

4.2 AndAR Model Viewer

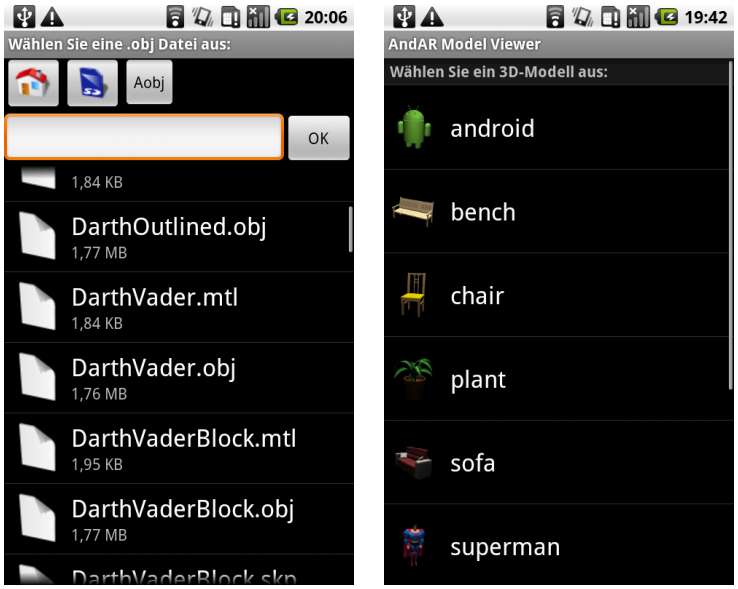
The AndAR Model Viewer is an example application that makes use of the AndAR framework. It allows the user to view obj models on Augmented Reality Markers.

This is done by extending the *ARObjects* by a class that is capable of displaying complex 3D models. This class stores all vertices, normals, texture coordinates and materials. As those data types are independent from the obj format, the application might be extended to support other model format, too. Furthermore this application makes use of specifying a custom renderer. However it is only used to initialize the lighting.

The application is bundled with internal obj models. Those reside in the *assets* folder of the Eclipse project. One may add new obj files to that folder, which will be in turn automatically be part of the application. All models in this folder are accessible through the applications main menu, which can be seen in figure 4.2(b). Additionally the user may load a custom obj model, as seen in figure 4.2(a). In order to select the model file, a *Intent* of the application OI File Manager is used. With an *Intent* an Android application may offer functionality to third party applications.

The obj parser, that was developed for this application, supports materials and textures. Anyhow there are some restrictions to the supported obj models:

- Every face must have normals specified.
- The object must be triangulated, this means exactly 3 vertices per face.



(a) The menu in which the user may select a custom obj file.

(b) The main menu of the AndAR Model Viewer.

Figure 4.2: Screenshots of the AndAR Model Viewer's menus.

AndAR Model Viewer is available in German, English, Portuguese and Chinese.[1] In figure 4.2 you can see screenshots of the application in action.

[1] Portuguese and Chinese being contributed by users of the application.

(a) An AR plant next to it's real counterpart.

(b) An AR chair.

(c) An AR marker on a towel on the Ipanema Beach in Rio de Janeiro.

Figure 4.3: Screenshots of the AndAR Model Viewer.

5 Conclusion

This paper has investigated Augmented Reality on Android smartphones. Therefore an overview over AR technologies has been provided. The Android platform has been introduced and capabilities of it have been presented. Furthermore the restrictions encountered on mobile devices have been determined. Besides design considerations for AR applications on mobile devices have been presented.

With all that in mind, a AR application framework has been developed during the work of this paper. This application leverages the commonly used ARToolkit library. This application framework is one the few providing AR on Android as defined by [1] nearly in real time. The framework runs on Android 1.5 through 2.2 and many different smartphones[1]. Additionally an example application, that is using the AndAR framework, capable of rendering complex 3D models on AR markers was developed. This application shows that todays smartphones are capable of being used for AR purposes.

The software was released under the GPL and is publicly available on the project's website[2]. This allows others to use the software as a foundation for their very own AR applications on Android.

Possbile future work includes the following:

- The tracking of the markers could be improved by using faster algorithms, e.g. as described in [21]. However this would imply changes in the underlying toolkit.

- ARToolkit+ is the C++ version of ARToolkit optimized for mobile devices. The application could be speeded up by replacing the underlying AR library. This is currently not possible because there is no Standard Template Library available.

[1] see `http://code.google.com/p/andar/wiki/SupportedMobilePhones`
[2] `http://code.google.com/p/andar/`

Though this might be subject to change in future Android versions.

- Currently all computational intensive things are done directly on the phone. Todays smartphones have many interfaces over which they can communicate with other phones or servers. Those interfaces are getting faster and faster. One could examine if it is possbile to move those computational intensive parts onto a server, letting the phone do nothing but displaying the end result. Nevertheless this would be a totally different approach, then the one of AndAR. Which means one had to start nearly from scratch.

- The Model Viewer currently only supports one model format. It could be extended to support multiple model formats.

- The conversion of the color space is done through equations. It might be faster to use a look up table of precalculated values.

- There are some Android smartphones supporting the OpenGL extensions Vertex Buffer Objects and Frame Buffer Objects. This might increase the performance of both AndAR and the Model Viewer.

- In the new release of Android codenamed Froyo you may add your own preview buffers to the camera. Those will be used instead of allocating new memory for every preview frame. This would cause the garbage collector to clean up less often, which in turn would increase the performance of the application.

Bibliography

[1] Ronald Azuma. A survey of augmented reality. *Presence*, 6:355–385, 1995. URL `http://www.cs.unc.edu/~azuma/ARpresence.pdf`.

[2] Arno Becker and Marcus Pant. *Android - Grundlagen und Programmierung*. dpunkt.verlag, Heidelberg, 2009.

[3] Dan Bornstein. Dalvik VM internals. Presentation, Google Inc., May 2008. URL `http://sites.google.com/site/io/dalvik-vm-internals`. last checked: 20.02.2010.

[4] Patrick Brady. Anatomy & physiology of an android. Presentation, Google Inc., May 2008. URL `http://sites.google.com/site/io/anatomy--physiology-of-an-android`. last checked: 20.02.2010.

[5] Tim Bray. On android compatibility. URL `http://android-developers.blogspot.com/2010/05/on-android-compatibility.html`. last checked: 01.06.2010.

[6] Andrew I. Comport, Éric Marchand, and François Chaumette. A real-time tracker for markerless augmented reality. In *ISMAR '03: Proceedings of the 2nd IEEE/ACM International Symposium on Mixed and Augmented Reality*, page 36, Washington, DC, USA, 2003. IEEE Computer Society. ISBN 0-7695-2006-5.

[7] J. Fischer. *Rendering Methods for Augmented Reality*. Dissertation, University of Tübingen, 2006. URL `http://www.janfischer.com/publications/Fischer06_PhdDissertation_HighRes.pdf`.

[8] Anders Henrysson, Mark Billinghurst, and Mark Ollila. Face to face collaborative ar on mobile phones. In *ISMAR '05: Proceedings of the 4th IEEE/ACM*

International Symposium on Mixed and Augmented Reality, pages 80–89, Washington, DC, USA, 2005. IEEE Computer Society. ISBN 0-7695-2459-1. doi: http://dx.doi.org/10.1109/ISMAR.2005.32.

[9] Wolfgang Höhl. *Interactive environments with open-source-software*. Springer, Wien, 2009. ISBN 978-3-211-79169-1.

[10] Google Inc. Android compatibility definition: Android 1.6, . URL `http://static.googleusercontent.com/external_content/untrusted_dlcp/source.android.com/de//compatibility/android-1.6-cdd.pdf`. last checked: 07.06.2010.

[11] Google Inc. Android 2.1 compatibility definition, . URL `http://static.googleusercontent.com/external_content/untrusted_dlcp/source.android.com/de//compatibility/android-2.1-cdd.pdf`. last checked: 07.06.2010.

[12] Paul Milgram, Haruo Takemura, Akira Utsumi, and Fumio Kishino. Augmented reality: A class of displays on the reality-virtuality continuum. *SPIE*, 2351:282–292, 1994.

[13] Ohan Oda, Levi J. Lister, Sean White, and Steven Feiner. Developing an augmented reality racing game. In *INTETAIN '08: Proceedings of the 2nd international conference on INtelligent TEchnologies for interactive enterTAINment*, pages 1–8, ICST, Brussels, Belgium, 2007. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering). ISBN 978-963-9799-13-4.

[14] Tom Olson. Polygons in your pocket: Introducing OpenGL ES. URL `http://www.opengl.org/pipeline/article/vol003_2/`. last checked: 22.02.2010.

[15] o.V. Der Blick in den Körper - Erweiterte Realität in der computergestützten Chirurgie, . URL `http://www.in.tum.de/forschung/forschungs-highlights/medical-augmented-reality.html`. last checked: 20.02.2010.

[16] o.V. Artoolkit documentation, . URL `http://www.hitl.washington.edu/`

`artoolkit/documentation/index.html`. last checked: 22.02.2010.

[17] Y. Pang, M. L. Yuan, A. Y. C. Nee, S. K. Ong, and Kamal Youcef-Toumi. A markerless registration method for augmented reality based on affine properties. In *AUIC '06: Proceedings of the 7th Australasian User interface conference*, pages 25–32, Darlinghurst, Australia, 2006. Australian Computer Society, Inc. ISBN 1-920682-32-5.

[18] Katharina Pentenrieder, Christian Bade, Fabian Doil, and Peter Meier. Augmented reality-based factory planning - an application tailored to industrial needs. In *ISMAR '07: Proceedings of the 2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality*, pages 1–9, Washington, DC, USA, 2007. IEEE Computer Society. ISBN 978-1-4244-1749-0. doi: http://dx.doi.org/10.1109/ISMAR.2007.4538822.

[19] Johannes Tümler. *Untersuchungen zu nutzerbezogenen und technischen Aspekten beim Langzeiteinsatz mobiler Augmented Reality Systeme in industriellen Anwendungen.* PhD thesis, OvGU Magdeburg, Fakultät für Informatik, 2009. URL `http://edoc.bibliothek.uni-halle.de/servlets/DocumentServlet?id=7909`.

[20] Daniel Wagner. Studierstube Tracker. URL `http://studierstube.icg.tu-graz.ac.at/handheld_ar/stbtracker.php`. last checked: 20.02.2010.

[21] Daniel Wagner. *HandheldAugmented Reality.* PhD dissertation, Graz University of Technology, Institute for Computer Graphics and Vision, October 2007. URL `http://studierstube.icg.tu-graz.ac.at/thesis/Wagner_PhDthesis_final.pdf`.

[22] Daniel Wagner and Dieter Schmalstieg. Artoolkitplus for pose tracking on mobile devices. In *CVWW'07: Proceedings of 12th Computer Vision Winter Workshop*, pages 139–146, Graz University of Technology, Institute for Computer Graphics and Vision, February 2007. URL `http://www.icg.tu-graz.ac.at/Members/daniel/Publications/ARToolKitPlus`.

[23] Daniel Wagner and Dieter Schmalstieg. Making augmented reality practical on

mobile phones, part 1. *IEEE Comput. Graph. Appl.*, 29(3):12–15, 2009. ISSN 0272-1716. doi: http://dx.doi.org/10.1109/MCG.2009.46.

[24] Daniel Wagner and Dieter Schmalstieg. Making augmented reality practical on mobile phones, part 2. *IEEE Comput. Graph. Appl.*, 29(4):6–9, 2009. ISSN 0272-1716. doi: http://dx.doi.org/10.1109/MCG.2009.67.

[25] Richard S. Wright, Benjamin Lipchak, and Nicholas Haemel. *OpenGL SuperBible: comprehensive tutorial and reference*. Addison-Wesly, fourth edition edition, 2007.