UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Grega Podlesek

Implementacija in uporaba knjižnice za prikaz bližnjih zanimivih točk z obogateno resničnostjo na mobilnih napravah

DIPLOMSKO DELO
NA UNIVERZITETNEM ŠTUDIJU

Mentor: prof. dr. Saša Divjak

Ljubljana, 2011

Univerza
v Ljubljani

Fakulteta za računalništvo
in informatiko

Tržaška 25
1000 Ljubljana, Slovenija
telefon: 01 476 84 11
faks: 01 426 46 47
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 01708/2010
Datum: 04.10.2010

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **GREGA PODLESEK**

Naslov: **IMPLEMENTACIJA IN UPORABA KNJIŽNICE ZA PRIKAZ BLIŽNJIH ZANIMIVIH TOČK Z OBOGATENO RESNIČNOSTJO NA MOBILNIH NAPRAVAH**

IMPLEMENTATION AND USAGE OF A LIBRARY FOR THE VISUALISATION OF POINTS OF INTEREST WITH AUGMENTED REALITY ON MOBILE DEVICES

Vrsta naloge: Diplomsko delo univerzitetnega študija

Tematika naloge:

Raziščite obstoječe aplikacije in platforme, ki omogočajo prikaz zanimivih točk s pomočjo obogatene resničnosti na mobilnih napravah. Razvijte primerno knjižnico, ki naj omogoči večjo fleksibilnost v primeri z že obstoječimi platformami pri prikazu zanimivih točk, odzivanju na uporabnikove dotike in obveščanju uporabnika o dogodkih v ozadju. Demonstrirajte uporabo razvite knjižnice na primerni konkretni aplikaciji.

Mentor:

prof. dr. Saša Divjak

Dekan:

prof. dr. Nikolaj Zimic

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Namesto te strani **vstavite** original izdane teme diplomskega dela s podpisom mentorja in dekana ter z žigom fakultete, ki ga diplomant dvigne v študentskem referatu, preden odda izdelek v vezavo!

IZJAVA O AVTORSTVU

diplomskega dela

Spodaj podpisani Grega Podlesek,

z vpisno številko 63040130,

sem avtor diplomskega dela z naslovom:

Implementacija in uporaba knjižnice za prikaz bližnjih zanimivih točk z obogateno resničnostjo na mobilnih napravah

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom
  prof. dr. Saše Divjaka
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani, dne 11.04.2011 Podpis avtorja:

Zahvala

Zahvaljujem se družini in prijateljem, ki so mi v času študija stali ob strani, me spodbujali in se z mano veselili uspehov. Zahvaljujem se tudi mentorju prof. dr. Saši Divjaku za pomoč in nasvete pri nastajanju diplome.

Kazalo

Seznam uporabljenih kratic in simbolov

- GPS (Global Positioning System) - sistem globalnega določanja položaja.
- GSM (Global System for Mobile communications) - svetovni standard mobilnih komunikacij.
- API (Application Programming Interface).

Povzetek

Poznamo več načinov prikaza bližnjih zanimivih točk na mobilnih napravah. Eden zanimivejših je obogatena resničnost, pri kateri na živo sliko iz kamere za vsako zanimivo točko prilepimo oznako, ki kaže v smeri te točke. Za prikaz teh oznak potrebujemo njihove zaslonske koordinate.

Lokacije zanimivih točk so določene z zemljepisno dolžino in širino ter nadmorsko višino. Naprej moramo iz teh podatkov izračunati koordinate v koordinatnem sistemu realnega sveta (ki ima središče v napravi). Za to moramo poznati lokacijo naprave, ki jo dobimo s pomočjo GPS ali drugega lokatorja (npr. triangulacije s pomočjo GSM baznih postaj in/ali wi-fi dostopnih točk). Te koordinate moramo nato transformirati v koordinate v koordinatnem sistemu naprave, za kar potrebujemo njeno orientacijo. To dobimo s pomočjo senzorja pospeškov (ta nam da vektor, ki kaže proti tlom) in senzorja magnetnega polja (da vektor, ki kaže proti severnemu magnetnemu polu). S pomočjo dobljenih vektorjev izračunamo rotacijsko matriko, s katero opravimo omenjeno transformacijo. Iz dobljenih koordinat nato izračunamo zaslonske koordinate, za kar moramo poznati zorni kot kamere.

V pričujočem diplomskem delu smo najprej raziskali obstoječe aplikacije in platforme, ki izkoriščajo opisani način prikaza zanimivih točk. To so Layar in Wikitude (aplikaciji z veliko bazo zanimivih točk, a zelo natrpanim uporabniškim vmesnikom), Android-AR-Kit (slab poizkus implementacije knjižnice) in Mixare (odprtokodna aplikacija z dobrim vmesnikom in zasnovo, namenjena prikazu poljubnih virov zanimivih točk).

Med naštetimi rešitvami manjka kvalitetna programska knjižnica, ki smo jo zato razvili v okviru tega diplomskega dela. S to knjižnico, razvito za sistem Android, smo dosegli veliko večjo fleksibilnost, kot jo ponujajo obstoječe platforme (Layar, Wikitude, Mixare) tako pri prikazu zanimivih točk, odzivanju na uporabnikove dotike in obveščanju uporabnika o dogodkih v ozadju. Na koncu smo demonstrirali uporabo naše knjižnice v aplikaciji Odpiralni Časi.

Ključne besede:

obogatena resničnost, mobilne naprave, zanimive točke, programska knjižnica, Android

Abstract

There are many ways to display nearby points of interest (POI) on mobile devices. One of more interesting is augmented reality, where a camera preview is displayed on the screen and a marker is overlaid over it for each of these points of interest, pointing in it's direction. In order to show these markers we need to calculate their screen coordinates.

The location of each POI is defined with it's latitude, longitude and altitude. First, we need to convert these coordinates to the real world coordinate system (with its center at the device's location). In order to do this, we need the device's current location, which we can get via GPS or other locator (e.g. triangulation ob GSM cells and/or wi-fi access points). For the next step, we need the device's orientation, which we can get from the acceleration (returns a vector, pointing at the ground) and magnetic field (returns a vector pointing at magnetic north) sensors. Using these vectors, we can calculate the rotational matrix, which we can than use to transform the point of interest's real world coordinates to the device coordinate system. From these, we can get the screen coordinates. For these final transformation we need to know the camera's angle of view.

In these thesis we began with an overview of existing application and platforms using the described way of displaying points of interest. Some of these are Layar and Wikitude (applications with a large collection of POI, but with cluttered interfaces), Android-AR-Kit (a poor shot at implementing a library) and Mixare (open source application with a well designed interface and backend, designed to display POI from custom data sources or applications).

One thing that is missing, is a quality library, so we developed one in these thesis. Our library, developed for the Android system, offers grater flexibility than the existing platforms (Layar, Wikitude, Mixare) at displaying POI, responding to user clicks and notifying user of background activities. We also present the application Odpiralni Časi, using our library with real world data.

Key words:

augmented reality, mobile devices, points of interest, programing library, Android

Poglavje 1

Uvod

Na mobilnih napravah se v zadnjem času pojavlja vedno več storitev oz. aplikacij, ki ponujajo pregled in iskanje po določeni vrsti zanimivih točk (angl. *POI - Points Of Interest*) v okolici uporabnika. Pri tem se poraja vprašanje načina prikaza informacij in lokacij teh točk. V večini primerov se za to uporabi seznam, v katerem se za vsako zanimivo točko prikaže nekaj osnovnih informacij (kratek opis, ime, naslov, oddaljenost, majhna slika in podobno). Običajno je na voljo tudi prikaz zemljevida z označenimi lokacijami točk skupaj s trenutno lokacijo uporabnika.

V tem diplomskem delu bomo predstavili alternativo tem prikazom, *obogateno resničnost*. To je direkten ali indirekten pogled na resnični svet, ki je obogaten z računalniško izdelanimi učinki. Ti učinki so največkrat vizualni in/ali zvočni, od enostavnih označb, do slik, 3D modelov, videa in animacije [4, 12].

Z obogateno resničnostjo se vedno pogosteje srečujemo tudi v vsakdanjem življenju. Dober primer so športni prenosi, na primer nogometni, kjer se pri analizi tekem poudari različne podrobnosti igre s potmi igralcev ter žoge in podobno. Pogosto so ti prenosi obogateni z oglasnimi sporočili, ki se npr. prikažejo na delu igrišča (Oglasi so tako lahko bolj lokalizirani).

Primer obogatene resničnosti je tudi *HUD (Heads-up display)* [5], to je prosojen zaslon, ki prikazuje informacije uporabniku, ne da bi mu bilo potrebno pogledati odvrniti od trenutne točke gledanja. Večinoma se uporablja v letalstvu (zlasti vojaškem), počasi pa se uvaja tudi v avtomobile.

Obogatena resničnost se uporablja tudi za zabavo. V tej kategoriji je še posebej zanimiv *AR.Drone* [3]. To je daljinsko vodeni helikopter s štirimi propelerji, ki ga upravljamo s pripadajočo aplikacijo na napravi iPhone (ali iPad), ki komunicira s helikopterjem preko wi-fi omrežja. Helikopter ima na

sprednji strani pritrjeno kamero, slika s katere se v živo prenaša aplikaciji. Tako uporabnik vidi skozi "oči" helikopterja. Obogatena resničnost se uporabi, ko imamo na voljo več kot en AR.Drone. Aplikacija omogoča navidezne boje s temi helikopterji, kjer lahko streljamo na nasprotnikov AR.Drone, pri čemer se izstrelki izrisujejo nad videom iz kamere.

Vendar nas najbolj zanima uporaba obogatene resničnosti za prikaz okoliških zanimivih točk. Za ta namen jo izkoristimo tako, da na živo sliko iz kamere za vsako zanimivo točko prilepimo oznako, ki kaže v smeri te točke.

Najprej bomo raziskali teoretično osnovo za prikaz zanimivih točk v obogateni resničnosti na mobilnih napravah, nato bomo predstavili nekaj obstoječih aplikacij oz. platform iz tega področja. Sledila bo predstavitev knjižnice za tovrstno obogateno resničnost, ki je bila razvita v okviru tega diplomskega dela, za konec pa bomo predstavili še uporabo te knjižnice v aplikaciji Odpiralni Časi [8].

Poglavje 2

Potrebna strojna oprema

Če hočemo "prilepiti" informacije na realni svet, moramo tega spraviti v digitalno obliko, ali pa prikazati informacije uporabniku pred očmi na prosojnem zaslonu. Mobilne naprave na splošno druge opcije ne ponujajo, so pa skoraj vse opremljene za prvo možnost, saj je fotoaparat/kamera pri njih že standardna oprema. Realni svet snemamo s kamero in posnetek v realnem času prikazujemo na ekranu, hkrati pa ga obogatimo z našimi informacijami.

Za takšno aplikacijo je potrebna ustrezna strojna oprema, ki pa je prisotna pri večini sodobnih pametnih telefonov in tablic.

2.1 Kamera

Kamera je potrebna za snemanje realnega sveta. Video, ki ga snema kamera nikamor ne shranjujemo, temveč le prikazujemo v živo (t.i. *predogled*). Dobro je, da kamera lahko snema s čim več sličicami na sekundo, saj bo tako prikazana slika bolj tekoča. Resolucija kamere ni pomembna, saj smo omejeni z resolucijo zaslona, ki pa jo kamere v sodobnih mobilnih napravah zlahka presegajo.

2.2 GPS ali drug mehanizem pridobivanja lokacije

Lokacije zanimivih točk prikazujemo glede na trenutno lokacijo, ki jo moramo avtomatsko pridobiti. To pomeni, da naprava potrebuje lokator: GPS sprejemnik ali kakšen drug način pridobivanja lokacije (triangulacija s pomočjo GSM baznih postaj in/ali wi-fi dostopnih točk). Trenutna lokacija je predstavljena z

zemljepisno dolžino in širino, izraženi v kotnih stopinjah, ter nadmorsko višino (ta sicer ni vedno na voljo), izraženo v metrih.

2.3 Orientacijski senzorji

Poleg trenutne lokacije potrebujemo tudi informacijo o trenutni orientaciji naprave. Sodobne mobilne naprave imajo vgrajeno celo vrsto različnih senzorjev. Za ugotavljanje orientacije se uporabljata senzor pospeškov in senzor magnetnega polja.

2.3.1 Senzor pospeškov

Ta senzor zaznava pospeške naprave v treh dimenzijah. Pogosto ga poimenujemo tudi gravitacijski senzor, saj ne zna ločiti med pospeškom, ki ga ustvarja gibanje, in gravitacijo. Iz tega razloga senzor pospeškov, ko je v mirovanju, na navpični osi vedno zazna pospešek $-9,8 m/s^2$.

S senzorjem pospeškov lahko torej zaznamo spremembe v gibanju naprave. Podatke uporabimo za kontrolo aplikacij z gestami, zaznavanje obračanja in nagnjenosti naprave, zaznavanje padca oz. meta naprave, proženje različnih akcij ob nepredvidenem premiku naprave (kot varnostni mehanizem) in podobno.

Od senzorja pospeškov dobimo vektor, ki kaže v smeri pospeška naprave, njegova magnituda pa odraža moč pospeška. Pri uporabi tega vektorja moramo vedno upoštevati, da je v njem vključen gravitacijski pospešek.

2.3.2 Senzor magnetnega polja

Senzor meri trenutno moč magnetnega polja in se lahko uporablja kot kompas. Od senzorja dobimo vektor, ki kaže v smeri severnega magnetnega pola. Ponavadi želimo izvedeti lokacijo oz. smer geografskega severnega pola, ki pa se lahko, odvisno od naše trenutne lokacije, za nekaj stopinj razlikuje od magnetnega severnega pola, kar moramo pri uporabi vektorja upoštevati.

2.4 Veljavnost izmerjenih vrednosti

V nekaterih primerih so izmerjene vrednosti s senzorjev in s tem rotacijska matrika neveljavne:

- kadar je naprava v prostem padu in je v stanju, podobnem breztežnosti,

- kadar pospešuje,
- kadar je blizu severnega pola,
- kadar je postavljena v močno magnetno polje

Poglavje 3

Izračun koordinat za izris oznak zanimivih točk

3.1 Ugotavljanje rotacije naprave s pomočjo senzorjev

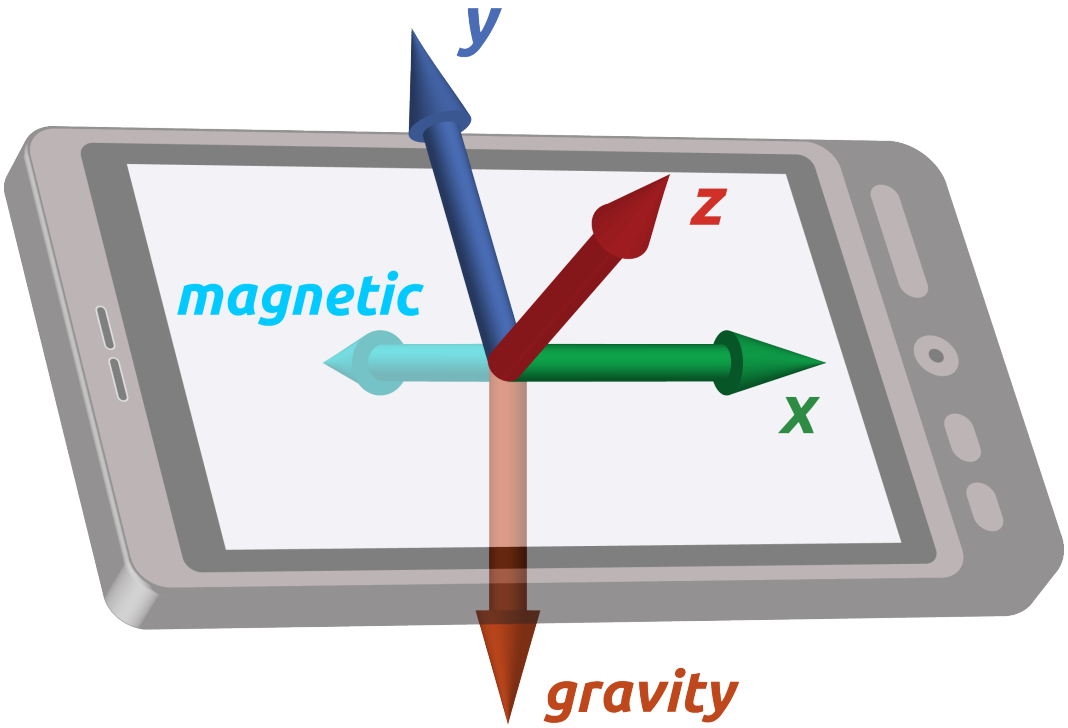
S kombinacijo senzorjev pospeškov in magnetnega polja lahko ugotovimo trenutno orientacijo naprave [11].

Podatki, ki jih dobimo od senzorjev pospeškov (vektor gravitacije) in magnetnega polja so definirani v koordinatnem sistemu naprave. Ta je definiran glede na napravo v vodoravni poziciji na naslednji način: x os je vodoravna in kaže proti desni, y os je navpična in kaže navzgor, z os pa kaže navzven iz ekrana na sprednji strani. Na sliki 3.1 je prikazana sheme tega koordinatnega sistema.

Na sliki vidimo tudi smer (vektor) gravitacijskega pospeška, kot nam jo posreduje senzor pospeškov. Ker ta vedno kaže navzdol (v koordinatnem sistemu realnega sveta) se njegove koordinate pri obračanju naprave spreminjajo. Iz teh koordinat lahko izračunamo, kako je nagnjena naprava glede na tla. Tako nam manjka le še rotacija glede na navpičnico, ki jo dobimo s pomočjo senzorja magnetnega polja. Le ta nam da vektor, ki kaže proti severnemu magnetnemu polu.

S pomočjo senzorjev smo dobili velikost rotacij okoli vseh treh osi, definiranih v koordinatnem sistemu naprave. Toda težava je, da so položaji zanimivih točk definirani v koordinatnem sistemu realnega sveta. Ta je definiran z naslednjimi koordinatnimi osmi:

- X, ki je definirana kot vektorski produkt $Y \times Z$ (vzporedna s tlemi in



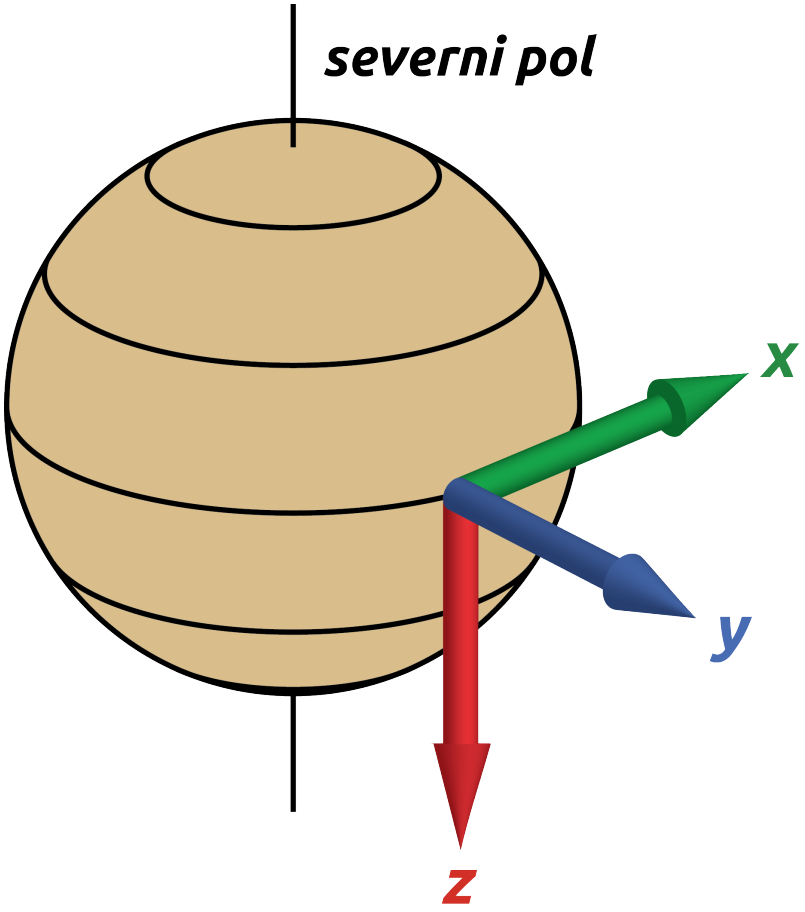
Slika 3.1: Koordinatni sistem naprave ter vektorja gravitacije in magnetnega polja, ki sta definirana v tem koordinatnem sistemu. Smer vektorja gravitacije nam da rotacijo naprave okoli obeh vodoravnih osi (v koordinatnem sistemu realnega sveta), vektor magnetnega polja (ki je usmerjen proti severu), pa nam da rotacijo okoli vertikalne osi.

kaže proti vzhodu),

- Y, ki je pravokotna na tla in kaže proti nebu in
- Z, ki je vzporedna s tlemi in kaže v smer, nasprotno severnemu magnetnemu polu.

Shemo tega koordinatnega sistema lahko vidimo na sliki 3.2.

Da bi lahko prikazali oznake zanimivih točk, potrebujemo transformacijo, ki bo preslikala koordinate teh točk iz koordinatnega sistema realnega sveta v koordinatni sistem naprave.



Slika 3.2: Koordinatni sistem realnega sveta glede na Zemljo.

3.2 Rotacijska matrika za preslikavo v koordinatni sistem naprave

Potrebujemo torej rotacijsko matriko, ki bo zarotirala koordinate v realnem svetu tako, kot je zarotirana naprava.

Rotacijska matrika je kvadratna matrika, s katero lahko na enostaven način izvajamo rotacije v evklidskem prostoru. Za izračun rotacije neke točke, predstavimo to točko s stolpčnim vektorjem v. Zarotiran vektor v' dobimo z matričnim množenjem $v' = Rv$, kjer je R rotacijska matrika [9].

Z rotacijsko matriko želimo zarotirati točke, definirane v fiksnem koordinatnem sistemu (v našem primeru koordinatni sistem realnega sveta) v nov koordinatni sistem (v našem primeru koordinatni sistem naprave), definiran z

enotskimi vektorji $\hat{u}$, $\hat{v}$ in $\hat{w}$. Ti predstavljajo koordinatne osi novega koordinatnega sistema in so definirani v fiksnem koordinatnem sistemu. Rotacijsko matriko lahko definiramo na več načinov, za nas je najbolj uporabna naslednja definicija:

$$R = \begin{bmatrix} \hat{u}_x & \hat{u}_y & \hat{u}_z \\ \hat{v}_x & \hat{v}_y & \hat{v}_z \\ \hat{w}_x & \hat{w}_y & \hat{w}_z \end{bmatrix} \tag{3.1}$$

Vektorje $\hat{u}$, $\hat{v}$ in $\hat{w}$ izračunamo na naslednji način:

$$\begin{aligned} \hat{u} &= \hat{gravity} \times \hat{geomagnetic} \\ \hat{w} &= \hat{gravity} \times \hat{u} \\ \hat{v} &= -\hat{gravity} \end{aligned}$$

kjer je *gravity* gravitacijski vektor (usmerjen proti tlom), dobljen od senzorja pospeškov, in *geomagnetic* vektor magnetnega polja (usmerjen proti severnemu magnetnemu polu), dobljen od senzorja magnetnega polja.

S pomočjo rotacijske matrike lahko sedaj dobimo koordinate posamezne zanimive točke v koordinatnem sistemu naprave. Za to najprej potrebujemo koordinate teh točk v koordinatnem sistemu realnega sveta.

Koordinate posameznih zanimivih točk imamo na voljo kot vektor $geo = (lng, lat, alt)$, kjer so *lng* (zemljepisna dolžina), *lat* (zemljepisna širina) in *alt* (nadmorska višina) absolutne koordinate s koordinatnim središčem v Atlantskem oceanu. Poleg tega sta *lng* in *lat* v kotnih stopnjah, *alt* pa v metrih. Za transformacijo moramo premakniti vektor *geo* v koordinatni sistem realnega sveta, ki ima središče na trenutni lokaciji naprave, in pretvoriti kotne stopinje v metre. *geo* premaknemo z odštevanjem trenutne lokacije l:

$$geo - l = loc.$$

Rezultat je vektor *loc*, ki ga sedaj lahko transformiramo z matriko R, da dobimo vektor m, ki predstavlja koordinate določene zanimive točke v koordinatnem sistemu naprave:

$$m = R * loc.$$

3.3 Pretvorba koordinat zanimivih točk v zaslonske koordinate

S pomočjo senzorjev smo dobili transformacijo med koordinatnim sistemom realnega sveta in koordinatnim sistemom naprave. Z njeno pomočjo smo dobili koordinate zanimivih točk v koordinatnem sistemu naprave. Vendar to ni dovolj. Za prikaz zanimivih točk na zaslonu namreč potrebujemo vedeti:

1. ali je točka sploh vidna oz. ali je smer, v kateri se nahaja, znotraj trenutnega vidnega polja kamere in
2. v primeru, da je zanimiva točka vidna, kakšne so njene zaslonske koordinate.

Za ugotavljanje vidnosti in izračun zaslonskih koordinat poleg orientacije naprave potrebujemo še zorni kot (α) kamere. Ta je odvisen od velikosti slikovnega senzorja (d) in goriščne razdalje (f) objektiva.

$$\alpha = 2 \arctan \frac{d}{2f}$$

Pri ugotavljanju teh parametrov lahko naletimo na težave. Pogosto ti podatki za posamezno napravo niso nikjer objavljeni, tako da moramo zorni kot izmeriti, kar pa ni vedno izvedljivo, saj ponavadi nimamo fizičnega dostopa do naprave. Drugi problem je raznolikost naprav, na katerih deluje naša aplikacija, saj imajo kamere na vsaki izmed naprav lahko drugačen zorni kot. To je še posebej problem pri sistemih, kot je Android, ki tečejo na velikem številu različnih naprav (in s tem tudi naša aplikacija). Na srečo ima večina mobilnih naprav zelo podoben zorni kot in lahko zato uporabimo povprečje.

Zdaj imamo potrebne podatke za izračun zaslonskih koordinat zanimivih točk. Za določeno točko želimo izračunati zaslonske koordinate (x, y, z). Upoštevati moramo, da je središče zaslona v zgornjem levem kotu, kar pomeni, da je potrebno koordinatama x in y prišteti polovico širine (w) oz. višine (h) zaslona. Koordinate dobimo s pomočjo vektorja m (lokacija zanimive točke, izražena v koordinatnem sistemu naprave) na naslednji način:

$$\begin{aligned} x &= f_p \frac{m_x}{-m_z} + \frac{w}{2} \\ y &= f_p \frac{m_y}{-m_z} + \frac{h}{2} \\ z &= m_z \end{aligned} \tag{3.2}$$

Pri tem je f_p goriščna razdalja kamere, izražena v pikslih. Velja:

$$f_p = \frac{\frac{w}{2}}{\tan\frac{\alpha}{2}},$$

kjer je w širina zaslona v pikslih, α pa zorni kot kamere.

Vidnost zanimive točke je odvisna od koordinate z in kotov $\beta_h = \arctan\frac{m_x}{-m_z}$ in $\beta_v = \arctan\frac{m_y}{-m_z}$. V primeru, da je $z < 0$ je točka za kamero in zato nevidna. Kota β_h in β_v morata biti manjša od polovice horizontalnega oz. vertikalnega zornega kota kamere. Preverjanje vidnosti lahko poenostavimo tako, da namesto izračuna kotov uporabimo kar izračunani koordinati x in y ter preverimo, če se nahajata znotraj dimenzij zaslona.

3.4 Celotna transformacija

Poizkusimo zdaj na hitro obnoviti do sedaj opisane transformacije in izračune. Najprej smo pridobili lokacijo naprave, ki predstavlja središče tako koordinatnega sistema naprave, kot koordinatnega sistema realnega sveta. Nato smo s pomočjo senzorja pospeškov in senzorja magnetnega polja dobili rotacijsko matriko, ki predstavlja transformacijo med obema koordinatnima sistemoma.

Lokacije zanimivih točk so podane z zemljepisno širino, zemljepisno višino in nadmorsko višino. Te koordinate smo najprej s pomočjo trenutne lokacije premaknili v koordinatni sistem realnega sveta in pretvorili v skupno mersko enoto (metre), nato pa z rotacijsko matriko transformirali v koordinatni sistem naprave. Na koncu smo iz teh koordinat s pomočjo informacij o fizičnih lastnosti kamere (zorni kot) izračunali točko na zaslonu, kjer lahko sedaj prikažemo oznake zanimivih točk.

Poglavje 4

Obstoječe aplikacije in platforme

Na tržišču obstaja že nekaj aplikacij, ki izkoriščajo obogateno resničnost za prikaz bližnjih zanimivih točk. Predstavimo nekaj najbolj zanimivih in popularnih.

4.1 Layar

Layar Reality Browser [6], ki ga ponuja Nizozemsko podjetje Layar, je verjetno najbolj znana mobilna aplikacija, ki uporablja obogateno resničnost kot svojo primarno funkcijo. Sama aplikacija Layar deluje kot platforma, informacije za prikaz (zanimive točke) pa zagotavljajo t.i. plasti (angl. *layers*). Te ponujajo lokalne informacije za turiste (bližnji hoteli, zanimivosti), informacije o bližnjih restavracijah in barih, o dogodkih v bližini (koncerti, kino predstave), o nepremičninah, pa tudi informacije tehnične narave (bližnje wi-fi dostopne točke, *id* bližnjih GSM celic). Nekaj plasti se povezuje tudi s socialnimi omrežji in podobnimi internetnimi storitvami (*Twitter*, *Foursquare*, *YouTube*, *Flickr* itd.). Med plastmi se najdejo tudi igre, kot je igranje verzije legendarne igre *Pac-Man*, kjer se sprehajamo in "jemo" točke in sadje, lov na predmete po okolici (t.i. *scavenger hunt*), kot tudi lov na navidezna bitja *Woomba*.

Obogatena resničnost ni edini način prikaza teh informacij, saj jih Layar lahko prikaže tudi v seznamu ali na zemljevidu. V obogateni resničnosti lahko prikažemo le posamezno plast naenkrat, a imamo na voljo tudi osnovni pogled (slika 4.1b), pri katerem so v seznamu podane zanimive točke v bližini, izbrane iz več plasti. Nad tem seznamom je poenostavljena različica obogatene resničnosti, brez žive slike iz kamere. Na žalost ni na voljo različica tega

seznama v obogateni resničnosti.

Layar je na voljo za sisteme Android, iOS in Symbian. Na trgu je že nekaj časa in ima že čez milijon uporabnikov ter več kot 1500 razvitih plasti. Platforma je prejela v letih 2010 in 2011 več nagrad.

Platforma je odprta za zunanje razvijalce, ki želijo implementirati lastne plasti. Na voljo je API za razvoj plasti, ki jih nato lahko objavimo, nakar bodo vključene v aplikacijo. Za svojo plast lahko tudi zaračunamo, kar pa naredijo le redki razvijalci. Za uporabo teh plasti lahko uporabniki plačajo preko storitve *PayPal*.

Poleg vključitve lastne plasti v aplikacijo Layar, imamo na platformi iOS na voljo tudi *Layar Player*, s katerim lahko vključimo zmogljivosti Layar v svojo lastno aplikacijo.

V primeru, da je plast, ki jo razvijemo za Layar del večje storitve, je lahko ta poteza tudi dobra promocija, saj lahko s tem dosežemo celotno bazo uporabnikov Layar aplikacije, ki pa, kot smo že omenili, ni majhna.

Sam pogled obogatene resničnosti je prikazan na sliki 4.1a. Poleg oznak zanimivih točk prikazuje "radar", velikost prikazanega območja, natančnost trenutne lokacije in seveda, informacije o trenutno opazovani zanimivi točki. Te informacije so prikazane v skupnem okvirju, ki zaseda skoraj celoten spodnji del zaslona, kar pa je lahko precej moteče, saj se nam vidno polje zmanjša za polovico. Posamezne zanimive točke so prikazane s sličico (angl. *thumbnail*). Vmesnik se prilagaja rotacijam telefona, tako da ga lahko uporabljamo tako v navpičnem, kot vodoravnem položaju.

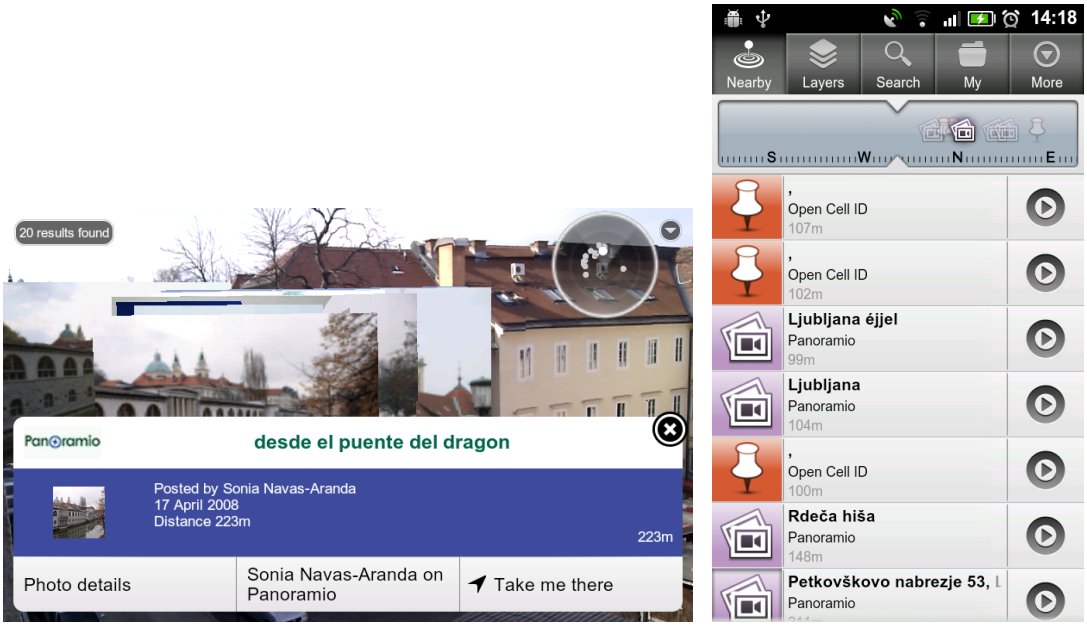
4.2 Wikitude

Wikitude World Browser [10], ki ga ponuja podjetje Mobilizy, je bil prvi brskalnik z obogateno resničnostjo na voljo po celotnem svetu in je poleg Layar verjetno najbolj znana takšna aplikacija.

Wikitude ima zanimive točke razdeljene v t.i. svetove (angl. *worlds*), podobno kot ima Layar plasti. Informacije, ki jih nudijo ti svetovi so zelo raznolike in zavzemajo podobne teme kot Layar plasti.

Wikitude je na voljo za sisteme Android, iOS in Symbian. Prejel je že celo vrsto nagrad na različnih svetovnih tekmovanjih.

Tudi Wikitude omogoča zunanjim razvijalcem, da naredijo lasten svet, podatke katerega lahko naložijo na spletne strežnike Wikitude ali pa na lasten strežnik. Poleg tega Wikitude omogoča tudi manj veščim uporabnikom, da sami dodajo posamezne zanimive točke ali kar celotne svetove s pomočjo *Goo-*



(a) Pogled obogatene resničnosti (b) Osnovni pogled

Slika 4.1: Zaslonske slike aplikacije Layar, vidimo, da je pogled obogatene resničnosti precej nastlan z informacijami in s tem nepregleden

gle Maps.

Pogled obogatene resničnosti je prikazan na sliki 4.2. Podoben je pogledu pri aplikaciji Layar: prikazuje oznake zanimivih točk, "radar", velikost prikazanega območja, ki jo lahko na enostaven način prilagodimo, ter skupen okvir za prikaz informacij zanimivih točk. Ta je malo manjši kot pri Layar in rahlo prosojen, tako da je manj moteč. Ta okvir lahko tudi skrijemo. Posamezno oznako zanimive točke prikaže kot ikono z naslovom pod njo. Pogled obogatene resničnosti je vedno obrnjen vodoravno.

4.3 Android-AR-Kit

Android-AR-Kit [2] je odprtokodna knjižnica, ki omogoča, da v okviru lastne aplikacije prikažemo pogled obogatene resničnosti s poljubnimi informacijami. Na žalost je zelo pomanjkljiva, tako pri možnostih prikaza, kot pri samem računanju transformacij (nagibanje naprave pokvari pravilen prikaz oznak zanimivih točk). Poleg tega se je razvoj knjižnice ustavil že decembra 2009.

Slika 4.2: Zaslonska slika pogleda obogatene resničnosti v aplikaciji Wikitude

4.4 Mixare

Mixare *(mix Augmented Reality Engine)* [7] je odprtokoden brskalnik v obogateni resničnosti, objavljen pod licenco GPLv3. Na voljo je za sistem Android ter iPhone 3G in novejše.

Mixare je na voljo kot samostojna aplikacija, ki prikazuje zanimive točke iz storitev *Wikipedia*, *Twitter*, *Buzz* in *OpenStreetMap*. Poleg tega pa lahko sprejema vire podatkov tudi iz drugih virov. Odziva se namreč na poseben *myme type* "application/mixare-json", kar pomeni, da lahko sprožimo zagon Mixare kar iz internetnega brskalnika. Kot namiguje že *myme type*, podatke posredujemo v JSON formatu. Na podoben način lahko zaženemo Mixare iz lastne aplikacije, pri čemer posredujemo vir podatkov v omenjenem formatu. V primeru, da to ni dovolj, lahko prilagodimo Mixare svojim potrebam in ga vključimo v lastno aplikacijo, saj je izvorna koda prosto dostopna. To na žalost ni preveč enostavno, saj je Mixare zasnovana kot aplikacija in ne kot knjižnica.

Pogled obogatene resničnosti prikazuje "radar" in smer v kotnih stopinjah (npr. "95° E"). Oznake zanimivih točk so prikazane z ikono in naslovom ter kratkim opisom pod njo. Klik na eno izmed oznak odpre spletno stran, na katero se zanimiva točka nanaša, v okvirju, ki zavzema skoraj celoten zaslon.

Poglavje 5

Knjižnica MixareLib

Kot vidimo, obstaja že kar nekaj implementacij obogatene resničnosti na mobilnih napravah. Med njimi pa manjka dobra knjižnica na platformi Android (ali katerikoli drugi platformi), ki bi jo lahko vključili v lastno aplikacijo. Android-AR-Kit ima sicer ta namen, ki pa mu ga ne uspe doseči.

Od takšne knjižnice pričakujemo:

- čim bolj kvalitetno izračunavanje rotacije naprave in s tem natančen prikaz lokacij zanimivih točk,
- posamezne točke naj bodo predstavljene s sličico in kratkim opisom, a ta prikaz naj bo fleksibilen,
- podpora za odzivanje na pritiske s prstom na eno izmed zanimivih točk,
- prikaz "radarja" za hiter pregled lokacij zanimivih točk v bližini,
- prikaz radija, znotraj katerega se nahajajo prikazane točke,
- omogoča naj prikaz sporočil (npr. "Nalagam podatke iz strežnika...").

Implementacije take knjižnice bi se lahko lotili od začetka, po drugi strani pa se nam Mixare, kot odprtokodna aplikacija ponuja kar sama, da jo prilagodimo v takšno knjižnico.

V okviru tega diplomskega dela je bila narejena ta prilagoditev v knjižnico. Pri tem je bilo potrebno:

- odstraniti vse, kar je pri knjižnici nepotrebno. Torej razrede, ki omogočajo Mixare, da deluje kot samostojna aplikacija, razrede za prenos in prebiranje podatkov, ki jih Mixare sicer prikazuje in podobno.

- spremeniti razrede in metode tako, da podpirajo prikaz poljubnih podatkov oz. zanimivih točk
- dodati sistem za obveščanje, ki pa že vključuje obveščanje o nekaterih dogodkih (več v razdelku 5.1.1).

Med to prilagoditvijo je bilo narejenih tudi nekaj manjših optimizacij ter prestrukturiranja razredov.

5.1 Delovanje

Rezultat opisanega je delujoča knjižnica, ki zadostuje vsem prej naštetim zahtevam. Pri predstavitvi naše knjižnice bomo opisali delovanje najpomembnejših razredov in njihove medsebojne interakcije. Za lažje razumevanje je na sliki 5.1 prikazan UML diagram teh razredov.

5.1.1 ARActivity

To je glavni razred knjižnice, ki povezuje vse komponente in razširja razred `Activity` [1]. Ta predstavlja posamezno aktivnost znotraj aplikacije, ki se trenutno izvaja in katere vmesnik je prikazan in na voljo uporabniku za interakcijo. `ARActivity` vključuje pogleda `camView` (razreda `CameraSurface`), ki prikazuje živo sliko iz kamere, in `augView` (razreda `AugmentedView`), ki prikazuje komponente obogatene resničnosti.

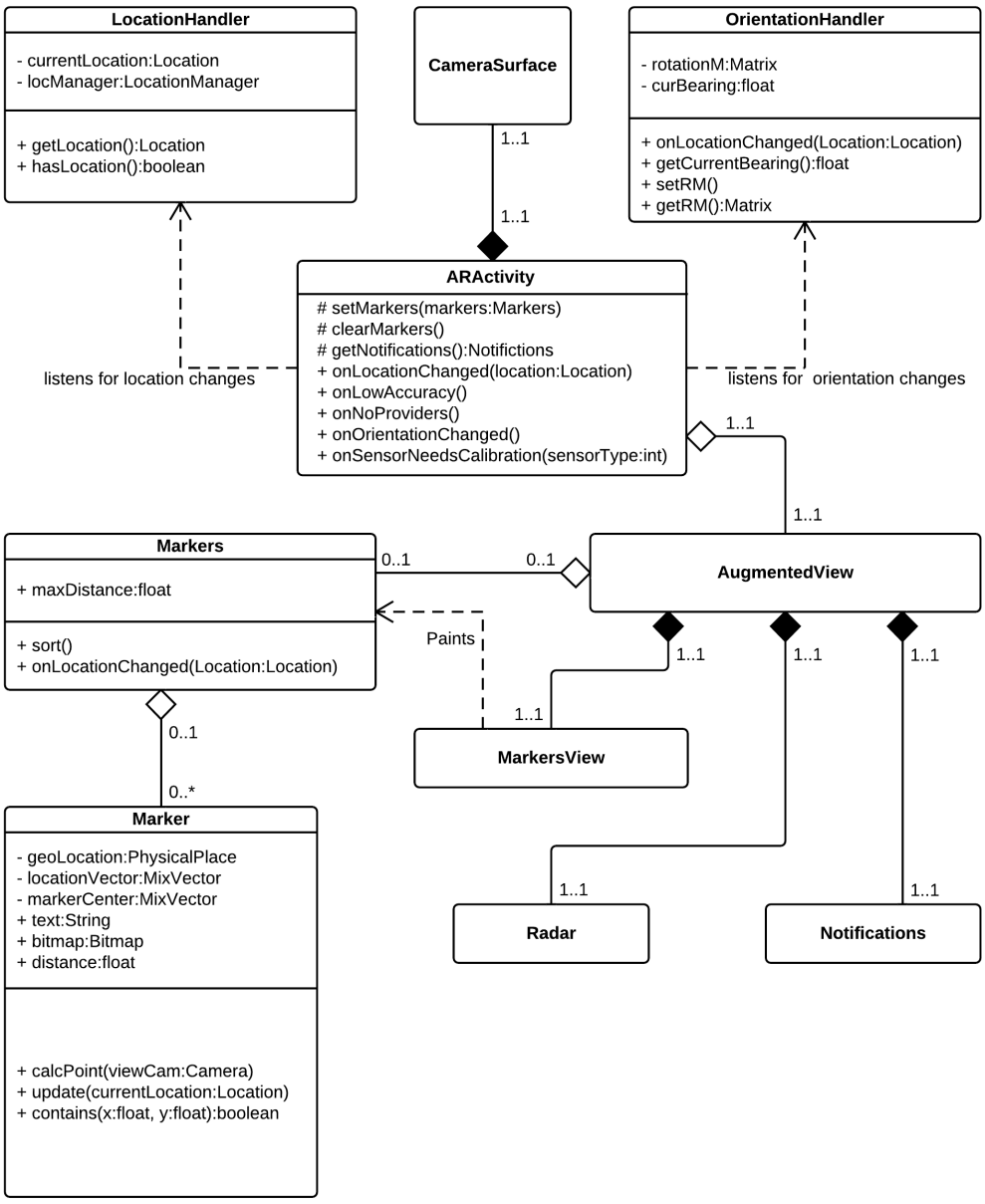
`ARActivity` "posluša" spremembe lokacije in orientacije naprave, ki mu jih posredujeta objekta `locationHandler` (razreda `LocationHandler`) in `orientationHandler` (razreda `OrientationHandler`), ter jih posreduje pogledu `augView`. Lokacijo posreduje tudi objektu `orientationHandle`.

`ARActivity` skrbi tudi za osnovno obveščanje uporabnika o:

- čakanju na lokacijo, v primeru, da trenutne lokacije še nismo pridobili in čakamo, da nam jo posreduje eden od vklopljenih lokatorjev.
- natančnosti trenutne lokacije (v metrih).

in kadar

- nobeden od lokatorjev ni vklopljen, kar pomeni, da ne moremo pridobiti lokacije,
- je natančnost trenutne lokacije nizka in GPS ni vklopljen. Pri tem svetujemo vklop GPS,



Slika 5.1: UML diagram knjižnice MixareLib. Prikazani so najpomembnejši razredi, metode in parametri.

- je magnetni senzor ali senzor pospeškov potrebno kalibrirati. To se zgodi v primeru, da je natančnost enega od teh senzorjev nizka. Več o kalibraciji senzorjev v razdelku 7.2.

Kot vidimo, se večina obvestil nanaša na natančnost trenutne lokacije. Uporabnost obogatene resničnosti je namreč od nje precej odvisna. Brez uporabe GPS in samo z uporabo triangulacije preko baznih postaj, je lahko lokacija napačna tudi za več sto metrov ali celo nekaj kilometrov (še posebej zunaj mest, kjer je malo baznih postaj), kar precej zmanjša uporabnost obogatene resničnosti.

5.1.2 LocationHandler

Ta razred skrbi za pridobivanje trenutne lokacije in za obveščanje `ARActivity` o njenih spremembah, stanju lokatorjev in podobno. Deluje kot nekakšen ovoj za storitev `android.location.LocationManager`, ki nudi aplikacijam obveščanje o spremembah lokacije. `LocationHandler` skrbi za njeno pridobivanje iz vseh vklopljenih lokatorjev (po navadi GPS in/ali triangulacija preko baznih postaj) in preklapljanje med lokatorji v primeru, da uporabnik katerega izklopi ali vklopi.

5.1.3 OrientationHandler

Ta razred "posluša" spremembe v pospešku in magnetnem polju, ki mu jih sporočata senzorja pospeškov in magnetnega polja. Ko dobi nov podatek od enega izmed teh senzorjev, izračuna novo rotacijsko matriko (glej razdelek 3.2). To matriko bi lahko uporabili neposredno za transformacijo koordinat zanimive točke, a se taka uporaba izkaže za neprimerno. Senzorji so namreč občutljivi na šum, tresenje rok in podobno, kar povzroči migotanje prikazanih oznak zanimivih točk. Problem lahko enostavno rešimo tako, da izračunamo povprečje zadnjih nekaj rotacijskih matrik. V naši implementaciji računamo povprečje zadnjih 60 matrik.

`OrientationHandler` spremlja tudi spremembe trenutne lokacije, ki jo potrebuje za izračun odstopanja severnega magnetnega pola od geografskega. To odstopanje je predstavljeno z rotacijo okoli y osi v obliki rotacijske matrike, ki se pri izračunu glavne rotacijske matrike tej doda z množenjem.

Poleg rotacijske matrike, pri vsakem posodabljanju izračunamo tudi smer gledanja glede na vertikalo (angl. *bearing*), ki ga potrebujemo pri prikazu kompasa:

$$look = R^T * \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

$$bearing = \begin{cases} \arccos\left(\frac{look_x}{d}\right) & \text{ko je } look_z \geq 0 \\ 360^{\text{o}} - \arccos\left(\frac{look_x}{d}\right) & \text{ko je } look_z < 0 \end{cases}$$

`OrientationHandler` skrbi tudi za obveščanje o nizki natančnosti senzorjev.

5.1.4 CameraSurface

Predstavlja prvo polovico obogateno resničnosti. Prikazuje namreč živo sliko (predogled) iz kamere. Kamera nudi več različnih velikosti in razmerji (angl. *aspect ratio*) predogledov in glavni problem, ki ga moramo pri tem rešiti je, katero velikost predogleda in razmerje izbrati.

Predogled, ki ga izberemo, mora ustrezati dvema kriterijema:

- predogled prikazujemo na zaslonu naprave, zato bi v idealnem primeru izbrali takega z enakimi dimenzijami kot zaslon. Ker pa to ni vedno mogoče, izberemo predogled, ki je čim bližje velikosti in razmerju zaslona.
- mora biti manjši od velikosti zaslona, ker na nekaterih napravah (npr. HTC Hero z Android 2.1) večji predogledi ne bodo delovali. Na nekaterih napravah tak predogled ni na voljo (na nekaterih telefonih Samsung je najmanjši predogled večji od zaslona), zato uporabimo prevzetega: 480x320.

5.1.5 AugmentedView

Ta razred skrbi za prikaz navideznega dela obogatene resničnosti. Vključuje poglede `markersView`, ki skrbi za prikaz oznak zanimivih točk, `notifications`, ki prikazuje obvestila, in `radar`. Hrani tudi seznam oznak zanimivih točk `markers`, površino za risanje `paintSurface`, na katero rišejo omenjeni pogledi (`markersView`, `radar` in `notifications`), in `cam` (razreda `Camera`), ki hrani lastnosti kamere in skrbi za transformacije.

`AugmentedView` skrbi tudi za odzivanje na uporabnikove klike oz. pritiske na zaslonu. Te dogodke posreduje pogledu `markersView`.

Glavna metoda za izrisovanje je `onDraw(Canvas canvas)`. `AugmentedView` v tej metodi, poleg inicializacije objekta `cam` ob prvem klicu, izvede klic metod `onDraw(PaintSurface ps)` v vsebovanih pogledih (`markersView`, `radar` in `notifications`) in pri tem posreduje svojo površino za risanje `paintSurface`,

5.1.6 MarkersView

Izrisuje oznake (angl. *markers*), ki kažejo v smeri naših zanimivih točk, tako, da kliče metodo `Marker.paint(PaintSurface ps)` za vsako vidno oznako. Pred tem jih je potrebno razvrstiti glede na razdaljo, saj se oznake nekaterih točk, ki se nahajajo v približno enaki smeri, lahko prekrivajo in želimo, da se bližnje izrišejo nad bolj oddaljenimi. Zato oznake razvrstimo tako, da se bolj oddaljene izrišejo pred manj oddaljenimi. To razvrščanje ni potrebno ob vsakem izrisu, temveč le ob spremembah lokacije in seznama zanimivih točk.

Objekt `markersView` sprejema obvestila o pritiskih na zaslonu (dogodkih) od `augmentedView`. Posamezen dogodek vsebuje tudi koordinate pritiska. S pomočjo teh koordinat in podatkov o merah in koordinatah oznak lahko ugotovimo katero oznako, če sploh, je uporabnik izbral. To oznako nato o tem obvestimo preko metode `Marker.onTouch()`. Razred, ki razširja razred `Marker`, lahko izkoristi to metodo za svoje potrebe (npr. za prikaz več informacij o izbrani zanimivi točki).

5.1.7 Markers

Je seznam oznak zanimivih točk. Razširja `ArrayList<Marker>` in tako podpira vse običajne operacije nad seznamom, poleg tega pa skrbi za izračun največje oddaljenosti med vsemi oznakami, že omenjeno razvrščanje oznak in obveščanje oznak o spremembi trenutne lokacije, o katerih ga obvesti `ARActivity`.

5.1.8 Marker

Ta razred predstavlja že omenjeno oznako določene zanimive točke. Vsak Marker hrani naslednje parametre:

- `PhysicalPlace geoLocation` lokacija zanimive točke definirana z geografsko višino in širino ter nadmorsko višino,
- `MixVector locationVector` lokacija zanimive točke v koordinatnem sistemu realnega sveta s središčem v napravi in koordinatami v metrih, kot je opisana v razdelku 3.1,

- `MixVector markerCenter` zaslonske koordinate zanimive točke
- `float distance` oddaljenost zanimive točke od trenutne lokacije (v metrih). Izračuna se hkrati z vektorjem `locationVector`, saj je oddaljenost enaka dolžini vektorja `locationVector`,
- `Bitmap bitmap` sličica (angl. *thumbnail*), ki predstavlja del oznake,
- `String text` besedilo, ki ga izrišemo pod sličico.

Cilj posameznega objekta razreda `Marker` je izris oznake na pravo mesto na površini za risanje `paintSurface`. Preden lahko oznako izrišemo, moramo izračunati njene koordinate na zaslonu in preveriti ali je sploh vidna. To pomeni, da moramo preslikati koordinate posamezne zanimive točke, definirane v koordinatnem sistemu realnega sveta (`locationVector`), v zaslonske koordinate (`markerCenter`). To operacijo smo opisali v razdelkih 3.2 in 3.3.

Vektorja `locationVector` na začetku še nimamo, saj so koordinate zanimive točke definirane z vektorjem `geoLocation`. Pretvorba je relativno enostavna:

koordinata x je razlika med geografsko dolžino zanimive točke in geografsko dolžino trenutne lokacije v metrih,

koordinata y je razlika med nadmorsko višino zanimive točke in nadmorsko višino trenutne lokacije v metrih.

koordinata z je razlika med geografsko širino trenutne lokacije in geografsko širino zanimive točke v metrih,

To pretvorbo moramo izvršiti le, kadar se spremeni trenutna lokacija.

Dobili smo `locationVector`, tako da se lahko lotimo transformacije njegovih koordinat v koordinatni sistem naprave. To storimo z rotacijsko matriko, ki jo je izračunal `orientationHandler`. Dobljene koordinate moramo nato preslikati v zaslonske, kar izračunamo po enačbah 3.2 in dobimo vektor `markerCenter`.

Od izrisovanja nas loči le še en korak. Preveriti moramo, če je oznako sploh potrebno izrisati oz. če je zanimiva točka vidna skozi kamero. Ker smo že izračunali zaslonske koordinate, enostavno preverimo, če je kateri del oznake viden znotraj mej zaslona. Veljati mora torej:

$$-\frac{marker_{width}}{2} < markerCenter_x < screen_{width} + \frac{marker_{width}}{2}$$
$$-\frac{marker_{height}}{2} < markerCenter_y < screen_{height} + \frac{marker_{height}}{2}$$
$$markerCenter_z < -1$$

kjer sta $marker_{width}$ in $marker_{height}$ širina in višina oznake, $screen_{width}$ in $screen_{height}$ pa širina in višina zaslona.

Zdaj vemo ali naj sploh prikažemo oznako in kje, tako da se lahko lotimo risanja. Narišemo sličico in pod njo v okvirčku besedilo. A tu še nismo končali. Želeli bi, da je oznaka vedno obrnjena pokončno glede na tla, ne glede na nagibanje naprave. Potrebujemo torej kot rotacije okoli z osi naprave. Ta kot lahko dobimo tako, da poleg zaslonskih koordinat zanimive točke, izračunamo še zaslonske koordinate točke, ki je tik pod zanimivo točko. Med tema dvema točkama nato potegnemo navidezno premico in izračunamo kot med to premico in navpičnico. Dobili smo kot, za katerega moramo nagniti oznako.

5.1.9 Radar

Skrbi za izris "radarja". `Radar` je nekakšen poenostavljen pogled na okolico iz ptičje perspektive. V ozadju izrišemo tri koncentrične krožnice in v zgornjem delu meji vidnega polja kamere (ki je neposredno odvisen od zornega kota). Na to ozadje izrišemo točke, ki predstavljajo lokacije zanimivih točk, glede na trenutno smer (angl. *bearing*).

Na vrhu "radarja" izrišemo trenutno smer v kotnih stopinjah skupaj s približno smerjo: N (sever), S (jug), NE (severovzhod), itd. Pod "radarjem" pa izpišemo radij trenutnega območja, oz. oddaljenost najbolj oddaljene zanimive točke.

5.1.10 Notifications

Je razred, ki srbi za prikaz obvestil. Hrani seznam sporočil, ki jih prikazuje izmenično v intervalu treh sekund. Omogoča dodajanje in odstranjevanje teh sporočil in enega sporočila s prioriteto, ki ima prednost pred ostalimi in ostane prikazano tudi, če so v seznamu druga sporočila. Primer prioritetnega sporočila je, da noben od lokatorjev ni vklopljen.

5.1.11 Drugi razredi

Poleg vseh naštetih razredov, je v knjižnici še nekaj bolj pomožnih razredov, ki jih velja omeniti:

PaintSurface je nekakšen ovoj za `android.graphics.Canvas`, ki nudi nekaj pomožnih metod, kot na primer `paintObj(ScreenObj obj, float x, float y, float rotation, float scale)` za izris objekta `obj` pod kotom `rotation` in povečanega za faktor `scale`.

ScreenObj je abstrakten razred, ki predstavlja objekte, ki jih lahko narišemo na objekt razreda `PaintSurface`. Nekaj njegovih podrazredov smo že spoznali: `Marker`, `Radar` in `Notifications`.

TextObj je še en podrazred razreda `ScreenObj` in izriše določeno besedilo v okvirju. Lahko mu nastavimo velikost pisave, širino robu, barve besedila, ozadja in okvirja ter omejitev širine (to zagotovi tako, da besedilo prelomi v več vrst, če je potrebno). Uporablja se za prikaz besedila pod oznako in obvestil.

PhysicalPlace pomeni lokacijo v realnem svetu, določeno z geografsko višino in širino ter nadmorsko višino. Metoda `locationToVector(...)` pretvori to lokacijo v koordinate v koordinatnem sistemu naprave.

Matrix pomeni matriko in nudi matrične operacije: seštevanje, skalarno in matrično množenje, inverzijo, transponiranje, izračun determinante in kreiranje enostavnih rotacijskih matrik za rotacijo okoli x, y ali z osi.

MixVector pomeni tridimenzionalni vektor in nudi vektorske operacije: seštevanje, odštevanje, množenje, deljenje, skalarni in vektorski produkt in normo.

Poglavje 6

Primer uporabe knjižnice v aplikaciji Odpiralni Časi

V prejšnjem poglavju smo predstavili delovanje naše knjižnice MixareLib, tu pa želimo predstaviti še primer njene uporabe v konkretnem programu in z realnimi podatki.

Gre za aplikacijo Odpiralni Časi [8] na mobilni platformi Android, ki izkorišča obogateno resničnost za prikaz bližnjih restavracij, barov, bank, trgovin ipd. in njihovih odpiralnih časov.

6.1 Na kratko o aplikaciji Odpiralni Časi

Aplikacija nudi naslednje informacije o poslovalnicah:

- Ime in opis
- trenutno stanje (odprto ali zaprto in kdaj zapira)
- razdalja od trenutne lokacije
- naslov
- odpiralni čas
- študentski boni
- brezžični internet (wi-fi)
- slike

- in drugo

Poslovalnice lahko iščemo po kategorijah in podkategorijah ali pa s podrobno poizvedbo. Pri poizvedbi se upošteva trenutna lokacija, če je to le mogoče (vklopljen GPS ali drug način pridobivanja lokacije). Rezultate si lahko ogledamo v seznamu, na zemljevidu, ki prikazuje lokacije vseh poslovalnic skupaj s trenutno lokacijo, ali v obogateni resničnosti. Uporabniki lahko dodajajo nove poslovalnice (ki pa gredo čez potrditev) in slike. Vse podrobne informacije o določeni poslovalnici si lahko ogledajo v podrobnem pogledu, ki prikazuje vse omenjene informacije, galerijo slik in zemljevid z označeno trenutno lokacijo in lokacijo poslovalnice.

Informacije o poslovalnicah aplikacija pridobi iz strežnika, ki ji posreduje seznam poslovalnic, ki ustreza poizvedbi in trenutni lokaciji.

Nas od vseh omenjenih funkcij, v tem diplomskem delu, zanima le prikaz poslovalnic v obogateni resničnosti.

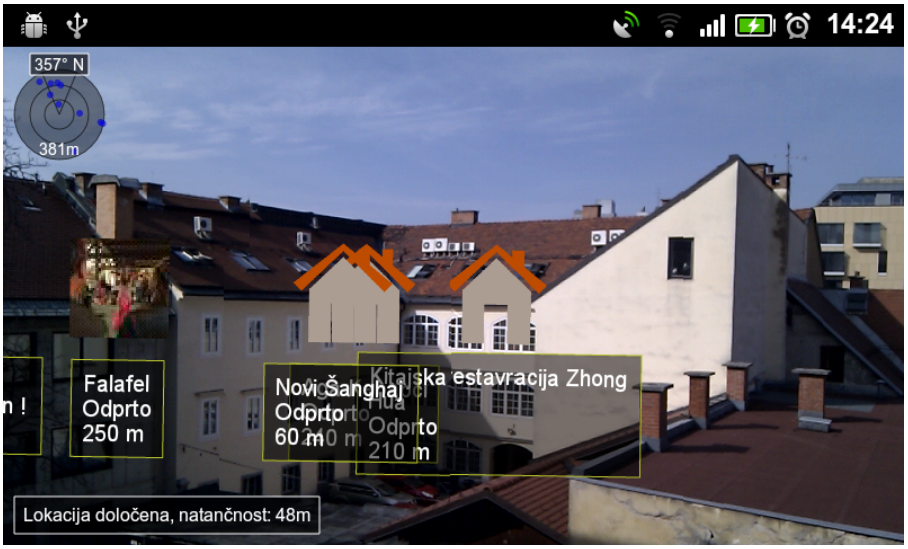
6.2 Obogatena resničnost v Odpiralnih Časih

Za prikaz poslovalnic v obogateni resničnosti uporabljamo našo knjižnico. V ta namen smo razširili razreda `ARActivity` in `Marker` in dobili razreda `Spotter` in `SpotMarker`.

`Spotter` skrbi za prenos podatkov o poslovalnicah iz strežnika, za kar potrebuje trenutno lokacijo. Za to razširja metodo `ARActivity.onLocationChanged (Location location)`, ki se izvede ob spremembi lokacije. Šele ko ima na voljo lokacijo, v ozadju odda zahtevo strežniku in pridobi podatke o bližnjih poslovalnicah. Nato za vsako poslovalnico naredi objekt razreda `SpotMarker`, ki poskrbi za oblikovanje besedila in prenos sličice v ozadju. Te objekte nato zapakira v seznam razreda `Markers` in jih nastavi za prikaz.

`Spotter` poskrbi tudi za obvestitev uporabnika med prenašanjem podatkov. Pri tem nastavi prioritetno sporočilo "Prenašam podatke...".

Naš pogled obogatene resničnosti v Odpiralnih Časih lahko vidimo na sliki 6.1.



Slika 6.1: Zaslonska slika obogatene resničnosti v aplikaciji Odpiralni Časi.

Poglavje 7

Zaključki

7.1 Rezultati

V tem diplomskem delu smo opisali delovanje obogatene resničnosti za prikaz zanimivih točk, raziskali nekaj obstoječih aplikacij oz. platform, ki izkoriščajo to tehnologijo in razvili knjižnico, ki omogoča, da na enostaven način dodamo to zmogljivost poljubni aplikaciji na sistemu Android. S knjižnico smo dosegli veliko večjo fleksibilnost, kot jo ponujajo obstoječe platforme (Layar, Wikitude, Mixare) tako pri prikazu zanimivih točk, odzivanju na uporabnikove dotike in obveščanju uporabnika o dogodkih v ozadju. Na koncu smo prikazali uporabo knjižnice v aplikaciji Odpiralni Časi.

7.2 Težave in nadaljnje delo

Pri implementaciji obogatene resničnosti seveda ni šlo brez težav. Prva težava je šum, ki se pojavi pri merjenju s senzorji pospeška in magnetnega polja. Težavo smo rešili tako, da rotacijsko matriko, ki jo izračunamo iz podatkov merjen, povprečimo. A ta rešitev nas pripelje do nove težave. Spremembe rotacijske matrike so namreč zdaj počasne tudi takrat, kadar uporabnik napravo premakne. Posledica so mehki premiki oznak, ki se premaknejo na pravo mesto z rahlo zamudo. Oznake tako ne sledijo premikom slike kamere v realnem času in zato niso videti "prilepljene" na določeno točko v realnem svetu, kot bi si v idealnem primeru želeli.

Informacija, ki jo naša implementacija ne izkorišča (in druge prav tako ne), je slika iz kamere. To namreč le prikazujemo. Z uporabo tehnik računalniškega vida bi lahko izkoristili tudi ta vir informacij za bolj natančno detekcijo pre-

mikov in rotacij naprave.

Še ena težava s senzorji pospeška in magnetnega polja je njihova (ne)natančnost. Del težave je potreba po občasni kalibraciji teh senzorjev. Kalibriramo jih tako, da z napravo nekajkrat orišemo navidezno številko 8 po zraku. Nekalibrirani senzorji namreč vračajo zaporedje nenatančnih, včasih zelo nihajočih vrednosti. Posledica so nepravilne pozicije oznak. A to je težava strojne opreme, na katero nimamo vpliva. Vse kar lahko naredimo je, da o potrebi po kalibraciji opozorimo uporabnika.

Drugi del težave z (ne)natančnostjo senzorjev je razlika v tej med različnimi napravami. Ker je število naprav s sistemom Android že veliko in se samo še povečuje, poleg tega pa je spekter teh naprav in s tem kvalitet senzorjev širok, ta problem ni zanemarljiv. Težavo bi lahko omilili s testiranjem knjižnice na čim več različnih napravah in posledično manjšimi prilagoditvami zaznave tudi slabšim senzorjem.

Problemi pa niso omejeni le na senzorje. Pogosto se zgodi, da se več zanimivih točk nahaja v (približno) isti smeri glede na trenutno lokacijo, zato njihove oznake narišemo na (približno) istem mestu. Posledica sta dva problema: kako prikazati te oznake, da bodo vse čim bolj vidne, in kako lahko uporabnik izbere (pritisne na) oznako, ki je prekrita z drugo. Platformi Layar in Wikitude rešujeta to težavo s posebnim okvirjem na spodnji strani zaslona, ki prikazuje informacije o posameznih zanimivih točkah in omogoča navigacijo med njimi. Slabost te rešitve je zmanjšan (skoraj prepolovljen) prostor za prikaz oznak zanimivih točk.

Aplikacija Mixare in naša knjižnica prvo težavo rešita z delno prosojnostjo napisov, druga pa ostaja odprta. Možna rešitev le-te bi bil prikaz manjšega okenca s seznamom tistih zanimivih točk, ki so pod uporabnikovim prstom ob pritisku na zaslon, seveda le takrat, kadar je teh točk več. S tem bi ohranili ves prostor za prikaz oznak, hkrati pa zagotovili možnost izbire vsake zanimive točke.

Na koncu omenimo še eno težavo bolj strojne narave. Za obogateno resničnost izkoriščamo veliko naprav: kamero, senzorje in GPS, za pridobitev podatkov pa tudi modem. Vse to seveda poleg običajnih zmogljivosti (procesorja, pomnilnika, zaslona itd.), ki jih potrebujemo za poganjanje in prikaz aplikacije. Vse te naprave, še posebej pa prve tri omenjene, za delovanje potrebujejo precej energije. Prikaz obogatene resničnosti zato močno obremeni akumulator, ki ga mobilne naprave uporabljajo kot vir energije. Ta se zato ob daljši uporabi obogatene resničnosti lahko hitro izprazni.

Slike

Literatura

[1] Activity. Dostopno na: http://developer.android.com/reference/android/app/Activity.html, 2011.

[2] Android-AR-Kit. Dostopno na: https://github.com/haseman/Android-AR-Kit/, 2011.

[3] AR.Drone. Dostopno na: http://ardrone.parrot.com/, 2011.

[4] Augmented reality. Dostopno na: http://en.wikipedia.org/wiki/Augmented_reality, 2011.

[5] HUD (Heads-up display). Dostopno na: http://en.wikipedia.org/wiki/Head-up_display, 2011.

[6] Layar Reality Browser. Dostopno na: http://www.layar.com/download/android/, 2011.

[7] Mixare (mix Augmented Reality Engine). Dostopno na: http://www.mixare.org/, 2011.

[8] Odpiralni Časi. Dostopno na: http://www.odpiralnicasi.com/, 2011.

[9] Rotation matrix. Dostopno na: http://en.wikipedia.org/wiki/Rotation_matrix, 2011.

[10] Wikitude World Browser. Dostopno na: http://www.wikitude.org/, 2011.

[11] Reto Meier. *Professional Android 2 Application Development.* Wiley Publishing, Inc., 10475 Crosspoint Boulevard, Indianapolis, IN 46256, 2010. chapter 14, pages 457-470.

[12] prof. dr. Aleš Leonardis and Tomaž Kranjec. Detekcija 3D objektov in prepoznavanje ključnih točk v realnem času (FERN), Obogatena resničnost. Dostopno na: http://vicos.fri.uni-lj.si/data/alesl/semKranjec.pdf, 2009.