UNIVERZA V LJUBLJANI

FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Marko Troha

# Mobilna letalska navigacija z obogateno resničnostjo

DIPLOMSKO DELO

VISOKOŠOLSKI STROKOVNI ŠTUDIJSKI PROGRAM PRVE STOPNJE RAČUNALNIŠTVO IN INFORMATIKA

Ljubljana, 2014

UNIVERZA V LJUBLJANI

FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Marko Troha

# Mobilna letalska navigacija z obogateno resničnostjo

DIPLOMSKO DELO

VISOKOŠOLSKI STROKOVNI ŠTUDIJSKI PROGRAM PRVE STOPNJE RAČUNALNIŠTVO IN INFORMATIKA

MENTOR: viš. pred. dr. Robert Rozman

Ljubljana, 2014

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Izdelajte programsko opremo za mobilno navigacijsko napravo – pripomoček, ki bo pilotom v športnem letalstvu omogočal lažjo orientacijo in navigacijo še posebej pri slabši vidljivosti. Za to izberite platformo, ki bo omogočala čimbolj praktično izvedbo in uporabo te opreme s pomočjo t.i. obogatene resničnosti, kjer na živo sliko s kamere dodate pomembnejše navigacijske točke (letališča, oddajnike). Preučite delovanje vseh potrebnih tipal in določite ter implementirajte še vse potrebne postopke za izračune pri določanju lokacije in nagiba naprave ter vseh potrebnih projekcij za prikaz vnaprej vnesenih navigacijskih točk, ki pilotu prikazujejo njihov položaj v skladu s trenutnim pogledom na napravi.

# IZJAVA O AVTORSTVU DIPLOMSKEGA DELA

Spodaj podpisani Marko Troha, z vpisno številko **63020304**, sem avtor diplomskega dela z naslovom:

*Mobilna letalska navigacija z obogateno resničnostjo*

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom viš. pred. dr. Roberta Rozmana
- 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 na svetovnem spletu preko univerzitetnega spletnega arhiva.

V Ljubljani, dne 16. septembra 2014 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.*

# Kazalo

# Seznam uporabljenih kratic

| kratica | angleško | slovensko |
|---|---|---|
| **GPS** | global positioning system | globalni pozicijski sistem |
| **VHF** | very high frequency range radio (30MHz – 300MHz) | radio za zelo visoke frekvence (30 MHz–300 MHz) |
| **VOR** | VHF Omni-directional radio | vsesmeren radio VHF |
| **NDB** | non-directional beacon | neusmerjen oddajnik |
| **VFR** | visual flight rules | pravila vizualnega letenja |
| **IFR** | instrument flight rules | pravila inštrumentalnega letenja |
| **JDK** | Java development kit | razvojno okolje za Javo |
| **SDK** | software development kit | programsko razvojno okolje |
| **WYSIWYG** | what-you-see-is-what-you-get | princip grafičnega oblikovanja |
| **GLSL** | OpenGL shading language | programski jezik knjižnice OpenGL za senčilnike |
| **WGS84** | world geodetic system 1984 | svetovni geodetski sistem iz leta 1984 |

# Povzetek

Cilj diplomske naloge je izdelati pripomoček za večjo varnost v športnem letalstvu, ki bo pilotu med letom pomagal pri orientaciji in navigaciji, zlasti v razmerah s slabšo vidljivostjo. Zaradi čimbolj praktične izvedbe in uporabe pripomočka ta temelji na platformi Android, na kateri velika večina današnjih elektronskih naprav že ponuja dovolj tipal za orientiranje in določanje lokacije teh naprav (tipala kot so GPS, magnetometer, pospeškometer, žiroskop). Aplikacija je izdelana tako, da zajema in obdeluje podatke vseh teh orodij, zasnovana pa je na konceptu obogatene resničnosti, kjer se na podlago žive slike s kamere, ki jo prav tako ima večina elektronskih naprav, izrisujejo ključne orientacijsko-navigacijske točke v letalstvu (letališča, sredstva VOR in sredstva NDB).

**Ključne besede:** letalstvo, navigacija, obogatena resničnost, Android

# Abstract

The goal of this thesis is to create a tool for greater safety in general aviation, to help the pilot in orienting and navigating while flying, particularly in low visibility conditions. In order to enhance the practicality and usage of the implementation, the application is based on Android platform, which is present on the vast majority of today's electronic devices and already provides sufficient tools for orientation and determining the location of these devices (sensors such as GPS, magnetometer, accelerometer, as well as gyroscope). The application is designed to capture and process data from all these tools; it is based on the concept of augmented reality, where live image from the camera (also present on most electronic devices) represents background, which is then overlaid with key orientation and navigation points in aviation (airports, VOR and NDB beacons).



**Keywords:** aviation, navigation, augmented reality, Android

# Poglavje 1 Uvod

Znova in znova lahko v medijih zasledimo, da se dogajajo letalske nesreče. Tisti, ki letalstvo spremljajo pobližje, o tem slišijo in vidijo še več. Cilj te diplomske naloge je preučiti možnosti in zasnovati pripomoček za izboljšanje varnosti pri letenju. Civilno letalstvo ima že dalj časa vpeljana zelo dobro opredeljena pravila in postopke, ki bistveno pripomorejo k zagotavljanju varnosti v vseh ozirih, tako na zemlji kot v zraku. Zato se bomo osredotočili na športno letalstvo, kjer so pravila bolj ohlapna kot v civilnem letalstvu. Športni pilot se namreč ne ravna po pravilih inštrumentalnega letenja (IFR), ampak po pravilih vizualnega letenja (VFR) [11]. Tako se mu tudi ni potrebno zanašati na radio-navigacijska sredstva, ki mu lahko služijo za orientacijo. Ta sredstva so radijski oddajniki z javno objavljeno zemljepisno širino in dolžino, njihovo frekvenco oddajanja in identifikacijsko kodo. V razmerah s slabšo vidljivostjo zagotavljajo pilotom orientacijo in navigacijo od vzleta do pristanka. Če se torej sredi leta razmere bistveno poslabšajo in se športni pilot ne zna orientirati s pomočjo oddajnikov, lahko zelo hitro zaide v resne težave. Toliko bolj, če leti na nepoznanem terenu, npr. v tujini.

V ta namen se torej pripravi aplikacija na platformi Android (v nadaljevanju platforma), ki na zaslon razmešča in izrisuje pozicije oddajnikov in letališč. Aplikacija prikazuje živo sliko s kamere in na tej sliki na podlagi orientacije in geolokacije naprave označi lokacije (točke) oddajnikov in letališč. Tako se lahko pilot in njegov kopilot enostavno orientirata tudi v razmerah s slabšo vidljivostjo. Za potrebe izračuna in prikaza je treba navigacijske točke predhodno vnesti v napravo, saj med letom običajno ni časa za iskanje koordinat točk s počasno internetno povezavo.

# Poglavje 2 Platforma Android

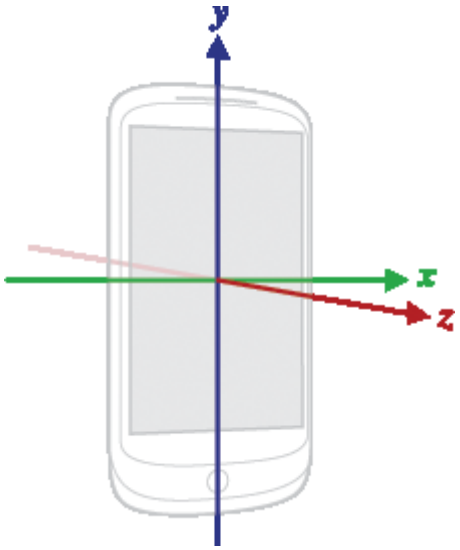
Kot ogrodje naše aplikacije smo vzeli platformo Android, ki izpolnjuje določene predpogoje za rešitev našega problema. Aplikacija potrebuje vhodne podatke o trenutni geolokaciji in orientaciji naprave ter prikaz žive slike s kamere na zaslonu.

Velika večina mobilnih naprav Android že vsebuje vso potrebno strojno opremo. Ker je tu govora o potrošniški elektroniki, je treba upoštevati morebitno nenatančnost merilnih inštrumentov – tipal [7], ki so v napravi. Zato je toliko bolj pomembno, da se pred zajemom in interpretacijo meritev te inštrumente umeri. Če ima inštrument več tipal (npr. za vsako os/dimenzijo svojega), je potrebno umeriti vsakega posebej. Ob tako umerjenih inštrumentih dobimo korekcijske faktorje, ki jih moramo upoštevati pri odčitavanju vrednosti in pred njihovo nadaljnjo interpretacijo.

Dodatno grafično ogrodje, ki ga ponuja platforma, se imenuje OpenGL ES. Z ogrodjem si lahko precej poenostavimo vizualno predstavitev objektov v 3-dimenzionalnem prostoru, saj ima vgrajeno celo vrsto različnih računskih, geometrijskih, transformacijskih in drugih operacij. Vse te operacije so tudi strojno pospešene na ločenem grafičnem procesorju, zaradi česar je glavni procesor v veliki meri razbremenjen zahtevnih grafičnih izračunov in lahko opravlja druge naloge.

## 2.1 Merilnik pospeškov

Pospeškometer v realnem času meri pospešek naprave v treh dimenzijah prostora (vertikala, horizontala, globina), ki jih prikazuje Slika 2.1. Če naprava miruje ali pa se nepospešeno giblje v določeni smeri, lahko na vseh treh oseh tipala izmerimo in odčitamo, kolikšen delež težnosti je izražen v smeri posameznih osi. Če vse tri vrednosti vektorsko seštejemo, dobimo vektor težnosti. Vektor je tako usmerjen proti središču Zemlje in ima vrednost $g \approx -9{,}81\ m/s^2$. Za nadaljnje izračune je pomembna zgolj smer vektorja, zato ga lahko normaliziramo.




Slika 2.1: Prikaz osi pospeškometra. [13]

## 2.2 Merilnik magnetnega polja

Magnetometer v realnem času meri intenzivnost magnetnega polja v vseh treh dimenzijah prostora. Izmerjene vrednosti so različne glede na to, kje v magnetnem polju se naprava nahaja. Če okrog naprave ni prisotno umetno povzročeno magnetno polje (trajni magnet ali električen vir magnetnega polja), naprava zazna in izmeri zgolj intenzivnost magnetnega polja Zemlje. Iz vseh treh izmerjenih vrednosti lahko izračunamo vektor, ki je usmerjen vzdolž magnetnih silnic in ima moč enako intenzivnosti magnetnega polja. Ta vektor je treba razstaviti na horizontalno in vertikalno komponento. Pri tem mora biti horizontalna komponenta pravokotna na vektor težnosti, vertikalna pa z njim vzporedna. Za potrebe določanja smeri neba je zanimiva zgolj horizontalna komponenta. Ker pa vemo, da Zemljin magnetni severni pol ni poravnan z njenim pravim severnim polom, je treba horizontalni komponenti prišteti še odmik.

## 2.3 Žiroskop

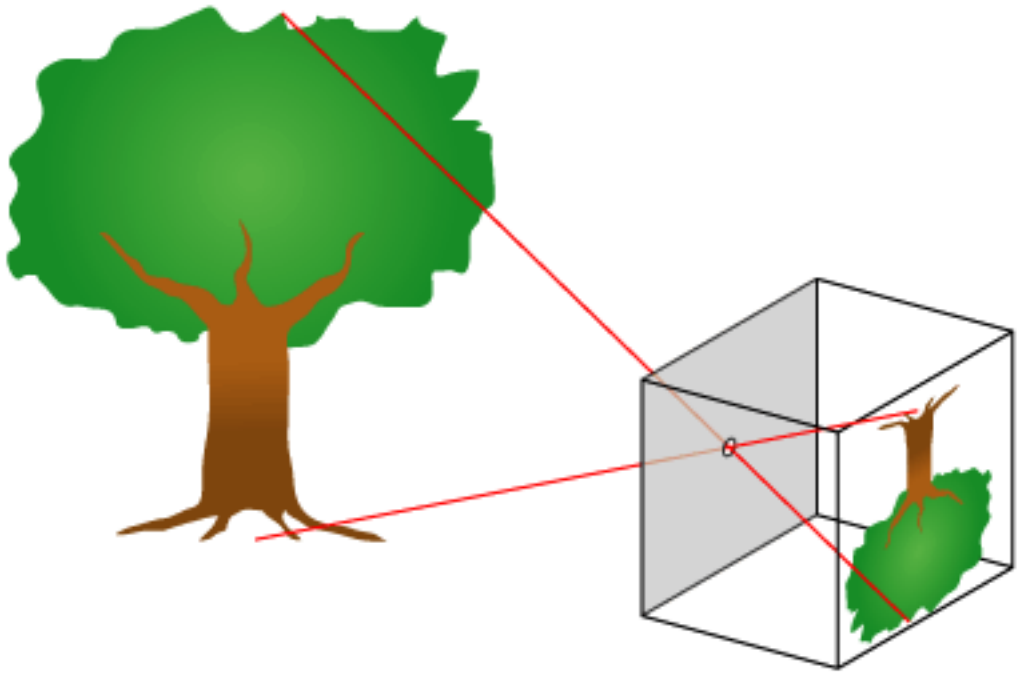
Žiroskop v realnem času meri vrtilno hitrost okrog vseh treh svojih osi. Meritve so izražene v radianih na sekundo (rad/s). Za potrebe določanja orientacije naprave v prostoru bi lahko žiroskop tudi opustili, vendar njegova uporaba bistveno izboljša zaznavo orientacije naprave v določenih pogojih (npr. med linearnim pospeševanjem naprave ali hitrimi zasuki).

## 2.4 Tipalo GPS

Tipalo za zaznavo signalov GPS in njihovo obdelavo poskrbi za določanje geolokacije naprave na planetu Zemlja. Za svoje delovanje mora v danem trenutku sprejeti signal z vsaj štirih satelitov in meriti čas, ki ga signal potrebuje od posameznega satelita do naprave. Čas oddaje signala je zapisan v samem signalu.

## 2.5 Kamera

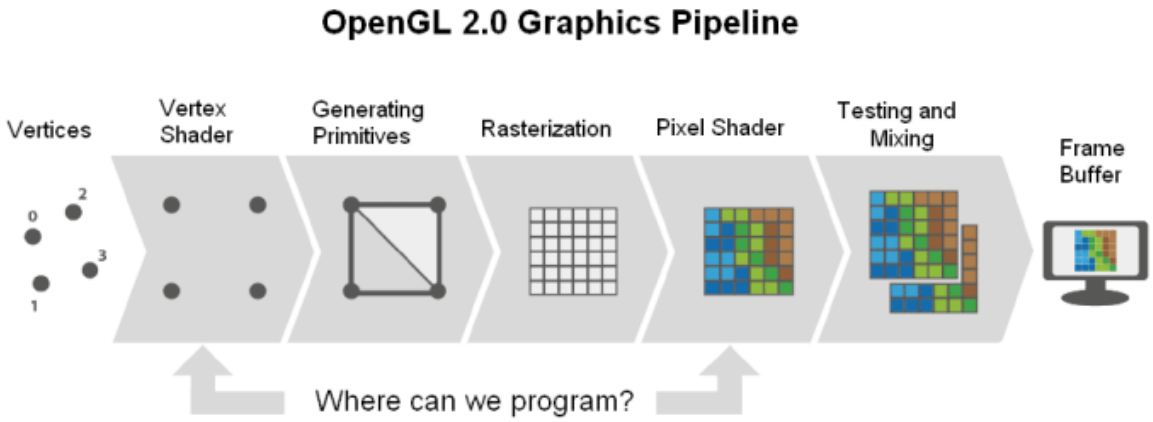
Kamere na prenosnih napravah lahko zajemajo posamezne slike ali pa sekvence slik. Takim kameram običajno pravimo kamere »pin-hole«. Zanje je značilno, da imajo tovarniško nastavljeno in nespremenljivo goriščno razdaljo in določeno število prednastavljenih velikosti zajetih slik. Uporabnik (aplikacija) lahko izbira samo med temi velikostmi. Število zajetih slik na sekundo je odvisno od osvetlitve ambienta in se samodejno prilagaja.



Slika 2.2: Zelo poenostavljena shema kamere »pin-hole«. [5]

# Poglavje 3 Grafično ogrodje OpenGL

Za prikaz pomembnih navigacijskih in orientacijskih točk je bilo izbrano ogrodje OpenGL ES 2.0 [9], [1], [2], [3]. Za razliko od različice 1.1, ki podpira zgolj statičen cevovod s širokim naborom funkcij, omogoča različica 2.0 zmogljivejši programabilen cevovod. Programabilni so tako senčilniki vozlišč kot tudi senčilniki fragmentov (delčki poligona). Slika 3.1 prikazuje posamezne stopnje cevovoda OpenGL, ki si sledijo od leve proti desni. Programer najprej poda seznam vozlišč in eno izmed metod, ki opisuje, kako so ta vozlišča med seboj povezana. Način, kako se ta vozlišča razporedijo v tridimenzionalnem prostoru, določa program senčilnika vozlišč. Kako se posamezni deli poligona obarvajo, pa določa program senčilnika fragmentov.




Slika 3.1: Poenostavljena shema cevovoda OpenGL; senčilnik fragmentov je tu prikazan kot senčilnik slikovnih pik (ang. *pixel shader*). [14]

## 3.1 Inicializacija

Inicializacija OpenGL se začne z definiranjem velikosti prikaza na zaslonu. OpenGL izrisuje podobe na komponento GLSurfaceView, ta pa je umeščena v grafični vmesnik aplikacije. Ker ni nujno, da komponenta zaseda celotno površino zaslona, je potrebno v OpenGL nastaviti dimenzije te komponente. Nadalje je potrebno določiti projekcijo in v ta namen izračunati projekcijsko matriko[1]. Ta matrika se potem uporabi pri vseh senčilnikih vozlišč. Zatem se v

[1] Več o projekcijah in projekcijskih matrikah v poglavju 3.3.

OpenGL naložijo vse teksture in prevedejo vsi programi za senčenje (v nadaljevanju senčilniki).

## 3.2 Senčilniki

Senčilniki so programi, napisani v jeziku GLSL (ang. *GL Shader Language*), ki se izvajajo neposredno na grafičnem procesorju. Tak program je v aplikaciji zapisan v obliki izvorne kode in shranjen kot znakovno zaporedje (ang. *string*). Aplikacija med svojim delovanjem izvorno kodo senčilnika naloži in prevede v binarno obliko. Po en senčilnik vozlišč in en senčilnik fragmentov skupaj tvorita program. Prevedeta (ang. *compile*) se vsak posebej, potem pa se povežeta (ang. *link*). Program je vedno sestavljen iz enega senčilnika vozlišč in enega senčilnika fragmentov. V ta namen se lahko poljubno kombinira že prevedene senčilnike. Preveden senčilnik se lahko uporabi večkrat – v več programih.

### *3.2.1 Senčilniki vozlišč*

Ti senčilniki vedno določajo transformacijo tridimenzionalne točke (ang. *vertex*) na dvodimenzionalno projekcijsko ravnino. Dodatno lahko vsakemu vozlišču določijo barvo, normalizacijski vektor, koordinate teksture itd. Primer takega senčilnika prikazuje Slika 3.2. Vhodne parametre za ta del programa nastavi aplikacija, ki se izvede za vsako (vidno) vozlišče posebej. Izhodni parametri se potem pošljejo senčilnikom fragmentov.

V aplikaciji so spisani in uporabljani trije različni senčilniki vozlišč:

- *multi_color_vertex_shader.glsl*
  Uporablja se za izračun vozlišč, kjer ima vsako vozlišče svojo barvo.

- *single_color_vertex_shader.glsl*
  Uporablja se za izračun vozlišč z enako barvo.

- *texture_vertex_shader.glsl*
  Uporablja se za izračun vozlišč, katerih poligoni prikazujejo teksturo.

```
uniform mat4 mMatrix;
uniform mat4 vMatrix;
uniform mat4 pMatrix;
attribute vec4 vertexPosition;
attribute vec2 texturePosition;
varying vec2 textureCoordinate;

void main()
{
	gl_Position = pMatrix * vMatrix * mMatrix * vertexPosition;
	textureCoordinate = texturePosition;
}
```

Slika 3.2: Primer senčilnika vozlišč skupaj s teksturnimi koordinatami.

### 3.2.2 Senčilniki fragmentov

Senčilniki fragmentov so pravzaprav druga stopnja senčenja. Vhodni parametri za njihovo delovanje se prenesejo (in po potrebi interpolirajo med posameznimi vozlišči) iz senčilnikov vozlišč, določene parametre pa je mogoče nastaviti tudi v aplikaciji. Za vsak fragment poligona se torej izračuna njegova barva. Ta je sicer lahko poljubno kompleksno izračunana, vendar naša aplikacija uporabi zgolj fiksno določeno barvo za vse fragmente poligona.

V aplikaciji so spisani in uporabljani štirje različni senčilniki fragmentov:

- *multi_color_fragment_shader.glsl*
  Uporablja se za izris fragmentov poligona, kjer ima vsako vozlišče svojo barvo.

- *single_color_fragment_shader.glsl*
  Uporablja se za izris fragmentov poligona enotne barve.

- *texture_color_fragment_shader.glsl*
  Uporablja se za izris fragmentov poligona, ki prikazuje teksturo.

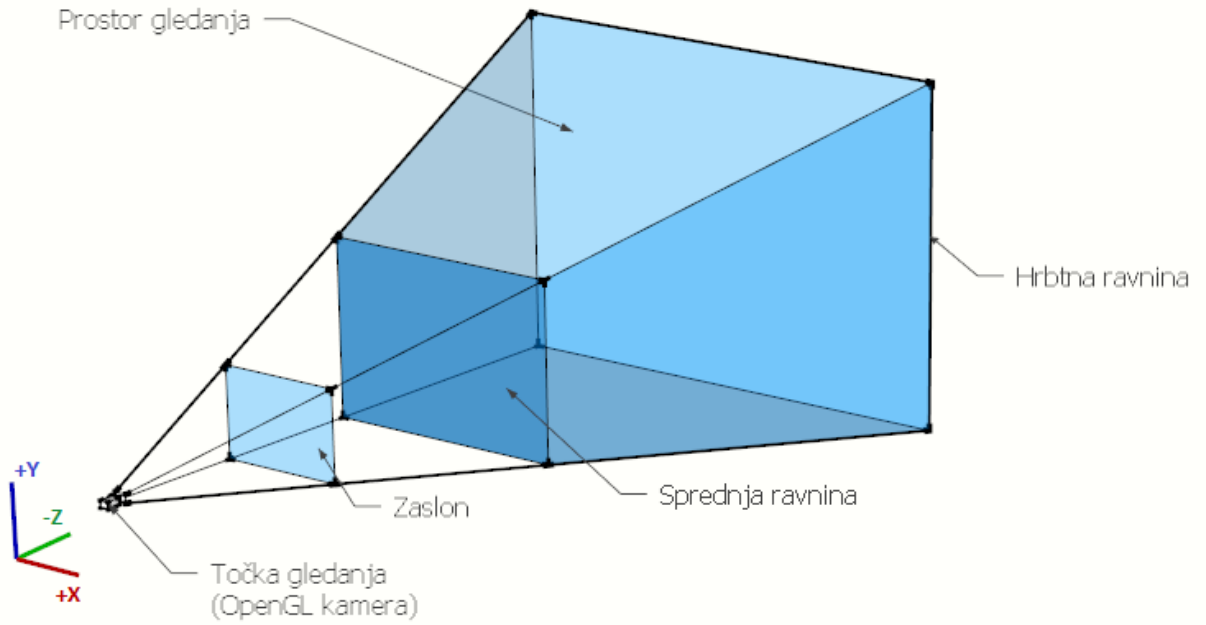
- *texture_camera_fragment_shader.glsl*
  Uporablja se za izris fragmentov poligona, ki prikazuje posebno teksturo za kamero.

## 3.3 Projekcije v OpenGL

Projekcija [4] pravzaprav določa prostor gledanja. V OpenGL obstajata v grobem dve projekciji: pravokotna vzporedna projekcija in perspektivna projekcija.

### 3.3.1 Perspektivna projekcija

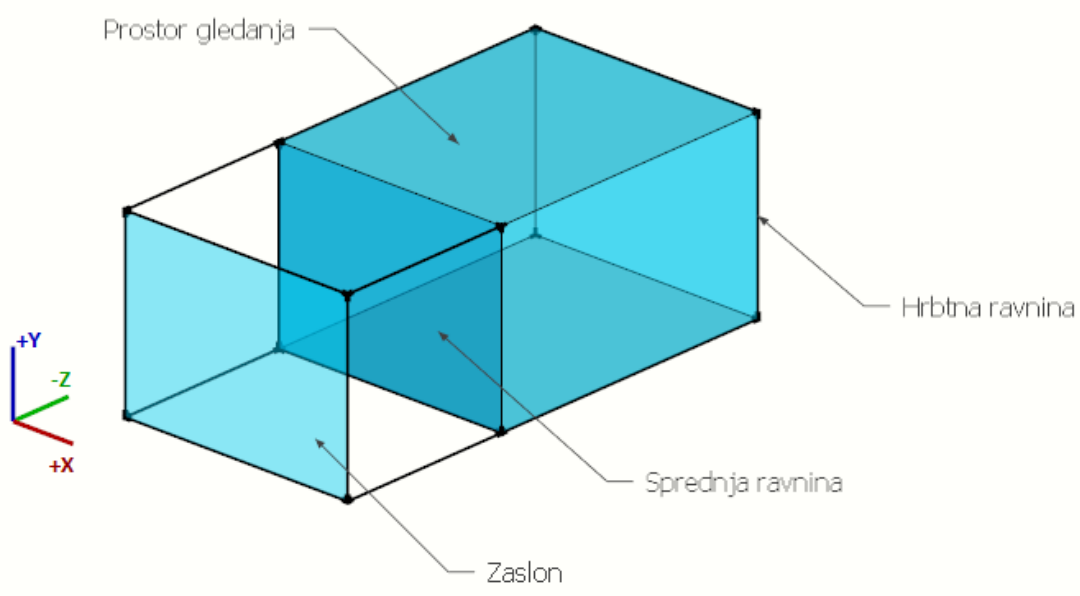
Za prikaz zanimivih točk se uporabi perspektivna projekcija (ang. *perspective projection*). V perspektivni projekciji so bolj oddaljene točke prikazane kot manjše in pravilno transformirane. Slika 3.3 prikazuje, kako je projekcija določena s sprednjo in hrbtno ravnino – ki sta med seboj vzporedni – in z razmerjem med višino in širino zaslona (ang. *aspect ratio*).




Slika 3.3: Perspektivna projekcija.

### 3.3.2 Pravokotna vzporedna projekcija

Slika kamere je odraz realne podobe sveta in tako vedno prikazuje podobo v perspektivni projekciji. Ko pa se sliko postavi v prostor OpenGL, je popolnoma nepotrebno, da se jo postavi ponovno v perspektivno projekcijo, ker ne potrebuje dodatnih transformacij. Postavi se jo v pravokotno vzporedno projekcijo (ang. *ortographic projection*), da se ohrani kote slike ne glede na postavitev in orientacijo točke gledanja, kot prikazuje Slika 3.4. S to projekcijo je tudi zelo enostavno določiti faktor povečave, kar je tudi opisano na koncu poglavja 6.1.3.




Slika 3.4: Pravokotna vzporedna projekcija.

### 3.3.3 Poravnava projekcij

Da bi izrisane točke v perspektivni projekciji sovpadale s sliko kamere v ortogonalni projekciji, je treba ti dve projekciji med seboj uskladiti. Implementirano je zelo osnovno usklajevanje projekcij z zgolj določanjem oddaljenosti sprednje ravnine perspektivne projekcije od točke gledanja (glej Slika 3.3). Pravilno nastavitev oddaljenosti je najlažje določiti z umerjanjem pred uporabo aplikacije. Za vsako napravo je to treba storiti zgolj enkrat, po namestitvi aplikacije. Od takrat dalje aplikacija vedno uporabi nastavljeno vrednost.

Če uporabnik opazi, da prikazane točke ne sledijo sliki kamere (ob zasuku naprave se točke premikajo z drugačno hitrostjo kot slika), mora izvesti ponovno umerjanje.

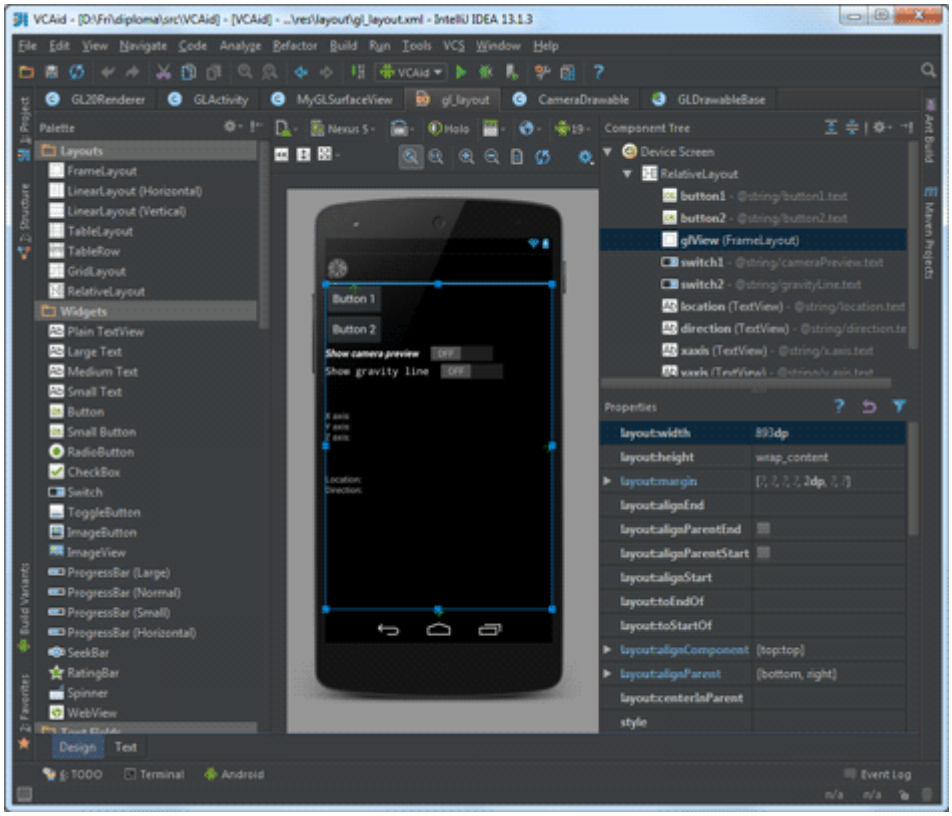
# Poglavje 4 Smernice za izdelavo diplomskega dela

Že pred izdelavo samega dela je bil zadan cilj, da se delo opravi kakovostno in v skladu s splošno sprejetimi pravili in standardi. Za lažje doseganja tega cilja je dobro sproti skrbeti, da delo poteka v skladu s pravili. V veliko pomoč so tudi uporabljena orodja, ki uporabnika ravijalca opozarjajo na ta pravila in standarde.

## 4.1 Uporabljene tehnologije in orodja

Kadarkoli se razvija aplikacijo za platformo Android, je treba najprej namestiti programski paket JDK. Zatem potrebujemo še paket Android SDK. V teh paketih je skupek orodij, potreben za prevajanje, razhroščevanje, analizo in druge postopke za razvoj aplikacij. Nekatera izmed teh orodij se uporabljajo posredno preko integracijskega okolja. Eno izmed takih okolij je IntelliJ IDEA, ki primarno omogoča razvoj javanskih aplikacij, dodatno pa nudi podporo za ogromno število ogrodij in tehnologij, med drugim tudi za razvoj aplikacij za operacijski sistem Android. Vizualno podobo lahko na preprost način izdelamo z grafičnim razporejanjem komponent po navideznem zaslonu (Slika 4.1). Orodje ima vnaprej pripravljene predloge za javanske razrede in samo poskrbi za definicije v datoteki *AndroidManifest.xml*.

Za grafično oblikovanje diagramov je bilo uporabljeno orodje yEd, za oblikovanje tridimenzionalnih modelov pa orodje SketchUp. Za enostavnejše podobe je dovolj uporaben tudi Microsoft Paint.




Slika 4.1: Prikaz grafičnega urejanja WYSIWYG v okolju IDEA.

## 4.2 Metode in dobre prakse

Izdelava aplikacije je potekala v skladu z dobrimi praksami na področju razvoja programske opreme. Uporabljena je bila že obstoječa programska oprema na platformi Android (funkcije za matrično računanje in geodetska matematika). Med razvojem se je sproti preverjala kvaliteta izvorne kode s pomočjo orodij Jenkins CI in SonarQube ter testi JUnit. Za sledenje revizijam tako izvorne kode kot tudi vse dokumentacije je bilo uporabljeno orodje Subversion. Za lažje razumevanje izvorne kode je večji del oblikovana po smernicah JavaRanch [6] in obogatena z veliko komentarji.

# Poglavje 5 Izdelava mobilne aplikacije

V tem poglavju je v grobem razložena arhitektura aplikacije, podrobneje pa so opisani podatkovni tokovi, zajem in obdelava signalov ter izračun in prikaz zanimivih točk.

## 5.1 Arhitekturna zasnova aplikacije

Aplikacija je arhitekturno zasnovana tako, da se dele implementacije lahko hitro izlušči in ponovno uporabi za rešitev kakšnega drugega podobnega problema. Tako lahko med seboj razločimo med naslednjimi programskimi sklopi:

- procesor podatkov, ki obdeluje podatke s tipal;
- uporabniški vmesnik, ki skrbi za razporejanje komponent na zaslon;
- računski algoritem, ki preračunava koordinate vseh točk;
- izrisovalnik OpenGL, ki prikazuje sliko s kamere in jo prekriva z navigacijskimi točkami;
- algoritem, ki poskrbi za pravilno shranjevanje in nalaganje virov (slik, točk, objektov).

Vsak izmed naštetih sklopov izvaja zgolj tiste operacije, ki so potrebne za njegovo delovanje, rezultate operacij pa pošilja ostalim sklopom.

## 5.2 Podatkovni tokovi

V aplikaciji obstajata dva podatkovna tokova. Prvi tok obsega zajem podatkov tipal naprave, njihovo obdelavo in uporabo. Drugi tok pa so slikovni podatki s kamere, ki služijo za grafično podlago, na katero se dodajajo navigacijske točke.

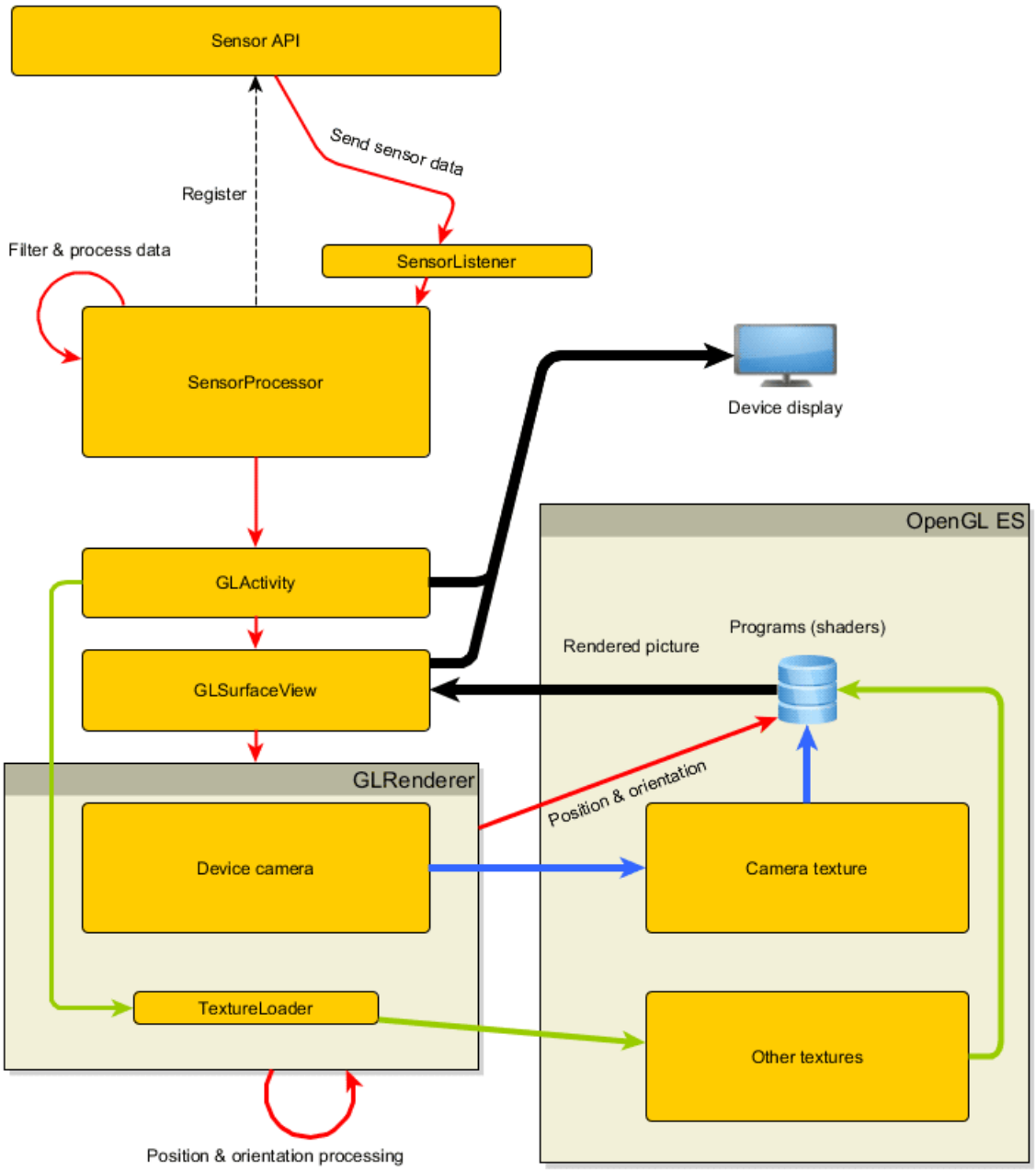
### *5.2.1 Zajem in obdelava signalov tipal*

Aplikacija sporoči platformi, da želi prejemati podatke od tipal – za nas so zanimivi pospeškometer, magnetometer in tipalo GPS. Takoj ko se aplikacija prijavi na sprejemanje

podatkov od tipal, jih platforma začne tudi pošiljati. Sveži podatki prihajajo s frekvenco pribl. 120–200 Hz. Aplikacija jih sprejme in odlaga v čakalno vrsto. Iz čakalne vrste se pobirajo s frekvenco 30 Hz in se pošiljajo obdelovalnemu algoritmu. Vrsta se pri tem izprazni. Obdelovalni algoritem, ki teče v svoji niti, te podatke pred uporabo obdela s filtrom mediane. Podatki iz magnetometra in pospeškometra se uporabijo za izračun orientacije naprave.

### 5.2.2 Slikovni tok

Pri slikovnem toku je bistveni del konfiguracija kamere in povezava med kamero in ogrodjem OpenGL ES. Poseben algoritem za odločanje, ki je bil v celoti razvit v okviru diplomskega dela, odloči o najprimernejši velikosti zajema slike s kamere. Odločitev sprejme na podlagi parametrov kamere (goriščna razdalja, razmerje dimenzij slike), možnih velikosti slik, ki jih kamera podpira, in ciljne velikosti okna, kjer se bo prikazovala slika. Po izbiri teh nastavitev se nastavi še površina (ang. surface), kjer se izrisuje zajeta slika kamere. Ker želimo čez sliko izrisovati dodatne objekte, ti objekti pa se izrisujejo v ogrodju OpenGL, moramo izrisovalno površino zgraditi v navezi s teksturo OpenGL. Kamera ob vsaki novi zajeti sliki osveži teksturo v medpomnilniku OpenGL, ogrodje pa jo potem izriše na zaslon.




Slika 5.1: Posplošen diagram tokov podatkov (rdeč tok so podatki o lokaciji in orientaciji naprave, moder tok so slikovni podatki iz kamere, zelen tok so teksture, črn pa končna slika).

## 5.3 Koordinatni sistemi

Da lahko pravilno razumemo, kako se zanimive točke izrisujejo na zaslon, moramo najprej razumeti štiri različne vrste koordinatnih sistemov, s katerimi imamo opravka. Vedeti moramo tudi kako določeno točko preslikati iz enega takega koordinatnega sistema v drugega.

### *5.3.1 Koordinatni sistem naprave*

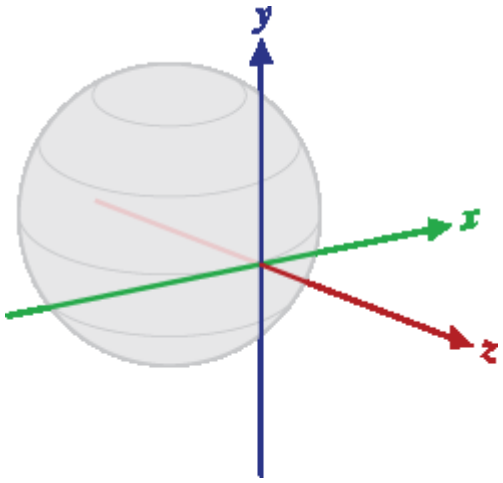
Koordinatni sistem naprave je definiran glede na zaslon naprave, kadar je ta v svoji naravni legi. Če napravo zasučemo v katerokoli drugo lego, se njen koordinatni sistem ne spremeni. Privzeta lega telefona je običajno pokončna (Slika 5.2), za tablični računalnik pa ležeča. Tipali pospeškov in magnetizma uporabljata isti koordinatni sistem.


Slika 5.2: Koordinatni sistem naprave. [12]

### *5.3.2 Referenčni koordinatni sistem*

Referenčni koordinatni sistem ali sistem realnega sveta uporabljamo za zapis koordinat zanimivih točk. Definiran je z naslednjimi osmi (Slika 5.3):

- os X je definirana kot vektorski produkt $Y \times Z$; usmerjena je približno proti vzhodu in vzporedna s tlemi;
- os Y je usmerjena proti magnetnemu severnemu polu in vzporedna s tlemi;
- os Z je usmerjena iz središča zemlje proti nebu in je pravokotna glede na tla.




Slika 5.3: Referenčni koordinatni sistem. [8]

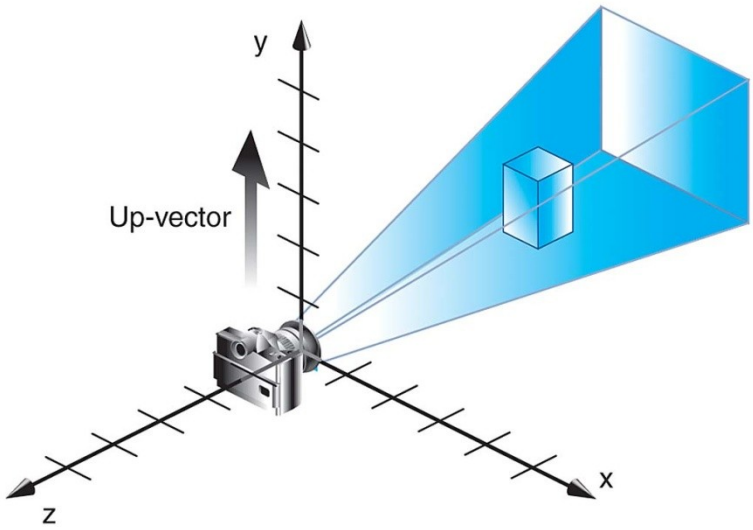
### 5.3.3 Koordinatni sistem zaslona

Zaslon naprave je dvodimenzionalen, za izris podob na zaslon pa skrbi ogrodje OpenGL. Za preslikavo in transformacije iz trodimenzionalnega v dvodimenzionalni prostor torej skrbi OpenGL glede na to, kje v prostoru se nahajajo objekti in kje točka gledišča (kamera).

### 5.3.4 Koordinatni sistem OpenGL

Eden izmed pomemben konstruktov v ogrodju OpenGL je tudi kamera [4]. S kamero namreč določamo točko gledanja in smer zajema tridimenzionalnega prizora. Koordinatni sistem OpenGL (Slika 5.4) je desnosučni, njegove osi pa so glede na kamero definirane takole:

- os X je vodoravna in usmerjena proti desni;
- os Y je navpična in usmerjena navzgor;
- os Z pa je pravokotna glede na zaslon in usmerjena iz zaslona naprave.




Slika 5.4: Kamera ogrodja OpenGL in njen koordinatni sistem. V vidnem polju kamere je trenutno kvader.

## 5.4 Obdelava podatkov tipal

Obdelava podatkov tipal se izvaja ločeno v svoji niti. Tako se lahko podatki obdelujejo zelo hitro in se pri tem ne ovira izrisovanje grafičnega vmesnika v glavni niti aplikacije. Uporabniška izkušnja je s tem izboljšana, saj je aplikacija na račun tega odzivnejša.

### *5.4.1 Zajem in predobdelava podatkov – »SensorProcessor«*

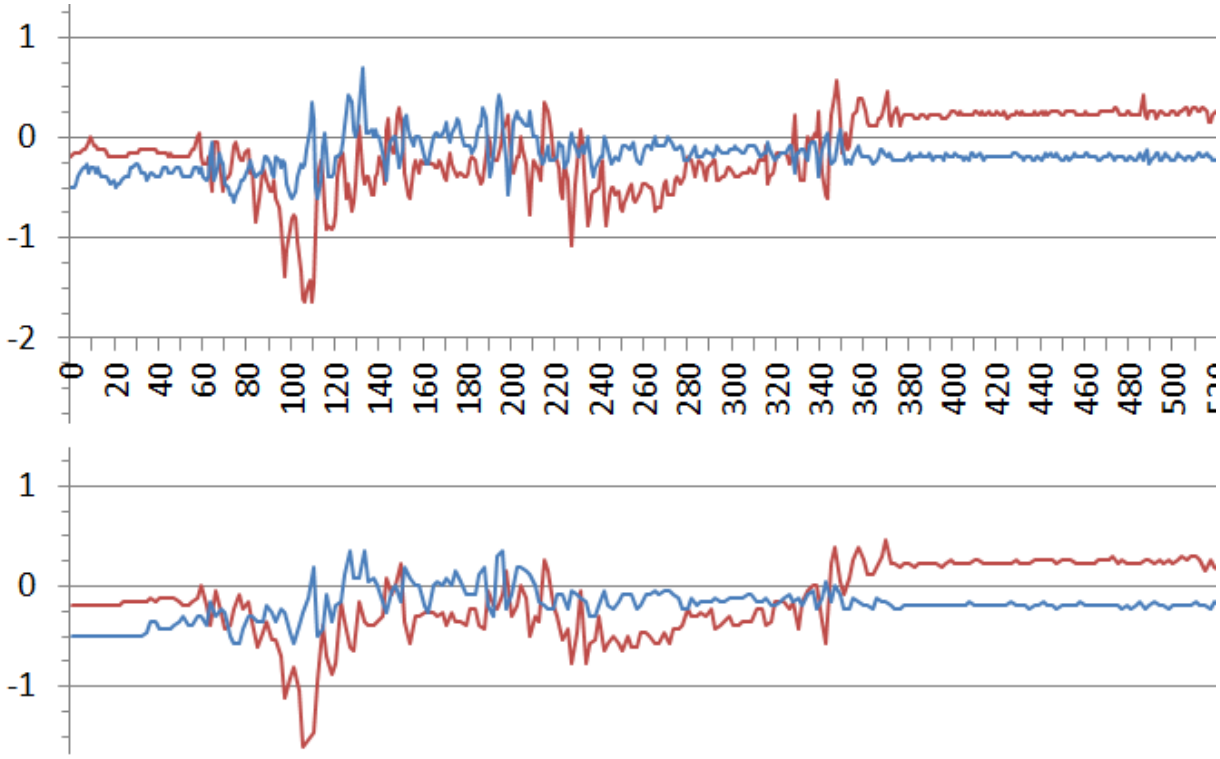
Zajeti podatki tipal vsebujejo določeno količino šuma in so kot taki neuporabni za obdelavo. Ta šum je treba iz njih izločiti tako, da se obenem v čimvečji meri ohrani osnovni signal.

#### 5.4.1.1 Rotacijski podatki

Magnetometer in pospeškometer pošiljata surove podatke s frekvenco pribl. 200 Hz. Te podatke s pomočjo razredov »SensorListener« in »LocationListener« zajema »SensorProcessor« in jih shranjuje v čakalno vrsto. Iz čakalne vrste jih s frekvenco 30 Hz zajema obdelovalni algoritem tako, da vsakič vrsto izprazni. Tako zajeti podatki imajo precej šuma in jih je zato potrebno filtrirati; najprej se jih sortira po vrednosti, končna vrednost pa se ne izračuna, temveč se izbere kot srednja vrednost sortiranih podatkov. Ta implementacija filtra se imenuje medianin filter. Filtrira se vsako os vsakega tipala posebej (5.1). Vrednosti $X_m$, $Y_m$, $Z_m$ so komponente vektorja magnetnega polja, vrednosti $X_a$, $Y_a$ in $Z_a$ pa so komponente vektorja pospeška.

$$\begin{bmatrix} x_{m1} & x_{m2} & x_{m3} & & x_{mn} \\ y_{m1} & y_{m2} & y_{m3} & \cdots & y_{mn} \\ z_{m1} & z_{m2} & z_{m3} & & z_{mn} \end{bmatrix} \xrightarrow{median\ filter} \begin{bmatrix} x_m \\ y_m \\ z_m \end{bmatrix}$$
$$\begin{bmatrix} x_{a1} & x_{a2} & x_{a3} & & x_{an} \\ y_{a1} & y_{a2} & y_{a3} & \cdots & y_{an} \\ z_{a1} & z_{a2} & z_{a3} & & z_{an} \end{bmatrix} \xrightarrow{median\ filter} \begin{bmatrix} x_a \\ y_a \\ z_a \end{bmatrix} \quad (5.1)$$

Da si je rezultat filtriranja lažje predstavljati, je na Slika 5.5 grafično prikazana razlika med zajetimi in filtriranimi podatki. Zgornji graf prikazuje odčitane vrednosti pospeškometra na oseh X in Y, spodnji graf pa te iste vrednosti, le da so filtrirane. Vrednosti so v $m/s^2$, na abscisni osi pa je naveden čas v milisekundah. Vidi se, da izbrani filter odstrani zgolj večje anomalije v signalu, obenem pa uspešno ohrani odzivnost signala.




Slika 5.5: Grafični prikaz odčitanih vrednosti: neobdelanih (zgoraj) in obdelanih (spodaj).

Iz tako filtriranih podatkov magnetometra in pospeškometra se potem izračuna rotacijsko matriko R (5.2), ki opisuje rotacijo (transformacijo) koordinat iz koordinatnega sistema naprave v referenčni koordinatni sistem. Platforma ponuja že izgrajeno metodo za izračun rotacijske matrike:

```
SensorManager.getRotationMatrix(R, null, accelerationVector,
magneticVector);
```

$$R = \begin{bmatrix} r_0 & r_1 & r_2 & 0 \\ r_4 & r_5 & r_6 & 0 \\ r_8 & r_9 & r_{10} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.2}$$

Iz rotacijske matrike lahko enostavno izračunamo rotacijo (5.3) naprave glede na osi referenčnega koordinatnega sistema.

$$\begin{aligned} smer &= \tan^{-1}\frac{r_1}{r_5} + D_m \\ naklon &= \sin^{-1} r_9 \\ nagib &= \tan^{-1}\frac{r_8}{r_{10}} \end{aligned} \tag{5.3}$$

#### 5.4.1.2 Lokacijski podatki

Lokacijsko tipalo (GPS) pošilja surove podatke s frekvenco 100–130 Hz. Te podatke zajema »SensorProcessor« in jih shranjuje v čakalno vrsto. Iz čakalne vrste jih zajema isti obdelovalni algoritem kot za rotacijske podatke in obenem vrsto izprazni. Tako kot rotacijski imajo tudi ti podatki precej šuma in jih je zato potrebno filtrirati. Tudi tu se uporablja filter mediane. Filtrira se vsako veličino posebej (5.4): zemljepisno širino, zemljepisno dolžino in tudi manj pomembne višino, smer ter hitrost.

$$\begin{bmatrix} x_1 & x_2 & x_3 & & x_n \\ y_1 & y_2 & y_3 & \cdots & y_n \\ z_1 & z_2 & z_3 & & z_n \end{bmatrix} \xrightarrow{median\ filter} \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{5.4}$$

Na podlagi povprečenih vrednosti zemljepisne širine, zemljepisne dolžine, višine in trenutnega časa poseben algoritem izračuna magnetno polje na lokaciji naprave, s tem pa se pridobi tudi podatek o odklonu magnetnega severa od pravega severa. Ta magnetni odklon ($D_m$) je treba upoštevati pri izračunu smeri naprave (5.3).

### 5.4.2 Posredovanje podatkov – »ISensorProcessorListener«

Ko so podatki o lokaciji in rotaciji naprave obdelani do te mere, da so dovolj stabilni za uporabo, se preko instance razreda »IsensorProcessorListener« prenesejo v algoritem za izračun lokacije in rotacije kamere v prostoru OpenGL.

# Poglavje 6 Izračun in prikaz navigacijskih točk

Pridobljene obdelane podatke o orientaciji in lokaciji naprave se uporabi za izračun lokacij točk na zaslonu. Za pravilen prikaz točk pa je treba najprej pravilno nastaviti kamero, sicer se točke prikzujejo na napačnem predelu slike.

## 6.1 Nastavitev kamere

Različne naprave imajo lahko različne kamere, ki se med seboj razlikujejo po goriščni razdalji, vidnem polju ali razmerju med višino in širino. Nastavitev je mogoče narediti s pomočjo odločitvenega algoritma, katerega vhodni podatki so parametri kamere (vidno polje, razmerje med višino in širino) in velikost prikaznega okna.

### *6.1.1 Razmerje velikosti prikaznega okna*

Prikazno okno ima dimenziji širino $w$ in višino $h$. Razmerje med širino in višino $ar$ je izračunano po formuli $ar_s = h_s/w_s$, pri čemer s označuje zaslon. Četudi aplikacija vedno deluje v ležeči postavitvi, je lahko prikazno okno večje po višini kot po širini (ne zaseda celotne širine zaslona). V takih primerih je $ar_s > 1$.

### *6.1.2 Razmerje velikosti slike kamere*

Kamera zna sliko običajno zajemati v več različnih velikostih in s tem tudi razmerjih. Praviloma tovrstne kamere nikoli ne zajemajo kvadratne slike in v ležeči postavitvi naprave je slika večja po širini kot po višini. Ker so velikosti zajemanja slik tovarniško določene in nespremenljive, je naloga algoritma, da izbere najustreznejšo velikost slike pri zajemanju.

### *6.1.3 Izbor velikosti in razmerja*

Če se razmerje velikosti okna in razmerje velikosti kamere ne ujemata, je treba del slike kamere odrezati. Bodisi zgornji in spodnji bodisi levi in desni rob. Lahko bi sicer predpostavili, da je najboljša izbira kar največja slika, vendar se lahko zgodi, da je vodoravni kot zajema slike manjši kot pri ostalih velikostih. Včasih je bolj pomembno, da ima slika večje vidno polje, kot pa da je zaradi večjih dimenzij bolj natančna, ostra; sploh pa če

zavzema le del zaslona. Spet druga skrajnost je prav tako neustrezna: lahko se izkaže, da se razmerje slike popolnoma ujema z njenim prikaznim oknom, vendar je ta krepko premajhna in ob povečavi postane nejasna.

Algoritem za vse velikosti slike posebej izračuna površino slike $S_c = h_c * w_c$, njeno razmerje $ar_c = h_c/w_c < 1$ ter količnik razmerij $r = ar_c/ar_s$, pri čemer c označuje kamero, s pa zaslon. Dodatno se za potrebe odločanja izračuna uteži $u_i$ glede na količnik razmerij, kar prikazuje enačba (6.1). Število n označuje količino vseh različnih velikosti zajema slik.

$$\forall i \in \{0, \quad 1, \quad \dots \quad n-1\}: u_i = \begin{cases} \sqrt[4]{-\cos(r_i \times \pi)}, & 1 \le r_i \le 1.5 \\ \cos(r_i \times \pi)^2, & 0.5 \le r_i < 1 \\ 0, & r_i < 0.5 \bigwedge r_i > 1.5 \end{cases} \tag{6.1}$$

Algoritem nadalje popravi uteži tako, da jim prišteje še količnik (6.3) med posamezno površino slike in največjo površino izmed vseh slik (6.2).

$$S_{c_{max}} = \max_{0 \le i < n} S_{c_i} \tag{6.2}$$

$$\forall i \in \{0, \quad 1, \quad \dots \quad n-1\}: u_i = u_i + {S_{c_i}}/{S_{c_{max}}} \tag{6.3}$$

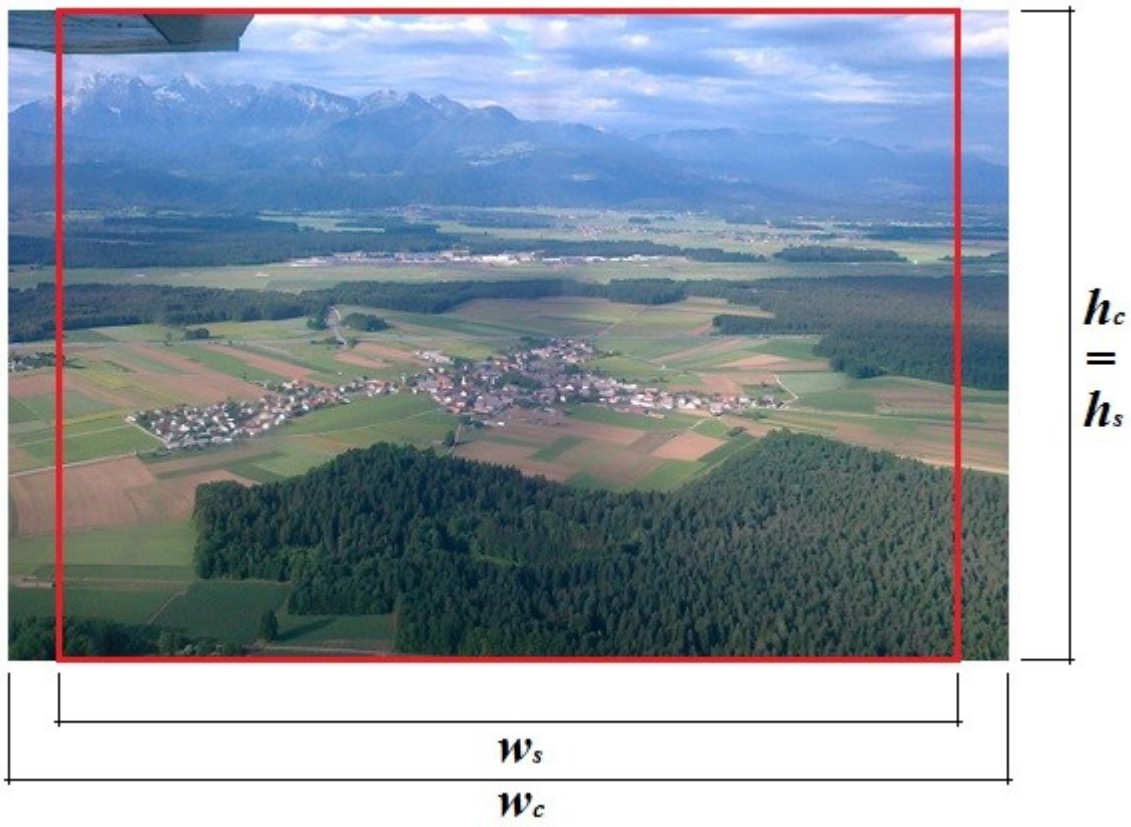
Algoritem nazadnje izbere velikost zajete slike z največjo utežjo. V primeru, da je s tem izbrani količnik razmerij $r < 1$ (sliki je potrebno porezati levi in desni rob), se to stori tako, da se poveča faktorje povečave pravokotne vzporedne projekcije v OpenGL. Platforma Android tukaj ponovno priskoči na pomoč z že vgrajeno funkcijo za izračun pravokotne vzporedne projekcijske matrike:

```
Matrix.orthoM(projectionMatrix, 0, -r, r, -arS * r, arS * r, -1001, 1001);
```

Pridobljena matrika razširi ali zoži sliko tako, da ohrani njeno razmerje med širino in višino, zgornji in spodnji rob pa sta poravnana z robom prikazovalne površine. Istočasno je seveda potrebno za enake vrednosti popraviti tudi perspektivno projekcijo s katero so prikazane navigacijske točke. Funkcijo za izračun matrike perspektivne projekcije ravno tako vsebuje platforma Android:

```
Matrix.frustumM(projectionMatrix, 0, -r, r, -arS * r, arS * r, 0.5, 10000);
```

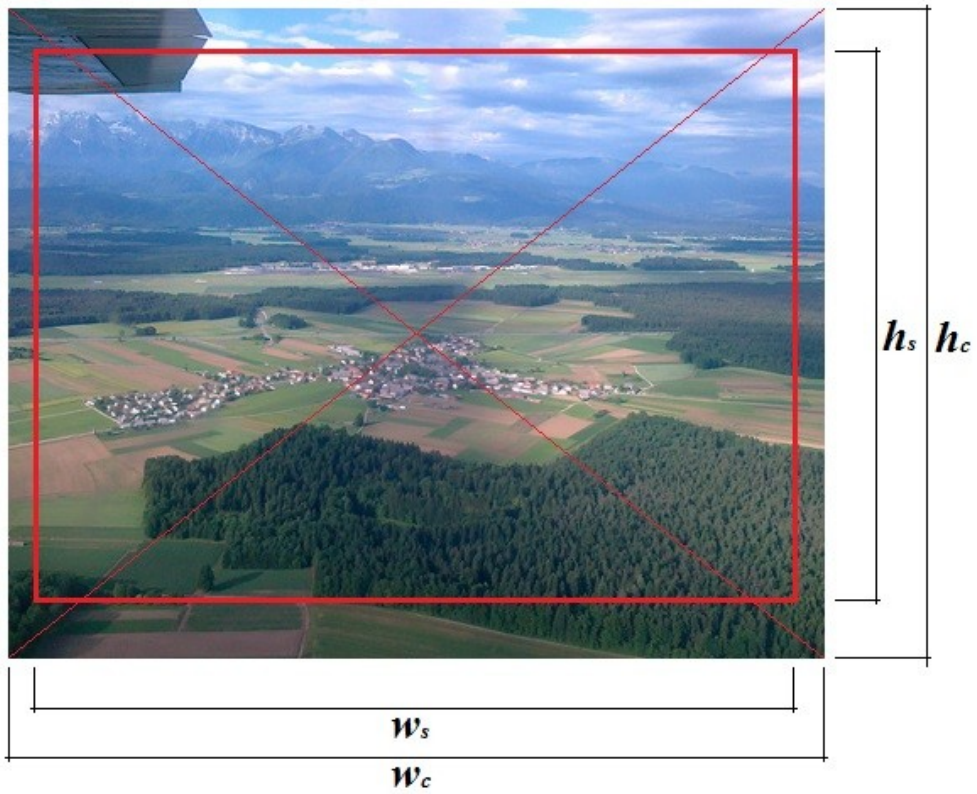
Rezultat te transformacije prikazuje Slika 6.1.



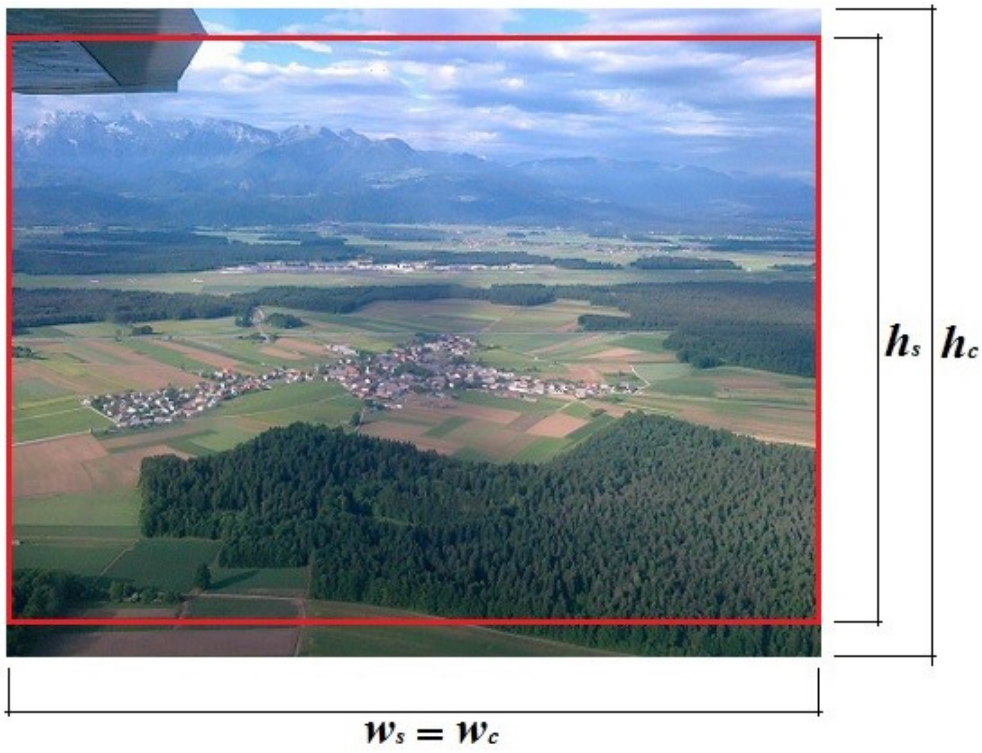

Slika 6.1: Zgornji in spodnji rob sta poravnana z prikazovalno površino.

V primeru, da je izbran količnik razmerij $r > 1$ (sliki je potrebno porezati zgornji in spodnji rob), perspektivne projekcije ni treba prilagajati. Slika se bo po širini prilagodila prikaznemu oknu, zgornji in spodnji rob pa bosta porezana – segala bosta izven vidnega polja.

Slika 6.2 prikazuje razmerja velikosti pred transformacijo pravokotne vzporedne projekcijske matrike, Slika 6.3 pa prikazuje razmerja velikosti po tej transformaciji.




Slika 6.2: Razmerja velikosti pred transformacijo.




Slika 6.3: Razmerja velikosti po transformaciji.

## 6.2 Izračun navigacijskih točk

Zaradi uporabe metod OpenGL je sam izris navigacijskih točk trivialen. Točke je treba zgolj pravilno postaviti v koordinatni sistem OpenGL. Problem, ki se ga tukaj rešuje, je, kako pretvoriti geografske koordinate točke (koordinate so v stopinjah) v koordinate OpenGL (koordinate so brez merskih enot).

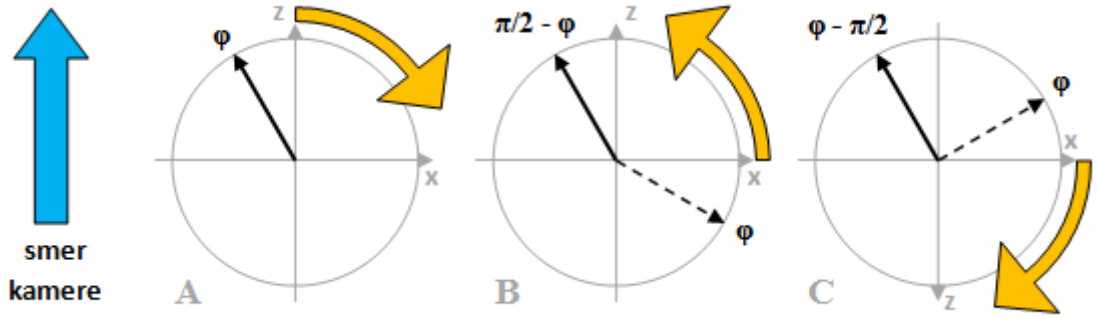
### *6.2.1 Preslikava geografskih točk v prostor OpenGL*

Platforma ima vgrajeno fukcijo za izračun oddaljenosti in nebesne smeri med dvema geografskima točkama, ki je v veliko pomoč, saj je izračun [15] matematično zelo kompleksen – po standardu WGS84 [17]. Točki sta podani z zemljepisno širino in dolžino, izračunana dolžina je v metrih, smer pa v stopinjah.

```
Location.distanceBetween(deviceLatitude, deviceLongitude, pointLatitude,
pointLongitude, results);
```

Funkcija se uporabi tako, da se kot prvo točko vnese koordinate trenutne lokacije naprave (pridobljene iz tipala GPS), kot drugo točko pa se vnese koordinate točke za izris. Izračunana oddaljenost in smer predstavljata polarne koordinate (R, φ), ki se jih pretvori v kartezične koordinate (x, z). Pri nebesnih smereh je izhodišče sever (naj bo ta na pozitivnem poltraku osi Z) in smeri se povečujejo v smeri urinega kazalca – Slika 6.4. – sistem A. V klasični matematiki je izhodišče pozitivni poltrak koordinatne osi X, smeri pa se povečujejo v nasprotni smeri urinega kazalca – Slika 6.4. – sistem B. Iz sistema A v sistem B se smer preslika po formuli $\varphi' = \frac{\pi}{2} - \varphi$. Ker pa aplikacija uporablja koordinatni sistem OpenGL, kjer je os Z obrnjena – Slika 6.4 – sistem C, je ustrezna formula za preslikavo iz sistema A v sistem C: $\varphi' = \varphi - \frac{\pi}{2}$.

Aplikacija prikazuje točke do oddaljenosti R = 200 km (200.000 m), zato je treba pridobljeno razdaljo ustrezno deliti. Glede na to, da perspektivna projekcija zaobsega največjo oddaljenost 10.000 enot, je za deljenje primeren količnik 20. Če je količnik deljenja premajhen, lahko točka pade za hrbtno ravnino projekcije in se tako ne prikaže na zaslonu.




Slika 6.4: Sistem A prikazuje nebesne smeri, sistem B prikazuje običajni koordinatni sistem, sistem C prikazuje koordinatni sistem OpenGL.

# Poglavje 7 Primer uporabe

Najprej se aplikacija namesti na napravo. Ker aplikacija ni na voljo na uradni distribucijski strani podjetja Google, je treba na napravi Android omogočiti namestitev aplikacije iz neznanih virov. Aplikacija zahteva naslednje pravice:

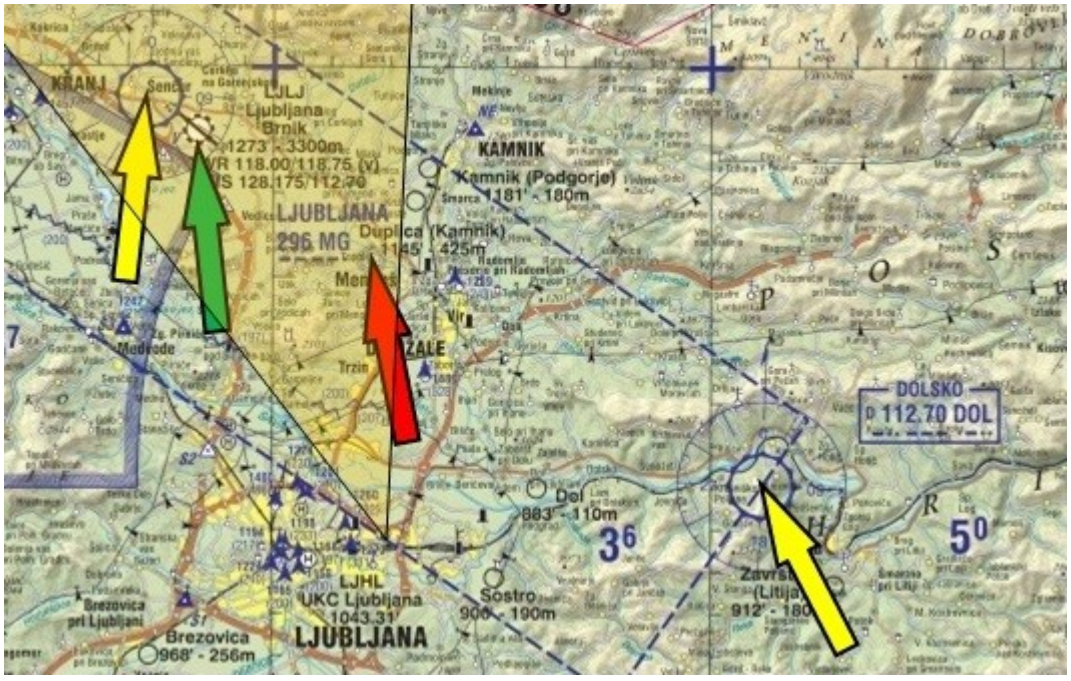
- dostop do kamere;
- dostop do lokacije;
- dostop do natančne lokacije (GPS);
- dovoljenje za preprečevanje napravi, da po določenem času samodejno ugasne zaslon.

Za svoje delovanje aplikacija zahteva tudi naslednjo strojno opremo: kamera, magnetometer, pospeškometer in tipalo GPS. Naprava mora podpirati tudi OpenGL ES 2.0. Če niso izpolnjene vse strojne in programske zahteve, aplikacija ne bo delovala ali pa se celo ne bo uspela namestiti.

Uporabnik najprej omogoči tipalo GPS, potem pa zažene aplikacijo. Ko se aplikacija zažene, je takoj vidna slika kamere. Takoj ko je lokacija GPS na voljo, se ob sliki začnejo izpisovati podatki. Hkrati se na sliki začnejo izrisovati zanimive točke. Večja kot je točka, bližje napravi se nahaja. Zelene točke označujejo letališča, rumene označujejo sredstva VOR, rdeče pa sredstva NDB.

Slika 7.1 je nastala ob preizkušanju delovanja aplikacije in prikazuje zaslon naprave v stanju, ko je ta stacionirana v BTC-ju v Ljubljani in je usmerjena proti točkam, kot je prikazano na Slika 7.2. Slednja prikazuje izsek karte VFR za Slovenijo [16]. Označene so posamezne navigacijske točke ter področje, ki ga prikazuje kamera naprave.


Slika 7.1: Prikaz zaslona naprave z označenimi oddajniki in letališčem ob pogledu iz stolpnice podjetja Comtrade v Ljubljani.




Slika 7.2: Zemljevid s pozicijami točk in navigacijske naprave; z zeleno je označeno letališče Jožeta Pučnika, z rumeno sta označeni sredstvi VOR Dolsko in Ljubljana, z rdečo je označeno sredstvo NDB Ljubljana.

# Poglavje 8 Sklepne ugotovitve

Cilj diplomske naloge je bil torej dosežen. Razvita je bila prototipna aplikacija za mobilno letalsko navigacijo na osnovi obogatene resničnosti, s katero si lahko piloti pomagajo pri navigaciji v slabih vremenskih razmerah. Uporaba aplikacije je še lažja, če z njo operira pilot – sopotnik.

Pred razvojem aplikacije so bile preučene posamezne metode in tehnologije razvoja (postavitev točk v tridimenzionalni prostor OpenGL, prikaz žive slike v OpenGL, obdelava podatkov tipal ...). Posebej pomembna dela naloge sta bila pravilna nastavitev kamere in poravnava projekcij. Brez teh navidezno postranskih delov bi bil vsakršen trud s še tako napredno obdelavo podatkov zaman.

Aplikacija je v grobem sestavljena iz treh delov: zajem in obdelava podatkov tipal, grafični uporabniški vmesnik, preračunavanje in prikaz vseh elementov obogatene resničnosti. Posamezni deli aplikacije so implementirani neodvisno drug od drugega, tako da se lahko posamezen del izvzame in uporabi drugje v podoben namen. Prav tako se lahko določen del zamenja z drugim, morda boljšim delom. Na tak način se lahko aplikaciji z lahkoto popolnoma spremeni namen in ta namesto navigacijskih točk v letalstvu prikazuje druge lokacije (npr. lokacije bank, bolnišnic, knjižnjic, restavracij, ...)

Aplikacija za svoj namen deluje dovolj solidno, vendar ima kar nekaj možnosti za izboljšave. V največji meri bi pripomogla že sama strojna oprema (tipala, kamera), kar se bo v prihodnosti najverjetneje tudi zgodilo. Tipala bodo natančnejša in odčitki vrednosti bolj kakovostni. Z izbiro boljšega ali dodatnega filtra bi bilo mogoče podatke tipal natančneje obdelati, kar bi pripomoglo k stabilizaciji točk na zaslonu. Pri kamerah je vedno prisotno popačenje slike, kar pa je mogoče popraviti z umerjanjem kamere in pravilnim izrisom slike. Uporabniško izkušnjo bi bilo zanimivo obogatiti z vidika interaktivnosti na ta način, da bi se prikazane točke odzivale na dotik, del grafičnega vmesnika pa bi prikazoval dodatne informacije o izbrani točki.

Vnos točk v napravo prek uporabniškega vmesnika v trenutni implementaciji aplikacije ni mogoč. Točke so pred samim prevajanjem statično uvožene s spletne strani OurAirports [10].

# Literatura


[1] D. Bomfim. All about OpenGL ES 2.x - (part 1/3). (4. februar 2014). *db-in* [Elektronski]. Dosegljivo: http://blog.db-in.com/all-about-opengl-es-2-x-part-1/. [Dostopano: 18. 6. 2014].

[2] D. Bomfim. All about OpenGL ES 2.x - (part 2/3). (4. februar 2014). *db-in* [Elektronski]. Dosegljivo: http://blog.db-in.com/all-about-opengl-es-2-x-part-2/. [Dostopano: 18. 6. 2014].

[3] D. Bomfim. All about OpenGL ES 2.x - (part 3/3). (4. februar 2014). *db-in* [Elektronski]. Dosegljivo: http://blog.db-in.com/all-about-opengl-es-2-x-part-3/. [Dostopano: 18. 6. 2014].

[4] D. Bomfim. Cameras on OpenGL ES 2.x - The ModelViewProjection Matrix. (4. februar 2014). *db-in* [Elektronski]. Dosegljivo: http://blog.db-in.com/cameras-on-opengl-es-2-x/. [Dostopano: 18. 6. 2014].

[5] *Cosmic diagrams* [Elektronski]. Dosegljivo: http://www.kabbalahmeditation.org/cosmicdiagrams.htm. [Dostopano: 24. 8. 2014].

[6] *JavaRanch - Java Programming Style Guide* [Elektronski]. Dosegljivo: http://www.javaranch.com/style.jsp. [Dostopano: 13. 7. 2014].

[7] *Location and Sensors APIs | Android Developers* [Elektronski]. Dosegljivo: http://developer.android.com/guide/topics/sensors/. [Dostopano: 10. 8. 2014].

[8] *Motion Sensors | Android Developers* [Elektronski]. Dosegljivo: http://developer.android.com/guide/topics/sensors/sensors_motion.html. [Dostopano: 8. 10. 2014].

[9] *OpenGL ES 2_X - The Standard for Embedded Accelerated 3D Graphics* [Elektronski]. Dosegljivo: http://www.khronos.org/opengles/2_X. [Dostopano: 10. 8. 2014].

[10] OurAirports. (14. september 2014). *Data downloads @ OurAirports* [Elektronski]. Dosegljivo: http://ourairports.com/data/. [Dostopano: 14.9.2014].

[11] Pravila vizualnega letenja. (12. april 2014). *Pravila vizualnega letenja - Wikipedia, prosta enciklopedija* [Elektronski]. Dosegljivo: http://sl.wikipedia.org/wiki/Pravila_vizualnega_letenja. [Dostopano: 22. 7. 2014].

[12] R. Chowdhury. Learn more about Android smartphone sensors. (30. julij 2013). *Learn more about Android smartphone sensors | Techtunes | scream* [Elektronski]. Dosegljivo: http://www.techtunes.com.bd/android/tune-id/227947. [Dostopano: 3. 7. 2014].

[13] *Sensors Overview | Android Developers* [Elektronski]. Dosegljivo: http://developer.android.com/guide/topics/sensors/sensors_overview.html. [Dostopano: 24. 8. 2014].

[14] T. Vladimir. Hydrodynamic Process Simulation on a Mobile Device. (17. februar 2013). *Azoft* [Elektronski]. Dosegljivo: http://rnd.azoft.com/fluid-dynamics-simulation-on-ios/. [Dostopano: 15.9.2014].

[15] T. Vincenty. "Direct and inverse solutions of geodesics on the ellipsoid with application of nested equations," *Survey Review*, vol. 23, št. 176, str. 88-93, april 1975.

[16] VFR CIVILNO-VOJAŠKA LETALSKA NAVIGACIJSKA KARTA REPUBLIKE SLOVENIJE 1 : 250.000 (2014). *Slovenia Control - VFR karta* [Elektronski]. Dosegljivo: http://www.sloveniacontrol.si/informacije/vfr-karta-2014. [Dostopano: 26. 8. 2014].

[17] World Geodetic System. (24. junij 2004). *World Geodetic System* [Elektronski]. Dosegljivo: http://en.wikipedia.org/wiki/WGS84. [Dostopano: 10. 8. 2014].