VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

ROZŠÍŘENÁ REALITA NA ANDROID - ZOBRAZOVÁNÍ PRVKŮ Z MAPY V TERÉNU

AUGMENTED REALITY ON ANDROID - DISPLAYING OF MAP FEATURES IN TERRAIN

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **TIMOTEJ HALÁS**
AUTHOR

VEDOUCÍ PRÁCE **prof. Ing. ADAM HEROUT, Ph.D.**
SUPERVISOR

BRNO 2019

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2018/2019

Zadání bakalářské práce

Student: **Halás Timotej**
Program: Informační technologie
Název: **Rozšířená realita na Android - zobrazování prvků z mapy v terénu**
Augmented Reality on Android - Displaying of Map Features in Terrain
Kategorie: Uživatelská rozhraní

Zadání:

1. Seznamte se s problematikou rozšířené reality, zaměřte se na vývoj aplikací AR pro Android.
2. Seznamte se s Open Street Maps (či dalšími volně dostupnými mapami) a s manipulací s mapovými daty.
3. Prototypujte a experimentujte s dílčími prvky rozšířené reality v krajině.
4. Navrhněte a implementujte aplikaci rozšířené reality pro zobrazování prvků mapy v krajině.
5. Laďte uživatelské rozhraní a technické řešení aplikace směrem k dokonalosti.
6. Zhodnoťte dosažené výsledky a navrhněte možnosti pokračování projektu; vytvořte plakátek a krátké video pro prezentování projektu.

Literatura:

- Steve Krug: Don't Make Me Think, Revisited: A Common Sense Approach to Web Usability, ISBN-13: 978-0321965516
- Susan M. Weinschenk: 100 věcí, které by měl každý designér vědět o lidech, Computer Press, Brno 2012
- Richard Szeliski: Computer Vision: Algorithms and Applications, Springer, 2011
- Dieter Schmalstieg, Tobias Hollerer: Augmented Reality: Principles and Practice, ISBN: 978-0321883575
- Android Developers: https://developer.android.com/index.html

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 a 2, značné rozpracování bodů 3 a 4.

Podrobné závazné pokyny pro vypracování práce viz http://www.fit.vutbr.cz/info/szz/

Vedoucí práce: **Herout Adam, prof. Ing., Ph.D.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2018
Datum odevzdání: 15. května 2019
Datum schválení: 6. listopadu 2018

Abstrakt

Cieľom tejto práce je vytvoriť aplikáciu rozšírenej reality pre operačný systém Android. Na základe náhľadu z kamery, mapových dát a polohy zariadenia určenej modulom GPS je vytvorená scéna zobrazujúca doplňujúce turistické informácie v teréne. Ako zdroj informácií a výškových dát je využitá mapa OpenStreetMap a Google Elevation API. Pozícia v kartézskom súradnicovom systéme je určená pomocou geografickej polohy s použitím geodetického systému WGS84. Pre vykresľovanie je využitý herný framework LibGDX. Výsledná aplikácia bola otestovaná v teréne a je užitočná pri spoznávaní okolitej krajiny. Taktiež veľmi pomohla v navigácii k zaujímavým bodom vďaka zobrazovaniu ich vzdialenosti od bodu pozorovania. Výsledkom tejto práce je aplikácia vhodná pre získavanie nových informácií o pozorovanom okolí.

Abstract

The aim of this thesis is to create an application of augmented reality for Android operating system. Based on the preview from the camera, map data and the location of the device determined by the GPS module, a scene is created showing additional tourist information in the field. As a source of information and altitude data, the OpenStreetMap map and the Google Elevation API are used. The position in the Cartesian coordinate system is determined by geographic location using the WGS84 geodetic system. LibGDX game framework is used for rendering. The resulting application was tested in the field and it is useful in getting to know the surrounding landscape. It has also helped to navigate to points of interest by displaying their distance from the point of observation. The result of this thesis is an application suitable for exploring the observed environment.

Kľúčové slová

rozšírená, realita, Android, mapa, senzory, GPS, OpenStreetMap, Overpass, LibGDX, WGS84, EGM96

Keywords

augumented, reality, Android, map, sensors, GPS, OpenStreetMap, Overpass, LibGDX, WGS84, EGM96

Citácia

HALÁS, Timotej. *Rozšířená realita na Android - zobrazování prvků z mapy v terénu.* Brno, 2019. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce prof. Ing. Adam Herout, Ph.D.

Rozšířená realita na Android - zobrazování prvků z mapy v terénu

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením pána prof. Ing. Adama Herouta, Ph.D a uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

.....
Timotej Halás
15. mája 2019

Poďakovanie

Ďakujem predovšetkým môjmu vedúcemu, pánovi prof. Ing. Adamovi Heroutovi, Ph.D, za skvelé vedenie, motiváciu a za všetky rady a pripomienky k mojej práci, ktoré mi pomohli k jej zdokonaľovaniu. Keďže v strese sa niekedy ťažko pracuje, chcem taktiež poďakovať mame a babke, že ho prežívali za mňa. Ďalej chcem poďakovať mame, sestre a priateľke za pomoc s gramatickou korektúrou práce. Nakoniec chcem poďakovať tatovi, ktorý výslednú aplikáciu otestoval a všetkým kamarátom, s ktorými som prácu mohol konzultovať.

Obsah

Kapitola 1

Úvod

Počiatky rozšírenej reality siahajú už do začiatkov dvadsiateho storočia. Prvé zariadenia slúžiace pre tento účel boli ťažké, veľké a drahé. V dnešnej dobe už skoro každý človek vlastní mobilný telefón, ktorý vo väčšine prípadov obsahuje potrebnú výbavu a dosahuje dostatočný výkon pre použitie rozšírenej reality. Mobilný telefón tejto doby obsahuje obrazový snímač, displej, na ktorom bude obraz zo snímača zobrazený a výkonný procesor. Tieto komponenty pre tento účel stačia. Mobilné však telefóny naviac obsahujú pohybové senzory a modul GPS, ktoré dojem rozšírenej reality umocňujú. Preto sa rozšírená realita stáva stále populárnejšou a pre jej vyskúšanie stačí nainštalovať aplikáciu v mobilnom telefóne.

Rozšírená realita má široké spektrum využití či už pre účely vzdelávania, v architektúre, propagácii produktov, alebo môže slúžiť napríklad ako pomocník v bežných činnostiach, ako meraní dĺžok a vzdialeností namiesto klasických meracích zariadení, ako meter. V tejto práci bude jej využitie zamerané na rozširovanie vrcholov, studničiek, vodopádov a iných turistických bodov v teréne o nové informácie.

Túto tému som si vybral z toho dôvodu, že rozšírená realita a vývoj pre operačný systém Android ma zaujímali už dlhú dobu, no nemal som príležitosť si ich možnosť vyskúšať.

Cieľom práce je vytvorenie aplikácie, ktorá bude užitočná hlavne pri turistike. Bude zobrazovať buď názvy, alebo typy bodov v okolí užívateľa. Taktiež bude zobrazovať vzdialenosť užívateľa od týchto bodov a pri vrcholoch bude zobrazená aj nadmorská výška.

V kapitole 2 sú popísané hlavné pojmy týkajúce sa tejto problematiky. Sú tu tiež popísané kroky pre začiatok vývoja pre operačný systém Android, spôsob vykresľovania náhľadu z obrazového senzoru a použitia senzorov a modulu GPS. Ďalej je popísané využitie máp v tejto práci, ako aj spôsob, ako použiť dáta z mapy v kartézskom súradnicovom systéme, keďže mapy používajú geografický systém. Na konci je popísaný spôsob, akým bude vykresľovaná grafika.

Kapitola 3 popisuje postup návrhu výslednej aplikácie. Je tu popísané, ako budú využité senzory v mobilnom telefóne, ako budú získavané dáta z máp a spôsob určenia polohy objektu v scéne, ktorá bude vykresľovaná. Túto scénu bude potrebné spravovať, kvôli počtu objektov získaných z mapy, a to bude tiež v tejto kapitole rozoberané. Nakoniec bude popísaný spôsob návrhu grafického užívateľského rozhrania.

O implementácii výslednej aplikácie je písané v kapitole 4. Na začiatku je popísaná štruktúra projektu, ktorej znalosť je potrebná v prípade rozširovania aplikácie. Ďalej je tu popísaný spôsob implementácie geodetického systému WGS84, a tiež to, akým spôsobom je spravovaná a vykresľovaná scéna. Nakoniec sa tu nachádza výsledok implementácie obsahujúci snímky obrazovky výslednej aplikácie.

Kapitola 2

Techniky a technológie rozšírenej reality a zobrazovania prvkov v teréne

Táto kapitola sa zameriava na témy, ktoré sú potrebné pre návrh a implementáciu aplikácie rozšírenej reality a zobrazovania prvkov v teréne.

Zameriava sa na vysvetlenie, čo je to rozšírená realita a vysvetľuje základy vývoja pre operačný systém Android, keďže v minulosti som nemal žiadne skúsenosti s vývojom aplikácií pre túto platformu. Ďalej popisuje spôsoby akými sa dajú použiť rôzne senzory v zariadeniach s operačným systémom Android, ako obrazový senzor, pohybové, pozičné a polohové senzory. Špecifikuje ako funguje mapa OpenStreetMap (OSM) a ako sa z nej dajú získať mapové dáta. Potom vysvetľuje súradnicový systém WGS84, ktorý je bežný pre určovanie lokácie a bude potrebný pre určovanie polohy v kartézskom súradnicovom systéme. Nakoniec rozoberá herný framework LibGDX, ktorý bude použitý na vykresľovanie všetkých grafických údajov v aplikácii.

2.1 Rozšírená realita

Rozšírená realita [30] obsahuje zmes reálnych a syntetických elementov scény. Oproti virtuálnej realite, ktorá sa snaží ponoriť užívateľa do kompletne počítačom generovaného sveta, rozšírená realita rozširuje vnímanie reálneho sveta užívateľom o nové prvky. Tieto prvky obsahujú informácie, ktoré nie sú prirodzene súčasťou scény, pomáhajú ľuďom v navigácii a môžu zefektívniť prácu v reálnom svete.

Hlavnou výzvou pre vytvorenie rozšírenej reality je udržovanie presnej registrácie medzi reálnymi a počítačom generovanými objektami. Virtuálne objekty si musia udržovať svoje zarovnanie s pozíciami a orientáciami reálnych objektov v trojdimenzionálnom priestore, keď v tomto priestore užívateľ mení svoju pozíciu. Toto zarovnanie závisí na presnom sledovaní pozície a orientácie sledovacej pózy relatívne k prostrediu alebo reálnym objektom, takže presnosť odhadnutej pózy priamo určuje vizuálne vnímanú presnosť zarovnania a registrácie. Sledovacie zariadenie definuje virtuálnu kameru, ktorá je použitá pre projekciu virtuálnych objektov na obraz reálneho sveta.

2.1.1 Spôsoby zobrazovania rozšírenej reality

Dôležitým faktorom v rozšírenej realite je spôsob jej zobrazovania. Týchto spôsobov je niekoľko. Zobrazovače môžu byť zložené z viacerých optických, elektronických alebo mechanických prvkov, ktoré tvoria cestu medzi zrakom pozorovateľa a objektom, ktorý bude rozšírený o nové informácie. Podľa toho kde medzi pozorovateľom a reálnym svetom je zobrazovač umiestnený, existujú tri spôsoby, ktoré sú popísané nižšie. [4]

Zobrazovač pripevnený o hlavu

Tento typ zobrazovačov vyžaduje nosenie zobrazovacieho systému na hlave pozorovateľa. Podľa generovania obrazu existujú nasledujúce typy zobrazovacích systémov:

- **Sietnicové zobrazovače** – nízko výkonné lasery pre zobrazovanie obrazu priamo na sietnicu oka
- **Displej na hlave (HMD)** – miniatúrne displeje pred očami pozorovateľa
- **Projektor na hlave** – miniatúrny projektor alebo LCD panel s podsvietením, ktoré premietajú obraz priamo na reálne okolie

Zobrazovač držaný v ruke

Príkladom tohto spôsobu je zariadenie (napríklad mobilný telefón), ktoré generuje obraz rozšírený o nové informácie. Tento spôsob kombinuje využitie procesora, pamäte, displeja a senzorov v zariadení pre zobrazovanie rozšírenej reality. Obrazový senzor je umiestnený na zadnej strane zariadenia a je natočený smerom od užívateľa. Obraz z obrazového senzoru, ktorý sníma reálne prostredie je použitý ako pozadie. Toto pozadie je prekresľované virtuálnym obsahom a výsledok sa zobrazuje na displeji zariadenia. Zobrazovač držaný v ruke reprezentuje dostupnejšiu alternatívu k zobrazovačom pripevneným o hlavu, pretože k týmto zariadeniam má v dnešnej dobe prístup väčšina ľudí, a teda majú väčší potenciál priniesť rozšírenú realitu na trh. Tento spôsob bude využitý v tejto práci.

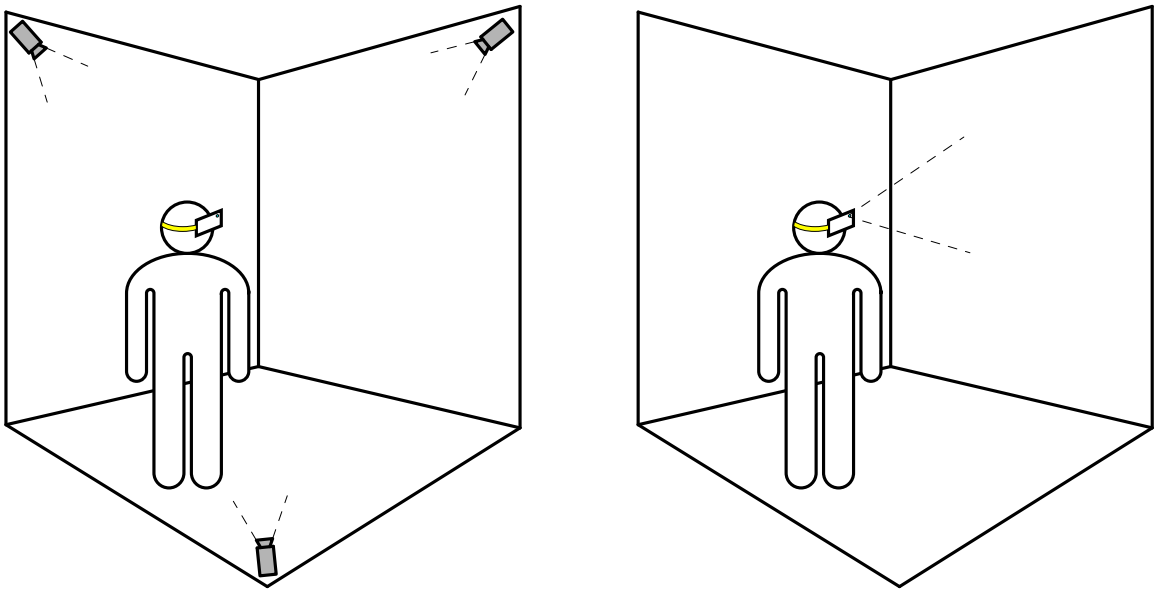
Zobrazovač v priestore

Na rozdiel od ostatných spôsobov, kedy je zobrazovač nejakým spôsobom držaný pri tele, tento spôsob oddeľuje väčšinu technológie od pozorovateľa a integruje ho do prostredia. Všeobecne sa zobrazovanie rozšírenej reality v priestore delí na tri prístupy:

- Displej, cez ktorý vidno pozadie doplnené o nové informácie
- Priesvitný materiál, na ktorý sú premietané nové informácie
- Priame doplnenie informácií (nevyužíva sa počítač na vykresľovanie, ale objekt je doplnený o informácie priamo)

2.1.2 Typy sledovania pózy

Sledovanie pózy [21] sa všeobecne rozdeľuje na dve metódy, a to sledovanie zvnútra von a zvonku dovnútra. Porovnanie medzi metódami je na obrázku 2.1.



Obr. 2.1: Rozdiel medzi odhadom pózy zvonka dovnútra (vľavo) a zvnútra von (vpravo). Vľavo odhad pózy vykonávajú externé zariadenia na sledovanie a vpravo odhad pózy vykonáva samotné zariadenie pre zobrazovanie rozšírenej reality. [21]

Sledovanie zvonku dovnútra

V sledovaní zvonku dovnútra sa užívateľova pozícia sleduje externým zariadením. Pre zisťovanie pozície sa na zariadenie zobrazujúce rozšírenú realitu pridávajú značky, ktoré sleduje externé zariadenie a pomocou nich určuje polohu a orientáciu v priestore. Veľkou nevýhodou tejto metódy je to, že rozsah pohybu užívateľa je obmedzený na zorné pole sledovacieho zariadenia. Ak sa užívateľ dostane mimo zorné pole sledovacieho zariadenia alebo rotácia zariadenia zobrazujúceho rozšírenú realitu je moc veľká, sledovanie sa preruší.

Sledovanie zvnútra von

Pri sledovaní zvnútra von sú senzory umiestnené priamo na užívateľovi alebo veľmi blízko užívateľa a jeho póza sa odhaduje zo sledovania senzorov. Táto metóda má niekoľko výhod oproti sledovaniu zvonku dovnútra. Tými sú napríklad výhoda toho, že rozsah pohybu užívateľa nie je obmedzený zorným poľom externého zariadenia alebo pri použití IMU (Inertial Measurements Units) sa dá vykonať fúzia vizuálnych a pohybových informácií veľmi efektívnym spôsobom.

2.2 Vývoj pre operačný systém Android

Keďže vývoj pre operačný systém Android bol pre mňa úplnou novinkou, musel som si získať informácie o tom ako vývoj začať.

2.2.1 Výber vývojového prostredia

Prvou vecou potrebnou pre vývoj aplikácií bolo výber vývojového prostredia. Rozhodol som sa pre štandardné IDE (Integrated Development Environment) Android Studio[1]. Toto vývojové prostredie podporuje emuláciu Android zariadenia, profilovanie v reálnom čase, inteligentné editovanie textu a iné vymoženosti použiteľné pri vývoji aplikácií pre operačný systém Android. Je založené na vývojovom prostredí IntelliJ IDEA[2] od výrobcu JetBrains, ktorého produkty veľmi rád používam. Prostredie Android Studia mi bolo známe hneď a nemal som problém si naň zvyknúť. [18]

2.2.2 Prvý testovací projekt

Predtým ako som sa pustil do vývoja aplikácie rozšírenej reality a zobrazovania prvkov z mapy v teréne, som si vytvoril testovací projekt, v ktorom som si skúšal základy vývoja. Vytvorenie projektu je veľmi jednoduché a robí sa bežným spôsobom pomocou kontextového menu **File -> New -> New Project...**.

Štruktúra projektu

Každý projekt v Android Studiu je zložený z jedného alebo viacerých modulov obsahujúcich zdrojové kódy a zo súborov so zdrojmi (obrázky, ikona v spúšťači a iné). [23] Typy modulov môžu byť:

- moduly s Android aplikáciami
- moduly s knižnicami
- moduly s Google App Engine

Každý aplikačný modul obsahuje nasledujúce priečinky:

- **manifests** – Obsahuje súbor `AndroidManifest.xml`
- **java** – Obsahuje zdrojové súbory v jazyku Java a JUnit testy
- **res** – Obsahuje všetky nekódové zdroje, ako XML rozloženia, texty alebo obrázky

Štruktúra Android projektu na disku sa líši od štruktúry ukázanej v Android Studiu.

Počiatočný bod aplikácie

Trieda `Activity` [14] (alebo jej rozšírenie trieda `Application`) je veľmi dôležitá časť aplikácie pre operačný systém Android. Spôsob akým sú aktivity spúšťané a pospájané do kopy je základnou časťou aplikačného modelu platformy. Na rozdiel od programovacích paradigiem, kde aplikácie sú spustené pomocou metódy `Activity`, systém Android iniciuje kód v inštancii triedy `Activity` zavolaním špecifických metód spätného volania, ktoré korešpondujú so špecifickými štádiami jej životného cyklu.

[1] https://developer.android.com/studio
[2] https://www.jetbrains.com/idea/

Manifest

Každý projekt musí obsahovať súbor `AndroidManifest.xml` v hlavnom adresári projektu. Tento súbor popisuje nevyhnutné informácie o aplikácii pre zostavenie projektu pre operačný systém Android, a taktiež pre Google Play v prípade distribúcie aplikácie. Nasledujúce odrážky popisujú niektoré z vecí nachádzajúcich sa v manifeste:

- Oprávnenia, ktoré aplikácia potrebuje pre jej správny chod, napríklad prístup k obrazovému senzoru, GPS, pripojeniu na internet alebo určovaniu polohy pomocou GPS
- Názov balíčku aplikácie, ktorý je väčšinou zhodný s menným priestorom kódu
- Názov aplikácie (viditeľný v spúšťači operačného systému Android)
- Definíciu každej aktivity, ktorá je obsiahnutá v aplikácii

Na čo slúži trieda `Activity`

Aktivita ponúka okno, v ktorom aplikácia kreslí jej užívateľské rozhranie. Toto okno väčšinou zapĺňa celý displej, no môže byť aj menšie a plávať nad ostatnými oknami. Všeobecne jedna aktivita implementuje jedno okno v aplikácii. Väčšinou je jedna aktivita špecifikovaná ako hlavná aktivita a spúšťa sa pri spustení aplikácie. Každá aktivita potom môže spúšťať iné aktivity pre vykonávanie ďalších akcií.

2.3 Náhľad z obrazového senzoru zariadenia s operačným systémom Android

V operačnom systéme Android existujú dva spôsoby zobrazovania náhľadu z obrazového senzoru. [5] Prvým je možnosť použitia triedy `Camera`. Tento spôsob je ale zastaralý, pomalý (nízka snímková frekvencia) a nie je odporúčaný oficiálnou dokumentáciu pre operačný systém Android. Druhý je spôsob použitie balíčka `android.hardware.camera2`. Tento balíček podporuje oveľa komplexnejšiu prácu s obrazovým senzorom zariadenia a je oveľa rýchlejší.

2.3.1 Balíček `android.hardware.camera2` a jeho použitie

Tento balíček [2] poskytuje rozhranie pre ovládanie jednotlivých obrazových senzorov v zariadení s operačným systémom Android.

Obrazový senzor modeluje ako reťazec, do ktorého vstupom je požiadavka o zachytenie jednej snímky, zachytí jednu snímku na každú požiadavku, a taktiež výstupom je jeden výsledok obsahujúci metadáta a súbor výstupných obrazových vyrovnávacích pamätí pre požiadavku. Tieto požiadavky sú spracovávané v poradí v akom prišli a aktívne môžu byť viaceré požiadavky. Keďže obrazový senzor je reťazec s viacerými stupňami je potrebné mať viac aktívnych požiadaviek, aby bola zachovaná plná snímková frekvencia.

Získanie informácií o dostupných obrazových senzoroch

Na začiatok treba získať inštanciu triedy `CameraManager`. Po získaní je možné zistiť informácie o všetkých obrazových senzoroch a tie potom použiť.

Jednotlivé obrazové senzory poskytujú statické vlastnosti, ktoré popisujú hardvér obrazového senzoru, dostupné nastavenia a výstupné parametre zariadenia. Dajú sa napríklad

získať informácie o veľkosti obrazového senzoru a ohniskovej vzdialenosti, ktoré budú neskôr potrebné pre výpočet zorného poľa tohto senzoru (bližší popis v kapitole 2.8.1). Tieto sú dostupné pomocou objektu `CameraCharacteristics`, ktorý sa dá získať pomocou metódy `getCameraCharacteristics(String cameraId)` triedy `CameraManager`.

Vytvorenie relácie zachytávania dát z obrazového senzoru

Pre zachytenie alebo streamovanie snímok z obrazového senzoru sa najskôr musí vytvoriť relácia zachytávania snímok z obrazového senzoru spolu so súborom výstupných plôch (trieda `Surface`) pomocou metódy `createCaptureSession(SessionConfiguration)` triedy `CameraDevice`, na ktoré budú snímky vykresľované. Plochy musia byť nakonfigurované, aby mali veľkosti a formáty podporované daným obrazovým senzorom. Cieľová plocha môže byť získaná rôznymi spôsobmi, no pre túto prácu bude potrebná plocha vytvorená pomocou triedy `SurfaceTexture`. Táto plocha má formát `YUV_420_888`. Ak bude textúra z tejto plochy použitá v OpenGL je potrebné vo fragment shadery použiť sampler `samplerExternalOES` namiesto obyčajného sampleru `sampler2d`. Je to z toho dôvodu, že to vyžaduje trieda `SurfaceTexture` [11].

Vytvorenie požiadavky o zachytenie snímky

Aplikácia potom musí vytvoriť požiadavku o zachytenie (trieda `CaptureRequest`), ktorá definuje všetky parametre potrebné obrazovým senzorom (napr. automatická expozícia, citlivosť ISO, čas uzávierky pri manuálnej expozícii, optická stabilizácia obrazového senzoru a iné parametre). Požiadavka tiež obsahuje informácie o cieľovej ploche alebo plochách ak ich je viac. `CameraDevice` má factory metódu pre vytvorenie tvoriča požiadaviek pre daný prípad použitia, ktorá je optimalizovaná pre zariadenie s operačným systémom Android, na ktorom beží aplikácia.

Predanie požiadavky relácii zachytávania dát z obrazového senzoru

Po nastavení požiadavky, táto môže byť predaná aktívnej relácii zachytávania buď pre vytvorenie jednej snímky alebo pre nekonečné použitie (napr. náhľad z obrazového senzoru alebo nahrávanie videa). Akceptovaný je tiež súbor požiadaviek pre nasnímanie zhluku snímok.

Zachytávanie obrazových dát z obrazového senzoru

Po spracovaní požiadavky obrazový senzor vytvorí `TotalCaptureResult` objekt, ktorý obsahuje informácie o stave obrazového senzoru v čase vytvorenia snímky a nastavenia, ktoré boli použité, pretože sa môžu líšiť od nastavení požiadavky. Obrazový senzor taktiež pošle snímku do každej výstupnej plochy obsiahnutej v požiadavke. Tieto sú vytvárane asynchrónne relatívne k výstupu `CaptureResult` a niekedy podstatne neskôr.

2.4 Použitie senzorov v operačnom systéme Android

Zariadenia s operačným systémom Android vo väčšine prípadov obsahujú rôzne typy senzorov [19]. Tieto senzory slúžia napríklad na monitorovanie pohybu zariadenia v trojdimenzionálnom priestore, jeho umiestnenia v priestore alebo na sledovanie zmeny okolitého prostredia. Delia sa do týchto troch kategórií:

- **Pohybové senzory** – slúžia na meranie síl zrýchlenia a rotačných síl okolo troch osí na obrázku 2.2
- **Environmentálne senzory** – slúžia na meranie okolitých parametrov, ako teplota, tlak, vlhkosť alebo svetlo
- **Pozičné senzory** – slúžia na meranie fyzickej pozície zariadenia a orientácie zariadenia voči pólom zeme

Pre využitie týchto senzorov sa požíva Android sensor framework. Tento framework ponúka rôzne triedy a rozhrania, ktoré slúžia na vykonávanie úkonov so senzormi. Napríklad:

- zistenie dostupnosti senzoru v zariadení
- zistenie informácií o jednotlivých senzoroch, ako výrobca, rozsah dát, rozlíšenie senzoru a výkon senzoru
- získanie surových dát a nastavenie vzorkovacej frekvencie senzoru
- registrácia a odregistrácia sledovačov udalostí, ktoré sledujú zmeny senzorov

2.4.1 Android sensor framework

Tento framework [19] umožňuje prístup k rôznym typom senzorov. Niektoré sú softvérové senzory a iné sú hardvérové.

Hardvérové senzory sú fyzické komponenty v zariadení. Dáta získavajú priamym meraním okolitých vlastností, ako zrýchlenie, intenzita geomagnetického poľa alebo zmeny uhlu.

Softvérové senzory nie sú fyzické komponenty, no imitujú ich. Dáta získavajú z jedného alebo viacerých hardvérových senzorov, a preto sú niekedy prezývané virtuálnymi senzormi. Medzi tieto senzory patria napríklad senzor lineárneho zrýchlenia alebo gravitačný senzor.

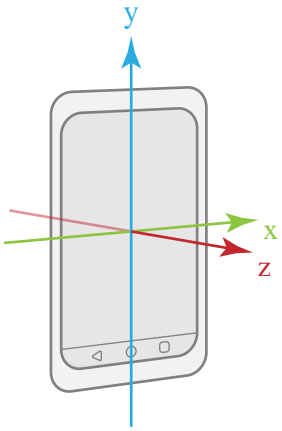
2.4.2 Súradnicový systém senzorov

Android sensor framework používa štandardný súradnicový systém [19] s troma osami pre vyjadrovanie hodnôt. Súradnicový systém je definovaný relatívne k displeju zariadenia, keď je zariadenie držané v štandardnej orientácii. Vtedy os x smeruje horizontálne doprava, os y smeruje smerom nahor a os z smeruje von z displeja zariadenia (obrázok 2.2).

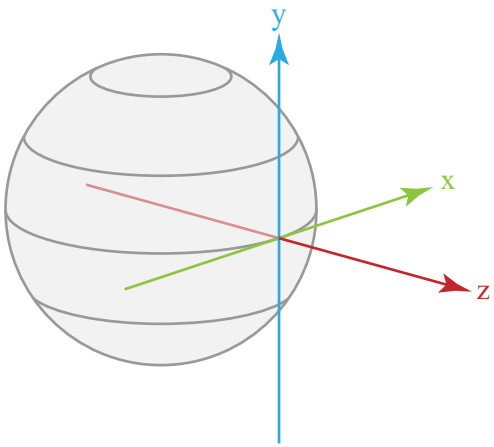
Osi súradnicového systému sa nevymieňajú, keď sa zmení orientácia zariadenia, a teda súradnicový systém sa nikdy nemení za pohybu zariadenia. Tak isto funguje aj súradnicový systém OpenGL.

2.4.3 Potrebné senzory pre rozšírenú realitu

V prípade zobrazovania prvkov z mapy v teréne je určite treba určiť smer natočenia zariadenia s ohľadom na svetové strany. Pre určenie tohto natočenia treba senzor magnetometer, ktorý sníma okolité magnetické pole. Ďalej je určite treba senzor gyroskop ktorý slúži na sledovanie uhla rotácie okolo osí súradnicového systému senzorov. Nakoniec je potreba senzor zrýchlenia, ktorým sa určí smer k ťažisku Zeme. [21]



Obr. 2.2: Súradnicový systém relatívny k displeju zariadenia. Os x smeruje horizontálne doprava, os y smeruje smerom nahor a os z smeruje smerom z displeja. [19]



Obr. 2.3: Súradnicový systém použitý pre senzor vektoru rotácie. Os x smeruje na západ a taktiež je rovnobežná s povrchom zeme, os y smeruje na sever a je rovnobežná s povrchom zeme v danom bode a os z je kolmá na povrch zeme v danom mieste. [29]

Senzor vektoru rotácie

Vektor rotácie reprezentuje orientáciu zariadenia ako kombináciu uhla a osi rotácie, okolo ktorej sa zariadenie rotovalo o uhol θ.

$$\mathbf{v} = (x,\ y,\ z) \tag{2.1}$$

$$\mathbf{u} = \left(\mathbf{v}_x \sin\frac{\theta}{2},\ \mathbf{v}_y \sin\frac{\theta}{2},\ \mathbf{v}_z \sin\frac{\theta}{2}\right) \tag{2.2}$$

$$q = \left(\cos\frac{\theta}{2},\ \mathbf{u}_x,\ \mathbf{u}_y,\ \mathbf{u}_z\right) \tag{2.3}$$

Os rotácie je definovaná jednotkovými vektorom $\mathbf{v}$, ako v rovnici 2.1. Vektor rotácie $\mathbf{u}$ (rovnica 2.2) je definovaný z jednotkového vektoru popisujúceho os rotácie. Dĺžka vektoru rotácie je rovná $\sin\frac{\theta}{2}$ a smer vektoru rotácie je rovný smeru osi rotácie. Tri prvky vektoru rotácie sú rovné posledným trom zložkám jednotkového quaternionu q v rovnici 2.3.

Prvky sú bez jednotky. Osi x, y a z sú definované rovnako, ako súradnicový systém na obrázku 2.2. Súradnicový systém je definovaný ako ortogonálna báza (obrázok 2.3) a má nasledujúce vlastnosti:

- Os x je definovaná ako vektorový súčin vektoru Y a Z. Je rovnobežná s povrchom zeme v aktuálnej polohe zariadenia a smeruje približne na východ.
- Os y je rovnobežná s povrchom zeme v aktuálnej lokácii zariadenia a smeruje ku geomagnetickému severnému pólu.
- Os z smeruje k oblohe a je kolmá na povrch zeme v aktuálnej lokácii zariadenia.

Dáta z tohto senzoru obsahujú štyri hodnoty quaternionu popísaného v rovnici 2.3, a teda jednotkový quaternion, ktorý popisuje rotáciu zariadenia okolo jeho osí (obrázok 2.2), keď tieto osi sú zhodné s osami súradnicového systému pre senzor vektoru rotácie (obrázok 2.3)

2.5 Geografická poloha zariadenia s operačným systémom Android

Pre získanie polohy zariadenia v systéme Android existujú dva spôsoby.

Prvý [3] je zložitejší a menej presný. Využíva sa v ňom API (Application programming interface) vstavané priamo v operačnom systéme Android.

Druhý spôsob [16] využíva Location API, ktoré je k dispozícii v službách Google Play a uľahčuje získavanie informácií o polohe pomocou automatizovaného sledovania polohy.

Údaj o polohe, ktorý je výstupom je inštancia triedy Location [15] a tá sa vždy skladá zo zemepisnej šírky, zemepisnej dĺžky, vo väčšine prípadov aj geodetickej výšky nad referenčným elipsoidom WGS84 a inými informáciami o polohe a smerovaní pohybu zariadenia. Význam týchto údajov a referenčný elipsoid WGS84 je hlbšie popísaný v kapitole 2.7.

2.5.1 Location API služieb Google Play

Pred použitím Location API služieb Google Play treba projekt nakonfigurovať [25] s SDK (Software development kit) služieb Google Play. Pre vývoj aplikácií s použitím SDK služieb Google Play sa dá použiť jedna z týchto možností:

- zariadenie s Androidom verzie minimálne 4.1 obsahujúce Obchod Google Play
- Android emulátor s AVD (Android Virtual Device), ktoré obsahuje Google APIs platformu založenú na verzii Androidu 4.2.2 alebo vyššej

Konfigurácia Location API služieb Google Play

Aplikácia používajúca Location API služieb Google Play si špecifikuje potrebnú úroveň presnosti určovania polohy a požadovaný interval získavania zmien. Zariadenie po špecifikovaní týchto parametrov automaticky nastaví systémové nastavenia. [6]

Pre nastavenie týchto parametrov je potrebne vytvoriť `LocationRequest`. Tieto parametre určujú úroveň presnosti pre požiadavky určovania polohy. Pre nastavenie niektorých parametrov slúžia tieto metódy:

- Metóda `setInterval(long)` určuje interval v milisekundách, po ktorých bude získavaná nová poloha. Ak je tento interval u iných aplikácií bežiacich v pozadí kratší tak poloha bude získavaná rýchlejšie aj v aplikácii bežiacej na popredí.
- Metóda `setFastestInterval(long)` určuje minimálny čas, za ktorý vie aplikácia spracovať novo získanú polohu. Toto nastavenie obmedzuje rýchlosť získavania dát o polohe z aplikácií bežiacich v pozadí, ktoré majú rýchlejší obnovovací interval a rýchlosť získavania dát z týchto aplikácií by mohli ovplyvniť chod aplikácie bežiacej v popredí. Napríklad by tieto dáta nemusela stíhať spracovávať.
- Metóda `setPriority(int)` nastavuje presnosť získanej polohy. Tá je určená poskytovateľom polohy, a teda buď GPS (Global Positioning System), WiFi a veže mobilných sietí. GPS je síce najpresnejšie, ale má aj najväčšiu spotrebu energie. Veže mobilných sietí a WiFi nie sú moc presné, ale spotreba je oveľa menšia.

Získanie poslednej známej polohy

Získanie poslednej známej polohy [27] slúži ako dobrý začiatok pre aplikáciu, ktorá už môže vedieť poslednú známu polohu pred spustením periodického získavania polohy.

Pomocou Location API služieb Google Play aplikácia môže požiadať o poslednú známu polohu zariadenia použitím triedy `FusedLocationProviderClient` [10] z Location API služieb Google Play. `FusedLocationProviderClient` sa stará o nízko úrovňovú polohovaciu technológiu a ponúka jednoduché API pre špecifikáciu požiadaviek popísaných v predošlej kapitole a taktiež optimalizuje spotrebu batérie pri získavaní polohy. Posledná známa poloha sa získa pomocou metódy `getLastLocation()`

Získavanie zmien polohy

Väčšinou je potrebné požadovanie periodických zmien polohy. Toto sa dá docieliť použitím metódy `requestLocationUpdates()` triedy `FusedLocationProviderClient`. Ako odpoveď API periodicky posiela aplikácii údaje o najpresnejšej polohe zariadenia podľa aktuálne dostupných poskytovateľov polohy, ako WiFi a GPS. Presnosť je určená poskytovateľmi, oprávneniami nastavenými v manifeste aplikácie a nastaveniami, ktoré boli nastavené pri konfigurácii Location API služieb Google Play. [24]

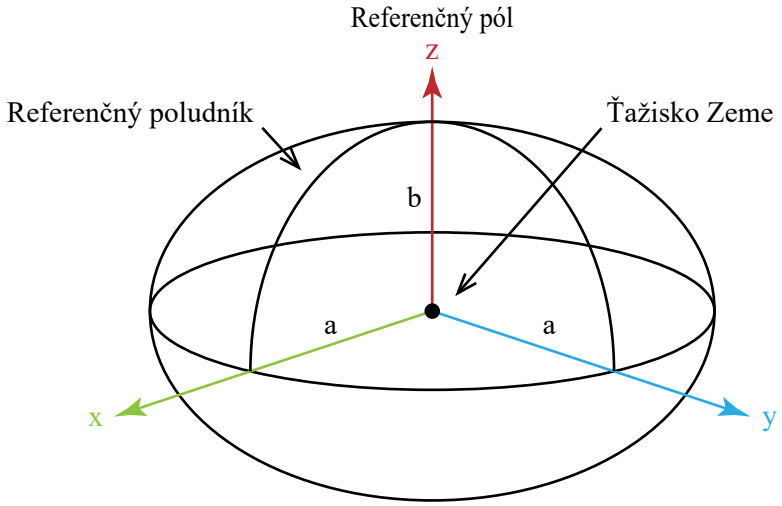
2.6 OpenStreetMap (OSM) ako zdroj dát

OpenStreetMap [1] (ďalej len OSM) je projekt, ktorý vytvára a distribuuje geografické dáta pre celú Zem. Je to editovateľná mapa budovaná dobrovoľnými prispievateľmi a môže do nej prispievať hocikto. Dáta z mapy môžu byť voľne využívané pre akýkoľvek dôvod. Tieto dáta využívajú súradnicový systém WGS84 (popísaný v kapitole 2.7). OSM má API, ktoré umožňuje pridávať, editovať alebo získavať dáta z mapy.

2.6.1 Typy dát v OSM

Základnými prvkami v dátach OSM sú elementy. Tieto elementy sú zložené z:

- **uzol** – definuje bod v priestore určený minimálne geografickými súradnicami (zemepisná šírka a zemepisná dĺžka) a každý uzol má unikátne identifikačné číslo. Môže definovať body ako vrcholy, zástavky mestskej hromadnej dopravy, semafory a iné.



Obr. 2.4: Súradnicový systém WGS84 ako elipsoid. Dĺžka `a` je dĺžka hlavnej polosi (vzdialenosť od ťažiska po rovník) a dĺžka `b` je dĺžka vedľajšej polosi (vzdialenosť od ťažiska po pól).[26]

- **cesta** – usporiadaný zoznam dvoch až dvetisíc uzlov (odkazovaných pomocou identifikačného čísla), ktoré definujú krivku. Cesta môže definovať napríklad dopravné cesty, chodníky, ale aj hranice štátu alebo tvar jazera.
- **relácia** – dátová štruktúra zložená z dvoch a viac elementov (uzlov, ciest alebo relácií), ktoré popisujú, ako tieto elementy spolu fungujú. Relácia môže byť napríklad diaľnica, ktorá sa skladá z viacerých ciest alebo rôzne mnohouholníkové tvary, ktoré sa nedajú popísať jednou cestou.

Všetky tieto elementy môžu obsahovať tagy, ktoré popisujú význam elementu, pri ktorom sú obsiahnuté. Tagy popisujú elementy spôsobom `"kľúč"="hodnota"`. Pre kľúče neexistujú žiadne pravidlá, ale mali by sa držať konvencií. Napríklad niektoré z tagov vrcholu Kriváň vo Vysokých Tatrách sú `"name"="Kriváň"`, ktorý určuje meno elementu a `"natural"="peak"`, ktorý určuje, že typ elementu je vrchol. [8]

2.6.2 Overpass API

Overpass API [20] je API, ktoré povoľuje iba čítanie mapy OSM a slúži na získavanie vybraných častí z dát mapy OSM. Funguje tým spôsobom, že klient vytvorí dopyt, ktorý pošle API a to mu pošle naspäť dáta zodpovedajúce dopytu, ktorý mu klient poslal.

Oproti hlavnému OSM API, ktoré je optimalizované pre editáciu, je Overpass API optimalizované pre klientov, ktorí chcú získať pár elementov alebo až niekoľko miliónov elementov za čo najkratšiu dobu. Tieto elementy môžu byť pretriedené napríklad podľa toho či sa nachádzajú v určitej lokácii, podľa tagov a iných kritérií. Pre triedenie slúži jazyk Overpass QL.

2.7 The World Geodetic System 1984 (WGS84)

WGS84 [26] reprezentuje aktuálne najlepší geodetický referenčný systém Zeme pre mapovanie a navigáciu. Definuje súradnicový systém, základné a odvodené konštanty, geopoten-

ciálny model EGM96 (popísaný v kapitole 2.7.3) a iné. Je definovaný elipsoidom so stredom nachádzajúcim sa v ťažisku Zeme (obrázok 2.4). Súradnice tohto systému vychádzajú z geodetických súradníc, a teda poloha je určená zemepisnou šírkou, dĺžkou a geodetickou výškou. Konštanty potrebné pre túto prácu sú dĺžka hlavnej polosi, dĺžka vedľajšej polosi a umocnenú prvú excentricitu. Konštanty sú nasledovné:

- Dĺžka hlavnej polosi $a = 6378137.0\ m$
- Dĺžka vedľajšej polosi $b = 6356752.3142\ m$
- Umocnená prvá excentricita $e^2 = 6.69437999014 * 10^{-3}$

Pomocou týchto konštánt, zemepisnej šírky, dĺžky a geodetickej výšky je možné zistiť súradnice bodu v kartézskom súradnicovom systéme.

2.7.1 Výpočet kartézskych súradníc bodu pomocou geodetických súradníc

Súradnice $(x,\ y,\ z)$ kartézskeho súradnicového systému sa dajú vypočítať zo známych geodetických súradníc $(\phi,\ \lambda,\ h)$ pomocou nasledujúcich rovníc:

$$x = \big(N + h\big) \cos\phi \cos\lambda \tag{2.4}$$

$$y = \big(N + h\big) \cos\phi \sin\lambda \tag{2.5}$$

$$z = \left(\frac{b^2}{a^2} N \ + h\right) \sin\phi \tag{2.6}$$

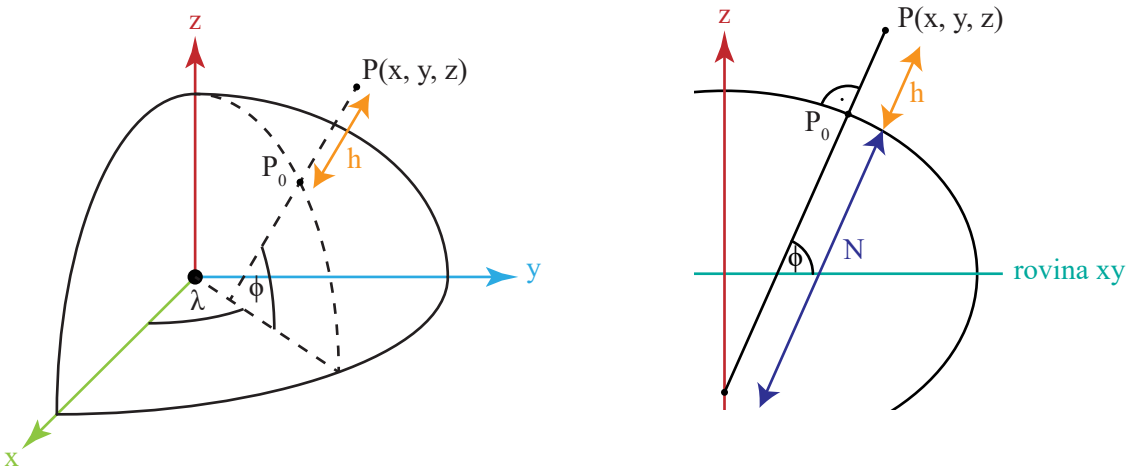
$$N = \frac{a}{\sqrt{1 - e^2 \sin^2\phi}} \tag{2.7}$$

Kde ϕ je zemepisná šírka, λ je zemepisná dĺžka a h je geodetická výška (výška relatívna k elipsoidu WGS84). N je vzdialenosť od bodu na povrchu elipsoidu (určeného zemepisnou šírkou a dĺžkou) po priesečník s osou Z. Vektor medzi týmito bodmi je rovnobežný s normálovým vektorom elipsoidu v danom bode (obrázok 2.5). [26]

2.7.2 Charakteristika geoidu

Geoid [9] je tvar, ktorý by mal povrch oceánu pod vplyvom gravitácie a rotácie Zeme samotnej, ak iné vplyvy, ako vietor a vlny budú zanedbané. Tento povrch prechádza aj cez kontinenty. Je to vlastne matematický popis Zeme. Aproximuje sa pomocou sférickej harmonickej funkcie. Povrch je hladký, ale nepravidelný. Jeho tvar je určený nerovnomerným rozložením hmoty na povrchu a vnútri Zeme. Dá sa zistiť iba pomocou rozsiahlych meraní a výpočtov. Tento koncept už je známy skoro dvesto rokov v histórii geodézie a geofyziky, ale merania s vysokou presnosťou existujú až od doby rozvíjajúcej sa satelitnej geodézie v neskorom dvadsiatom storočí.

Všetky body povrchu geoidu majú rovnakú efektívnu potenciálnu energiu (suma gravitačnej potenciálnej energie a odstredivej potenciálnej energie). Gravitačná sila pôsobí kolmo na povrch geoidu. Povrch geoidu je vyššie ako referenčný elipsoid, ak je na konkrétnom mieste pozitívna gravitačná anomália (prebytok hmoty) a naopak nižšie ako referenčný elipsoid, ak je na konkrétnom mieste negatívna gravitačná anomália (deficit hmoty).



Obr. 2.5: Ilustrácia hodnôt z rovníc 2.4, 2.5, 2.6 a 2.7 – vľavo perspektívny pohľad, vpravo prierez rovinou určenou osou z a bodom P_0 [26]. Bod $P(x,\ y,\ z)$ je určený pomocou geodetických súradníc $(\phi,\ \lambda,\ h)$. N je vzdialenosť od bodu P_0 (leží na povrchu elipsoidu) po priesečník s osou Z. h určuje geodetickú výšku nad elipsoidom WGS84 (vzdialenosť medzi bodmi P_0 a $P(x,\ y,\ z)$). Priamka prechádzajúca týmito bodmi je rovnobežná s normálou elipsoidu v bode P_0. ϕ je zemepisná šírka a λ je zemepisná výška. [26]

2.7.3 Earth Gravitational Model 1996 (EGM96)

EGM96 [26] je geopotenciálny model zeme, pomocou ktorého sa dá zistiť výškový rozdiel medzi geoidom a elipsoidom WGS84. Tento rozdiel sa pohybuje v rozmedzí približne od -100 m do 80 m.

Povrch elipsoidu WGS84 nepredstavuje hladinu oceánu. Povrch geoidu ale hladinu oceánu predstavuje. Model EGM96 slúži pre korekciu elipsoidu WGS84 aby sa čo najbližšie približoval geoidu Zeme.

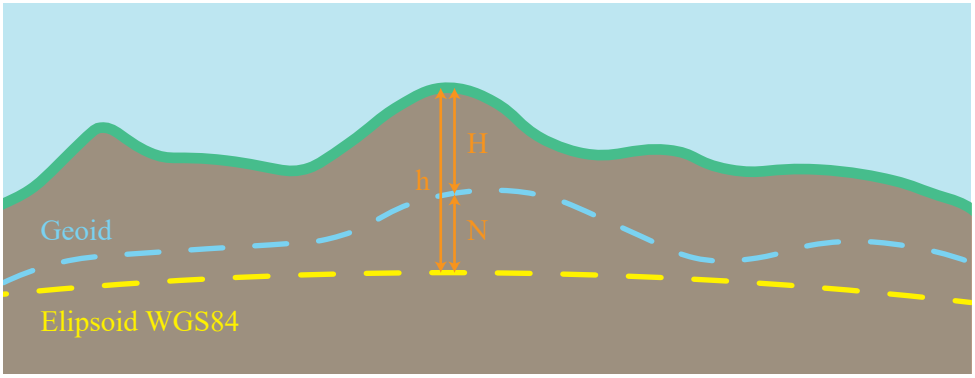
$$h = H + N \tag{2.8}$$

$$H = h - N \tag{2.9}$$

Rovnica 2.8 popisuje výpočet geodetickej výšky h (výška nad elipsoidom WGS84). Rovnica 2.9 popisuje výpočet nadmorskej výšky H (výška nad geoidom alebo ortometrická výška). Tieto hodnoty sú ilustrované na obrázku 2.6. Pre obidve rovnice platí, že N je výškový rozdiel medzi geoidom a elipsoidom WGS84 (vlnitosť). N sa dá vypočítať pomocou sférických harmonických koeficientov modelu EGM96.

Súvis s GPS

Satelity GPS obiehajúce okolo stredu gravitácie (ťažiska) Zeme môžu merať výšku iba relatívne k referenčnému elipsoidu WGS84 (geodetickú výšku). Pre zistenie reálnej nadmorskej výšky (výška relatívna ku geoidu) musí byť vykonaná korekcia pomocou tohto modelu rovnicou 2.9.



Obr. 2.6: Ilustrácia rozdielu medzi geoidom a elipsoidom relatívne k povrchu zeme. H je nadmorská výška (ortometrická výška alebo výška nad geoidom), h je geodetická výška (výška nad elipsoidom WGS84) a N je výškový rozdiel medzi geoidom a elipsoidom WGS84 (vlnitosť). [26]

2.8 Vykresľovanie grafiky v operačnom systéme Android

Pre vykresľovanie náročnejšej 2D a 3D grafiky slúži v operačnom systéme Open Graphics Library (OpenGL), konkrétne OpenGL ES API. OpenGL je multiplatformové grafické API, ktoré špecifikuje štandardný softvérový interface pre hardvér na spracovanie 3D grafiky. OpenGL ES je podmnožinou OpenGL špecifikácie a je určená pre vstavané zariadenia, ako mobilné zariadenia, tablety, herné konzoly a iné.

Pre zjednodušenie práce s grafikou existujú rôzne herné frameworky. Medzi veľmi známe frameworky patria napríklad Unity 3D[3], Unreal Engine[4] alebo CryEngine[5]. Tieto frameworky sú však až moc komplexné pre potreby tejto práce. Preto som sa rozhodol pohľadať nejakú alternatívu. Hľadal som framework, ktorý spĺňa nasledujúce vlastnosti:

- bude jednoduchý na používanie
- dobre optimalizovaný pre operačný systém Android
- bude sa dať jednoducho použiť v Android Studiu
- vlastné vývojové prostredie nie je potrebné

Všetky tieto vlastnosti spĺňa herný framework LibGDX[6]. LibGDX je relatívne nízko úrovňový, voľne dostupný, open source, multi-platformový herný vývojový framework.

2.8.1 Herný framework LibGDX

LibGDX je herný vývojový framework, ktorý poskytuje jednotné API funkčné na všetkých podporovaných platformách. Pre vykresľovanie používa OpenGL ES. Prototypuje sa v ňom jednoducho a rýchlo. Obsahuje triedy pre prácu s maticami, vektormi a quaternionmi. Taktiež obsahuje triedy pre jednoduché vytváranie modelov, textúr, shaderov, grafického užívateľského rozhrania a zjednodušuje prácu s nízko úrovňovými objektami v OpenGL, ako polia vrcholov alebo framebuffer objekty. [12]

[3] https://unity.com
[4] https://www.unrealengine.com
[5] https://www.cryengine.com
[6] https://libgdx.badlogicgames.com

Vytvorenie projektu a jeho štruktúra.

Projekt je možné vytvoriť niekoľkými spôsobmi, ale najjednoduchší je spôsob použitia aplikácie **gdx-setup.jar**[7]. Pomocou nej sa projekt vytvára len zadaním mena projektu, cesty k projektu, vybratím platforiem, pre ktoré bude vyvíjaný a vybratím rozšírení, ktoré budú v projekte zahrnuté. [7]

Keďže LibGDX je multiplatformový framework, projekt môže byť rozdelený na dva a viac modulov. Hlavný modul `core` obsahuje kód spoločný všetkým platformám, pre ktoré je projekt vyvíjaný. Ďalšie moduly majú názov podľa platformy. Táto práca je vyvíjaná pre operačný systém Android, a teda druhý modul bude mať názov `android`. Modul `android` obsahuje kód špecifický pre operačný systém a jeho štruktúra je rovnaká, ako je popísané v kapitole 2.2.2. [22]

Vytvorenie rozhrania s operačným systémom Android

Modul `android` taktiež slúži ako štartovný bod aplikácie (kapitola 2.2.2) pomocou triedy `AndroidApplication` frameworku LibGDX, ktorá rozširuje triedu `Activity`. Rozšírením triedy `AndroidApplication` je možné nastaviť rôzne nastavenia frameworku LibGDX a následne ho inicializovať. Nakoniec sa vytvorí inštancia triedy, ktorá rozširuje triedu `ApplicationListener` v module `core`, a ktorá už obsahuje samotný kód aplikácie. [28]

Pretože je projekt rozdelený na viac modulov, je potrebné vytvoriť rozhranie medzi týmito modulmi, a teda rozhranie, pomocou ktorého sa modul `core` dostane k funkciám danej platformy, ak je to potrebné. Funguje to takým spôsobom, že v module `core` sa vytvorí rozhranie, ktoré bude tento modul využívať pre prístup k funkciám platformy. V module určenému špecificky danej platforme bude toto rozhranie implementované, následne bude vytvorená jeho inštancia a táto bude pomocou konštruktora poslaná triede, ktorá rozširuje triedu `ApplicationListener`. [13]

Nastavenie perspektívnej kamery v LibGDX

Aby perspektíva videná cez obrazový senzor v zariadení a zobrazená na displeji sedela s perspektívou virtuálnej kamery, je potrebné nastaviť vertikálny uhol pohľadu virtuálnej kamery. V LibGDX trieda `PerspectiveCamera` obsahuje triednu premennú `fieldOfView`, ktorá určuje vertikálny uhol pohľadu kamery. Aby perspektíva obrazového senzoru v zariadení a virtuálnej kamery boli rovnaké musí byť premenná `fieldOfView` nastavená na rovnaký uhol pohľadu aký obrazovému senzoru v zariadení vytvára optika. Tento uhol sa dá vypočítať nasledujúcim spôsobom:

$$\alpha = 2 \arctan\left(\frac{s}{2f}\right) \frac{180}{\pi} \tag{2.10}$$

Pomocou rovnice 2.10 sa dá vypočítať uhol pohľadu α obrazového senzoru, kde s je dĺžka obrazového senzoru v ose, v ktorej sa bude počítať uhol pohľadu a f je ohnisková vzdialenosť. [17]

[7] https://libgdx.badlogicgames.com/nightlies/dist/gdx-setup.jar

Kapitola 3

Návrh aplikácie rozšírenej reality a zobrazovania prvkov z mapy v teréne

V tejto kapitole bude popísaný spôsob a postup návrhu aplikácie rozšírenej reality a zobrazovania prvkov z mapy v teréne. Bude popísaný spôsob, akým budú použité senzory v zariadení pre dosiahnutie čo najkvalitnejšej registrácie medzi virtuálnymi a reálnymi objektami v scéne. Ďalej bude popísané, ako budú získavané mapové dáta a akým spôsobom budú pretriedené a nakoniec použité. Kapitola sa bude taktiež venovať riešeniu problému s presnosťou dátového typu float vo veľkých vzdialenostiach od počiatku súradnicového systému. Potom bude popísaný spôsob použitia mapových dát pre určenie pozície objektu v scéne, ktorá bude vykresľovaná. Pre veľký počet objektov bude táto scéna spravovaná triedou `SceneManager`. Nakoniec bude popísaný návrh grafického užívateľského rozhrania.

3.1 Použitie senzorov

Základom rozšírenej reality je presná registrácia medzi reálnymi a počítačom generovanými objektami. Pre docielenie tejto registrácie je potrebné čo najpresnejšie sledovanie pózy, ktoré závisí na sledovaní orientácie a polohy v priestore. Aby bolo možné sledovať kvalitu registrácie, je potrebné použiť obrazový senzor v zariadení, ktorý bude slúžiť, ako pozadie, cez ktoré sa budú vykresľovať virtuálne objekty. Virtuálne objekty musia byť virtuálnou kamerou pozorované v správnej perspektíve, a to bude docielené správnym nastavením virtuálnej kamery tak, ako je popísané v kapitole 2.8.1. Namiesto spracovania obrazu z obrazového senzoru pre určenie orientácie bude využitý senzor vektoru rotácie (popísaný v kapitole 2.4.3), ktorým sa určí orientácia pomerne jednoducho a presne. Ďalej modul GPS popísaný v kapitole 2.5 využijem pre určenie polohy v priestore. Spojením dát z týchto senzorov sa dá docieliť celkom kvalitná registrácia.

3.1.1 Rotovanie virtuálnej kamery pohybovými senzormi

Prvou potrebnou vecou pri navrhovaní aplikácie bolo vymyslenie spôsobu synchronizácie rotácie zariadenia (obrazového senzoru v zariadení) s rotáciou virtuálnej kamery. Výstupné dáta senzoru vektoru rotácie (kapitola 2.4.3) obsahujú quaternion rotácie zariadenia a tým sa bude rotovať virtuálna kamera. Pred použitím tohto quaternionu ho treba premapovať do

koordinačného systému LibGDX (taký istý ako OpenGL). V tomto súradnicovom systéme je smer hore zhodný s rastúcou hodnotou osi y a smer dopredu s klesajúcou hodnotou osi z. V súradnicovom systéme vektoru rotácie je smer hore zhodný s rastúcou hodnotou osi z a smer dopredu s rastúcou hodnotou osi y. Po premapovaní bude rotácia virtuálnej kamery zhodná s rotáciu obrazového senzoru v priestore.

3.1.2 Náhľad z obrazového senzoru

Ako pozadie pri vykresľovaní objektov bude slúžiť náhľad z obrazového senzoru, ktorý bude získavaný spôsobom popísaným v kapitole 2.3. Pri používaní vstavanej aplikácie fotoaparátu som si všimol, že ak v móde natáčania videa zapnem funkciu 60 snímok za sekundu, tak náhľad z obrazového senzoru je veľmi ostrý a plynulý. Rozhodol som sa, že chcem docieliť niečo podobné aj v mojej aplikácii rozšírenej reality, pretože podľa môjho názoru väčšia plynulosť zlepší dojem z aplikácie. Nič podobné som zatiaľ pri iných aplikáciách rozšírenej reality nevidel. Väčšina aplikácií bola dosť pomalá a náhľad trhal. Otázkou ostávalo či zariadenie bude výkonovo zvládať tak rýchle vykresľovanie. Našťastie nebudú vykresľované moc náročné scény a počet vykreslených objektov bude predpokladám maximálne pár stoviek, čo by nemalo moc zaťažovať grafický čip v zariadení.

3.2 Polohové, výškové dáta a informácie o objektoch v teréne

Dáta budú sťahované z OSM (kapitola 2.6) pomocou Overpass API (kapitola 2.6.2). Pomocou tohto API sa dajú dáta sťahovať v rôznych formátoch ako `CSV` alebo `XML`. Keďže programovací jazyk Java podporuje parsovanie formátu `XML`, rozhodol som sa ho použiť.

Nie všetky dáta z okolia budú potrebné, a preto pred stiahnutím dát je potrebné určiť si typ dát, ktoré budú sťahované.

3.2.1 Výber typu objektov, o ktorých budú získané dáta

Keďže v aplikácii pôjde hlavne o zobrazovaní prvkov z mapy v teréne, najskôr bolo potrebné vybrať typy objektov, ktoré budú zobrazené. Rozhodol som sa pre rôzne turistické body. Ako pre turistu, sú pre mňa zaujímavé polohy nasledujúcich prvkov v teréne:

- vrcholy, sopky a výhľady
- jaskyne, vodopády, studničky a prírodné útvary, ako zaujímavé skaly
- informačné tabule, tabule s mapou a smerovacie stĺpy

Zameral som sa teda na získavanie informácií iba o týchto prvkoch v teréne z mapy OSM.

3.2.2 Získanie polohových dát a informácií o objektoch v teréne

Pomocou Overpass API budú získané dáta z okolia. Pre získanie dát je potrebné vytvoriť Overpass dopyt. Dopyt sa dá vytvoriť pomocou Overpass QL (Query Language) alebo XML. Podľa môjho názoru je Overpass QL veľmi jednoduchý na pochopenie, a preto som sa ho rozhodol použiť.

Dáta môžu byť pomocou dopytu triedené a preto bude vytvorený dopyt, pomocou ktorého budú získane iba uzly spĺňajúce typy objektov popísaných v kapitole 3.2.1.

Po vytvorení treba dopyt poslať na niektorý z Overpass serverov, ktorý následne pošle odpoveď s mapovými dátami. Tieto dáta môžu obsahovať okrem zemepisnej šírky a dĺžky aj iné informácie, ako napríklad meno či kategóriu vo forme tagov (kapitola 2.6.1). Neobsahujú však informáciu o nadmorskej výške.

3.2.3 Chýbajúce výškové dáta

Pre získanie výškových dát je potrebné použiť inú službu, pretože mapové dáta z OSM ich neobsahujú. Rozhodol som sa pre použitie Google Elevation API, pretože nadmorská výška sa tu dá zistiť veľmi presne a skúšobná verzia je zadarmo na rozdiel od konkurenčných služieb. Tak isto, ako aj pre Overpass API, aj pre Google Elevation API je potrebné vytvoriť dopyt. Ten obsahuje zoznam zemepisných šírok a dĺžok, pre ktoré nie je známa nadmorská výška. Pre programovací jazyk Java, ale existuje knižnica pre Google Maps Services[1], ktorá toto odosielanie zjednodušuje. Server nakoniec odpovie nadmorskými výškami pre polohy, ktoré mu boli zaslané.

3.2.4 Uloženie mapových dát

Aby pri každom spustení aplikácie nemuseli byť dáta z blízkeho okolia sťahované odznova, budú vždy po stiahnutí polohových dát, výškových dát a tagov tieto dáta uložené do pamäti zariadenia. Keďže uzly sú vždy identifikované unikátnym identifikačným číslom, budú tieto uzly vložené do mapy, kde kľúčom bude identifikačné číslo a hodnotou budú všetky údaje zistené z uzlu. Táto mapa bude následne serializovaná a uložená do súboru v pamäti zariadenia. Pri spustení aplikácie bude tento súbor prečítaný, deserializovaný a znova použitý pre získanie mapových dát.

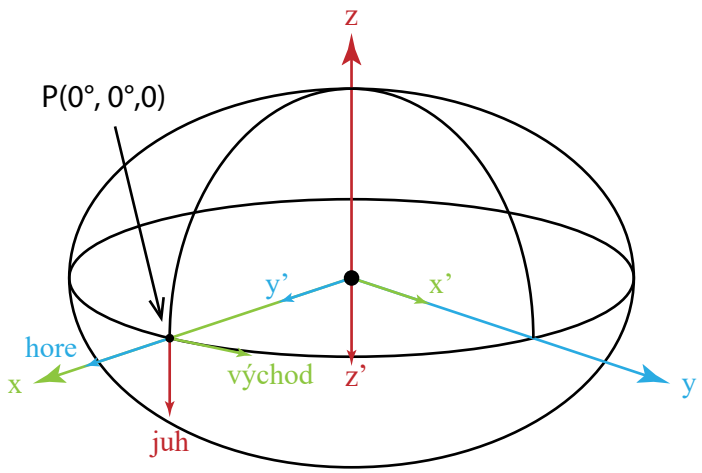
3.3 Pozícia a orientácia virtuálnej kamery a objektov v priestore

Keď sú všetky potrebné polohové dáta známe, je možné vypočítať pozíciu v kartézskom súradnicovom systéme. Pre toto bude slúžiť trieda `Location`, ktorej inštancia bude inicializovaná zemepisnou šírkou, dĺžkou a nadmorskou výškou. Bude mať metódu `toCartesian()`, ktorej návratová hodnota bude vektor obsahujúci súradnice bodu v kartézskom súradnicovom systéme. Výpočet bude vykonaný spôsobom popísaným v kapitole 2.7.1. Tento spôsob ale počíta s geodetickou výškou, no výška získaná pomocou Google Elevation API je nadmorská. Najskôr bude teda potrebné vykonať premenu nadmorskej výšky do geodetickej pomocou modelu EGM96, a to zistením vlnitosti v danej polohe a následným použitím rovnice 2.8 v kapitole 2.7.3.

3.3.1 Mapovanie súradnicového systému WGS84 do LibGDX

Ako referenčnú polohu pre určenie smerov hore, južne a východne som si vybral zemepisnú šírku 0° a dĺžku 0° s tým že juh bude v smere stúpajúcej hodnoty osi z a rovnobežne s povrchom Zeme a smer hore bude kolmý na povrch Zeme (obrázok 3.1). V súradnicovom systéme LibGDX je smer hore zhodný so smerom osi y a smer dopredu je zhodný so smerom klesajúcej hodnoty osi z, a teda os z stúpa v smere dozadu. Z obrázku 3.1 je vidieť, že rovnice

[1] https://github.com/googlemaps/google-maps-services-java



Obr. 3.1: Zobrazenie nového premapovaného súradnicového systému $(x',\ y',\ z')$ z pôvodného systému $(x,\ y,\ z)$ zhodného so systémom WGS84 (kapitola 2.7). Nový systém je určený pomocou bodu $P(0°,\ 0°,\ 0)$ (zemepisná šírka 0° a dĺžka 0° a geodetická výška je 0 m). V tomto bode je vo WGS84 smer hore zhodný s osou x, v LibGDX s osou y. Smer na východ v bode $P(0°,\ 0°,\ 0)$ je vo WGS84 zhodný s osou y, v LibGDX bude s osou x. Nakoniec smer na juh vo WGS84 je zhodný s osou opačnou osi z a v LibGDX bude tento smer zhodný s osou z.

nového súradnicového systému budú vyzerať nasledovne:

$$x' = y \ = \big(N + h\big) \cos\phi \sin\lambda \tag{3.1}$$

$$y' = x \ = \big(N + h\big) \cos\phi \cos\lambda \tag{3.2}$$

$$z' = -z \ = -\left(\frac{b^2}{a^2} N \ + h\right) \sin\phi \tag{3.3}$$

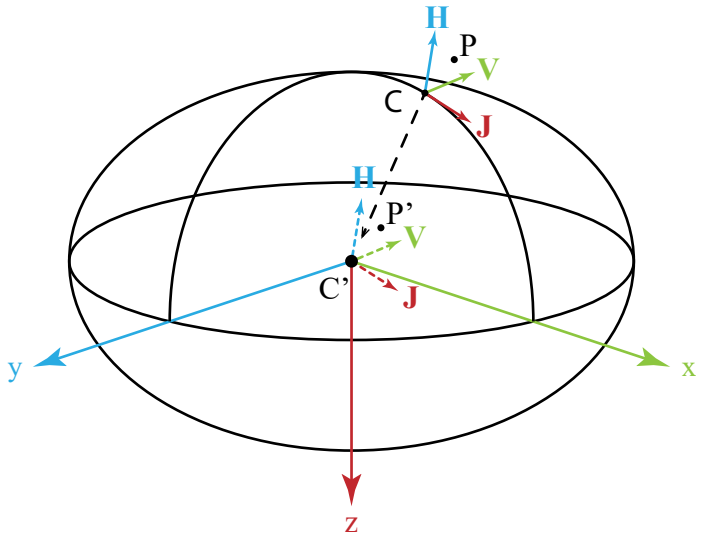
3.3.2 Problém presnosti dátového typu float a jeho riešenie

Keďže objekty sa budú v scéne nachádzať niekoľko miliónov metrov od počiatku súradnicového systému, prestáva dátový typ float mať potrebnú presnosť. Pri prvotných testoch bola virtuálna kamera umiestnená do takejto vzdialenosti a jej následné rotovanie spôsobovalo silné trasenie obrazu. Toto trasenie sa ale nevyskytovalo v blízkosti počiatku súradnicového systému.

Dostal som teda nápad premiestniť celú scénu (všetky vykresľované objekty) spolu s kamerou, tak aby sa kamera nachádzala vždy na počiatku súradnicového systému. Inak povedané treba vypočítať pozíciu objektov v lokálnom súradnicovom systéme kamery. Aby sa toto nemuselo vykonávať pri každej zmene polohy kamery, sa kamera môže pohybovať iba v určitej vzdialenosti od počiatku súradnicového systému. Ak túto vzdialenosť prekoná, celá scéna spolu s kamerou sa znova presunie, aby sa kamera nachádzala v počiatku súradnicového systému.

3.3.3 Presunutie kamery a scény do počiatku súradnicového systému

Pre presunutie kamery do počiatku súradnicového systému musí byť od bodu, v ktorom leží kamera odčítaný vektor utvorený medzi bodom, v ktorom leží kamera a počiatkom



Obr. 3.2: Ilustrácia presunu scény, ktorú tvorí bod P a kamera C do počiatku súradnicového systému. Bod P' a C' už sú presunuté s tým, že bod C' leží v počiatku súradnicového systému. Vektor **H** je vektor smerujúci hore a je kolmý na povrch Zeme, vektor **J** smeruje na juh a je rovnobežný s povrchom zeme. **V** ukazuje na východ a taktiež je rovnobežný s povrchom Zeme.

súradnicového systému. Tým vznikne bod (0, 0, 0). Pre posunutie celej scény musíme tento vektor odčítať od všetkých bodov v scéne. Toto je ilustrované na obrázku 3.2.

3.3.4 Určenie orientácie scény relatívne k polohe na elipsoide WGS84

Keďže Zem má tvar elipsoidu, všetky polohy s rozdielnou zemepisnou šírkou a dĺžkou budú mať smer hore (rastúci smer osi y), ktorý je kolmý na povrch Zeme rôzny. Tak isto je rôzny aj smer na juh (rastúci smer osi z), ktorý musí byť rovnobežný s povrchom Zeme. Smer na východ (rastúci smer osi x) sa mení iba v závislosti na zemepisnej výške. Vektory týchto smerov sa dajú vypočítať pomocou zemepisnej šírky ϕ a zemepisnej dĺžky λ týmto spôsobom:

$$\mathbf{H} = (\cos\phi \sin\lambda,\ \cos\phi\cos\lambda,\ -\sin\phi) \tag{3.4}$$

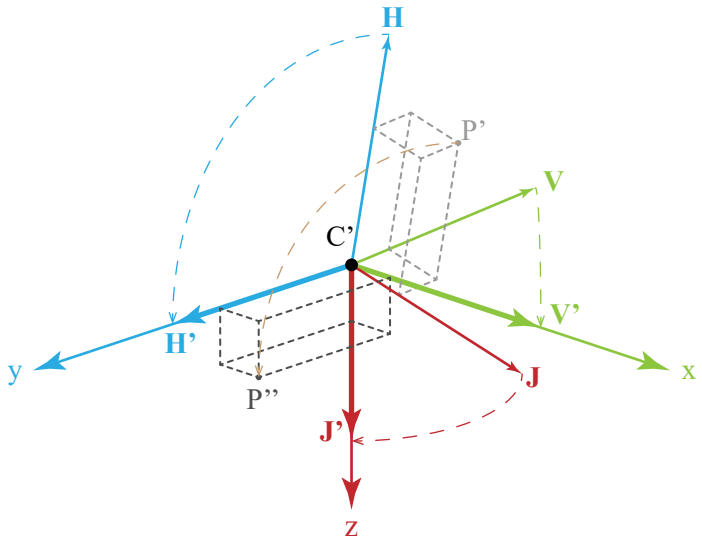
$$\mathbf{J} = (\sin\phi \sin\lambda,\ \sin\phi\cos\lambda, \cos\phi) \tag{3.5}$$

$$\mathbf{V} = (\cos\lambda,\ -\sin\lambda,\ 0) \tag{3.6}$$

Vektory 3.4, 3.5 a 3.6 som odvodil z vlastností gule a jej súvislosti s goniometriou. Smer hore určuje vektor 3.4, smer na juh vektor 3.5 a smer na východ vektor 3.6. Pre získanie týchto vektorov budú v triede `Location` metódy `getUpPointingVector()`, `getSouthPointingVector()` a `getEastPointingVector()`.

Po zistení týchto vektorov je možné vytvoriť transformačnú maticu. Táto matica sa vytvorí z vektorov vypočítaných zo zemepisnej šírky ϕ a zemepisnej dĺžky λ polohy zariadenia zistenej pomocou modulu GPS.

Pre vytvorenie tohto typu matice existuje v LibGDX metóda `set(Vector3 xAxis, Vector3 yAxis, Vector3 yAxis, Vector3 pos)` triedy `Matrix4`. Ako `xAxis` bude nastavený vektor smerujúci na východ, ako `yAxis` bude nastavený vektor smerujúci hore a ako



Obr. 3.3: Pokračovanie k obrázku 3.2. Zobrazené je len okolie počiatku súradnicového systému. Obrázok zobrazuje zmenu orientácie scény pomocou vektorov **H** (smer hore), **V** (smer východ) a **J** (smer juh). Tieto tri vektory musia byť transformované do vektorov rovnobežných so súradnicovým systémom. Spolu s nimi sa transformuje aj bod P' do bodu P''. P'' už je finálna poloha bodu.

`zAxis` bude nastavený vektor smerujúci na juh. Vektor `pos` bude nastavený na nulový vektor, pretože pozícia počiatku súradnicového systému, voči ktorému sa bude vykonávať transformácia je vždy v bode (0, 0, 0).

Po vynásobení touto maticou sa objekt (transformovaný pomocou metódy popísanej v kapitole 3.3.3) dostane do rovnakej relatívnej pozície ku virtuálnej kamere, akú má objekt v reálnom svete ku obrazovému senzoru v zariadení. Táto pozícia je už konečná a na nej sa bude vykresľovať objekt. Virtuálna kamera je v tomto momente orientovaná tak, že jej vektor pohľadu je rovnobežný s povrchom zeme a smeruje na sever (do smeru klesajúcej osi z).

3.4 Spravovanie objektov v scéne

Z mapy OSM bude sťahovaných veľmi veľa informácií o prvkoch z terénu a pri tisíc až desaťtisíc prvkoch by určite nebolo možné vykresľovať všetky tieto objekty bez vplyvu na snímkovú frekvenciu. Pre tento problém budú slúžiť triedy `SceneManager` a `SceneObjectFilter`, ktoré sa budú starať o spravovanie scény a triedenie objektov, ktoré budú vykresľované.

3.4.1 Trieda `SceneManager`

Trieda `SceneManager` bude slúžiť na spravovanie všetkých objektov, ktoré budú vykresľované. Bude obsahovať zoznam všetkých vykresliteľných objektov a pomocou triedy `SceneObjectFilter` popísanej v kapitole 3.4.2 z nich bude vyberať iba množinu objektov, ktoré sa budú vykresľovať. Tieto objekty ešte pretransformuje podľa pozície počiatku súradnicového systému a geografickej polohy zariadenia.

Sledovanie pozície kamery a úprava počiatku súradnicového systému

Na začiatok nastaví počiatok súradnicového systému na počiatočnú pozíciu kamery určenú pomocou rovníc 3.1, 3.2 a 3.3 a bude sledovať pohyb kamery v tomto priestore. Ak sa dostane do určitej vzdialenosti, nastaví túto novú pozíciu ako počiatok súradnicového systému, tak ako je popísané v kapitole 3.3.2.

Výpočet pozície v lokálnom súradnicovom systéme kamery

Pri každej zmene počiatku súradnicového systému vypočíta `SceneManager` pozície všetkých objektov v scéne relatívne ku kamere, a teda v jej lokálnom súradnicovom systéme. Použitá bude metóda popísaná v kapitole 3.3.3.

Transformácia celej scény podľa geografickej polohy kamery

Keď je vypočítaná pozícia v lokálnom súradnicovom systéme kamery, objekty ešte nie sú v dobrej pozícii relatívne k orientácii kamery. Je to spôsobené tým že Zem má tvar elipsoidu. Pre výpočet tej správnej a finálnej polohy je potrebné každý objekt transformovať pomocou transformačnej matice vytvorenej zo smerových vektorov určených z geografickej polohy kamery (obrázok 3.3). Postup vytvorenia a použitia tejto matice je popísaný v kapitole 3.3.4.

3.4.2 Trieda `SceneObjectFilter`

Trieda `SceneObjectFilter` najskôr vyberie všetky objekty, ktoré sú v dohľade kamery podľa parametru `far` pohľadového objemu (view frustum). Potom rozdelí pohľad na bloky, kde v každom bloku sa môže nachádzať iba jeden objekt. Je to preto lebo všetkých objektov budú tisíce, a ak by boli všetky vykreslené tak bude prostredie plné objektov a nebude prehľadné. Do tohto bloku je vybratý objekt podľa určitých kritérií. Tie sú popísané v kapitole 4.3.3.

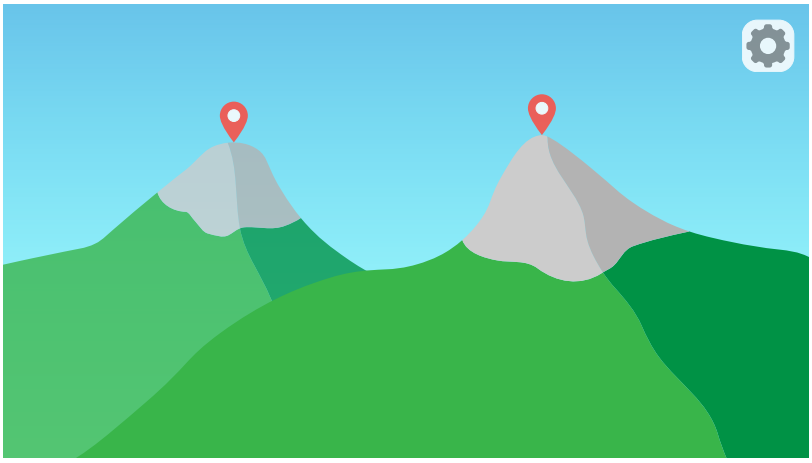
3.5 Grafické užívateľské rozhranie

Grafické užívateľské rozhranie je veľmi dôležitý prvok aplikácie, pretože tvorí prvotný dojem z aplikácie, a preto som sa zameral na vzhľad aplikácie. Zvolil som plochý dizajn grafického užívateľského rozhrania, pretože podľa môjho názoru vyzerá jednoducho a dobre. V takom plochom dizajne sú vytvorené aj ilustrácie grafického užívateľského rozhrania na obrázkoch 3.4 a 3.5.

3.5.1 Vzhľad objektov a ich zobrazené informácie

Aby sa dalo rozlišovať medzi rôznymi typmi objektov popísaných v kapitole 3.2.1, bude každý z typov mať vlastnú značku, ktorá bude vykresľovaná. Tieto značky budú vykreslené v 3D priestore na mieste, na ktorom sa nachádza objekt po všetkých transformáciách popísaných v kapitole 3.4.1. Pre predstavu je na obrázku 3.4 vyznačená červeno biela značka pri dvoch vrcholoch.

Značka bude vykreslená tak aby vždy bola natočená smerom ku kamere s obmedzením na ose y, a teda značka bude rotovaná iba okolo osi y. Viditeľná veľkosť značky nebude závislá na vzdialenosti od kamery. To znamená, že všetky značky nachádzajúce sa v scéne budú mať v pohľade "cez" kameru zhodnú veľkosť.

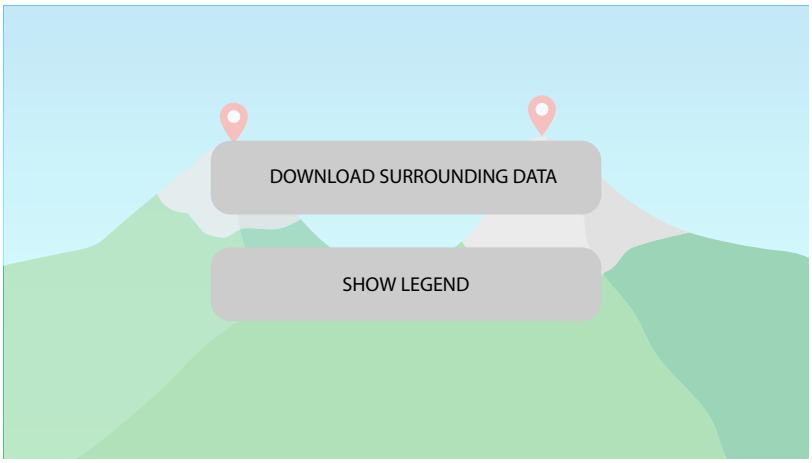


Obr. 3.4: Ilustrácia vzhľadu aplikácie po jej spustení. Pozadie tvorí náhľad z obrazového senzoru zobrazujúce hory. Červeno biele značky rozširujú vrcholy hôr o nejaké informácie, ktoré budú vypísané nad značkou. Namiesto týchto červeno bielych značiek bude mať každý typ objektu z kapitoly 3.2.1 vlastnú unikátnu značku, aby sa dalo medzi nimi rozlišovať už na prvý pohľad. V pravom hornom rohu je tlačidlo, ktoré po stlačení zobrazí menu s nastaveniami.

Značka bude doplnená informáciami o objekte, a to konkrétne názov objektu a vzdialenosť objektu od kamery. Pri väčšine typov objektov (kapitola 3.2.1) nie je potrebné vedieť ich nadmorskú výšku. Pri vrcholoch, sopkách a výhľadoch táto výška určite zaujímavá je, a preto pri značkách rozširujúcich niektorý z týchto typov objektov bude nadmorská výška zobrazená tiež.

3.5.2 Nastavenia

Aby bolo grafické užívateľské rozhranie prehľadné a neobsahovalo zbytočne veľa prvkov, budú nastavenia a funkcie mať vlastné miesto. Nastavenia sa budú otvárať pomocou tlačidla umiestnenom v pravom hornom rohu aplikácie. Po stlačení sa nastavenia vysunú z pravého okraja okna a prekryjú jeho aktuálny obsah. Pozadie nastavení bude priehľadné tak, ako na obrázku 3.5. Pre vypnutie nastavení bude stačiť ťuknúť kamkoľvek v nastaveniach, kde nebudú aktívne prvky (napríklad tlačidlo). Po ťuknutí sa okno s nastaveniami zasunie naspäť do pravého okraja okna. Vzhľad nastavení bude vyzerať podobne ako na obrázku 3.5.



Obr. 3.5: Ilustrácia vzhľadu nastavení aplikácie. Po stlačení tlačidla na pravo hore (obrázok 3.4) sa zobrazia nastavenia. Tu sa budú dať stiahnuť dáta o okolí z OSM alebo zobraziť legendu popisujúcu objekty, ktoré aplikácia pozná a môžu byť vykreslené.

Kapitola 4

Implementácia pre Android a výsledky

V tejto kapitole bude popísaný spôsob implementácie aplikácie rozšírenej reality a zobrazovania prvkov z mapy v teréne. Najskôr bude popísaná štruktúra projektu, ktorá popisuje rozloženie tried, rozhraní a balíčkov, v ktorých sa tieto nachádzajú. Ďalej sa zameria na konkrétny popis niektorých zaujímavých častí implementácie, ako implementácie geodetického systému WGS84, triedy, ktorá uchováva údaje o polohe na Zemi alebo tried pre spravovanie scény. Bude tiež popísaná implementácia objektov, ktoré sa nachádzajú v scéne a nakoniec budú zobrazené výsledky a snímky obrazovky priamo z obrazovky telefónu pri použití v teréne.

4.1 Štruktúra projektu

Projekt je rozdelený do dvoch modulov. Prvým je modul `android` a druhým je modul `core`. V obidvoch moduloch je adresár `java`, ktorý obsahuje implementáciu modulov. Tieto moduly sú implementované v mennom priestore (balíčku) `cz.vutbr.fit.xhalas10.bp`.

4.1.1 Modul core

V tomto module sa nachádza multiplatformová implementácia projektu. Obsahuje všetko od získavania dát z mapy až po spravovanie a vykresľovanie scény a grafického užívateľského rozhrania pomocou LibGDX.

Balíček `earth`

Balíček slúži na abstrakciu súradnicového systému WGS84. Bližší popis a spôsob implementácie tried implementujúcich systém WGS84 sa nachádza v kapitole 4.2. Ďalej sa tu nachádzajú triedy, ktoré implementujú virtuálnu kameru a objekty v scéne. Toto je popísané v kapitole 4.2.

Balíček `gui`

Tento balíček obsahuje dve triedy. Prvou je trieda `MySkin`, ktorá popisuje vzhľad prvkov grafického užívateľského rozhrania. Načítavajú sa tu tiež všetky textúry a fonty potrebné pre vykresľovanie grafického užívateľského rozhrania. Ďalšou triedou je trieda `UserInterface`.

V tejto triede je implementované grafické užívateľské rozhranie, ktoré je vykresľované. Sú v ňom implementované všetky tlačidlá a iné prvky rozhrania. Taktiež je tu implementovaná interakcia s hlavnou triedou `ARNature` pomocou týchto prvkov a taktiež animácie a prechody užívateľského rozhrania.

Balíček `multiplatform.interfaces`

V tomto balíčku sa nachádzajú všetky rozhrania, ktoré je potrebné implementovať zvlášť v každom operačnom systéme. Pomocou týchto rozhraní dokáže multiplatformová časť aplikácie komunikovať s časťou, ktorú je možné implementovať iba v určitom operačnom systéme.

Balíček `osm`

Tento balíček obsahuje triedy, ktoré slúžia na získavanie dát z mapy OSM pomocou Overpass API, pre spracovanie týchto prijatých dát do štruktúr a nakoniec pre získanie výškových dát pomocou Google Elevation API. Viac o OSM je v kapitole 2.6. Overpass API je detailnejšie rozobrané v kapitole 2.6.2. Návrh ako budú získavané dáta a ich spracovanie je rozobraté v kapitole 3.2.

Balíček `scene`

Rozhrania a triedy potrebné pre spravovanie a vykresľovanie scény sú súčasťou balíčka `scene`. Detaily o tomto balíčku sú popísané v kapitole 4.3.

Balíček `utils`

V tomto balíčku sa nachádzajú pomocné triedy potrebné pre projekt. Obsahuje tri triedy

Prvou triedou je `FontGenerator`. Táto trieda slúži na vytvorenie objektu `BitmapFont`[1], pomocou ktorej sa dá vykresľovať zadaný text. Pre vytvorenie objektu je potrebný súbor s fontom. Pre túto prácu som použil font OpenSans[2], pretože sa mi páči jeho vzhľad a jednoduchosť. V tomto fonte sú vykresľované všetky texty viditeľné v aplikácii. Trieda obsahuje statickú metódu `generateFont()`, ktorej parametrami sú cesta k súboru s fontom, veľkosť vygenerovaného fontu a prepínač, ktorý určuje či bude mať vygenerovaný font pozadie pre lepšiu čitateľnosť.

Ďalšou triedou je `TextureTextGenerator`. Dá sa pomocou nej vytvoriť textúra obsahujúca zadaný text. Má statickú metódu `generateTexture()`, ktorej parametre sú text, ktorý bude na textúre, veľkosť fontu, farba a prepínač určujúci hrubosť písma.

Poslednou triedou je `Vector3d`. Táto trieda implementuje vektor v trojdimenzionálnom priestore. Framework LibGDX už takúto triedu obsahuje (trieda `Vector3`[3]), ale tá pracuje iba s dátovým typom float. Trieda `Vector3d` pracuje s dátovým typom double a preto má v názve pridané písmeno „d“. Neimplementuje ale všetky funkcie triedy `Vector3`. Implementuje iba tie, ktoré sú potrebné pre túto aplikáciu.

[1] https://github.com/libgdx/libgdx/wiki/Bitmap-fonts
[2] https://fonts.google.com/specimen/Open+Sans
[3] https://github.com/libgdx/libgdx/wiki/Vectors%2C-matrices%2C-quaternions

Trieda `ARNature`

Táto trieda je hlavnou triedou aplikácie a riadi celý jej chod. Je to trieda, ktorej inštancia je vytvorená hlavnou aktivitou (trieda `AndroidLauncher`) aplikácie. Po vytvorení tejto triedy sa vytvoria inštancie tried `UserInterface`, `EarthCamera` a iných potrebných tried. Táto trieda obsahuje metódu `render()`, ktorá je zavolaná vždy, keď je pripravená nová snímka náhľadu z kamery. V tejto metóde sa najskôr vyčistí kresliaca plocha LibGDX. Ďalším krokom je aktualizácia rotácie kamery quaternionom zo senzorov a potom sa vykreslí aktuálna snímka náhľadu z kamery. Ak bola zaznamenaná zmena polohy zariadenia, táto nová poloha sa nastaví virtuálnej kamere v priestore. Nakoniec sa vykreslí celá scéna a grafické užívateľské rozhranie.

4.1.2 Modul `android`

V tomto module sa nachádza implementácia potrebná pre operačný systém Android a sú tu tiež implementované rozhrania potrebné pre modul `core`. Obsahuje implementáciu získavania dát zo senzorov, náhľad z obrazového senzoru a tiež získavanie polohy zariadenia.

Balíček `android.implementation`

Triedy v tomto balíčku implementujú rozhrania, ktoré sú v balíčku `multiplatform.interfaces` modulu `core`. Triedy sú rovnomenné v obidvoch balíčkoch s tým, že rozhrania ešte začínajú písmenom „I“ (Interface).

Trieda `CameraPreview` implementuje náhľad z kamery spôsobom popísaným v kapitole 2.3. Podarilo sa dosiahnuť náhľadu so snímkovou frekvenciou 60 snímok za sekundu podľa špecifikácie v návrhu v kapitole 3.1.2.

Ďalej trieda `DeviceLocation` implementuje získavanie dát o polohe zariadenia, tak ako je popísané v kapitole 2.5.

Trieda `MotionSensors` implementuje získavanie rotačného quaternionu zo senzorov. Implementácia bola vykonaná podľa postupu v kapitole 2.4 s tým, že quaternion, ktorý je výstupom z tejto triedy už je premapovaný podľa toho, ako je popísané v kapitole 3.1.1.

Poslednou triedou v tomto balíčku je trieda `Utils`. Hlavnou úlohou tejto triedy je vytvorenie jedného rozhrania, ktoré bude poslané do hlavnej triedy v balíčku `core`, a pomocou ktorej si bude môcť vypýtať implementácie ostatných rozhraní popísaných vyššie.

Trieda `AndroidLauncher`

Táto trieda rozširuje triedu `AndroidApplication`[4] frameworku LibGDX. Keďže Android potrebuje ako začiatočný bod aplikácie objekt triedy `Activity` (viac v kapitole 2.2.2), tak trieda `AndroidApplication` rozširuje túto triedu.

Trieda `AndroidLauncher` slúži hlavne pre konfiguráciu frameworku LibGDX a vytvorenie inštancií tried v balíčku `android.implementation`. Následne vytvorí a spustí inštanciu hlavnej triedy z balíčku `core`, ktorá sa potom stará o beh aplikácie. Taktiež sa stará o získanie povolení pre získavanie polohy a použitie obrazového senzoru v telefóne.

[4] https://libgdx.badlogicgames.com/ci/nightlies/docs/api/com/badlogic/gdx/backends/android/AndroidApplication.html

Manifest a zdroje

V tomto module sa nachádza súbor `AndroidManifest.xml`, ktorý popisuje, v ktorej aktivite začína aplikácia, oprávnenia, ktoré aplikácia potrebuje a iné informácie o aplikácii (viac o manifeste v kapitole 2.2.2). Taktiež obsahuje adresár `assets`, v ktorom sa nachádzajú všetky textúry, fonty, shadery a iné zdroje. Tieto budú pribalené k aplikácii pri kompilácii.

4.2 Implementácia geodetického systému WGS84

Triedy geodetického systému sú implementované v module `core` v balíčku `earth.wgs84`, ktorý sa nachádza v mennom priestore aplikácie. Tieto triedy sú dve. Prvá z nich je trieda `GeoidUndulation`, ktorá slúži na výpočet vlnitosti geoidu (rozdiel medzi nadmorskou výškou a geodetickou výškou, čiže výškou nad elipsoidom WGS84). Druhou je trieda `Location`, ktorá slúži na uloženie geografických súradníc. Pomocou nich potom vie vypočítať pozície v kartézskom súradnicovom systéme a smerové vektory na danej geografickej polohe.

4.2.1 Kalkulácia vlnitosti geoidu pomocou modelu EGM96

Algoritmus kalkulácie vlnitosti geoidu je implementovaný v triede `GeoidUndulation` pomocou návrhového vzoru jedináčik. Rozhodol som sa použiť tento návrhový vzor, pretože pre výpočet vlnitosti je potrebná jediná metóda a prístup k tejto metóde by mali mať všetky triedy bez vytvárania nových inštancií tejto triedy. Viac informácií o tom čo je to geoid a EGM96 je popísané v kapitolách 2.7.2 a 2.7.3.

Algoritmus

Vlnitosť geoidu je výškový rozdiel medzi nadmorskou výškou a geodetickou výškou. Tento rozdiel sa dá vypočítať pomocou modelu EGM96. Implementácia tohto výpočtu je ale veľmi zložitá. Našťastie agentúra NGA (NATIONAL GEOSPATIAL-INTELLIGENCE AGENCY) poskytuje tento algoritmus na svojej webovej stránke[5]. Konkrétne to je súbor F477.f. Je to zdrojový kód napísaný v jazyku Fortran. Na webovej stránke boli k tomuto programu dostupné aj súbory s koeficientami modelu EGM96, ktoré tento program využíva.

Program je ale napísaný v programovacom jazyku Fortran, a preto ho bolo potrebné prepísať do jazyka Java. Bolo to pomerne náročné, pretože jazyk Fortran som nikdy nevidel, jeho syntax mi bola úplne neznáma a kód bol veľmi neprehľadný. Program sa mi ale nakoniec podarilo prepísať a fungoval tak, ako mal aj po prepísaní do jazyku Java. Ako referenciu som používal už skompilovaný program pre Windows XP (fungoval aj vo Windows 10) z tej istej webovej stránky odkiaľ som získal zdrojový kód v jazyku Fortran.

Optimalizácia algoritmu

Pri skúmaní prepísaného kódu v Jave som si všimol, že ku štyrom poliam, ktoré boli vytvárané zo súborov s koeficientami a boli vytvorené pri inicializácii triedy sa neskôr už nič nezapisuje a iba sa z nich číta. Vytvorenie týchto polí trvalo niekoľko sekúnd pri spúšťaní aplikácie, pretože boli vytvárané zo súborov s koeficientami a tie majú dokopy približne 10,7 megabajtov. Popri vytváraní polí boli vykonávane aj matematické operácie, ktoré tiež zvyšovali dobu načítania. Napadlo ma teda, že sa tieto polia po vytvorení zoserializujú a uložia do súborov, ktoré sa potom pri spúšťaní aplikácie znova deserializujú a načítajú

[5] http://earth-info.nga.mil/GandG/wgs84/gravitymod/egm96/egm96.html

do polí. Tieto vytvorené súbory majú spolu približne 2 megabajty a budú priložené v balíčku s aplikáciou. Tento krok zrýchlil načítanie aplikácie z niekoľko sekúnd na pár stotín sekundy.

Štruktúra triedy

Metódy, ktoré boli implementované podľa algoritmu v jazyku Fortran nachádzajúce sa v triede `GeoidUndulation` majú privátnu viditeľnosť. Trieda má dve verejné metódy. Keďže implementuje návrhový vzor jedináčik, tak prvou je statická metóda `getInstance()`, ktorá vráti jedinú inštanciu triedy. Druhou verejnou metódou je metóda `getUndulation(double lat, double lon)`, ktorá po zadaní zemepisnej šírky a dĺžky vráti rozdiel medzi geoidom a elipsoidom WGS84. Tento rozdiel bude môcť byť použitý na výpočet nadmorskej výšky pomocou geodetickej výšky použitím rovnice 2.9 alebo naopak geodetickej výšky pomocou nadmorskej výšky použitím rovnice 2.8.

4.2.2 Trieda `Location`

Trieda `Location` slúži na uloženie geografických súradníc, získanie súradníc v kartézskom súradnicovom systéme a na získanie smerových vektorov v danej polohe. Inštancia tejto triedy sa inicializuje pomocou zemepisnej dĺžky, zemepisnej šírky a nadmorskej výšky. Obsahuje taktiež metódy pre nastavenie alebo získanie jednej z týchto hodnôt.

Štruktúra a popis metód

Trieda obsahuje metódu `toCartesian()`, pomocou ktorej sa dá vypočítať z geografických súradníc pozícia v kartézskom súradnicovom systéme. Pre toto je využitý spôsob popísaný v kapitole 2.7 rozšírený o poznatky z návrhu, ktoré sú vysvetlené v kapitole 3.3.1. Výpočet ale potrebuje geodetickú výšku. Preto vždy po zmene nadmorskej výšky, a teda buď pri nastavení nadmorskej výšky pomocou nastavovacej metódy alebo pri inicializácii inštancie je vypočítaná geodetická výška s použitím metódy `getUndulation(double lat, double lon)` triedy `GeoidUndulation`. Je pri tom využitá rovnica 2.8.

Metódy pre získanie smerových vektorov v polohe, na ktorú je nastavená inštancia triedy sú `getUpPointingVector()`, `getSouthPointingVector()` a `getEastPointingVector()`. Použitie týchto vektorov je popísané v kapitolách 3.3.4 a 3.4.1.

Optimalizácia

Keďže metódy pre výpočet smerových vektorov a metóda `toCartesian()` potrebujú pre svoj výpočet sínusy a kosínusy zemepisnej šírky a dĺžky, tak tieto nie sú počítané vždy pri zavolaní metódy. Vypočítajú sa vždy iba vtedy, keď sa zmení zemepisná poloha. Tak isto výpočet smerových vektorov alebo súradníc pre jednu a tú istú polohu je zbytočné opakovať. Preto sa výpočet vykoná iba po zmene niektorej z hodnôt geografickej polohy, na ktorú je inštancia inicializovaná.

4.3 Spravovanie scény

Pre spravovanie scény slúži balíček `scene` nachádzajúci sa v mennom priestore aplikácie v module `core`. Je to skupina tried a rozhraní, ktoré slúžia pre abstrakciu objektov v scéne

aby s nimi triedy `SceneManager` a `SceneObjectFilter` vedeli jednoducho a jednotne pracovať. Tieto triedy sa starajú o výber objektov, ktoré budú vykreslené podľa určitých kritérií, ich umiestnenie v scéne a nakoniec trieda `SceneManager` obsahuje metódu `renderScene()`, ktorá celú scénu vykreslí.

4.3.1 Rozhrania pre objekty v scéne

Rozhrania, s ktorými pracuje trieda `SceneManager` sa nachádzajú v balíčku `scene.interfaces`. Tieto rozhrania sú tri, a to konkrétne `ISceneCamera`, `ISceneObject` a `ISceneDrawableObject`. Objekty, s ktorými má trieda `SceneManager` pracovať musia niektoré z týchto rozhraní implementovať.

Rozhranie `ISceneObject`

Toto rozhranie slúži pre abstrakciu každého objektu v scéne. Obsahuje nasledujúce metódy:

- `calculateOriginRelativePosition()` – Slúži na prepočítanie polohy relatívnej k počiatku súradnicového systému. Aktuálny bod počiatku, voči ktorému je počítaná poloha, je uložený v inštancii triedy `SceneManager`. Tento počiatok samozrejme nemusí byť v polohe (0, 0, 0), ale kdekoľvek v scéne. Spôsob výpočtu je popísaný v kapitole 3.3.3.
- `getOriginRelativePosition()` – Vráti aktuálnu polohu relatívnu k počiatku súradnicového systému.
- `getScenePosition()` – Vráti absolútnu polohu v scéne. Pri implementácii tejto metódy je z geografickej polohy vypočítaná poloha v kartézskom súradnicovom systéme tak, ako je to navrhnuté v kapitole 3.1.
- `update()` – V tejto metóde je možné naimplementovať zmeny objektu, ktoré sa majú diať pred každým novým vykresľovaním

Rozhranie `ISceneDrawableObject`

Toto rozhranie taktiež rozširuje rozhranie `ISceneObject`. Slúži pre abstrakciu objektov, ktoré sa budú vykresľovať. Jednou z metód tohto rozhrania je metóda `getRenderableProvider()`. Táto metóda vráti objekt `RenderableProvider`[6] frameworku LibGDX, ktorý slúži pre rendrovanie pomocou triedy `ModelBatch`[7] alebo pre pridanie do vyrovnávacej pamäti, ktorú implementuje trieda `ModelCache`[8]. Tento objekt obsahuje dáta z inštancie modelu, ako vertexy a textúry.

Ďalšou metódou je metóda `getMaximumDrawableDistance()`, ktorá vráti vzdialenosť v metroch od virtuálnej kamery, ktorá je maximálna pre vykresľovanie daného objektu. Slúži pre preriedenie scény od objektov, ktoré nie sú až tak dôležité vo veľkej vzdialenosti (napríklad tabule s mapami alebo smerovacie stĺpy) alebo naopak hovorí, že objekt je dôležitý a má sa vykresľovať aj keď je vo veľkej vzdialenosti. Toto platí napríklad pre vrcholy hôr. Toto rozhranie implementujú triedy `Poi` a `Compass`.

[6] https://libgdx.badlogicgames.com/ci/nightlies/docs/api/com/badlogic/gdx/graphics/g3d/RenderableProvider.html
[7] https://github.com/libgdx/libgdx/wiki/ModelBatch
[8] https://github.com/libgdx/libgdx/wiki/ModelCache

Rozhranie `ISceneCamera`

Toto rozhranie rozširuje rozhranie `ISceneObject`, keďže kamera v scéne je tiež objekt, ktorý má svoju polohu, ale tiež to nie je objekt, ktorý sa dá vykresliť. Rozhranie obsahuje metódy `getSouthVector()`, `getUpVector()` a `getEastVector()`. Tieto metódy slúžia pre získanie smerových vektorov v danom bode scény. Pri implementácii je využitý spôsob popísaný v kapitole 3.3.4.

Ďalej obsahuje metódu `getCamera()`, ktorá vráti inštanciu triedy `Camera` frameworku LibGDX.

4.3.2 Trieda `SceneManager`

Táto trieda bola implementovaná podľa návrhu v kapitole 3.4.1. Obsahuje metódy pre pridávanie objektov, ktoré implementujú rozhranie `ISceneDrawableObject`. Keďže vykresľovaných objektov môže byť veľa, je dobré využiť triedu `ModelCache`, ktorá slúži na uloženie objektov do pamäti a následne je možné túto pamäť vykresliť, ako jeden celok. Preto sa vždy pri zmene počiatku súradnicového systému inicializuje objekt triedy `ModelCache` všetkými objektami, ktoré sa budú vykresľovať a potom sa tento objekt vykresľuje v metóde `renderScene()` pomocou triedy `ModelBatch`. Objekty, ktoré sa budú vykresľovať určuje trieda `SceneObjectFilter`.

4.3.3 Trieda `SceneObjectFilter`

V tejto triede je implementované triedenie objektov. Najskôr vyberie objekty, ktoré sú v dohľade virtuálnej kamery a nie sú bližšie ako 10 m. Z týchto objektov budú vyberané objekty, tak že na jeden blok pohľadu bude maximálne jeden objekt z objektov, ktoré sa v ňom môžu nachádzať. Tento objekt bude vybratý ak splňuje jedno z nasledujúcich kritérií:

- Ak má nový objekt väčšiu maximálnu vykresľovaciu vzdialenosť (metóda `getMaximumDrawableDistance()`), ako aktuálny objekt v bloku a je bližšie. Ak je ďalej ako aktuálny objekt, tak iba v tom prípade, že uhol pohľadu na nový objekt je väčší ako uhol pohľadu na aktuálny objekt.
- Ak má nový objekt maximálnu vykresľovaciu vzdialenosť rovnú aktuálnemu objektu v bloku a uhol pohľadu na nový objekt je väčší ako uhol pohľadu na aktuálny objekt.
- Ak má nový objekt menšiu maximálnu vykresľovaciu vzdialenosť ako aktuálny objekt v bloku, nachádza sa vo vzdialenosti od kamery menšej, ako je jeho maximálna vykresľovacia vzdialenosť a je bližšie, ako aktuálny objekt v bloku.

4.4 Triedy implementujúce rozhrania scény

Je potrebné vytvoriť triedy, ktoré implementujú metódy rozhraní v balíčku `scene.interfaces`, aby s objektami mohli pracovať triedy `SceneManager` a `SceneObjectFilter`. Triedy, ktoré tieto rozhrania implementujú sú popísané nižšie.

4.4.1 Abstraktná trieda `EarthObject`

Táto trieda implementuje všetky metódy rozhrania `ISceneObject` okrem metódy `update()`, pretože využitie tejto metódy bude závisieť na konkrétnom objekte.

4.4.2 Trieda `EarthCamera`

Trieda `EarthCamera` implementuje rozhranie `ISceneCamera` a rozširuje abstraktnú triedu `EarthObject`. Trieda pri inicializácii vypočíta vertikálny uhol pohľadu kamery spôsobom popísaným v kapitole 2.8.1. Vytvorí inštanciu triedy `PerspectiveCamera`, ktorú bude vracať pomocou metódy `getCamera()` rozhrania `ISceneCamera`. Metódu `update()` rozhrania `ISceneObject` implementuje takým spôsobom, že kamere nastaví pozíciu v scéne na pozíciu určenú pomocou metódy `calculateOriginRelativePosition()` a narotuje ju podľa aktuálneho quaternionu získaného zo senzorov.

4.4.3 Triedy `Compass` a `Poi`

Tieto triedy implementujú rozhranie `ISceneDrawableObject` a rozširujú abstraktnú triedu `EarthObject`

Trieda `Compass` vytvorí objekt triedy `Model`[9] zložený zo štvorca a textúry kompasu. Keďže kompas je umiestnený vždy pod kamerou, je metóda `update()` rozhrania `ISceneObject` implementovaná tak, že model transformuje na pozíciu, kde sa aktuálne nachádza kamera v scéne posunutú o 1,5 metra nižšie.

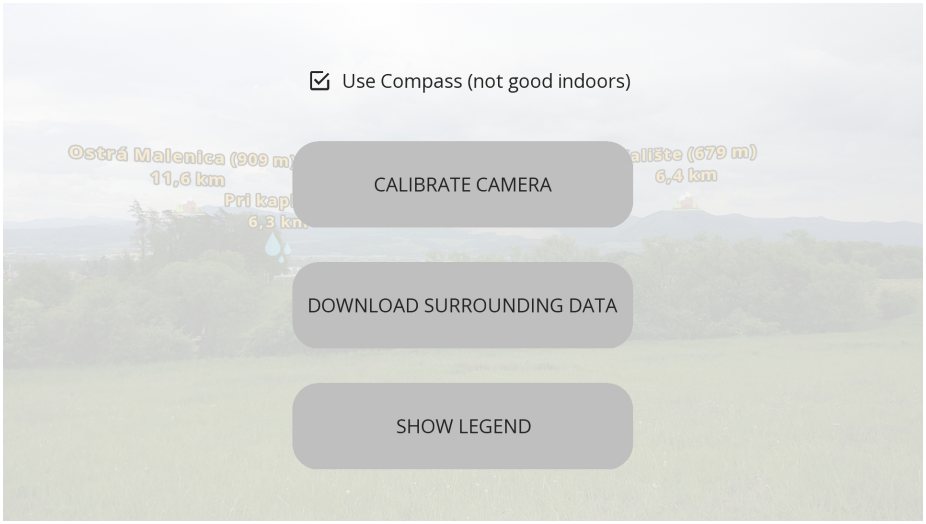
Trieda `Poi` vytvorí model skladajúci sa z troch obdĺžnikov. Prvý obsahuje textúru značky, druhý, ktorý sa nachádza nad prvým obsahuje text so vzdialenosťou objektu od kamery v metroch a tretí, ktorý je najvyššie obsahuje názov objektu a v niektorých prípadoch aj nadmorskú výšku (viac o vzhľade objektov je popísané v kapitole 3.5.1). Textúru značky vyberá podľa informácií získaných o objekte z mapy. Implementácia metódy `update()` rozhrania `ISceneObject` najskôr vytvorí text vzdialenosti od kamery a potom transformuje objekt. Transformuje ho tak, aby bol natočený smerom ku kamere, aby mal veľkosť závislú od vzdialenosti (aby mali pri pohľade z kamery všetky objekty rovnakú veľkosť) a nakoniec ho presunie do polohy relatívnej k počiatku súradnicového systému.

4.5 Výsledky implementácie

Aplikáciu sa podarilo úspešne naimplementovať. Podarilo sa splniť všetky body zadania. Aplikácia zobrazuje rozširujúce informácie o prvkoch v teréne. Jedna z vecí čo ma najviac teší je to, že náhľad z obrazového senzora beží na snímkovej frekvencii 60 snímok za sekundu. Vzhľad výslednej aplikácie je zobrazený na obrázkoch 4.1, 4.2 a 4.3. Video z testovania aplikácie v teréne je dostupné na YouTube[10] a na priloženom CD. Zdrojový kód aplikácie je dostupný v repozitári[11] na GitHube a taktiež na priloženom CD.

[9] https://github.com/libgdx/libgdx/wiki/Models
[10] https://www.youtube.com/watch?v=enuYCX_RXDc
[11] https://github.com/timoti111/VUT-FIT-IBP-AR

Obr. 4.1: Na tomto obrázku je snímka obrazovky priamo z aplikácie zobrazujúca pohľad na hory v diaľke. Vrcholy týchto hôr sú pomocou rozšírenej reality doplnené o informácie o ich názve, nadmorskej výške a vzdialenosti od pozorovateľa.



Obr. 4.2: Snímka obrazovky ukazujúca vzhľad nastavení aplikácie. Zaškrtávacie pole Use Compass slúži na aktiváciu kompasu, ktorý ale nemusí byť presný. Keď je toto pole odškrtnuté, je pre zisťovanie orientácie použitý gyroskop. Vtedy ale treba pomocou tlačidla CALIBRATE CAMERA nastaviť sever manuálne, napríklad podľa nejakého známeho bodu alebo iného kompasu. Tlačidlo DOWNLOAD SURROUNDING DATA, ako už z názvu plynie, stiahne okolité dáta z mapy a následne ich vykreslí. Tlačidlo SHOW LEGEND slúži pre zobrazenie legendy (obrázok 4.3).

Obr. 4.3: Snímka obrazovky zobrazujúca legendu. V tejto legende sa nachádza zoznam značiek, ktoré sú vykresľované na objektoch, ktoré dopĺňajú informáciami. V tomto zozname sa dá posúvať prostom pre zobrazenie ostatných položiek.

Kapitola 5

Záver

Cieľom tejto práce bolo vytvoriť aplikáciu rozšírenej reality na operačný systém Android so zameraním na zobrazovanie prvkov z mapy v teréne.

Hlavným úspechom je dobrá registrácia medzi virtuálnymi a reálnymi objektami iba s použitím pohybových senzorov a modulu GPS. Ďalším dôležitým výsledkom je to, že náhľad z obrazového senzora má snímkovú frekvenciu 60 snímkov za sekundu, a teda náhľad pri pohybe so zariadením ide úplne plynule. Táto plynulosť je viditeľná v priloženom videu. Podarilo sa tiež úspešne implementovať získavanie dát z mapy zameraných na turistiku pomocou Overpass API. Z týchto dát sú potom vytvorené objekty, ktoré sa budú vykresľovať a sú umiestnené do scény podľa geografickej polohy získanej z mapy. Pozícia scéne je určená pomocou geodetického systému WGS84.

Aj keď registrácia medzi virtuálnymi a reálnymi objektami je na dobrej úrovni, určite by sa dala ešte zlepšiť metódami spracovávania obrazu náhľadu z kamery. Pre toto by bolo vhodné použiť napríklad knižnicu ARCore[1], ktorá vyzerá veľmi zaujímavo a v budúcom projekte zameranom na rozšírenú realitu ju určite vyskúšam. Ďalšou vecou, na ktorej by sa dalo zapracovať je zdokonalenie algoritmu pre výber objektov, ktoré sa budú vykresľovať v scéne. Zaujímavou funkcionalitou by tiež určite bola možnosť výberu typov objektov, ktoré by boli pridané do scény, a ktoré nie. Tiež by sa dalo zdokonaliť grafické užívateľské rozhranie napríklad tým, že úkony ako sťahovanie, spracovávanie objektov alebo ich načítavanie by sa odohrávali v inom vlákne. Tým by sa dal oznamovať užívateľovi postup o vykonávaní daného úkonu. Poslednou vecou, ktorá by bola veľmi zaujímavá a určite aj užitočná pre navigáciu v teréne, je vykresľovanie trás turistických chodníkov spolu s ich farebným označením.

Táto práca ma naučila vytvoriť aplikáciu pre operačný systém Android. Už veľmi dávno som sa chcel pustiť do vývoja nejakej aplikácie pre operačný systém Android, ale nemal som žiadny nápad. Keďže zadanie je zamerané na zobrazovanie prvkov z mapy v teréne, musel som aplikáciu testovať takým spôsobom, že som sa išiel prejsť niekde na blízke kopce. Za toto som veľmi vďačný, pretože vždy je dobré si prečistiť hlavu a prísť na nové myšlienky. Tie mi veľakrát pomohli pri návrhu a implementácii.

[1] https://developers.google.com/ar/

Literatúra

[1] About OpenStreetMap. [Online; navštívené 20.04.2019].
URL https://wiki.openstreetmap.org/wiki/About_OpenStreetMap

[2] android.hardware.camera2. [Online; navštívené 20.04.2019].
URL https://developer.android.com/reference/android/hardware/camera2/package-summary.html

[3] android.location. [Online; navštívené 20.04.2019].
URL https://developer.android.com/reference/android/location/package-summary.html

[4] Bimber, O.; Raskar, R.: *Spatial augmented reality: merging real and virtual worlds*. AK Peters/CRC Press, 2005, doi:10.1201/b10624.

[5] Camera API. [Online; navštívené 20.04.2019].
URL https://developer.android.com/guide/topics/media/camera

[6] Change location settings. [Online; navštívené 20.04.2019].
URL
https://developer.android.com/training/location/change-location-settings

[7] Creating a LibGDX project. [Online; navštívené 20.04.2019].
URL https://libgdx.badlogicgames.com/documentation/gettingstarted/Creating%20Projects.html

[8] Elements. [Online; navštívené 20.04.2019].
URL https://wiki.openstreetmap.org/wiki/Elements

[9] Fowler, C. M. R.; Fowler, C. M. R.; Fowler, M.: *The solid earth: an introduction to global geophysics*. Cambridge University Press, 1990.

[10] FusedLocationProviderClient. [Online; navštívené 20.04.2019].
URL https://developers.google.com/android/reference/com/google/android/gms/location/FusedLocationProviderClient

[11] Get the last known location. [Online; navštívené 20.04.2019].
URL https://developer.android.com/training/location/retrieve-current

[12] Goals and Features. [Online; navštívené 20.04.2019].
URL https://libgdx.badlogicgames.com/features.html

[13] Interfacing with platform specific code. [Online; navštívené 20.04.2019].
URL https://github.com/libgdx/libgdx/wiki/Interfacing-with-platform-specific-code

[14] Introduction to Activities. [Online; navštívené 20.04.2019].
URL https://developer.android.com/guide/components/activities/intro-activities

[15] Location. [Online; navštívené 20.04.2019].
URL https://developer.android.com/reference/android/location/Location

[16] Location and context overview. [Online; navštívené 20.04.2019].
URL https://developer.android.com/training/location

[17] McCollough, E.: Photographic topography. *Industry: A Monthly Magazine Devoted to Science, Engineering and Mechanic Arts*, , č. 54-65, 1893.

[18] Meet Android Studio. [Online; navštívené 20.04.2019].
URL https://developer.android.com/studio/intro

[19] Motion sensors. [Online; navštívené 20.04.2019].
URL https://developer.android.com/guide/topics/sensors/sensors_overview

[20] Overpass API. [Online; navštívené 20.04.2019].
URL https://wiki.openstreetmap.org/wiki/Overpass_API

[21] Pagani, A.; Henriques, J.; Stricker, D.: SENSORS FOR LOCATION-BASED AUGMENTED REALITY THE EXAMPLE OF GALILEO AND EGNOS. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, ročník XLI-B1, 06 2016: s. 1173–1177, doi:10.5194/isprs-archives-XLI-B1-1173-2016.

[22] Project setup, running & debugging. [Online; navštívené 20.04.2019].
URL https://github.com/libgdx/libgdx/wiki/Project-setup,-running-&-debugging

[23] Project structure. [Online; navštívené 20.04.2019].
URL https://developer.android.com/studio/intro#project_structure

[24] Receive location updates. [Online; navštívené 20.04.2019].
URL https://developer.android.com/training/location/receive-location-updates

[25] Set Up Google Play Services. [Online; navštívené 20.04.2019].
URL https://developers.google.com/android/guides/setup

[26] Smith, R. W.: *Department of Defense World Geodetic System 1984: its definition and relationships with local geodetic systems*. Defense Mapping Agency, 1987.

[27] SurfaceTexture. [Online; navštívené 20.04.2019].
URL https://developer.android.com/reference/android/graphics/SurfaceTexture

[28] The application framework. [Online; navštívené 20.04.2019].
URL https://github.com/libgdx/libgdx/wiki/The-application-framework

[29] Use the rotation vector sensor. [Online; navštívené 20.04.2019].
URL https://developer.android.com/guide/topics/sensors/sensors_motion#sensors-motion-rotate

[30] You, S.; Neumann, U.: Fusion of vision and gyro tracking for robust augmented reality registration. In *Proceedings IEEE Virtual Reality 2001*, IEEE, 2001, s. 71–78.

Príloha A

Obsah priloženého pamäťového média

ARNature – zdrojový kód aplikácie

TechnicalReport – zdrojový kód tohto dokumentu

ARNature.apk – inštalačný súbor

poster.pdf – plagát

preview.mp4 – video ukazujúce činnosť aplikácie v teréne

README – návod na inštaláciu a použitie aplikácie

xhalas10_report.pdf – tento dokument