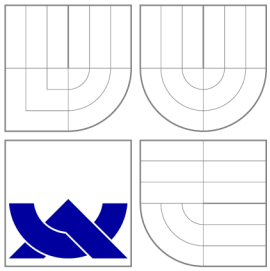
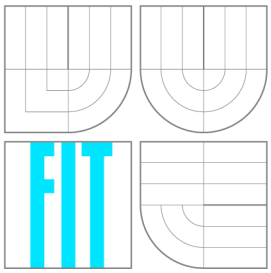


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

ROZŠÍŘENÁ REALITA PRO PLATFORMU ANDROID

AUGMENTED REALITY FOR ANDROID PLATFORM

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. RADEK LŽIČAŘ
AUTHOR

VEDOUCÍ PRÁCE — Ing. ALEŠ LÁNÍK
SUPERVISOR

BRNO 2009

SEM VLOŽTE ZADÁNÍ

Abstrakt

Tato diplomová práce pojednává o mobilní platformě Google Android, rozšířené realitě a knihovně ARToolKit. V rámci práce byla navržena a implementována aplikace demonstrující rozšířenou realitu na platformě Google Android.

Klíčová slova

rozšířená realita, mobilní telefon, smartphone, PDA, Java, ARToolKit, NyARToolkit, Google Android, OpenGL ES

Abstract

This thesis deals with Google Android platform, augmented reality and the ARToolKit library. An application demonstrating augmented reality for Google Android was designed and implemented in this thesis.

Keywords

augmented reality, mobilní telefon, smartphone, PDA, Java, ARToolKit, NyARToolkit, Google Android, OpenGL ES

Citace

Radek Lžičař: Rozšířená realita pro platformu Android, diplomová práce, Brno, FIT VUT v Brně, 2009

Rozšířená realita pro platformu Android

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana ing. Aleše Láníka

.....
Radek Lžičař
26. 5. 2009

Poděkování

Chtěl bych velmi poděkovat vedoucímu této práce ing. Aleši Láníkovi za cenné rady a podnětné připomínky k práci

Obsah

Kapitola 1

Úvod

Rapidně narůstající výpočetní výkon, paměťové možnosti a čím dál nižší pořizovací náklady mobilních zařízení otevírají nové možnosti jejich využití. Jednou z oblastí, kde začala mobilní zařízení v poslední době získavat uplatnění, je rozšířená realita – odvětví počítačového výzkumu, které se zaměřuje na možnosti kombinování reálného světa s daty generovanými počítačem. Mobilní telefony, smartphony, PDA apod. jsou lehkou a levnou hardwarovou platformou s jednoduchým, dobře známým uživatelským rozhraním. Moderní mobilní zařízení tohoto typu jsou v současnosti masově rozšířená a prakticky všechna jsou vybavena kamerou a displejem s vysokým rozlišením, což umožňuje tzv. průhledový („see-through“) způsob interakce. To z nich dělá zajímavou alternativu ke klasickým hardwarovým řešením pro rozšířenou realitu, jakými jsou například náhlavní displeje nebo počítače se statickou kamerou.

Tato diplomová práce si klade za cíl především demonstrovat možnosti rozšířené reality na přenosných zařízeních; za tímto účelem byla navržena a implementována aplikace určená pro novou mobilní platformu Google Android. Tato platforma je detailně popsána v kapitole 2. Kapitola 3 se věnuje obecně problematice rozšířené reality. Algoritmy implementované knihovnou ARToolKit, které tvoří jádro aplikace, jsou podrobně popsány v kapitole 4. Předmětem kapitoly 5 je návrh demonstrační aplikace. Její implementace je pak popsána v kapitole 6. Testování aplikace se věnuje kapitola 7. Poslední kapitola 8 pak hodnotí dosažené výsledky a nasťiňuje možné pokračování práce.

Kapitola 2

Google Android

Android je softwarová platforma určená především pro mobilní telefony, chytré telefony, PDA, GPS navigace a jiná mobilní zařízení. Jejími součástmi jsou operační systém, middleware a hlavní aplikace.

2.1 Historie

Roku 2005 koupila společnost Google malou společnost Android Inc., která byla zaměřená na software pro mobilní zařízení. Tento krok byl první známkou ochoty Googlu vstoupit na mobilní trh. V roce 2007 iniciovala spol. Google vznik aliance Open Handset Alliance (OHA), která má za cíl vytvářet otevřené standardy pro mobilní zařízení. Jejími členy je 47 významných společností jako např. Google, Intel, NVIDIA, Motorola, T-Mobile a další operátoři, výrobci mobilních zařízení, softwarové a jiné společnosti. Snahy OHA vytvořit otevřenou mobilní platformu odstartovaly velký boj s jejími hlavními konkurenty – společnostmi Microsoft, Apple, Symbian a dalšími. Microsoft vydal Windows Mobile 6.0 s vylepšenými Office Mobile a dalšími novinkami. Symbian, na jehož systémech běželo v té době více než 110 milionů mobilních zařízení, vydal OS v9.5, a Apple zaplavil trh iPhonem. Svět čekal na odpověd Googlu, kterou měl být takzvaný gPhone – telefon, který by mohl konkurovat iPhonu a jiným zařízením. OHA však přišla s mnohem lepším řešením – Google Android. Google Android je první skutečně otevřená mobilní platforma na světe, založená na Linuxu, s čistým a jedoduchým uživatelským rozhraním a možností vytvářet aplikace v jazyce Java. Společnost Google, místo toho aby vytvořila jeden jediný gPhone, přišla s platformou, kterou je možné integrovat do tisíců již existujících i zcela nových zařízení. [13] [3]

V roce 2007 společnost Google vydala vývojové nástroje – Android Software Development Toolkit – a vyhlásila soutěž Google Android Challenge. Cílem soutěže bylo podpořit vývojáře a co nejvíce urychlit vývoj softwaru pro platformu. Prozatím se uskutečnila dvě kola a společnost Google rozdala na odměnách 20 milionů dolarů. Až počátkem roku 2008 byly vydané kompletní zdrojové kódy Androidu. [1]

Na podzim roku 2008 se na trhu objevil první telefon běžící na platformě Android. Jedná se o komunikátor T-Mobile G1 (během vývoje známý jako HTC Dream). [3] [13]

[1] Všechny součásti platformy jsou k dispozici pod licencí „Apache free-software and open-source license“

2.2 Architektura

Aplikace Android je dodáván se sadou hlavních aplikací, mezi které patří internetový prohlížeč, kalendář, program pro SMS, kontakty, prohlížeč map a další. Vývoj aplikací pro platformu Android je možný pouze v jazyce Java.

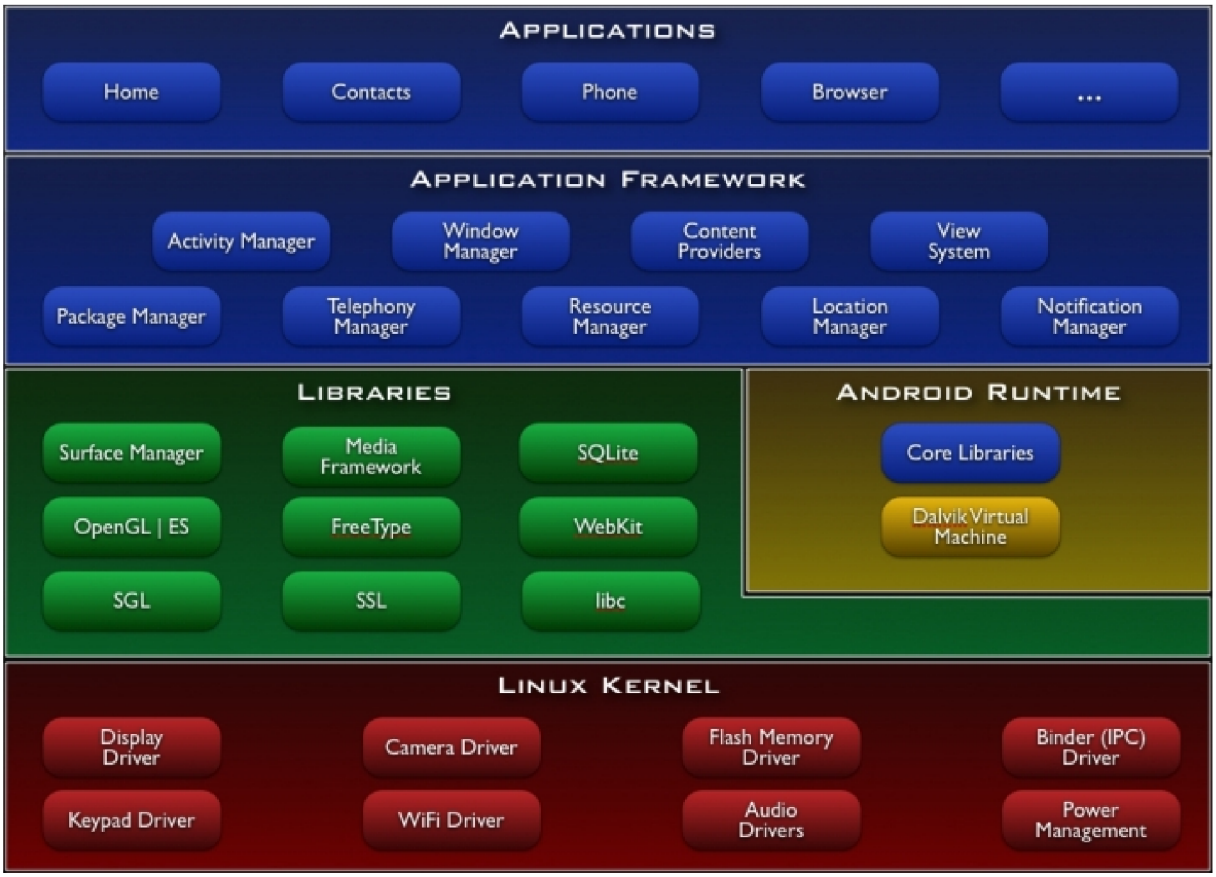
Aplikační framework Slouží pro psaní aplikací pro Android. Narozdíl od jiných mobilních prostředí jsou všechny aplikace pro Android rovnocenné. To znamená, že vývojáři mají přístup ke všem API, ke kterým mají přístup hlavní aplikace Androidu. Architektura je navržená s ohledem na jednoduché opakované využívání komponent – každá aplikace může „zveřejnit“ své možnosti a jakákoliv jiná aplikace je pak muže využívat. Stejný mechanismus umožnuje uživatelům nahrazovat existující komponenty.

Knihovny Android obsahuje C/C++ knihovny, využívané různými komponentami Androidu. Tyto knihovny jsou zpřístupněné vývojářům prostřednictvím aplikačního frameworku. Mezi hlavní knihovny patří např.:

- Systémová knihovna C – implementace standardni knihovny C (libc), optimalizovaná pro embedded zařízení
- Knihovny pro práci s médii – umožňují práci s velkým množstvím audio a video formátů a také statických obrázků, mezi podporované formáty patří JPG, PNG, MPEG4, AAC, AMR, H.264
- LibWebCore – moderní engine pro zobrazování webových stránek
- SGL – engine pro 2D grafiku (Scalable Graphics Engine)
- 3D knihovna – implementace založená na OpenGL ES, podporuje softwarové renderování i hardwarovou akceleraci
- FreeType – rendrování bitmapových a vektorových fontů
- SQLite – nenáročný relační databázový systém

Android Runtime Android obsahuje sadu hlavních knihoven (core libraries), které poskytují většinu funkcionality standardních knihoven jazyka Java. Každá aplikace běží ve svém vlastním procesu a ve vlastní instanci virtuálního stroje. Virtuální stroj používaný platformou Android se jmenuje Dalvik. Tento virtuální stroj byl vytvořen speciálně pro platformu Android. Slouží k běhu Java aplikací, převedených do kompaktního souboru typu Dalvik Executable (.dex). Dalvik je navržený s ohledem na atributy mobilních zařízení – má malé nároky na paměť i procesor.

Linuxové jádro Architektura Androidu je založená na Linuxovém jádru verze 2.6. To zajišťuje správu paměti, řízení procesů, síťové služby a další.



Obrázek 2.1: Architektura platformy Android [10]

2.3 Struktura aplikací

Aplikace Androidu se skládají z kombinace hlavních stavebních bloků. Hlavní čtyři stavební bloky jsou: Activity, Broadcast Intent Receiver, Service a Content Provider

2.3.1 Activity

Activity je nejčastěji používaný stavební blok. Activity obvykle odpovídá jedné obrazce aplikace. Komponenta Activity většinou zobrazuje uživatelské rozhraní složené z tzv. views (prvky uživatelského rozhraní jako např. textová pole, tlačítka, zaškrtávací pole atd., viz dále).

Intent

K přesunu mezi obrazovkami slouží třída Intent. Intent je prostředek, kterým může aplikace vyjádřit svůj požadavek na nějakou akci. Dvě nejdůležitější složky Intent jsou akce (MAIN, VIEW, PICK, EDIT atd.) a data, na kterých se má akce provést. Data jsou specifikována ve formě URI.[2] Pokud například aplikace potřebuje zobrazit údaje o nějaké osobě, může vytvořit Intent s akci VIEW a s URI, která reprezentuje danou osobu. Následující kód představuje ukázku použití Intentu, který způsobí vytočení telefonního čísla.

```
Intent i = new Intent("android.intent.action.CALL",
         ContentURI.create("tel:+420088533224"));
startActivity(i);
```

Intent Filter

Jaké akce je Activity (nebo Broadcast Receiver, viz níže) schopná zpracovat, popisuje Intent Filter. Například aplikace, která je schopná zobrazit kontaktní informace osoby, zpřístupní Intent Filter, který obsahuje informaci, že Activity umí zpracovat akci VIEW na datech reprezentujících informace o osobě. Activity zpřístupňují své Intent Filtery v souboru AndroidManifest.xml.

Aplikace, která chce spustit nějakou Activity, zavolá metodu startActivity() s parametrem specifikujícím daný požadavek ve formě Intent. Systém pak prohledá všechny nainstalované aplikace a vybere Activity, jejíž Intent filter nejlépe odpovídá danému Intent. Proces řešení Intentů probíhá za běhu, v okamžiku, kdy je zavolána metoda `startActivity()`. Tento přístup má dva hlavní přínosy: Activity může používat funkcionalitu jiných komponent jednoduše tím, že vytvoří požadavek ve formě Intent a mohou být kdykoliv nahrazeny jinou Activity s ekvivalentním Intent Filterem.

Životní cyklus Activity

Activity se může vyskytovat ve třech základní stavech:

- Je *bežící* (running), pokud je v popředí na obrazovce a má fokus pro uživatelské akce.
- Je *ozastavená* (paused), pokud ztratila fokus, ale je pořád viditelná pro uživatele.
- Je *zastavená* (stopped), pokud její okno je celé překryto jinou activtiy.

[2] Uniform Resource Identifier – řetězec znaků přesně specifikující zdroj nějakých dat

Pokud je Activity pozastavená nebo zastavená, může ji systém kdykoliv ukončit a uvolnit z paměti (např. v případě, že ostatní aplikaci potřebují paměť). O přechodech mezi jednotlivými stavy je Activity notifikována pomocí metod `onStart()`, `onPause()`, `onResume()` atd. V těchto metodách může Activity řešit různé záležitosti spojené se stavem, ve kterém se momentálně nachází – např. načítání dat při startu, uvolňování zdrojů při zastavení apod.

2.3.2 Broadcast Intent Receiver

Broadcast Intent Receiver slouží ke spouštění aplikace v reakci na nějakou externí událost, např. zvonění telefonu, příchozí SMS, určitá hodnota systémových hodin apod. Broadcast Intent Receivery nezobrazují žádné uživatelské rozhraní. Broadcast Intent Receivery jsou registrovány v AndroidManifest.xml. Aplikace mohou také posílat vlastní Broadcast Intenty ostatním aplikacím.

2.3.3 Service

Service (služba) je kód běžící na pozadí, zatímco uživatel může procházet ostatní aplikace. Příkladem může být hudební přehrávač, který hraje hudbu i v době, kdy uživatel spustil jinou aplikaci. K service je možné se připojit a komunikovat s ním pomocí rozhraní, které service zpřístupňuje. V případě přehrávače hudby tak může být například možné přehrávač pozastavit, přetočit apod.

2.3.4 Content Provider

Slouží ke sdílení dat mezi aplikacemi. Content Provider je třída, která implementuje standardní množinu metod, umožňující ostatním aplikacím ukládat a načítat data, která jsou spravována daným Content Providerem.

2.4 AndroidManifest.xml

Každá aplikace pro platformu Android musí obsahovat ve svém kořenovém adresáři soubor AndroidManifest.xml. Tento soubor dává systému základní informace o aplikaci. Manifest mimo jiné:

- Nastavuje název balíku aplikace. Název slouží jako jednoznačný identifikátor aplikace.
- Popisuje komponenty aplikace – activitiy, služby, broadcast receivery a content providery.
- Deklaruje povolení (permissions), která aplikace vyžaduje pro svůj běh (např. pro připojení k internetu).
- Deklaruje povolení, která musí mít ostatní aplikace, pokud chtějí s touto komunikovat.
- Deklaruje minimální verzi API, kterou aplikace vyžaduje.
- Vyjmenovává knihovny používané aplikací, které nejsou součástí hlavní knihovny (např. maps nebo awt).

Ukázka AndroidManifest.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
     package="cz.android.test">
    <uses-permission android:name="android.permission.INTERNET" />
    <application>
         <activity android:name=".MainActivity" android:label="Android Test">
            <Intent-filter>
                <action android:name="android.Intent.action.MAIN"/>
                <category android:name="android.Intent.category.LAUNCHER"/>
            </Intent-filter>
        </activity>
    </application>
</manifest>
```

V tomto AndroidManifest.xml aplikace specifikujem, že vyžaduje povolení pro přístup k internetu. Aplikace obsahuje jedinou Activity.

2.5 Uživatelské rozhraní v Androidu

Jak již bylo řečeno, uživatelské rozhraní (UI) se skládá z komponent zvaných View. Podtřídy třídy View reprezentují prvky uživatelského rozhraní jako například Button, ImageButton, EditText, CheckBox, RadioButton, AutoComplete.

Rozložení prvků na obrazovce je řízeno tzv. Layouts. Layout je objekt, který má na starosti určování velikosti a polohy prvků UI, které mu příslušejí. Mezi nejpoužívanější Layouts patří:

FrameLayout – nejjednodušší objekt typu Layout. Představuje prázdné místo, které později může být vyplněno jedním objektem, např. obrázkem.

LinearLayout – zarovnává prvky v jednom směru, buď vertikálně, nebo horizontálně.

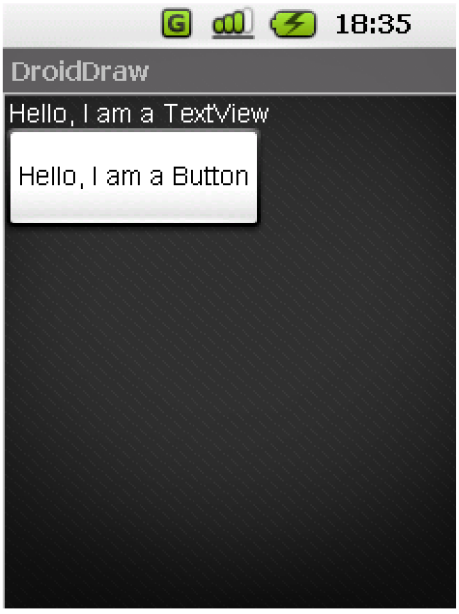
AbsoluteLayout – umožňuje specifikovat přesné souřadnice x, y na obrazovce, kde se mají prvky zobrazit.

TableLayout – zarovnává prvky do řádků a sloupců.

V Androidu existují dvě možnosti, jak vytvářet uživatelská rozhraní (UI):

- definovat rozložení jednotlivých elementu pomocí XML souborů
- vytvářet UI za běhu pomocí objektů View a GroupView

Android umožňuje kombinovat tyto metody pro vytváření a řízení uživatelského rozhraní. Je také možné odkazovat v programu na jednotlivé prvky UI definované v XML a měnit jejich stav. Výhoda definování UI v XML je ve větší přehlednosti a v oddělení od kódu – je možné měnit design UI bez nutnosti změn v programu a rekompilace. Navíc použití XML umožňuje snadnější vizualizaci UI a usnadňuje tak řešení případných problémů při jeho návrhu.



Obrázek 2.2: Ukázka UI

Každé XML má právě jeden kořenový element, jenž musí být buď View nebo ViewGroup. Do kořenového elemntu se vkládají vnořené elemnty, a tím se postupně vytváří hierarchie UI prvků. Následující výpis demonstruje použití LinerLayoutu, obsahujícího jeden Button (tlačítko) a TextView (needitovatelný textový řetězec). Výsledné rozvržení prvků je na obrázku 2.2.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
              android:layout_width="fill_parent"
              android:layout_height="fill_parent"
              android:orientation="vertical" >
    <TextView android:id="@+id/text"
              android:layout_width="wrap_content"
              android:layout_height="wrap_content"
              android:text="Hello, I˜am a TextView" />
    <Button android:id="@+id/button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Hello, I˜am a Button" />
</LinearLayout>
```

Znaky „@+“ v atributech `android:id` znamenají, že následující text se bude interpretovat jako identifikator, pomocí kterého pak bude možné na daný UI element odkazovat v programu. Atributy `android:layout_width` a `android:layout_height` určují výšku a šířku prvku. Hodnota `fill_parent` znamená, že daný element má mít stejný rozměr, jako jeho rodič. Hodnota `wrap_cotent` určuje, že se velikost prvku určí podle obsahu (např. u tlačítka bude v našem případě šířka a výška určená velikostí textu „Hello, I am a Button“).

2.6 Přístup k datům

Platforma Android poskytuje několik mechanismů pro práci s daty:

Zdroje Ve zdrojích (resources) se ukládají soubory, které aplikace využívá při svému běhu (obrázky, zvuky, textové řetezce, XML soubory definující UI apod.). Zdroje jsou po kompilaci součástí balíčku aplikace. Aplikace k nim přistupuje pomocí identifikátorů, které se jednoznačně přidělují podle názvu souboru a umístění v adresářové struktuře. Zdroje lze pak pomocí těchto identifikátorů z aplikace otevírat (pouze pro čtení).

Preference Preference jsou mechanismus pro ukládání menšího množství dat ve formě klíč-hodnota. Preference umožňují ukládání jednoduchých datových typů a typicky se využívají pro ukládání nastavení aplikace (např. uložení nastavené barvy pozadí atd.).

Soubory Je možné ukládat soubory přímo do paměti mobilního zařízení nebo na přenositelné medium. Ostatní aplikace k těmto souborům pak obecně nemají přístup.

Databáze Android podporuje vytváření SQLite databází. Každá databáze je privátní pro aplikaci, která ji vytvořila. Android poskytuje funkce umožňující ukládání užitečných komplexních datových struktur – je např. definován datový typ pro informace o kontaktu skládající se z polí pro jméno, příjmení, telefon, adresu, fotografii a další.

2.7 Bezpečnostní model

Android je víceprocesový systém, kde každá aplikace (a každá část systému) běží ve svém vlastním procesu. Většina bezpečnostních opatření mezi aplikacemi a systémem je zajištěna na úrovni procesů pomocí standardních Linuxových prostředků, jako jsou ID uživatelů a skupin. Dalším bezpečnostním mechanismem jsou tzv. povolení (permissions), která kladou omezení na specifické operace, které určitý proces může vykonávat. Zajištění ad-hoc přístupu ke specifickým datům řeší povolení vázaná na URI.

Bezpečnostní model Androidu je založen na tom, že žádná aplikace nesmí implicitně provádět žádné operace, které by mohly mít nepříznivý vliv na ostatní aplikace, operační systém nebo uživatele. Mezi tyto operace patří např. čtení a zápis soukromých dat (kontakty, e-maily, atd.), čtení a zápis souborů jiných aplikací, přístup k síti a další.

Proces, ve kterém je aplikace spuštěná, nemůže tyto operace provádět, pokud mu to není explicitně umožněno pomocí povolení. Požadavky na povolení k těmto operacím jsou v aplikacích, které je vyžadují, staticky uloženy, jsou tedy známy už v době instalace aplikace a nemohou se měnit. Žádosti o povolení mohou být systémem zpracována různými způsoby, typicky je systém povoluje/zamítá na základě certifikátů nebo dotazování uživatele.

Každá aplikace pro Android musí být podepsaná certifikátem[3], majitelem privátního klíče certifikátu je vývojář aplikace.

[3]Certifikát slouží pouze k identifikaci vývojáře a není nutné, aby byl podepsán certifikační autoritou

Kapitola 3

Rozšířená realita

3.1 Definice rozšířené reality

Rozšířená realita (Augmented Reality, AR) je druh virtuální reality (Virtual Reality, VR). Virtuální realita i rozšířená realita se zaměřují na poskytování informací uživatelům umístěným ve 3D prostředí. Zatímco však virtuální realita kompletně obklopuje uživatele umělým světem, rozšířená realita využívá existující reálné prostředí, které doplňuje o počítačem generované informace potřebné v příslušném kontextu. Paul Milgram rozšířenou realitu na tzv. ose virtuálního kontinua umisťuje mezi reálné a virtuální prostředí. (viz obr. 3.1) [15][8]

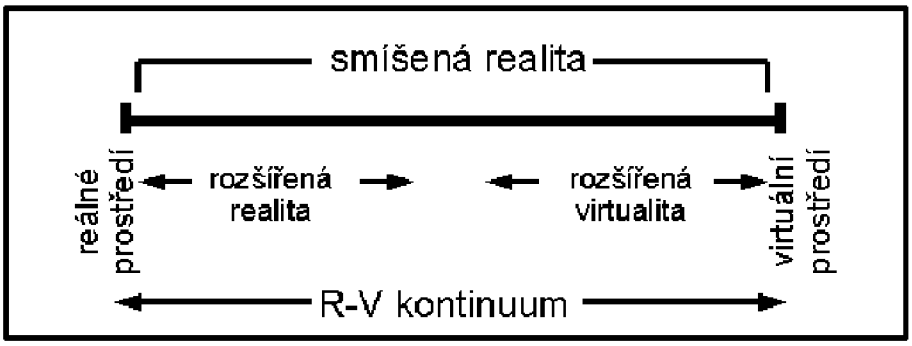
Jednotná definice rozšířené reality neexistuje. Definice AR vycházejí více či méně z definice virtuální reality – „prostředí vytvořené počítačem, trojrozměrné, interaktivní, kterým je uživatel obklopen“. V literatuře se setkáváme s definicemi, které pojem AR úzce spojují se speciálním hardwarem, typicky HMD.[1] Druhý typ definicí je obecnější a nevylučuje použití bežných zobrazovacích zařízení. Obecnější definice vznikly jako reakce na nově vznikající implementace, pro které definice spojené s hardwarem přestaly stačit.

R. Azuma [8] definuje systémy rozšířené reality jako systémy, pro které platí následující podmínky:

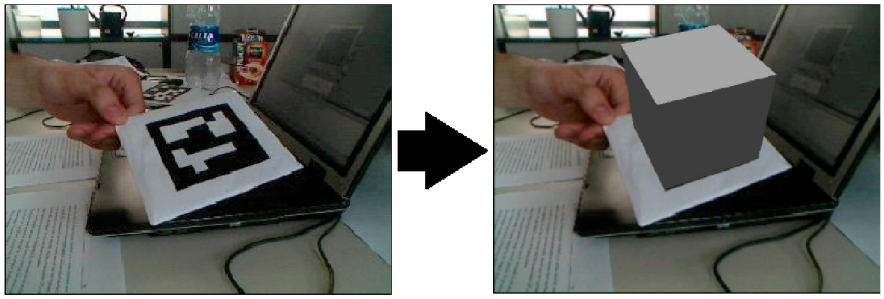
- Kombinují realné a virtuální.
- Jsou interaktivní v reálném čase.
- Jsou registrované ve 3D.

Definice je tedy podobná definici VR. V případě VR je však snaha, aby uživatel vnímal pouze virtuální prostředí. V AR je naproti tomu snaha zachovat kontakt uživatele s reálným světem. Dále existují pojmy rozšířená virtualita a smíšená realita. Rozšířená virtualita je definována jako vkládání obrazů reálných objektů do virtuální scény. Smíšená realita je termín zahrnující jak rozšířenou realitu, tak rozšířenou virtualitu.

[1] Head-Mounted display – průhledové náhlavní obrazovky



Obrázek 3.1: Osa virtuálního kontinua [15]



Obrázek 3.2: Sledování polohy kamery pomocí značky

3.2 Principy AR

Pro každý AR systém je především nutné správným způsobem přidávat virtuální objekty do reálné scény (tzv. registrace). K tomu je potřeba zjistit, kde se uživatel v reálném světě nachází a jakým směrem se dívá. Existují dva hlavní typy přístupů, které toto umožňují. Prvním z nich je využití polohového senzoru (používá se především u HMD). [12] Druhým přístupem je sledování směru pohledu uživatele pomocí značek, které jsou předem umístěné za tímto účelem do scény. Tento přístup bude využívat i aplikace vyvinutá v rámci této diplomové práce. Systémy, využívající tento přístup, pracují na následujícím principu:

1. Kamera snímá video s reálnou scénou a zasílá je do počítače.
2. Počítač v každém snímku videa hledá značky a snaží se je identifikovat.
3. Pokud nalezne nějakou značku, vypočítá polohu kamery relativně ke značce.
4. Data o nalezených značkách a poloze kamery vůči nim se zpracují, vypočtou se polohy virtuálních objektů.
5. Na monitoru nebo displeji se vykreslí reálná scéna spolu s virtuálními objekty.

Na obrázku 3.2 je znázorněno využití značky ve scéně k výpočtu polohy kamery vzhledem ke značce pro následné vložení virtuálního objektu.

Obrázek 3.3: Video průhledové HMD [9]

Obrázek 3.4: Opticky průhledové HMD [9]

Výslednou scénu je možné zobrazit buď na monitor počítače (popřípadě jiného zařízení) nebo pomocí specializovaného hardware – HMD. Jedná se o obrazovky, které uživatel nosí nasazené na hlavě. Existují dva hlavní typy HMD:

- video průhledové HMD (video see-through)
- opticky průhledové HMD (optical see-through)

Optical see-through HMD (obr. 3.4) fungují tak, že je obraz virtuálních objektů promítán na polopropustné zrcadlo, přes které se uživatel dívá na reálnou scénu.

U video see-through HMD (obr. 3.3) obrazovek je výsledku dosaženo tím, že na 3D brýlích je připevněna kamera snímající reálnou scénu ve směru pohledu uživatele, virtuální a reálný obraz je zkombinován a uživatel pak na obrazovce vidí tento sloučený obraz.

Obrázek 3.5: Ukázka hry ARQuake [4]

3.3 Využití AR

Rozšířená realita má široké uplatnění. Následuje stručný výčet několika oblastí využití AR [8] [15] [12] [2] [4].

Montáže a opravy AR lze využít při podpoře vykonávání komplexních úkolů v oblasti montáží a oprav. Například společnost Boeing vyvinula software, jenž unsadňuje montáž kabelů do letadel tím, že pomocí AR zvýrazňuje záchytné body, zásuvky apod.

Bezpečnostní složky V armádě už se používají přístroje typu HMD již delší dobu a to jak v letectví, tak pro pěší vojsko. Využití AR v bezpečnostních složkách zahrnuje zobrazování map, různých instrukcí, poloh nepřátel apod.

Medicína Rozšířená realita zde může pomáhat vizualizací skrytých objektů – při diagnostice nebo operaci může AR aplikace např. promítat virtuální „rentgenový“ pohled založený na datech z předchozí tomografie nebo na snímcích z ultrazvuku, získávaných v reálném čase.

Spolupráce distribuovaných týmů Například telekonference, společná práce na simulovaném 3D modelu apod.

Vzdělávání V dnešní době již existuje několik aplikací nacházejících uplatnění v muzeích. AR v nich umožňuje zobrazovat virtuální objekty, popisky, animace apod.

Zábava V herním průmyslu je možné očekávat velké využití rozšířené reality. Rozšířená realita vnáší do herního průmyslu zcela nové koncepty. Jako příklad uveďme hru ARQuake (viz obr. 3.5). Jednalo se o AR verzi populární 3D hry Quake. ARQuake umožňoval hráčům běhat v reálném světě a bojovat proti virtuálním nepřátelům. Z poslední doby je dobře známá hra The Eye of Judgement pro PlayStation 3, která používá rozšířenou realitu k zobrazování virtuálních nestvůr na reálných kartách.

3.4 Mobilní AR

Mnoho dřívějších i současných AR aplikací je založeno na klasickém přístupu využívajícím statické desktopové počítače s kamerou nebo HMD spolu s laptopem, umístěným v batohu na zádech. Tento přístup přináší vysoký výkon, ale také určité nevýhody – především vysokou cenu a omezení uživatelovy pohyblivosti. Tyto nedostatky zabraňují masovému rozšíření mezi široké spektrum netechnických uživatelů.

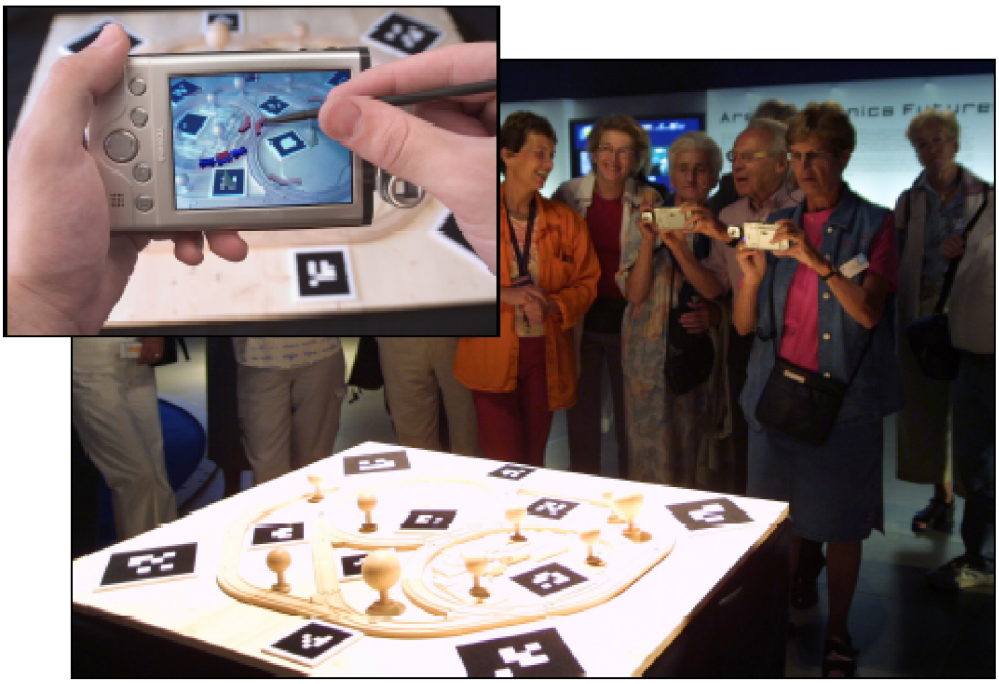
Naproti tomu přenosná zařízení mohou asistovat uživateli prakticky kdekoliv a jsou čím dál levnější a výkonnější. Proto se jeví jako vhodná hardwarová platforma pro aplikace AR.

V počátcích mobilní AR aplikací, tj. v době před několika lety, využívaly projekty zaměřené na mobilní AR zařízení jako jsou PDA apod. jen jako tzv. tenké klienty. Tato zařízení tedy sloužila pouze k jednoduchému zpracování vstupních dat a k zobrazování dat výstupních. Veškeré náročnější výpočty prováděly servery, např. osobní počítače, které s mobilními zařízeními komunikovaly prostřednictvím bezdrátových sítí. Příkladem těchto aplikací, používajících přístup s tenkými klienty, jsou např. AR-PDA, AR-Phone nebo Batportal.

V době, kdy tyto projekty vznikaly, byl přístup využívající tenké klienty kvůli omezeným možnostem tehdejších mobilních zařízení nutný. Během posledních několika let však začal výkon mobilních zařízení po všech stránkách rapidně narůstat. Pokročilá hardwarová řešení v dnešní době umožňují vyvíjet poměrně komplexní aplikace i pro mobilní zařízení. Navíc jsou moderní telefony, PDA, apod. vybaveny kvalitními kamerami a displeji s vysokými rozlišeními. To vše umožňuje používat tato zařízení jako zcela samostatné platformy pro AR aplikace, včetně výpočetně náročného zpracování obrazu a vykreslování 3D grafiky. Nutnost používat servery pro výpočty spojené s algoritmy AR tak odpadá. [16]

Mobilní zařízení je díky jejich vlastnostem možné používat jako see-through zařízení. V literatuře se pro tento způsob interakce používá metafora „magický objektiv“. Při adaptaci tohoto see-through přístupu na mobilní zařízení uživatel zamíří zařízením na nějakou část reálného světa a na displeji se zobrazí tato část rozšířená o virtuální objekty. Ukázka použití „magického objektivu“ je na obrázku 3.6. Jedná se o aplikaci The Invisible Train. Reálné dřevěné kolejiště je rozšířeno o virtuální animované vlaky a výhybky, ty jsou však vidět pouze v případě, že uživatel sleduje trať skrze obrazovku PDA.

V nedávné době vzniklo několik prací zabývajících se mobilní rozšířenou realitou. Anders Henrysson naportoval knihovnu ARToolKit (viz 4) pro zařízení se systémem Symbian. Mathias Moehring napsal alternativní knihovnu pro mobilní zařízení, založenou také na technikách počítačového vidění. Dalším podobně zaměřeným frameworkem je Studierstube ES. Tyto knihovny jsou významným krokem k urychlení rozvoje aplikací rozšířené reality pro mobilní zařízení. [16]



Obrázek 3.6: The Invisible Train [16]

Kapitola 4

ARToolKit

ARToolKit je softwarová knihovna napsaná v jazyce C určená ke tvorbě aplikací pro rozšířenou realitu.

Jedním ze zásadních problémů při vývoji aplikací pro rozšířenou realitu je sledování úhlu pohledu uživatele. Aby aplikace mohla správně vykreslovat virtuální objekty, musí umět s dostatečnou přesností zjistit, odkud a kam se uživatel dívá v reálném světě. Tento problém řeší ARToolKit pomocí algoritmů počítačového vidění. Knihovny ARToolKitu počítají reálnou pozici a orientaci kamery vztaženou k fyzickým značkám (markers) v reálném čase.

Mezi hlavní funkce a vlastnosti ARToolKitu patří [1]:

- detekce pozice a orientace kamery
- možnost používat vlastní vzory značek
- jednoduchá kalibrace kamery
- dostatečná rychlost pro aplikace běžící v reálném čase
- distribuce pro SGI IRIX, Linux, MacOS, Windows
- distribuce s kompletními zdrojovými kódy

  Knihovna ARToolKit je dostupná na adrese `http://www.hitl.washington.edu/artoolkit` (prosinec 2008).

4.1 Principy fungování ARToolKitu

V této podkapitole budou stručně popsány algoritmy a metody využívané balíkem ARToolKit. ARToolKit pracuje v několika krocích. Nejdříve se vstupní obraz z kamery převede pomocí prahování na černobílý.

Poté v tomto černobílém obraze ARToolKit hledá čtyřúhelníkové oblasti. U každé z těchto nalezených oblastí se provede test, zda se jedná o hledanou značku – vzor z vnitřku každé oblasti se porovná s šablonami předtrénovaných vzorů. ARToolKit potom ze získaných informací o velikosti a orientaci značky vypočítá polohu kamery vzhledem k této značce. Výstupem této fáze je matice velikosti 3x4, která obsahuje polohu kamery v reálném světě vůči značce. Tato matice se použije k nastavení souřadnic virtuální kamery. Poté už je možné vykreslovat virtuální objekt. Protože se souřadnice reálné a virtuální kamery shodují, vykreslený objekt přesne překrývá skutečnou značku.

4.1.1 Kalibrace kamery

Virtuální objekty, které v AR aplikacích vkládáme do scény, by měly co nejlépe přiléhat na reálné. Je tedy potřeba co nejpřesněji detekovat polohu a orientaci značek v obraze. Každá kamera však vnáší do vstupního obrazu jistou míru zkreslení (lens disortion) kvůli nedokonalostem v optické soustavě. Pro eliminaci tohoto zkreslení umožňuje ARToolKit provádět kalibraci kamery. Jedná se o proces, při kterém se pomocí snímaní kalibračního obrazce se známými vlastnostmi zjišťuje projekční matice kamery **P**. Kalibrační obrazec obsahuje body, jejichž souřadnice v lokální soustavě souřadnic jsou známy. Tyto souřadnice v souřadném systému obrazovky lze pak detekovat při procesu kalibrace. Vztah mezi souřadnicemi obrazovky (x_c, y_c), souřadnicemi kamery (X_c, Y_c, Z_c) a souřadnicemi kalibračního obrazce (X_t, Y_t, Z_t) lze vyjádřit rovnicí (4.1) [12][9].

$$\begin{bmatrix} hx_c \\ hy_c \\ h \\ 1 \end{bmatrix} = \mathbf{P} \begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} = \mathbf{P}.\mathbf{T_{ct}} \begin{bmatrix} X_t \\ Y_t \\ Z_t \\ 1 \end{bmatrix} = \mathbf{C}. \begin{bmatrix} X_t \\ Y_t \\ Z_t \\ 1 \end{bmatrix} = \begin{bmatrix} C_{11} & C_{12} & C_{13} & C_{14} \\ C_{21} & C_{22} & C_{23} & C_{24} \\ C_{31} & C_{32} & C_{33} & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_t \\ Y_t \\ Z_t \\ 1 \end{bmatrix}$$

$$\mathbf{P} = \begin{bmatrix} s_x f & 0 & x_0 & 0 \\ 0 & s_y f & y_0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \mathbf{T_{ct}} = \begin{bmatrix} R_{11} & R_{12} & R_{13} & T_x \\ R_{21} & R_{22} & R_{23} & T_y \\ R_{31} & R_{32} & R_{33} & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.1}$$

kde **P** je hledaná matice perspektivní transformace, f je ohnisková vzdálenost, s_x je koeficient zvetšení [pixel/mm] ve směru osy x, s_y je koeficient zvětšení ve směru osy y, (x_0, y_0) je místo průsečíku optické osy s obrazem, $\mathbf{T_{et}}$ reprezentuje translační a rotační transformaci ze souřadnic kalibračního obrazce na souřadnice kamery a **C** je transformační matice získaná kombinací **P** a $\mathbf{T_{et}}$.

V průběhu kalibrace je získáno pomocí snímání kalibračního obrazce mnoho párů (x_c, y_c) a (X_t, Y_t, Z_t). Z nich může být vypočítána matice **C**. Matici **C** však nelze obecně rozložit na matice **P** a $\mathbf{T_{ct}}$, protože matice **C** obsahuje 11 nezávislých proměnných, zatímco **P** a $\mathbf{T_{ct}}$ obsahují 4, resp. 6 nezávislých proměnných. Součet počtu nezávislých proměnných **C** a $\mathbf{T_{ct}}$ je tedy různý od počtu nezávislých proměnných matice **C**. Proto je do matice **P** přidána skalární proměnná k, jak je vidět v rovnici (4.2). V důsledku přidání této proměnné je možné matici **C** rozložit na **P** a $\mathbf{T_{ct}}$. [12][9]

$$\begin{bmatrix} hx_c \\ hy_c \\ h \\ 1 \end{bmatrix} = \begin{bmatrix} s_x f & k & x_0 & 0 \\ 0 & s_y f & y_0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} R_{11} & R_{12} & R_{13} & T_x \\ R_{21} & R_{22} & R_{23} & T_y \\ R_{31} & R_{32} & R_{33} & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_t \\ Y_t \\ Z_t \\ 1 \end{bmatrix} \tag{4.2}$$

Proměnná k vyjadřuje zkosení mezi x-ovou a y-ovou souřadnicí a měla by být v ideálním případě nulová.

4.1.2 Prahování

V první fázi hledání značek v obraze provádí ARToolKit binární prahování. Prahování (thresholding) je jedna z nejstarších, nejjednodušší a často používaných metod pro segmentaci obrazu. Algoritmus převádí vstupní obraz na obraz binární, tedy na obraz, kde má

každý pixel hodnotu 0 (pozadí) nebo 1 (popředí). Obecná definice prahování je:

$$g(i,j) \quad = \quad \begin{cases} 0, & f(i,j) \leq T \\ 1, & f(i,j) > T \end{cases}$$

kde $g(i,j)$ je segmentovaný obraz, T je práh a $f(i,j)$ vyjadřuje nějakou vlastnost zpracovávaného obrazu (v našem případě intenzitu). ARToolKit používá globální prahování, tzn. že pro celý obraz používá jednu hodnotu prahu. Nevýhodou tohoto přístupu je, že nedosahuje dobrých výsledků na obrazech s nehomogenním osvětlením.

4.1.3 Detekce souvislých oblastí

Dalším krokem, který ARToolKit provádí, je detekce souvislých oblastí (objektů). K tomu se využívá tzv. algoritmus barvení (Connected Component Labeling). Než přistoupíme k definici souvislé oblasti, definujme pojem *spojitá cesta*. *Spojitá cesta* je pojem definující cestu mezi body X a Y jako posloupnost n bodů $p_1, p_2, \ldots, p_n$, kde $p_1 = X$ a $p_n = Y$ a pro kterou platí, že pro všechny dvojice bodů p_i a p_{i+1} platí, že bod p_{i+1} leží v okolí bodu p_i. *Souvislá oblast* je oblast, pro kterou platí, že existuje spojitá cesta mezi libovolnými dvěma body, které v ní leží.

Algoritmus barvení přiřadí všem pixelům, které náleží do souvislé oblasti, stejné číslo („barvu“). Algoritmus je dvouprůchodový a funguje následujícím způsobem:

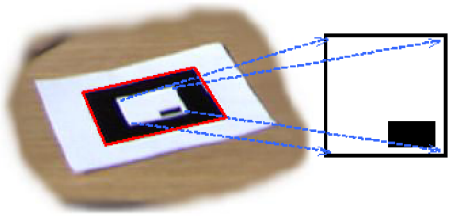
1. V prvním průchodu se každému bodu binárního obrazu P, který patří k popředí (tj. má hodnotu 1) přiřadí barva, která se určí z hodnot bodů z čtyřokolí (resp. osmiokolí) bodu P takto:
   - Pokud jsou všechny body ležící v okolí bodu P součástí pozadí (tj. mají-li hodnotu nula), pak se bodu P přiřadí nová, dosud nepřiřazená hodnotu.
   - Mají-li všechny body ležící v okolí bodu P nenulové body stejnou barvu, pak se bodu P přiřadí tato barva.
   - Existují-li v okolí bodu P nenulové body, které mají různou barvu, pak se bodu P přiřadí nejmenší z těchto barev a poznačí se ekvivalence barev.
2. V druhém průchodu se nahradí barvy, které jsou ekvivalentní, jednou barvou.

Výstupem algoritmu jsou tedy jednotlivé spojité oblasti, označené „barvami“. U každé z oblastí si ARToolKit uchovává pomocné informace jako např. souřadnice hraničního obdélníku.

4.1.4 Extrakce obrysů

Po detekci a označení oblastí algoritmem barvení je nutné provést extrakci obrysů nalezených oblastí. To se provádí následujícím způsobem (algoritmus sledování obrysu) [9]:

1. Najdi první bod P, který je obrysovým bodem objektu.
2. Přidej bod P na konec seznamu obrysových bodů objektu.
3. Najdi další obrysový bod P objektu proti směru hodinových ručiček.
4. Pokud je bod P různý od prvního bodu ze seznamu obrysových bodů, pokračuj krokem 2.



Obrázek 4.1: Normalizace značky [9]

Získané obrysy nalezených objektů ARToolKit aproximuje liniovými segmenty. V dalším zpracování se berou v úvahu jen objekty, jejichž obrysy jsou reprezentovány právě čtyřmi hranami – tedy oblasti, které mohou potenciálně být hledanými značkami [1][12][9].

Z parametrických rovnic hran se vypočítají souřadnice rohů hraničního čtyřúhelníku a uchovají se pro pozdější použití.

4.1.5 Identifikace značky

Výstupem předchozí fáze jsou souřadnice vrcholů čtyřúhelníkových oblastí. S těmito oblastmi se nyní provede porovnávání se vzorem (pattern matching), aby ze zjistilo, zda se skutečně jedná o značky ARToolKitu.

Před samotným porovnáním se musí nejprve vnitřek značky promítnout do čtvercové oblasti, jak je vidět na obrázku 4.1. Toho lze dosáhnout pomocí transformační matice s parametry $a_1, \ldots, a_8$, která reprezentuje transformaci mezi vrcholy čtyřúhelníku a rohy vzoru značky [12][9].

$$\begin{bmatrix} hX_i \\ hY_i \\ h \end{bmatrix} = \begin{bmatrix} a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 \\ a_7 & a_8 & 1 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} \tag{4.3}$$

V rovnici (4.3) reprezentují x_i a y_i souřadnice vrcholů čtyřúhelníku, X_i a Y_i reprezentují rohové body vzoru značky. Parametry $a_1, \ldots, a_8$ lze vypočítat ze soustavy osmi lineárních rovnic o osmi nezávislých proměnných (4.4). K výpočtu jsou potřeba alespoň čtyři páry bodů. Čtyřúhelník spolu se vzorovou značkou tuto podmínku splňují.

$$\begin{bmatrix} X_1 \\ Y_1 \\ \ldots \\ Y_4 \end{bmatrix} = \begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -X_1.x_1 & -X_1.y_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -Y_1.x_1 & -Y_1.y_1 \\ & & & & \ldots & & & \\ 0 & 0 & 0 & x_4 & y_4 & 1 & -Y_4.x_4 & -Y_4.x_4 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ \ldots \\ a_8 \end{bmatrix} \tag{4.4}$$

Získaná transformační matice slouží k namapování testovaného čtyřúhelníku na matici **E**, která má stejnou velikost jako matice vzoru **M**. Podobnost těchto matic je dána vzorcem (4.5)

$$p = \frac{\sum_m \sum_n \mathbf{E}[m,n].\mathbf{M}[m,n]}{\frac{c_\mathbf{M}}{c_\mathbf{E}}} \tag{4.5}$$

kde $c_\mathbf{M}$ a $c_\mathbf{E}$ označují geometrický průměr hodnot matice **M**, resp. matice **E**. Každý čtyřúhelník je porovnán se všemi značkami, které se mohou objevit ve scéně. Pro každý čtyřúhelník je zjištěna značka, která se mu nejvíce podobá (minimální hodnota p s využitím určitého prahu podobnosti). Porovnávání se značkami probíhá pro všechna možná natočení

(0°, 90°, 180°, 270°). V této fázi se každému čtyřúhelníku, u kterého proběhne úspěšně identifikace značky, přiřadí ID značky a uchová se i její natočení pro pozdější použití.

4.1.6 Zjištění pozice a orientace značky

Zjištění pozice a orientace značky je klíčovým úkolem ARToolKitu.

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} = \begin{bmatrix} V_{11} & V_{12} & V_{13} & W_x \\ V_{21} & V_{22} & V_{23} & W_y \\ V_{31} & V_{32} & V_{33} & W_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_m \\ Y_m \\ Z_m \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{V_{3x3}} & \mathbf{W_{3x1}} \\ 0\ 0\ 0 & 1 \end{bmatrix} \begin{bmatrix} X_m \\ Y_m \\ Z_m \\ 1 \end{bmatrix} = \mathbf{T_{cm}} \begin{bmatrix} X_m \\ Y_m \\ Z_m \\ 1 \end{bmatrix} \tag{4.6}$$

Klíčová transformace $\mathbf{T_{cm}}$ ze souřadnic značky na souřadnice kamery je získána pomocí analýzy vstupního obrazu [1][12].

Když se dvě paralelní hrany značky promítnou do obrazu, mají rovnice odpovídajících přímek v soustavě souřadnic obrazovky tvar:

$$\begin{aligned} l_1 &: a_1 x + b_1 y + c_1 = 0 \\ l_2 &: a_2 x + b_2 y + c_2 = 0 \end{aligned} \tag{4.7}$$

Pro každou značku v obraze byly hodnoty parametrů těchto přímek zjištěny v předchozích krocích. Dále známe matici perspektivní projekce $\mathbf{P}$, která je získána kalibrací kamery. Dosazením do (4.8) získáme rovnice rovin, které obsahují přímky l_1 a l_2 (4.9)

$$\mathbf{P} = \begin{bmatrix} P_{11} & P_{12} & P_{13} & 0 \\ 0 & P_{22} & P_{23} & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad \begin{bmatrix} hx_c \\ hy_c \\ h \\ 1 \end{bmatrix} = \mathbf{P} \begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} \tag{4.8}$$

$$\begin{aligned} a_1 P_{11} X_c + (a_1 P_{12} + b_1 P_{22}) Y_c + (a_1 P_{13} + b_1 P_{23} + c_1) Z_c = 0 \\ a_2 P_{11} X_c + (a_2 P_{12} + b_2 P_{22}) Y_c + (a_2 P_{13} + b_2 P_{23} + c_2) Z_c = 0 \end{aligned} \tag{4.9}$$

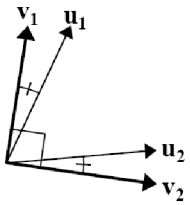
Nechť $\mathbf{n_1}$ a $\mathbf{n_2}$ jsou normálové vektory těchto rovin. Směrový vektor paralelních hran značky je pak $\mathbf{u_1} = \mathbf{n_1} \times \mathbf{n_2}$. Obdobným způsobem pak vypočítáme směrový vektor pro druhou dvojici paralelních hran značky.

Po normalizaci tedy získáme dva jednotkové směrové vektory:

$$\begin{aligned} \mathbf{u_1} &= \mathbf{n_1} \times \mathbf{n_2} \\ \mathbf{u_2} &= \mathbf{n_3} \times \mathbf{n_4} \end{aligned} \tag{4.10}$$

Vektory $\mathbf{u_1}$ a $\mathbf{u_2}$ by měly být na sebe kolmé, protože jsou získány ze dvou dvojic paralelních hran čtvercové značky. Při zpracování obrazu však dochází k chybám, které způsobují, že dané vektory nesvírají přesně pravý úhel. Proto jsou vypočítány výsledné jednotkové vektory $\mathbf{v_1}$ a $\mathbf{v_2}$, které tuto chybu kompenzují. Tyto vektory leží v rovině obsahující vektory $\mathbf{u_1}$ a $\mathbf{u_2}$, jak je vidět na obrázku 4.2. Vektory $\mathbf{v_1}$, $\mathbf{v_2}$ a vektor $\mathbf{v_3}$, který je na ně kolmý, tvoří rotační složku $\mathbf{V_{3x3}}$ transformační matice $\mathbf{T_{cm}}$ ze (4.6):

$$\mathbf{V_{3x3}} = \begin{bmatrix} \mathbf{v}_1^T & \mathbf{v}_2^T & \mathbf{v}_3^T \end{bmatrix} \tag{4.11}$$



Obrázek 4.2: Kolmé jednotkové vektory v_1 a v_2 [9]

Je tedy známa rotační část matice a lze vypočítat translační složku $\mathbf{W_{3x1}}$. Použijeme rovnice (4.6) a (4.8). Známe souřadnice čtyř vrcholů značky v soustavě souřadnic značky a souřadnice vrcholů značky v soustavě souřadnic obrazovky. Dosazením těchto souřadnic získáme soustavu osmi lineárních rovnic, z nichž je možné vypočítat hodnoty W_x, W_y a W_z.

Transformační matice získaná výše popsanou metodou může obsahovat chybu. Tato chyba může však být snížena následujícím postupem. Souřadnice vrcholů značky v souřadné soustavě obrazovky se vypočítají ze souřadnic vrcholů v souřadné soustavě značky pomocí transformační matice. Pak se optimalizují hodnoty rotační složky transformační matice, tak aby se snížila celková chyba mezi vypočítanými souřadnicemi a souřadnicemi, které byly naměřeny ve vstupním obraze. Po optimalizaci rotační složky se translační složka přepočítá pomocí výše zmíněné metody. Tento postup se několikrát zopakuje, což zajistí přesnější zjištění transformační matice. [1][12][9]

Kapitola 5

Návrh demonstrační aplikace

V této kapitole je popsán návrh aplikace demonstrující AR. Navržená aplikace vychází z flashové hry *Crazy Cube*.[1] Tato logická hra, ve které se na herní krychli spojují různě barevná políčka, je ze své podstaty velice vhodná k převodu do rozšířené reality s využitím mobilního zařízení pro průhledový způsob interakce. Výsledná aplikace –*AR Cube* – bude zajímavou formou demonstrovat všechny hlavní principy rozšířené reality.

5.1 Pravidla hry

Hra probíhá na třech stěnách krychle, přičemž každá stěna je rozdělena na 3x3, nebo 4x4 barevná pole. Na počátku hry se na herní kostce vyskytuje několik různě barevných dvojic výchozích polí. Všechna tato výchozí pole jsou označena černým čtvercem uprostřed. Ostatní pole jsou buď šedá nebo černá. Cílem hráče je změnit barvu šedých polí takovým způsobem, aby mezi všemi dvojicemi výchozích polí existovala cesta dané barvy. Každé pole může mít pouze jednu barvu, není tedy možné cesty křížit. Speciální černá pole nemohou být při tvorbě cesty mezi výchozími body použita – hráč jim nemůže měnit barvu.

Pokaždé, když dojde ke spojení všech dvojic výchozích bodů, hra přejde do další úrovně s jinou počáteční konfigurací.

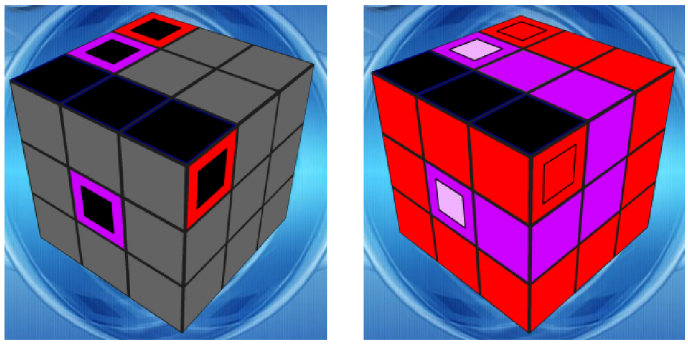
Příklad určité počáteční konfigurace kostky a odpovídající cílové konfigurace, kde jsou obě dvojice výchozí bodů propojené cestou, můžeme vidět na obr. 5.1.

5.2 AR verze

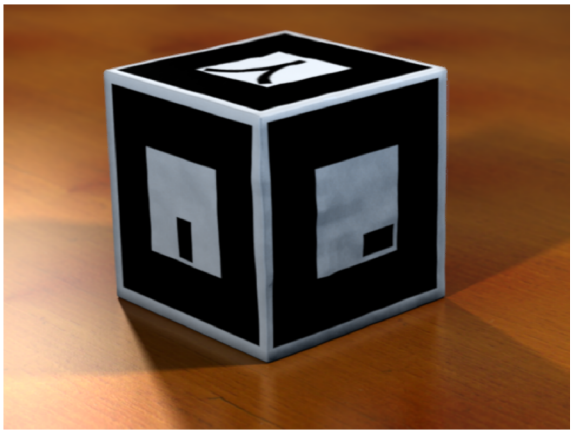
AR Cube bude využívat pro reprezentaci hrací krychle v reálném světě fyzickou kostku. Tato kostka bude mít na třech sousedních stěnách nalepeny různé značky ARToolKitu (viz obr. 5.2). Hráč zamíří svoje mobilní zařízení tak, aby v zorném úhlu kamery byla alespoň jedna ze značek ARToolKitu. To umožní aplikaci, aby pomocí AR algoritmů vypočítala polohu kamery vuči kostce. Na displej zařízení pak aplikace vykreslí reálnou scénu, doplněnou o virtuální model hrací kostky (obr. 5.1). Virtuální kostka bude přesně doléhat na reálnou kostku. Tím vznikne dojem hracího prostředí skutečně existujícího v reálném světě (viz „magický objektiv“, 3.4). Otáčením fyzické kostky nebo změnou umístění kamery může hráč prohlížet stěny virtuální kostky z různých poloh.

Uživatel bude na stěnách, které budou viditelné, moci pomocí dotykového displeje měnit barvy políček dle pravidel popsaných výše.

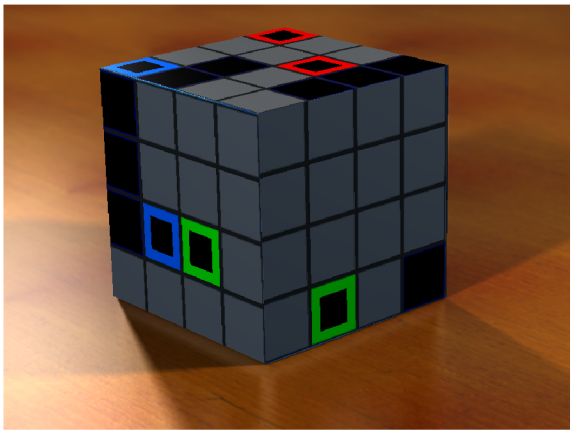
[1] http://www.2dplay.com/crazy-cube/crazy-cube-play.html (prosinec 2008)



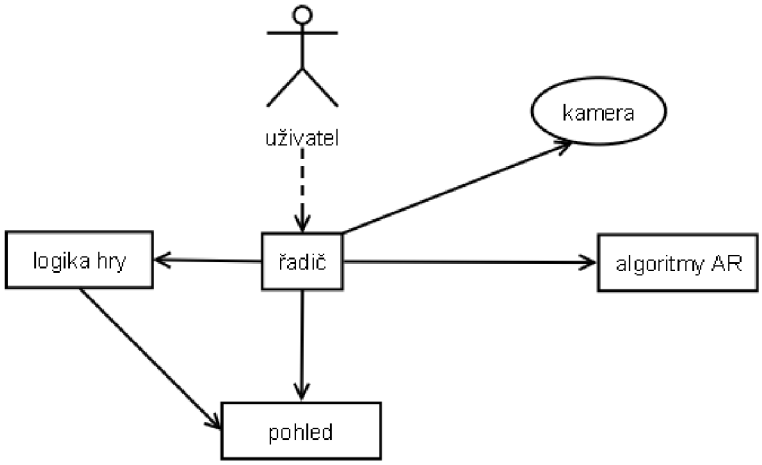
Obrázek 5.1: Příklad počáteční konfigurace a odpovídající cílové konfigurace, ve které jsou dvojice výchozích bodů propojeny cestou



Obrázek 5.2: Fyzická kostka, která má na stranách nalepené značky ARToolKitu



Obrázek 5.3: Reálná scéna doplněná o virtuální hrací kostku, která přesně překrývá fyzickou kostku



Obrázek 5.4: Schema aplikace

V AR Cube budou prezentační a aplikační logika od sebe odděleny. Aplikace se bude skládat z několika hlavních částí: řadiče, herní logiky, pohledu a algoritmů AR (viz obr. 5.4).

Pohled bude zajišťovat vykreslování hry. Veškeré renderování bude realizováno pomocí API OpenGL ES.

Výpočty spojené s AR obstará knihovna ARToolKit, resp. její port do jazyka Java NyARToolkit.

Řadič bude mít na starost koordinaci všech celků a interakci s uživatelem. Bude pořizovat náhled z kamery a po výpočtu polohy kamery vůči snímané kostce pomocí AR algoritmů ovlivní pohled tak, aby hrací krychle vykreslovaná pohledem přesně doléhala na reálnou kostku.

Logika hry bude umožňovat hrát hru podle daných pravidel – bude spouštět levely (tj. nastavovat konfigurace kostky), bude kontrolovat podmínku dokončení levelu (spojení výchozích políček). Od řadiče bude dostávat příkazy ke změnám barev políček (jako důsledek interakce s uživatelem). O změnách barvy políček bude informovat pohled, aby mohl patřičně upravit vykreslovaný 3D model.

Kapitola 6

Implementace aplikace

Tato kapitola popisuje způsob implementace demonstrační aplikace. Protože kompilace aplikace a její spuštění v emulátoru platformy Android je poměrně zdlouhavé, byl pro urychlení vývoje nejprve vytvořen prototyp aplikace pro PC. Na tomto prototypu byla ověřena funkčnost implementace hlavních částí aplikace – herní logiky, rozpoznávání stěn kostky v obraze z kamery a renderování hry pomocí OpenGL. Teprve po otestování funkčnosti na PC byla aplikace portována na platformu Android. V následujících podkapitolách je popsána struktura aplikace, princip funkce jednotlivých částí aplikace a některé problémy, které bylo nutné řešit při portování prototypu na platformu Android.

6.1 Struktura aplikace

Aplikace je rozdělena do několika balíků (viz obr. 6.1):

game Balík obsahuje třídy řešící logiku hry. V balíku jsou dále rozhraní umožňující implementujícím třídám přijímat notifikace o různých událostech spojených se hrou.

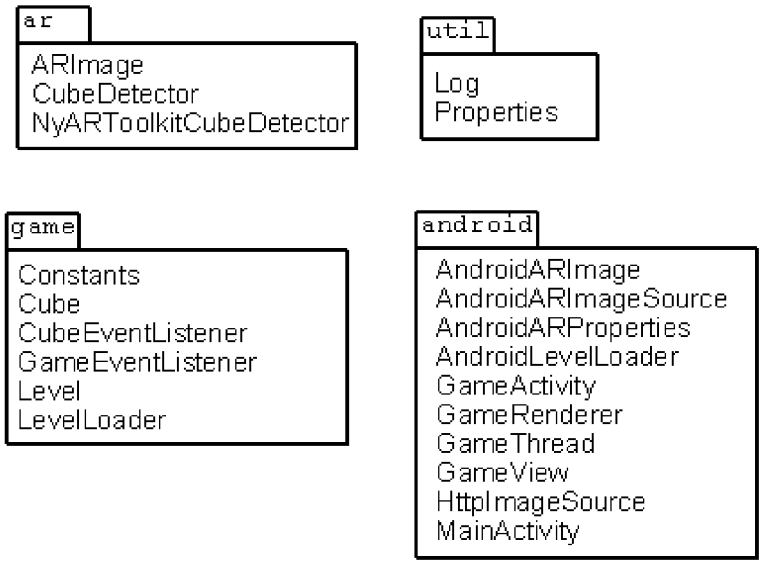
ar V tomto balíku jsou umístěny třídy zajišťující detekci stěn kostky se značkami ARToolKitu v obraze. Balík obsahuje obecné rozhraní pro detektor kostky a jeho implementaci využívající knihovnu NyARToolkit.

android Balík obsahuje hlavní třídu obstarávající běh hry, třídy zajišťující získávání snímků z kamery a třídy řešící uživatelské rozhraní – menu, renderování hry, interakci s uživatelem.

util Součástí tohoto balíku je třída pro ladící výpisy a pomocné rozhraní pro přístup ke sdíleným datům.

Aplikace byla navržena s ohledem na snadnou rozšiřitelnost a modifikovatelnost. Jednotlivé části aplikace lze snadno nahrazovat, upravovat nebo rozšiřovat bez nutnosti zásahu do ostatních částí. Logika hry, renderování hry a detektor kostky v obraze jsou od sebe odděleny. Kromě balíku android jsou všechny balíky platformově nezávislé, což usnadňuje případné portování.

Vztahy mezi třídami aplikace jsou znázorněny ve zjednodušeném diagramu tříd na obrázku 6.2.



Obrázek 6.1: Rozdělení do balíků

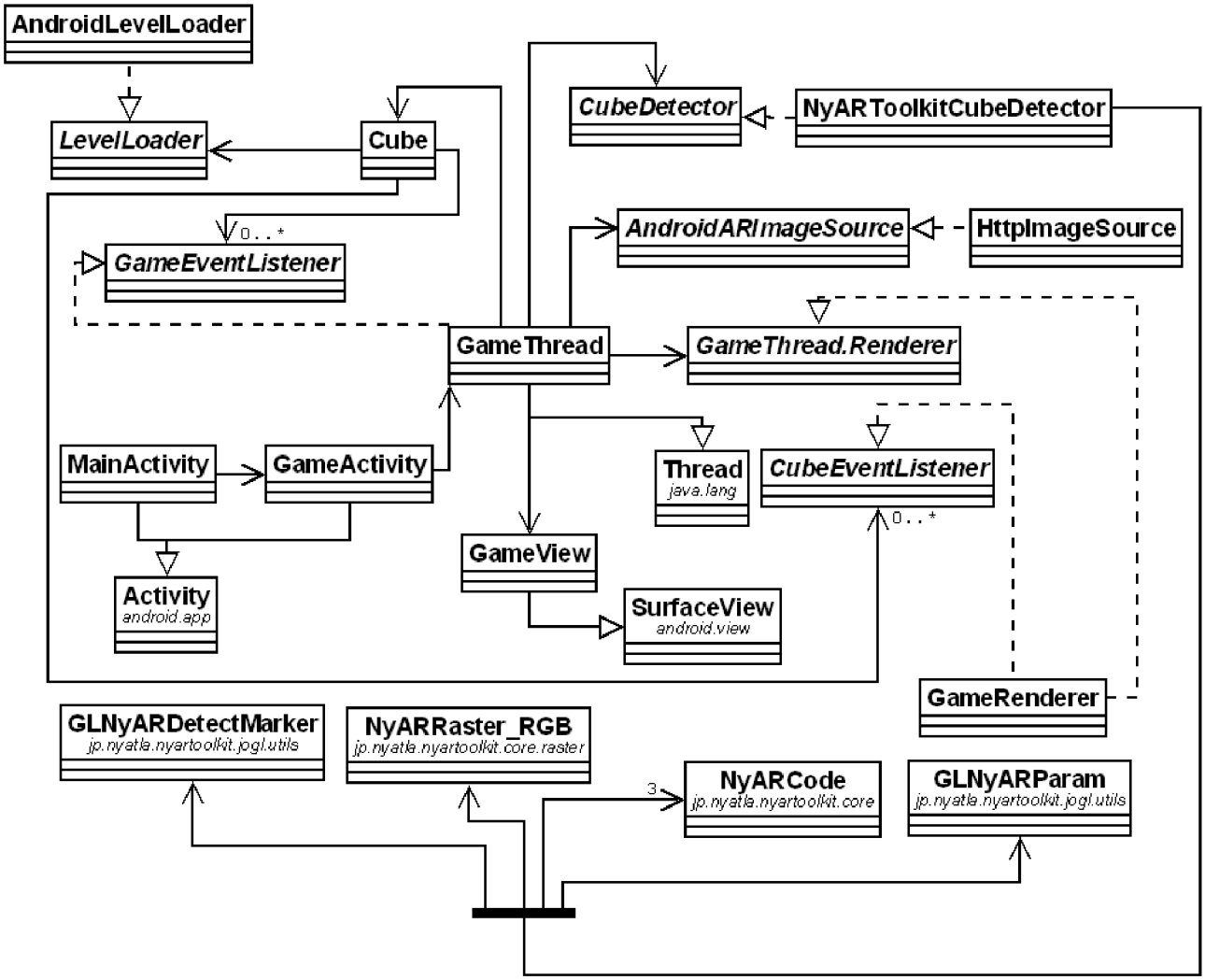
6.2 Logika hry

Logika hry je soustředěná ve třídě Cube. Tato třída umožňuje měnit barvu jednotlivých políček a testuje podmínku ukončení hry – spojení všech výchozích dvojic. Nastavování hodnot políček obstarává metoda `setPlaneData()`. Ve hře je vždy některá ze tří stěn aktivní – nastavuje se pomocí metody `setActivePlane()` – na této stěně pak lze pohybovat kurzorem pomocí metod `left()`, `right()`, `up()`, `down()` nebo přímo nastavovat pozici kurzoru pomocí metody `setPosition()`. Metoda `fire()` pak způsobí akci s políčkem, nad kterým je momentálně kurzor:

- Pokud je políčko jedním z výchozích políček, nastaví se aktuální barva na barvu tohoto políčka.
- Pokud je políčko standardní, změní se jeho barva na aktuální barvu.
- Pokud je políčko pod kurzorem speciální (tj. políčko, které není možno využít při tvorbě cesty mezi výchozími body – nelze mu měnit barvu), neprovede se při volání `fire()` žádná akce.

O každé změně barvy políčka, změně polohy kurzoru nebo změně velikosti stěny kostky (možné velikosti jsou 3x3 nebo 4x4 políčka) posílá Cube notifikace všem zaregistrovaným implementacím rozhraní CubeEventListener. V případě, že dojde ke spojení všech dvojic výchozích bodů, pošle Cube notifikaci všem registrovaným implementacím rozhraní GameEventListener (levelFinished()).

Pro spouštění levelů (neboli při nastavování konfigurací kostky) využívá třída Cube rozhraní LevelLoader, jehož implementace řeší (potenciálně platformově závislé) načítání konfigurací kostky ze souboru nebo sítě apod. V demonstrační aplikaci toto rozhraní implementuje třída AndroidLevelLoader, která jednotlivé úrovně načítá z textového souboru, uloženého v resources. LevelLoader poskytuje dvě metody: `getNumLevels()`, která vrací celkový počet levelů a `loadLevel()`, která umožňuje načíst jednotlivé levely. Načtení levelu



Obrázek 6.2: Zjednodušený diagram tříd aplikace AR Cube

se provádí po zavolání metody `startLevel()` třídy Cube. Po načtení se zavolá metoda `levelStarted()` všech zaregistrovaných implementacím rozhraní GameEventListener.

6.3 Sledování pozice a orientace kamery

6.3.1 Získání snímku z kamery

V současné době emulátor platformy Android neumožňuje emulaci kamery mobilního zařízení pomocí kamery připojené k PC – emulátor kameru mobilního zařízení pouze simuluje pomocí animace šachovnicového vzoru (viz obr). Z tohoto důvodu bylo nutné implementovat získávání snímků z kamery nepřímo – pomocí HTTP připojení k webovému serveru webkamery. Tento úkol obstarává třída HttpImageSource implementující rozhraní AndroidARImageSource. HttpImageSource umožňuje specifikovat v konstruktoru adresu a port webového serveru kamery. Hlavní metoda této třídy je getAndroidARImage(). Volání této metody způsobí vytvoření spojení s daným serverem a načtení obrázku ze serveru. Výstupem metody je získaný obrázek v podobě instance třídy AndroidARImage, která implementuje rozhraní ARImage. ARImage je obecné rozhraní, jež reprezentuje obrázky, u kterých lze získat RGB hodnoty jejich pixelů. Toto rozhraní je využíváno implementacemi třídy CubeDetector. Tato struktura umožňuje při portování pouze napsat implementaci ARImage tak, aby fungovala na cílové platformě, bez nutnosti změn v ostatních třídách. Stějně tak je možné napsat jinou implementaci AndroidARImageSource – například takovou, která by obrázek získavala přímo z kamery (pro použití na fyzickém zařízení).

6.3.2 Zpracování snímku

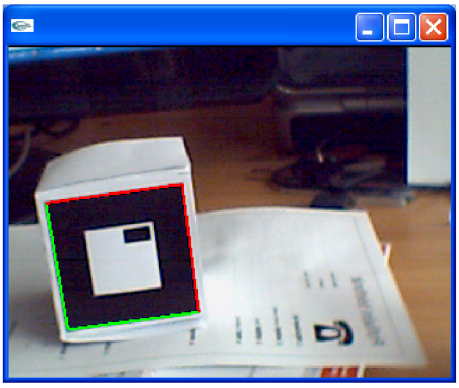
Při zpracovávání snímku z kamery je použita knihovna NyARToolkit. NyARToolkit je port knihovny ARToolKit (viz kapitolu) do několika různých jazyků (Java, C#, ActionScript, C++). Je dostupný na adrese http://nyatla.jp/NyARToolkit/wiki/index.php (květen 2009). NyARToolkit pro Javu nepoužívá žadné nativní knihovny, je proto vhodný i pro použití na platformě Android. Poměrně velkou nevýhodou NyARToolkitu je neexistence dokumentace v jiném jazyce než japonštině.

Trénování značek

Aby bylo možné detekci značek pomocí ARToolKitu, resp. NyARToolkitu používat, je nejprve nutné natrénovat rozpoznávání na reálných značkách, které budou využity v aplikaci. K tomu slouží utilita ARToolkitu mk_patt. Při procesu trénování uživatel pomocí této utility udělá kamerou snímek trénované značky. Značka musí být snímána ve správné poloze (ta je indikována červenými a zelenými linkami, viz obrázek 6.3). Aby byla zaručena co nejvyšší úspěšnost rozpoznávání, mělo by trénování probíhat v podobných podmínkách (osvětlení), v jakých později bude používána aplikace. Výstupem aplikace je datový soubor obsahující vzor uvnitř značky v různých natočeních (0°, 90°, 180°, 270°). Data z tohoto souboru jsou pak používána knihovnou NyARToolkit ve fázi identifikace značky.

Kalibrace kamery

Pro dosažení co nejlepších výsledků při hledání značek v obraze a výpočtu polohy kamery vůči značce je vhodné kameru kalibrovat. Pro demonstrační aplikaci byla provedena kalibrace kamery, se kterou byla aplikace testována (webová kamera Trust WB-1400T), pomocí



Obrázek 6.3: Utilita mk_patt

utility ARToolKitu calib_dist, která využívá pro vypočet parametrů kamery snímaní kalibračního vzoru. Výstupem je datový soubor obsahující zjištěné parametry kamery, které jsou v AR Cube využívány při detekci značek i při renderování.

Detekce značek, výpočet transformace

Detekce probíhá ve třídě NyARToolkitCubeDetector, která vyhledává v obraze stěny kostky označené značkami ARToolKitu. Třída pracuje následujícím způsobem. Nejprve načte z datového souboru parametry kamery získané kalibrací do instance GLNyARParam, poté načte vzory tří značek z datových souborů získaných při trénování do instancí třídy NyARCode. Nejdůležitější třída využívaná při hledání značek ARToolKitu je GLNyARDetectMarker. Tato třída umožňuje po nastavení prahu thresholdingu a předání parametrů kamery a vzorů značek hledaných v aplikaci provést zpracování obrazu, při kterém dochází k hlavním výpočtům spojeným s alogritmy AR. Výstupem volání metod této třídy jsou nalezené značky v obraze. O každé z nalezených značek jsou uchovány následující informace – index rozpoznaného vzoru (tj. o kterou z možných značek se jedná), confidence – míra věrohodnosti (tj. do jaké míry byl vzor z nalezené značky ztotožněn s jedním z hledaných vzorů) a modelview matice OpenGL, vyjadřující pozici a orientaci objektu vůči kameře. Pokud je v obraze zároveň vidět více značek (tzn. kostka je snímána z úhlu, ve kterém je vidět několik jejích stěn naráz), vybere NyARToolkitCubeDetector značku, která má nejvyšší confidence. Výstupem zpracování obrázku třídou NyARToolkitCubeDetector je (v případě úspěšného nalezení) číslo jedné z nalezených stěn a příslušná modelview matice. Třída NyARToolkitCubeDetector poskytuje dvě veřejné metody: processImage() provádějící zpracování obrazu a getModelviewMatrix(), která vrací číslo nalezené stěny a příslušnou OpenGL modelview matici.

6.4 Renderování hry

O renderování hrací kostky se stará třída GameRenderer implementující rozhraní CubeEventListener. Třída GameRenderer přijímá od třídy Cube notifikace o změnách barvy políček, změně velikosti kostky (např. při přechodu do dalšího levelu) a těmto změnám na hrací kostce přizpůsobuje vykreslování.

6.4.1 OpenGL ES

Renderování scény se ve hře provádí prostřednictvím OpenGL ES (OpenGL for Embedded Systems). Jedná se o multiplatformní API pro 2D a 3D grafiku na vestavěných systémech. OpenGL ES vzniklo jako podmnožina desktopového OpenGL a tvoří nízkoúrovňové rozhraní mezi softwarem a grafickou akcelerací. OpenGL ES vychází z klasického OpenGL, ale je přizpůsobeno charakteristickým vlastnostem mobilních zařízení – relativně málo paměti, nízký výpočetní výkon, nutnost malé spotřeby energie, nevhodnost floating point datových typů atd.

OpenGL ES API vs. OpenGL API

Jedním z hlavních rozdílů OpenGL ES oproti OpenGL je absence immediate módu – v OpenGL ES neexistují instrukce `glBegin`, `glEnd`, `glVertex`, `glTexCoord`. Vykreslování objektů tedy musí být prováděno pomocí polí, ve kterých jsou definovány vertexy, texturovací souřadnice, barvy, normály (instrukce `glDrawElements`, `glDrawArrays`, `glVertexPointer`, `glNormalPointer`, `glColorPointer`, `glTexCoordPointer`). Z OpenGL ES jsou dále vynechány některé instrukce pro práci s pixely a bitmapami (`glDrawPixels`, `glCopyPixels`, `glPixelZoom`, `glBitmap`, ...), je vynechán OpenGL Imaging Subset (histogramy, filtry, minmax). OpenGL ES neumožňuje vykreslovat polygony, čtyřúhelníky (quads) a pásy čtyřúhelníků (quad strips). Z transformačních funkcí nepodporuje OpenGL ES `glLoadTransposeMatrix`, `glMultTransposeMatrix`, `glTexGen`, `glGetTexGen`. Chybí také evaluátory, display-listy, selection buffer. OpenGL ES nepodporuje 1D a 3D textury a cube mapy a další funkce OpenGL, které jsou pro mobilní zařízení zbytečné nebo nepoužitelné. Do OpenGL ES byla oproti OpenGL přidána podpora pro celočíselné datové typy. [6]

EGL

EGL (Embedded System Graphics Library) je rozhraní mezi vykreslovacími API jako např. OpenGL ES a okenním systémem nativní platformy. EGL poskytuje mechanismy pro vytváření renderovacích povrchů (okna, případně pbuffery, pixmapy), na které mohou klientská API kreslit a která mohou sdílet. EGL poskytuje metody pro řízení grafických kontextů. V Androidu (od verze 0.9) nelze vykreslovat pomocí OpenGL bez použití EGL pro inicializaci grafického kontextu a povrchu pro vykreslování.

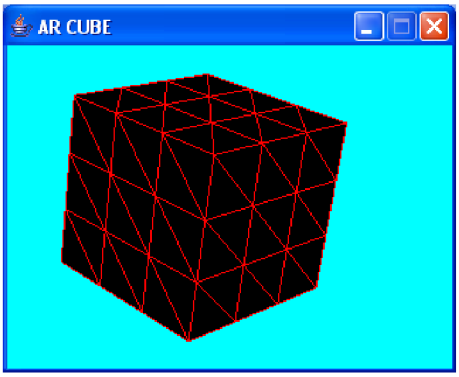
6.4.2 Implementace

Vykreslování

Při inicializaci GameRendereru se nastaví projekční matice, která definuje těleso záběru kamery, a to na hodnotu, jež byla vypočítána třídou GLNyARParam z parametrů kamery načtených z datového souboru, vzniklého kalibrací.

Vzhledem k tomu, že – jak již bylo řečeno – OpenGL ES nepodporuje immediate mód, bylo pro reprezentaci kostky použito několik bufferů:

Buffer souřadnic vertexů Obsahuje souřadnice vrcholů tvořících model hrací kostky. Souřadnice jsou nastaveny při inicializaci hry a mění se pouze při změně počtu políček na stěnách kostky. Model krychle je složen z trojúhelníků, jak je vidět na obrázku 6.4. Čtyřúhelníky nejsou v OpenGL ES podporovány.



Obrázek 6.4: Model kostky

Obrázek 6.5: Model kostky s texturou

Buffer texturovacích souřadnic Texturovací souřadnice určují způsob namapování textury na model hrací kostky. Ve hře je použita jediná textura, která obsahuje grafiku políček všech typů v různých barvách. Texturovací souřadnice se v průběhu hry mění.

Buffer souřadnic barev Barevné souřadnice jsou při inicializaci hry nastaveny na hodnoty, podle kterých pak lze jednotlivá políčka identifikovat – používají se při pickingu.

Řadič po detekci kostky předá GameRendereru číslo nalezené stěny a modelview matici, která vyjadřuje polohu této stěny vůči kameře pomocí metody `setModelviewMatrix()`. vykreslování modelu kostky s použitím bufferů zmíněných výše pak probíhá v metodě `renderToBuffer()`.

Picking

Ve 3D aplikacích je někdy nezbytné zjistit, který objekt se při kliknutí na obrazovku vyskytuje pod kurzorem. Tomuto procesu zjišťování objektu scény na daných souřadnicích se říká picking. V demonstrační AR aplikaci se picking využívá ke zjišťování políčka na krychli, na které uživatel poklepal na dotykovém displeji. V AR Cube je za tímto účelem použita tzv. color picking. Tato metoda obecně pracuje na následujícím principu: každému objektu (resp. různým částem jednoho objektu) se přiřadí jednoznačná barva. V okamžiku nutnosti zjištění objektu na určitých souřadnicích se scéna vykreslí s vypnutým osvětlením, texturováním

a mlhou, přičemž každý objekt (resp. část objektu) je vykreslen barvou, která mu byla přiřazena. Poté se pomocí funkce OpenGL `glReadPixels` přečte barva pixelu vykresleného na daných souřadnicích. Tato barva identifikuje objekt nebo jeho část. Pro uživatele není vykreslení v módu pro picking nikdy viditelné.

Na obrázku 6.6 vpravo je ukázka kostky vykreslené v módu pro picking, kde každé políčko je jednoznačně identifikované svou barvou. Na témže obrázku vlevo stejná scéna vykreslená tak, jak ji vidí hráč – s texturou a pozadím.

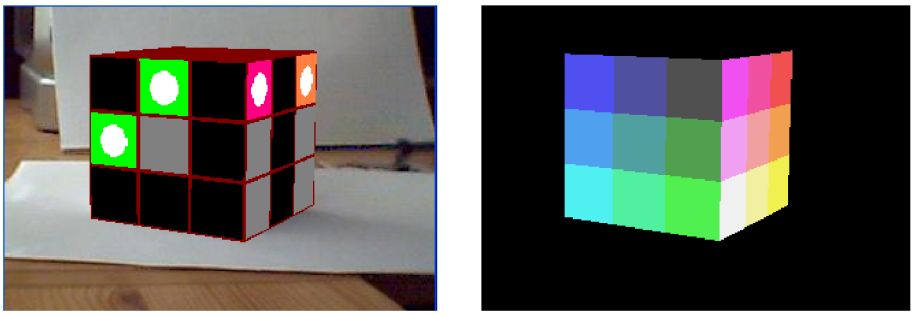
Portování na Android

Prototyp aplikace využíval Java OpenGL (JOGL) – jedná se o knihovnu, která umožňuje použití OpenGL v jazyce Java. Pro usnadnění pozdějšího portování byl prototyp psán bez použití instrukcí, které nejsou podporovány v OpenGL ES (jedinou výjimku tvořila instrukce `glDrawPixels`). Při portování renderovací části aplikace na platformu Android se vyskytlo několik problémů.

Nejprve bylo nutné přepsat některé části renderovací třídy aplikace tak, aby využívaly EGL – například vytvoření povrchu pro vykreslování, nastavení parametrů (barevná hloubka, hloubka depth bufferu, ...).

Ukázalo se, že Android neumožňuje volat metody OpenGL z různých vláken. Kvůli tomu musela být přepracována struktura aplikace takovým způsobem, aby se inicializace GameRendereru, vykreslování a picking prováděly ze stejného vlákna.

Dalším problémem bylo vykreslování pozadí – náhledu z kamery. Jak bylo zmíněno výše, OpenGL ES API neobsahuje funkci glDrawPixels, která byla použita pro vykreslování pozadí v protoypu aplikace. Původním záměrem bylo použít k vykreslování snímků z kamery standardní 2D API Androidu. Android snadné kombinování 2D a 3D grafiky umožňoval, ale bohužel pouze do verze 0.9, kdy byly v API Androidu provedeny poměrně velké změny týkající se OpenGL ES. Od verze 0.9 je jedinou komponentou, na kterou lze vykreslovat pomocí OpenGL ES, SurfaceView a nelze při tom kombinovat 2D a 3D volání. Proto byly další snahy namířeny na zprůhlednění komponenty SurfaceView, na které bylo prováděno vykreslování scény. Různé komponenty uživatelského rozhraní (Views) lze v Androidu vrstvit na sebe a pomocí částečně průhledných komponent tak lze například zobrazit 2D ovládací prvky nad 3D scénou. Bohužel se ukázalo, že není možné SurfaceView zprůhlednit tak, aby byla vidět komponenta pod ní (na kterou by se vykresloval náhled z kamery). Komponenta SurfaceView funguje jiným způsobem než ostatní Views – z jistých důvodů lze umisťovat Views viditelně nad její povrch, ale nemůže být vidět nic pod ní. Jedním z možných řešení tohoto problému by bylo pixely vykreslované scény zpětně získávat pomocí `glReadPixels()` a tyto pixely pak spolu s pozadím vykreslovat na jiné komponentě pomocí 2D API. Tato varianta by ale byla zbytečně neefektivní. Konečné řešení tohoto problému spočívá ve vykreslování 3D scény do nativní bitmapy platformy Android pomocí speciálního typu EGL povrchu. 3D scéna se vykreslí do bitmapy s průhledným pozadím a tato bitmapa se pak vykreslí nad náhledem z kamery. Toto relativně elegantní řešení nebylo bohužel zřejmé, protože starší verze Androidu EGL povrchy umožňující vykreslování do nativních bitmap nepodporovaly (volání příslušné EGL funkce nemělo efekt) a u novějších verzí (1.0 a výš) implementace tohoto typu EGL povrchu sice existuje, ale dokumentace se o této skutečnosti nikde nezmiňuje.



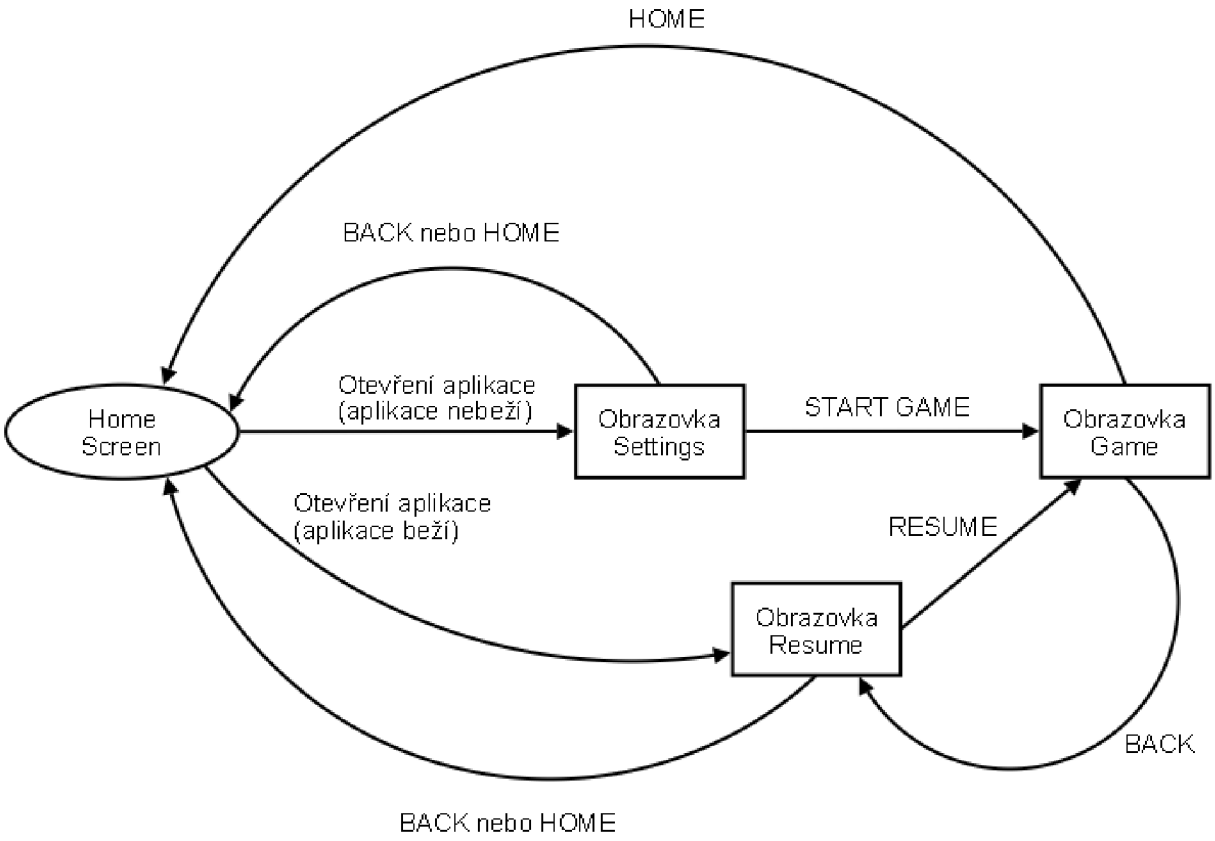
Obrázek 6.6: Scéna vykreslená v normálním módu a v módu pro picking

6.5 Řadič

GameThread je třída reprezentující řadič ze schématu 5.4. Instance této třídy koordinuje běh hlavních částí aplikace. Hlavní smyčka třídy GameThread běží ve vlastním vlákně a provádí následující operace:

1. Získání obrázku z kamery pomocí AndroidARImageSource.
2. Detekce stěny kostky pomocí CubeDetector.
3. Nastavení příslušné modelview transformace u Rendereru.
4. Zpracování případných událostí od dotykového displeje – pomocí Rendereru zjištění místa na kostce, na které uživatel poklepal a případná změna barvy políčka voláním metod třídy Cube.
5. Vykreslení pozadí.
6. Vyrenderování kostky pomocí Rendereru do bufferu a vykreslení bufferu na pozadí.

GameThread implementuje rozhraní GameEventListener – od třídy Cube dostává notifikace o spuštění a ukončení levelů. Tyto notifikace využívá k zobrazování informačních textů (pomocí GameActivity) a spouštění následujícího levelu po dohrání aktuálního. Pro rendering využívá povrch (surface) z GameView. Od GameView získává informace o událostech na dotykovém displeji. Zpracování událostí od dotykového displeje se provádí v hlavním cyklu, protože pro zjišťování místa, které na kostce uživatel vybral pomocí dotykového displeje, Renderer využívá volání OpenGL ES a v Androidu – jak bylo již řečeno – je nutné provádět všechna volání OpenGL ze stejného vlákna.



Obrázek 6.7: Obrazovky aplikace a přechody mezi nimi

6.6 Uživatelské rozhraní

Na obrázku 6.7 je schema hlavních obrazovek a přechodů mezi nimi.

Obrazovka Home je systémová obrazovka, ze které uživatel spouští aplikace. Do této obrazovky se uživatel může kdykoliv dostat stisknutím tlačítka HOME na zařízení. Po spuštění aplikace se zobrazí obrazovka Settings. Tuto obrazovku reprezentuje třída MainActivity – jedná se o vstupní bod aplikace. Zobrazí menu umožňující nastavení adresy webového serveru kamery, ke kterému se má hra připojit, a rozlišení snímků z kamery. Na obrázku B.1 je vidět rozvržení úvodní obrazovky[1]. Po kliknutí na tlačítko START GAME se spustí inicializace hry – v průběhu inicializace je na obrazovce zobrazen Progress Dialog. Během načítání může uživatel provést návrat do obrazovky Home pomocí tlačítka HOME – načítání hry pak bude probíhat na pozadí a o jeho ukončení bude uživatel informován pomocí Toast (viz dále).

Po inicializaci se zobrazí hlavní obrazovka – obrazovka Game viz obr. B.2). Tuto obrazovku představuje třída GameActivity. GameActivity vytváří instanci GameView pro vykreslování hry a kontextové menu pro restart a přeskakování levelů (viz obr. B.3). Stisknutím tlačítka BACK na této obrazovce způsobí zobrazení obrazovky Resume.

Obrazovka Resume je stejně jako obrazovka Settings zajišťována třídou MainActivity. Nabízí možnost pokračování v pozastavené hře (tlačítko RESUME GAME). Stisk tlačítka

[1]Rozvržení obrazovky je specifikováno prostřednictvím XML souboru (viz 2.5).

BACK na této obrazovce ukončí aplikaci. Do obrazovky Resume se uživatel dostane také v případě, že aplikaci znovu spustil z obrazovky Home po přerušení hry tlačítkem HOME.

Pro krátké informační zprávy, jako např. informace o ukončení levelu nebo informace o chybě při získávání obrázku z webového serveru kamery, používá AR Cube třídu Toast. Toast je speciální View určené právě pro takovéto zprávy. Toast se vždy zobrazí na určitou dobu nad aktuálním oknem a nikdy nezíská fokus (tzn. uživatel může dál pokračovat v právě prováděné akci, zobrazený Toast ho nepřeruší). Ukázka Toast je na obrázku B.3.

Kapitola 7

Testování

Testování probíhalo průběžně po celou dobu vývoje a jeho výsledky se odrážely do úprav v implementaci. Renderování, herní logika a algoritmy AR byly testovány z větší části na prototypu aplikace na PC. Teprve po důkladném otestování bylo provedeno portování aplikace na platformu Android. Testování naportované aplikace probíhalo pouze v emulátoru – fyzické zařízení s operačním systémem Android nebylo k dispozici.

7.1 Sledování pozice a orientace kamery

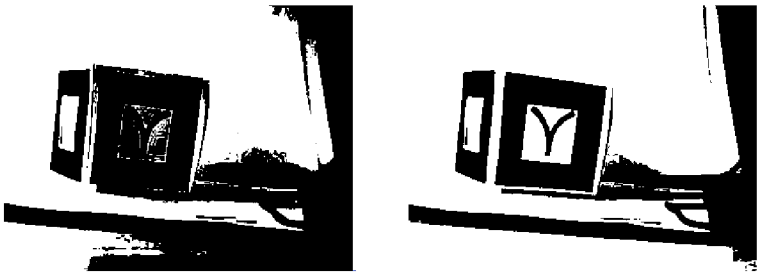
Testování části aplikace, která obstarává detekci kostky v obraze a výpočet polohy kamery vůči nim, bylo zaměřeno na experimentování s volbou značek, osvětlením, rozlišením vstupního obrazu atd.

7.1.1 Výběr značek

Schopnost sledovat značku pomocí ARToolKitu je ovlivněna mimo jiné složitostí značky. Dokumentace doporučuje používat nízkofrekvenční vzory (velké bílé a černé plochy). Toto doporučení vyplývá ze způsobu fungování algoritmů ARToolKitu. Při rozpoznávání značek je vzor zaznamenán na relativně malém počtu pixelů, a proto pokud by se značky lišily pouze ve vysokofrekvenční detailech, bylo by pro ARToolKit (resp. NyARToolkit) nemožné je identifikovat. Z těchto důvodů byly vybrány značky s relativně jednoduchými vzory a s nízkou mírou vzájemné podobnosti. Použité značky jsou vidět na šabloně hrací kostky (obr. A.1). Velikost hrany používané kostky je 7 cm.

7.1.2 Osvětlení, pozadí, rozlišení

Jedním z parametrů, které měly při testování nejvetší vliv na správné sledování polohy kamery, bylo osvětlení. Z experimentů s osvětlením vyplývá, že pro efektivní detekci značek ARToolKitu je dobré vyhnout se silným umělým zdrojům světla – tyto zdroje mohou způsobovat nerovnoměrné osvětlení a odlesky. Jelikož ARToolKit poskytuje pouze nejjednodušší metodu prahování – používá konstantí hodnotu prahu pro celý obraz –, je poměrně náchylný na rovnoměrnost osvětlení. Na obrázku 7.1 jsou ukázky snímků z kamery po prahování: snímek vlevo byl pořízen s použitím nevhodně nasměrovaného umělého zdroje světla. Vpravo je ukázka vyprahovaného snímku stejné scény pořízeného při rovnoměrném osvětlení.



Obrázek 7.1: Vliv osvětlení na výsledek prahování

Při testování byly prováděny experimenty s rozpoznáváním značek na kostce na různých pozadích. Ukázalo se, že pozadí, na kterém je kostka snímána, nemá téměř žádný vliv na detekci značek. Značky na stěnách používané kostky mají kolem sebe bílý okraj o šířce 0,5 cm, což zajišťuje po vyprahování dostatečné oddělení značky v obraze od okolí a snadnou detekci i na členitém pozadí.

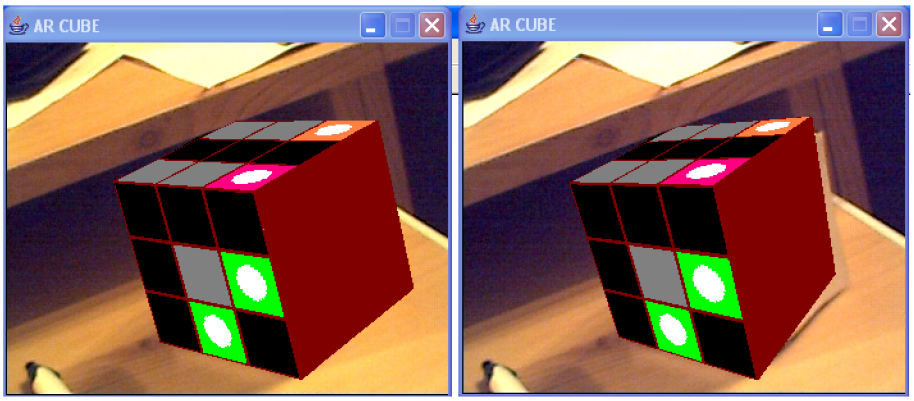
Dalším parametrem ovlivňujícím detekci značek je použité rozlišení kamery. Kamera, se kterou byla aplikace testována, umožňuje používat rozlišení 160 × 120, 176 × 144, 320 × 240, 352 × 288 a 640 × 480 (interpolované). Při testování byla funkčnost aplikace vyzkoušena ve všech těchto rozlišeních kromě interpolovaného. Výsledek testování byl ve shodě s očekáváním – vyšší rozlišení zlepšuje přesnost detekce a zvyšuje vzdálenost, na kterou lze značky rozpoznávat. Zvýšení rozlišení se ale bohužel nepříznivě projeví ve zpomalení aplikace (viz dále).

7.1.3 Vliv kalibrace kamery

Správná kalibrace zvyšuje dosah detekce značek a zpřesňuje výpočet polohy kamery vůči značkám. Parametry kamery se navíc využívají při renderování pro nastavení projekční matice. Aplikace byla nejprve testována s přednastavenými parametry kamery, které jsou součástí balíku ARToolKit. Po ověření základní funkčnosti detekce byla provedena jednokroková kalibrace používané kamery pomocí calib_dist. Podle očekávání kalibrace zlepšila přesnost vykreslování virtuální kostky na fyzické. Na obrázku 7.2 je vidět, jak vypadá scéna vzniklá s použitím správných kalibračních parametrů, vpravo je stejná scéna vypočítaná a vykreslená s použitím přednastavených parametrů. Všimněte si, že při použití přednastavených parametrů model kostky nepřekrývá na snímku celou fyzickou kostku.

7.1.4 Rychlost

Rychlost zpracování vstupu z kamery je pro běh aplikace klíčová. Bohužel se ukázalo, že v emulátoru Androidu na použitém PC (1.8 GHz, 512 MB RAM) je rychlost zpracování snímků nedostatečná. Hra je v emulátoru sice hratelná, ale dochází při ní k nepříjemným prodlevám a ani při použití nejmenšího rozlišení není zdaleka možné dosáhnout interaktivity v reálném čase, tedy jednoho z hlavních atributů aplikací rozšířené reality. Prototyp běžící na stejném PC mimo emulátor naopak dosahoval při zpracování snímků poměrně dobré rychlosti – při použití maximálního rozlišení bylo možné dosáhnout zpracování přibližně 80 snímků za sekundu. Jak je vidět v tabulce 7.1.4, zpracovávání jednoho snímku v aplikaci AR Cube běžící v emulátoru probíhá až 150x pomaleji než v prototypu aplikace na PC. Hlavní příčiny nedostatečné rychlosti aplikace spuštěné v emulátoru jsou následující:



Obrázek 7.2: Vliv správné kalibrace kamery na přesnost kombinování virtuálního a reálného prostředí

Rozlišení	PC [ms]	emulátor Androidu [ms]
160 x 120	3	220
176 x 144	4	450
320 x 240	10	1400
352 x 288	12	1800

Tabulka 7.1: Porovnání průměrné doby zpracování jednoho snímku v různých rozlišeních na PC a v emulátoru Androidu

- Celý systém beží na emulované ARM architektuře[1] – softwarově je emulován ARM procesor, správa paměti a další funkce. Tento způsob fungování emulátoru klade vysoké nároky jak na paměť, tak na výkon procesoru hostitelského PC. Na počítači používaném při testování byl běh emulátoru velmi pomalý a ani jednoduché aplikace v něm spuštěné většinou nefungovaly příliš reaktivně.
- Problémem je také příliš malý výkon virtuálního stroje Dalvik. V současné době Dalvik nepoužívá JIT[2] a další běžné optimializace, což se velmi negativně odráží v jeho výkonu. [7]
- Knihovna NyARToolkit silně využívá floating point aritmetiku, což aplikaci zpomaluje. V emulátoru Androidu jsou operace s racionálními datovými typy zhruba dvakrát pomalejší než operace s celočíselnými datovými typy. (V případě virtuálního stroje s JIT nebo hardwarové akcelerace floating point by byl rozdíl v rychlostech markantnější.) Operace s racionálními datovými typy nejsou obecně při vývoji pro mobilní zařízení doporučována (především kvůli absenci matematického koprocesoru na většině takovýchto zařízení).

[1] ARM je RISC architektura používaná především v mobilních zařízeních

[2] Just-In-Time kompilace – technika, při které je bytecode za běhu kompilován do nativního strojového kódu

7.2 Testy specifické pro platformu Android

Aplikace běžící na mobilních zařízeních mohou být ve svém běhu přerušeny externími událostmi – příchozím hovorem nebo příchozí SMS. Emulátor umožňuje testovat chování aplikace v těchto situacích následujícím způsobem:

1. Pomocí telnetu se připojíme ke konzoli emulátoru:

   ```
   telnet localhost 5554
   ```

2. Příchozí hovor pak vyvoláme přikazem odeslaným pomocí telnetu:

   ```
   gsm call (telefonní číslo)
   ```

   Příchozí SMS umožňuje simulovat příkaz:

   ```
   sms send (telefonní číslo) (text zprávy)
   ```

Příchozí volání i SMS byly simulovány v každé fázi běhu aplikace (úvodní obrazovka, načítání, hlavní obrazovka). Aplikace fungovala správně – v případě příchozího hovoru se hlavní vlákno aplikace pozastaví, po jeho ukončení se opět spustí. Na příchozí SMS aplikace nijak nereaguje – platforma Android na SMS upozorňuje notifikací v horní části obrazovky, na běh aplikace tato událost nemá vliv.

Kapitola 8

Závěr

Hlavním cílem této práce bylo navrhnout a implementovat aplikaci demonstrující možnosti mobilní rozšířené reality na platformě Google Android. Tento úkol byl splněn. Implementovaná aplikace umožňuje hrát na mobilním zařízení logickou hru s použitím průhledového způsobu interakce – na displeji se kombinuje okolní realita snímaná kamerou a virtuální hrací prostředí. Uživatel hru ovládá s využitím reálné kostky: může ji pozorovat z různých úhlů, fyzicky s ní manipulovat apod. Na displeji zařízení se pak tato kostka překresluje virtuální kostkou, na níž probíhá samotná hra. Aplikace tak demonstruje hlavní principy rozšířené reality a ilustruje vhodnost mobilních zařízení coby levné a přenositelné hardwarové platformy pro AR aplikace.

Bohužel běh implementované aplikace na počítači, na kterém probíhalo testování, je v emulátoru velmi pomalý – v některých situacích až 150krát pomalejší, než běh stejného programu spuštěného přímo na počítači. To je způsobeno jednak tím, že emulace platformy je sama o sobě velmi náročná, a jednak tím, že použitá knihovna NyARToolkit není optimalizovaná pro mobilní zařízení. Na fyzickém zařízení aplikace testována nebyla.

Existuje mnoho možných úprav a rozšíření stávající aplikace. Aplikace poskytuje poměrně velký prostor pro zlepšování výkonu – případné pokračování práce by mělo být zaměřeno především na optimalizaci implementace algoritmů spojených s AR výpočty tak, aby bylo jejich použití na cílové platformě efektivnější. Bylo by například možné upravit knihovnu NyARToolkit tak, aby pracovala pouze s celočíselnou aritmetikou. Dalo by se také experimentovat s použitím jiných knihoven pro práci s rozšířenou realitou, potažmo s principiálně jiným způsobem sledování pozice a orientace kamery. Společnost Google oznámila, že se pro platformu Android připravuje podpora volání nativního kódu, nebude tedy nutné omezovat se na knihovny napsané v jazyce Java. Použití nativních knihoven spolu s připravovanou podporou Just-In-Time kompilace ve virtuálním stroji by mohlo přinést velmi významné zrychlení běhu aplikace. Dále je možné upravovat hru samotnou – aplikace by mohla být rozšířena o detaily zlepšující herní zážitek jako např. měření času potřebného k dohrání jednotlivých úrovní, odesílání skóre na internet, stahování nových úrovní z internetu, použití zvukových efektů a hudby apod.

Literatura

[1] *ARToolKit Homepage*, HIT Lab, University of Washington, 2008, dokument dostupný na URL http://www.hitl.washington.edu/artoolkit/ (prosinec 2008)

[2] *Augmented reality*, Wikipedia, the free encyklopedia, 2008, dokument dostupný na URL http://en.wikipedia.org/wiki/Augmented_reality (prosinec 2008)

[3] *Open Handset Alliance*, Wikipedia, the free encyklopedia, 2009, dokument dostupný na URL http://en.wikipedia.org/wiki/Open_Handset_Alliance (květen 2009)

[4] *ARQuake*, Wikipedia, the free encyklopedia, 2008, dokument dostupný na URL http://en.wikipedia.org/wiki/ARQuake (prosinec 2008)

[5] *OpenGL ES*, Wikipedia, the free encyklopedia, 2008, dokument dostupný na URL http://en.wikipedia.org/wiki/OpenGL_ES (květen 2009)

[6] Astle, D.: *OpenGL ES Walktrough* http://www.khronos.org/developers/library/sf_2005/opengles_general//Qualcomm_OpenGL-ES-Walkthrough.ppt

[7] Albayrak, S., Batyuk, L., Camtepe, A., Schmidt, A., Schmidt, H.: *Developing and Benchmarking Native Linux Applications on Android*, dokument dostupný na URL http://www.mobilware.org/2009/presentations/mobilware09-LeonidBatyuk.pdf (květen 2009)

[8] Azuma, R. T.: *A Survey of Augmented Reality*, dokument dostupný na URL http://www.cs.unc.edu/~azuma/ARpresence.pdf (prosinec 2009)

[9] Beran, V.: *Augmented Reality in Videoconference System* [diplomová práce]. Vysoké učení technické v Brně, Fakulta informačních technologií

[10] *Developing Android Applications*, Google, 2008, dokument dostupný na URL http://code.google.com/intl/cs/android/devel/index.html (prosinec 2008)

[11] Hirzer, M.: *Marker Detection for Augmented Reality Applications*, dokument dostupný na URL http://studierstube.icg.tu-graz.ac.at/thesis/marker_detection.pdf (prosinec 2009)

[12] Kato, H., Billinghurst, M.: *Marker Tracking and HMD Calibration for a Video-based Augmented Reality Conferencing System*, dokument dostupný na URL http://www.hitl.washington.edu/artoolkit/Papers/IWAR99.kato.pdf (prosinec 2009)

[13] Katysovas, T.: *A first look at Google Android*, dokument dostupný na URL http://www.kandroid.org/s2/doc/android.pdf (prosinec 2008)

[14] Láník, A.: *Detekce výrobků na dopravníkovém pásu* [diplomová práce]. Vysoké učení technické v Brně, Fakulta informačních technologií

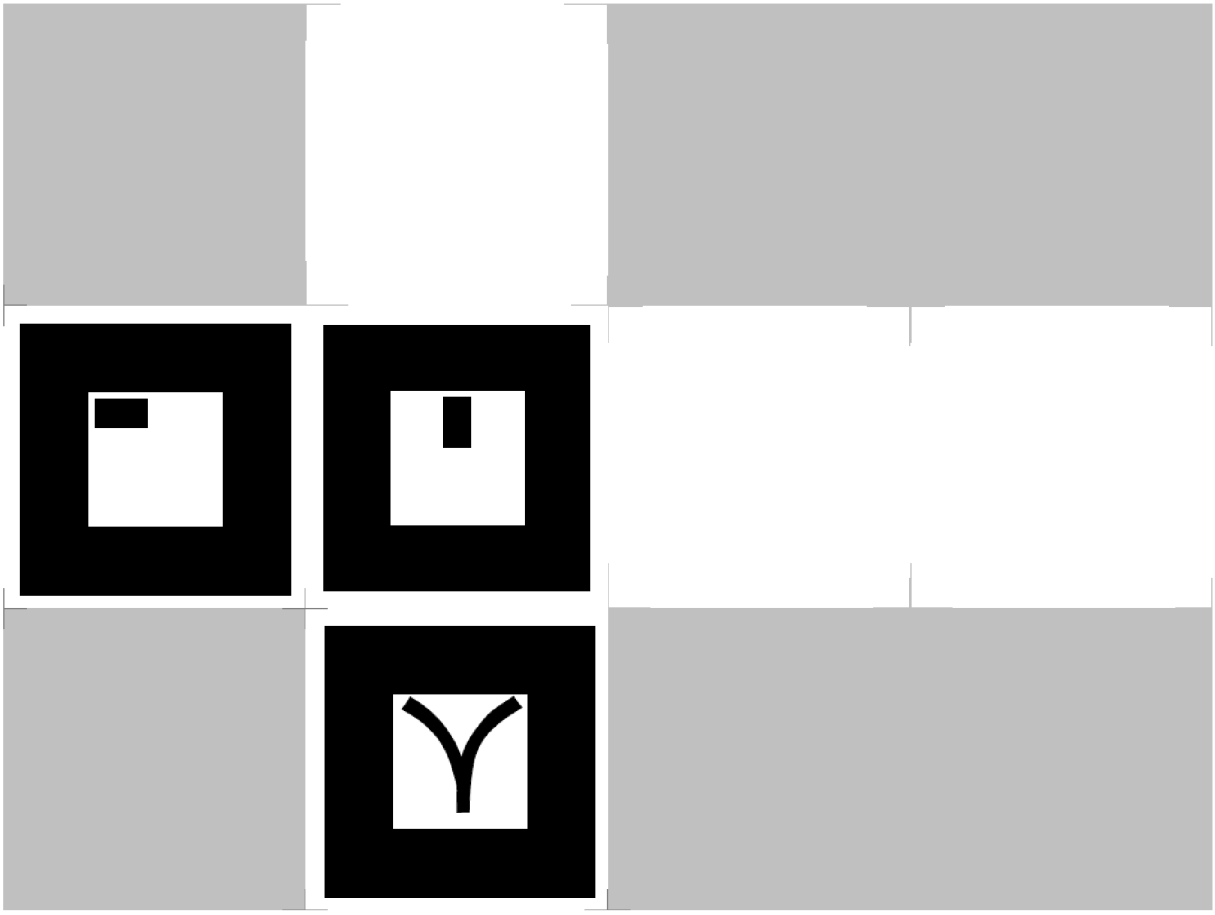
[15] Matěna, L.: *Parametry systému pro rozšířenou virtuální realitu* [diplomová práce]. Masarykova univerzita, Fakulta informatiky

[16] Mulloni, A.: *A collaborative and location-aware application based on augmented reality for mobile devices* [diplomová práce]. Universita degli Studi di Udine, Facolta di Scienze Matematiche Fisiche e Naturali

[17] Sairio, M.: *Augmented Reality*, dokument dostupný na URL http://www.tml.tkk.fi/Studies/Tik-111.590/2001s/papers/mikko_sairio.pdf (prosinec 2009)

Dodatek A

Šablona hrací kostky

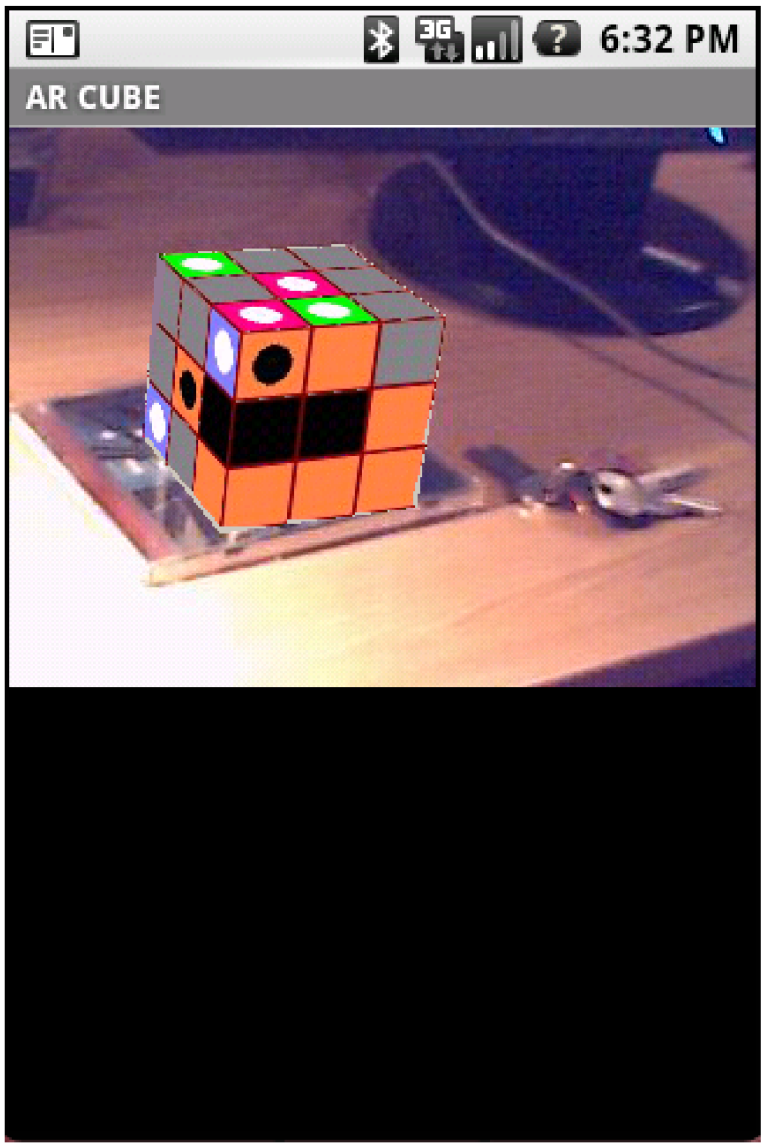


Obrázek A.1: Šablona hraci kostky

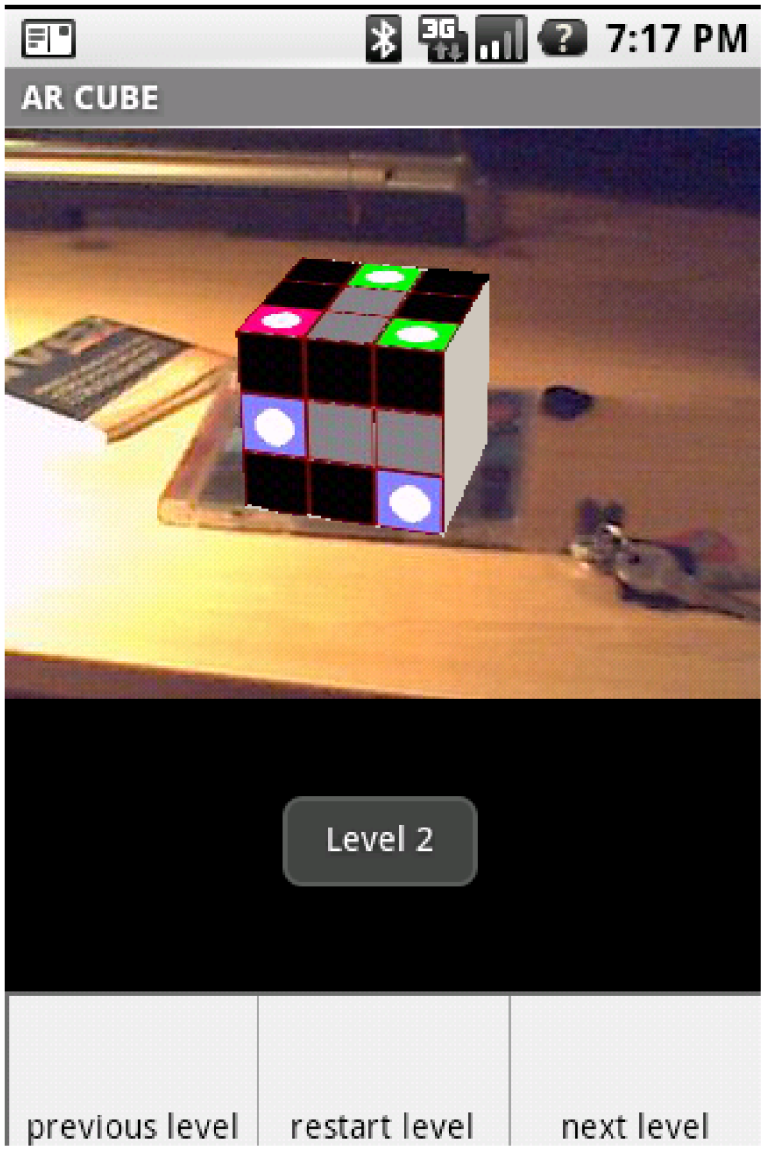
Dodatek B

Ukázky uživatelského rozhraní

Obrázek B.1: Obrazovka pro nastavení a spuštění hry



Obrázek B.2: Hlavní obrazovka



Obrázek B.3: Hlavní obrazovka hry se zobrazeným menu