Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Rozšírená realita na platforme Android pre identifikáciu horských vrcholov

Bakalárska práca

2017
Veronika Mečiarová

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Rozšírená realita na platforme Android pre identifikáciu horských vrcholov

Bakalárska práca

Študijný program:	Informatika
Študijný odbor:	2508 Informatika
Školiace pracovisko:	Katedra informatiky
Školiteľ:	RNDr. Richard Ostertág, PhD.

Bratislava, 2017
Veronika Mečiarová

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Veronika Mečiarová
Študijný program: informatika (Jednoodborové štúdium, bakalársky I. st., denná forma)
Študijný odbor: informatika
Typ záverečnej práce: bakalárska
Jazyk záverečnej práce: slovenský
Sekundárny jazyk: anglický

Názov: Rozšírená realita na platforme Android pre identifikáciu horských vrcholov
Augmented reality on the Android platform for the identification of mountain peaks

Cieľ: Cieľom práce je vytvoriť aplikáciu pre Android zariadenia, ktorá bude vkladať názvy horských vrcholov do živého obrazu snímaného kamerou zariadenia. Za týmto účelom sa bude využívať technológia rozšírenej reality a senzory dostupné na zariadení (okrem kamery to bude najmä GPS, akcelerometer a magnetometer). Správne umiestnenie popisov do snímaného obrazu bude riadené senzormi zariadenia. Program z digitálneho výškového modelu vytvorí aj reliéf krajiny, ktorý pomôže pri ručnom zarovnaní reálneho obrazu a modelu.

Vedúci: RNDr. Richard Ostertág, PhD.
Katedra: FMFI.KI - Katedra informatiky
Vedúci katedry: prof. RNDr. Martin Škoviera, PhD.

Spôsob sprístupnenia elektronickej verzie práce:
bez obmedzenia

Dátum zadania: 19.10.2016

Dátum schválenia: 24.10.2016 doc. RNDr. Daniel Olejár, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

Poďakovanie: Chcem sa poďakovať predovšetkým môjmu školiteľovi, RNDr.Richardovi Ostertágovi, PhD., za všetky usmernenia, odbornú pomoc, trpezlivosť a podporu počas písania práce.

Abstrakt

Cieľom práce je vytvoriť Android aplikáciu na rozpoznávanie horských vrcholov. Pomocou technológie rozšírenej reality sa budú názvy vrcholov spolu s počítačovým modelom reliéfu zobrazovať do náhľadu kamery, čo užívateľovi pomôže ľahšie rozpoznať daný vrchol. Aplikácia využíva senzory, GPS a databázu na vytvorenie a správne umiestnenie siluety panorámy do obrazu kamery. Správnemu umiestneniu panorámy môže pomôcť užívateľ manuálnym posunom modelu po obrazovke.

Kľúčové slová: rozšírená realita, detekcia hrán, senzory, panoráma, Android

Abstract

The aim of the present paper is to create an Android application to identify mountain peaks. Via augmented reality technology the names of peaks together with the computer relief model will be displayed to the camera preview, helping so the user to distinguish a given peak more easily. The application uses sensors, GPS and a database. It creates a silhouette of panorama and place it correctly to the camera frame. User can place the panorama rightly by moving the model manually on the screen.

Keywords: augmented reality, edge detection, sensors, panorama, Android

Obsah

Zoznam obrázkov

Zoznam ukážok kódu

Úvod

Cieľom tejto práce je vytvoriť aplikáciu pre Android, ktorá pomôže užívateľovi rozpoznať vrcholy pohorí v jeho okolí. Využívame technológiu rozšírenej reality, pomocou ktorej do náhľadu kamery vykreslíme na obrazovku počítačom vytvorenú siluetu okolitej krajiny spolu s popisom vrcholov, ktoré sú na nej zobrazené. Aplikácia bude mať praktické využitie najmä pri turistike na území Slovenska.

V práci budeme používať kameru, dáta získané zo senzorov, údaje o lokalizácií smartfónu a databázu vrcholov a nadmorských výšok.

Bakalárska práca nepriamo nadväzuje na ročníkový projekt, ktorý sme vytvárali v akademickom roku 2015-2016. V ročníkovom projekte sme vytvorili prvotnú verziu tejto aplikácie. Program vytvorený v rámci ročníkového projektu bol jednoduchší. V tejto práci sme vedeli získať dáta zo senzorov, určiť svetovú stranu, GPS súradnice a nadmorskú výšku. Z týchto údajov sme vedeli odhadnúť, kde približne na obrazovke sa nachádzajú vrcholy uložené v internom zozname a na to miesto sme ho po stlačení tlačidla vykreslili. V bakalárskej práci pozmeníme ročníkový projekt a rozšírime ho o nové funkcie.

V súčasnosti existuje viacero podobných aplikácií. Jednou z nich je PeakFinder Earth. Táto aplikácia však nepoužíva technológiu rozšírenej reality a panorámu zobrazuje bez náhľadu kamery. Ďalším typom sú aplikácie ako napríklad aplikácia Nájdi kopec. Táto aplikácia, naopak, využíva technológiu rozšírenej reality, ale nezobrazuje siluetu pohorí a užívateľ si nevie sám zarovnať siluetu na sledovaný reliéf.

Táto práca pozostáva zo štyroch kapitol. V prvej kapitole sa oboznámime s pojmom rozšírená realita a popíšeme možnosť jej implementácie na platforme Android. Druhá kapitola je zameraná na senzory a GPS, ktoré nám poskytujú informácie o lokalizácii a orientácii zariadenia. V tretej kapitole sa budeme venovať vytvoreniu modelu panorámy, anotácií vrcholov na tomto modeli a zarovnávaniu siluety na sledovaný reliéf. Posledná, štvrtá kapitola, opisuje algoritmus detekcie hrán, ktorý sme využili pri zobrazovaní siluety počítačového modelu na obraz kamery.

Kapitola 1

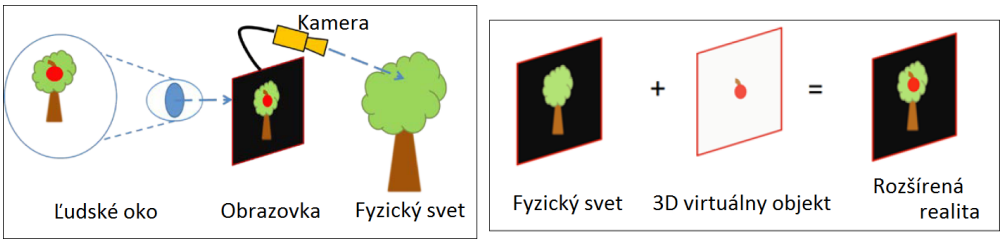
Rozšírená realita

V tejto kapitole si stručne predstavíme fungovanie aplikácie a následne si predstavíme pojem rozšírenej reality, uvedieme rozdiel medzi virtuálnou, skutočnou a rozšírenou realitou. Spomenieme históriu, súčasnosť, ale aj možnú budúcnosť tejto technológie. Vychádzame pri tom z odbornej literatúry zaoberajúcej sa rozšírenou realitou [35, 31]. V závere kapitoly popíšeme implementáciu tejto časti v našej aplikácii, konkrétne vlastnosti kamery a spôsob vkladania grafických virtuálnych objektov.

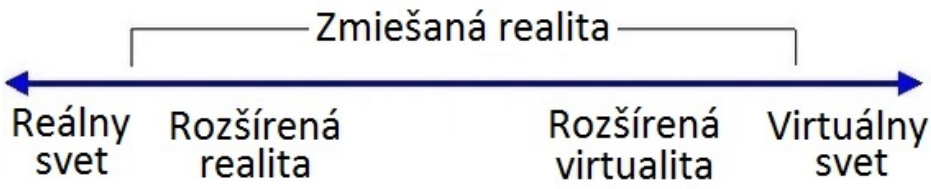
1.1 Chod aplikácie

Aby sme vedeli lepšie pochopiť využitie rozšírenej reality, ako aj iných prvkov, v našej aplikácii, na začiatok si stručne popíšeme fungovanie aplikácie.

Aktuálna verzia aplikácie pozostáva z viacerých krokov. Po spustení aplikácie spustíme náhľad kamery a načítame dáta z databázy vrcholov. Následne zistíme lokalizáciu, prípadne požiadame užívateľa o zapnutie GPS v nastaveniach zaradenia. Po získaní hodnôt z GPS vyžiadame prístup k sieti. Ak máme prístup na internet, pošleme správu obsahujúcu naše aktuálne geografické súradnice na vzdialený server. Server po prijatí správy podľa zadaných hodnôt zistí z databázy našu nadmorskú výšku a vypočíta model panorámy v okolí používateľa. Obrázok panorámy spolu s ďalšími údajmi o okolí pošle späť užívateľovi. Obrázok panorámy následne spracujeme. Vykonáme na ňom detekciu hrán a odstránime čierne pozadie. Do takto vzniknutej siluety vložíme názvy vrcholov z databázy vrcholov a vyobrazíme na obrazovku podľa veľkosti obrazovky, zorných uhlov kamery a otočenia smartfónu. Zobrazenie panorámy prispôsobujeme otočeniu smartfónu v reálnom čase. Užívateľ si môže sám upravovať časť siluety, ktorá má byť zobrazená na obrazovke, pohybom prstov po dotykovej obrazovke zariadenia. Takto môže zväčšovať, zmenšovať a presúvať dvoma smermi siluetu podľa potreby.



Obr. 1.1: Princíp vizuálne rozšírenej reality. [31]



Obr. 1.2: Os virtuálnej súvislosti. [34]

1.2 Pojem rozšírená realita

Rozšírená realita (angl. augmented reality) je technológia, ktorá sledovaný reálny svet obohacuje o pridané digitálne informácie.

R. T. Azuma [27] definuje rozšírenú virtuálnu realitu ako systém s nasledujúcimi tromi vlastnosťami:

1. Kombinácia reality a virtuálnych prvkov
2. Interaktivita v reálnom čase
3. Registrácia v 3D priestore

Na rozdiel od virtuálnej reality, ktorá vytvára celkom umelé prostredie, rozšírená realita umožňuje prostredníctvom kamery a obrazovky vidieť súčasne časť skutočného a časť virtuálneho sveta. Ide teda o vizuálnu kombináciu virtuálnej a skutočnej reality. Digitálne vytvorené prvky, často interaktívne, prekrývajú objekty reálneho sveta zobrazované na obrazovku prostredníctvom kamery v reálnom čase. Obrázok 1.1 schematicky bližšie objasňuje pojem rozšírenej reality. Na osi virtuálnej súvislosti (obrázok 1.2) je rozšírená realita vnímaná ako jeden z medzistupňov medzi realitou a virtuálnym svetom. Spolu s rozšírenou virtualitou patria do zmiešanej reality. Zatiaľ čo rozšírená virtualita je na osi bližšie virtuálnemu svetu, rozšírená realita je bližšia reálnemu svetu [34].

Tieto technológie nám ponúkajú nový spôsob pozerania na svet, ktorý je obohatený o text, obrázky, animácie, 3D modely alebo rôzne efekty. Zjednodušene môžeme rozšírenú realitu chápať ako pozeranie na svet cez sklo, na ktorom je nakreslený objekt.

Pojem rozšírenej reality sa používa aj pri sluchovo vnímanej realite. Tu využívame mikrofón na snímanie reálneho sveta a reproduktor alebo slúchadlá na doplnenie digi-

tálnej informácie. V tejto práci sa však budeme zaujímať výlučne o vizuálne rozšírenú realitu.

1.3 História, súčasnosť, budúcnosť

Prvýkrát aplikoval virtuálnu realitu Ivan Sutherland v roku 1964. Tento vynález mal názov „Damoklov meč“ (angl. „The Sword of Damocles“ [22]). Bola to obrazovka pripevnená na hlave užívateľa, ktorá do reálneho sveta dokresľovala 3D objekty. Toto dielo môžeme označiť ako predchodcu v súčasnosti veľmi úspešných Google Glass.

V priebehu posledných rokov je čoraz väčší počet aplikácií, filmov a rôznych hier využívajúcich technológiu rozšírenej reality. Najznámejšia aplikácia, ktorá využíva túto technológiu je Pokémon GO, ktorá na obrazovke smartfónu vykresľuje virtuálne postavičky do reálneho sveta s možnosťou interakcie.

Technológia rozšírenej reality má veľký potenciál spôsobiť pokroky v zdravotníctve, architektúre, strojníctve i mnohých ďalších oblastiach.

1.4 Implementácia na platforme Android

Implementácia rozšírenej reality pozostáva z dvoch hlavných krokov: zachytenie reálneho sveta prostredníctvom kamery zariadenia a zobrazovanie digitálnych objektov pomocou grafickej knižnice do snímaného obrazu.

V ďalších častiach si popíšeme implementáciu týchto dvoch bodov.

1.4.1 Počítačové videnie

Počítačové videnie [25] (angl. computer vision) je veda, ktorá za sa zaoberá schopnosťou vizuálne vnímať realitu prostredníctvom počítača. Základným prvkom je snímanie okolia pomocou kamery. Jednotlivé zachytené snímky následne môžeme spracovávať alebo upravovať využitím rôznych algoritmov. Medzi najpoužívanejšie algoritmy na prácu s obrazom patrí zmena farieb obrazu, rozpoznávanie tváre, identifikácia objektov, sledovanie pohybujúcich sa objektov, ale aj detekcia hrán, ktorú budeme v inom kontexte využívať.

OpenCV4Android

`OpenCV` [16] (z angl. Open Source Computer Vision Library) je voľne prístupná knižnica, ktorá obsahuje viac než 2500 algoritmov zameraných na počítačové videnie a strojové učenie. Pomocou knižnice `OpenCV` vieme rôzne algoritmy na modifikáciu obrazu vykonávať v reálnom čase, teda priamo počas ich snímania.

```
1 <uses-permission android:name="android.permission.CAMERA"/>
2
3 <uses-feature android:name="android.hardware.camera" android:required=
    "false"/>
4 <uses-feature android:name="android.hardware.camera.autofocus" android
    :required="false"/>
5 <uses-feature android:name="android.hardware.camera.front" android:
    required="false"/>
6 <uses-feature android:name="android.hardware.camera.front.autofocus"
    android:required="false"/>
```

Ukážka kódu 1.1: Povolenie na prístup ku kamere zariadenia [17]

Pre naše účely budeme používať `OpenCV` knižnicu určenú pre platformu Android, s názvom `OpenCV4Andorid`. Z tejto knižnice využijeme metódy na prácu s obrazom, teda časť zaoberajúcu sa počítačovým videním.

Kamera

Kamera je jedným zo základných prvkov aplikácie. Preto je pre kompletný chod aplikácie nevyhnutné, aby zariadenie, na ktorom chceme spustiť aplikáciu, obsahovalo kameru, teda vstupné zariadenie určené na snímanie obrazu.

V prípade, že aplikácia na systéme Android potrebuje pracovať napríklad s hardvérovým zariadením, ukladacím priestorom alebo dátami inej aplikácie, potrebuje, z dôvodu bezpečnosti, vyžiadať od užívateľa prístupové práva. V tomto prípade potrebujeme získať povolenie na prístup ku kamere. Ukážka 1.1 zobrazuje časť súboru $AndroidManifest.xml$, ktorá umožňuje prístup ku kamere zariadenia. Na prvom riadku je žiadosť o povolenie prístupu. Cieľom ostatných štyroch riadkov je upresniť závislosti aplikácie od konkrétnych čŕt zariadenia. Príkaz $android : required = "false"$ vyjadruje nezávislosť aplikácie od tohto zariadenia alebo vlastnosti. Znamená to, že aplikácia túto črtu použije, ak je dostupná, ale nepotrebuje ju nutne k svojmu fungovaniu. Ak by sme použili $android : required = "true"$ znamenalo by to, že aplikácia nie je navrhnutá na fungovanie bez tejto črty. [23]

Po zapnutí aplikácie chceme, aby sa nám na celú obrazovku (angl. fullscreen) zobrazil obraz snímaný kamerou v reálnom čase. Nastavenie módu na celú obrazovku vidíme vo výpise 1.2 zo súboru $AndroidManifest.xml$ v treťom riadku. V 6. riadku tejto ukážky je nastavenie, aby bola aplikácia zobrazovaná na šírku obrazovky (angl. landscape), teda otočená o 90°.

Hlavná trieda aplikácie, *MainActivity*, implementuje rozhranie `CameraBridgeViewBase.CvCameraViewListener2` z `OpenCV` knižnice. Toto rozhranie nám dáva prístup k jednotlivým snímkam kamery. Žiadne snímané údaje nezaznamenávame, snímky len zobrazujeme na obrazovku. Takýto mód kamery nazývame náhľad (angl. preview mode).

```
<application
          ...
    android:theme="@android:style/Theme.NoTitleBar.Fullscreen" >

    <activity android:name="MainActivity"
          android:screenOrientation="landscape"
                ...
           >
          ...
    </activity>
</application>
```

Ukážka kódu 1.2: Nastavenie módu na celú obrazovku a otočenie obrazovky

V nasledujúcej časti popíšeme základné nastavenia parametrov kamery, ktoré sú dôležité pre plynulé fungovanie aplikácie. Na získanie a nastavenie hodnôt budeme používať knižnicu `android.hardware.Camera.Parameters` a na prístup ku kamere zariadenia použijeme triedu z Android knižnice `android.hardware.Camera`.

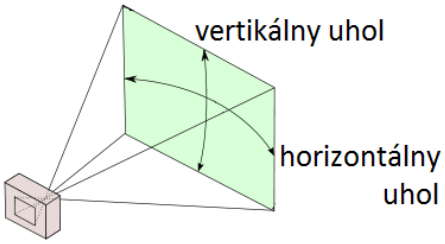
Rozlíšenie kamery

Dôležitým parametrom kamery je jej rozlíšenie. Rozlíšenie kamery je počet pixelov zobrazených na obrazovke. Živý prenos obrazu s vysokým rozlíšením z kamery na displej výrazne zaťažuje CPU. Pritom pre našu aplikáciu nie je nutná úroveň detailov, ktorú poskytujú moderné fotoaparáty. Väčšina mobilných telefónov má 8 až 13 Mpx fotoaparát, no stretávame sa aj s extrémnymi hodnotami až do 41 Mpx [15]. Reálne sa tieto hodnoty využívajú len pri fotografovaní. Preto v našej aplikácií nastavíme menšie rozlíšenie, ktoré pre naše účely postačuje a zároveň minimálne zaťažuje CPU. Vďaka tomu sme schopní obraz živo premietať a ešte doň vkladať ďalšie grafické informácie.

Pre účely našej aplikácie sme potrebovali zvoliť vhodný kompromis medzi kvalitou obrazu a rýchlosťou. Zdrojový kód 1.3 ukazuje, ako sme implementovali nastavenie rozlíšenia kamery. Android trieda `Camera.Parameters` [4] nám ponúka funkciu `getSupportedPreviewSizes`, ktorá nám vytvorí zoznam podporovaných rozlíšení daného zariadenia. Skúšali sme možné rozlíšenia v náhľadovom móde pre smartfón Samsung Galaxy S5 Neo. Pre tento model boli podporované nasledovné rozmery: 1920x1080, 1440x1080, 1072x1072, 1280x720, 1056x704, 1024x768, 960x720, 720x720, 720x480, 640x480, 352x288, 320x240 a 176x144. Z týchto hodnôt sme pre našu aplikáciu vybrali rozlíšenie odporúčané v literatúre zaoberajúcej sa vývojom rozšírenej reality [31] v druhej kapitole. Odporúčané je rozlíšenie kamery 640x480 pixelov, známe aj ako VGA (Video Graphics Array) rozlíšenie. Táto hodnota sa použije len v prípade, že ho dané zariadenie podporuje, v opačnom prípade zvolíme čo najpodobnejšie rozlíšenie.

```
1  private void setCameraResolution() {
2      int recommendedWidth = 640;
3      int recommendedHeight = 480;
4
5      Camera.Parameters parameters = camera.getParameters();
6      List<Camera.Size> cameraSupportedSizes = parameters.
           getSupportedPreviewSizes();
7      boolean foundRecommendedWidth = false;
8      int minDiff = Integer.MAX_VALUE;
9      Camera.Size minDiffSize = null;
10
11     for (Camera.Size size : cameraSupportedSizes) {
12         if (size.width == recommendedWidth && size.height ==
               recommendedHeight) {
13             foundRecommendedWidth = true;
14             break;
15         }
16         int diff = Math.abs(size.width-recommendedWidth);
17         if (diff < minDiff) {
18             minDiff = diff;
19             minDiffSize = size;
20         }
21     }
22
23     if (foundRecommendedWidth) {
24         parameters.setPreviewSize(recommendedWidth,recommendedHeight);
25         camera.setParameters(parameters);
26     } else {
27         if (minDiffSize != null) {
28             parameters.setPreviewSize(minDiffSize.width, minDiffSize.
                   height);
29             camera.setParameters(parameters);
30         }
31     }
32 }
```

Ukážka kódu 1.3: Nastavenie rozlíšenia kamery [31]



Obr. 1.3: Horizontálny a vertikálny zorný uhol. [1]

Snímková frekvencia

Ďalším dôležitým parametrom je frekvencia snímkovania (angl. frame rate), ktorá udáva počet snímok za sekundu. Jednotka je fps (z angl. frame per second). Postup je analogický ako pri nastavovaní rozlíšenia, len s tým rozdielom, že z triedy `Camera.Parameters` sme tento raz použili funkciu `getSupportedPreviewFpsRange` na získanie rôznych typov nastavenia. Pre každé nastavenie je daná minimálna a maximálna frekvencia. Opäť sme skúšali dostupné hodnoty. Podporované nastavenia rozsahov snímkových frekvencií pre Samsung Galaxy S5 Neo sú buď stabilné hodnoty 15fps, 24fps, 30fps alebo môžeme zvoliť rozsah 15fps-30fps. Podobne ako v prípade rozlíšenia, ak sú dostupné, použijeme hodnoty, ktoré odporúča spomínaná literatúra. V tomto prípade je najvhodnejšia 3. možnosť nastavenia, teda práve stabilná hodnota 30fps. Ak nie sú dostupné, použijeme dostupnú hodnotu čo najbližšiu k odporúčanej hodnote.

Zorný uhol

Okrem nastavenia predchádzajúcich parametrov potrebujeme zistiť horizontálny a vertikálny zorný uhol kamery, viď obrázok 1.3. Zistenie veľkosti týchto uhlov je pre nás dôležité, aby sme vedeli čo najpresnejšie odhadnúť, ktorú časť pohoria vidí používateľ cez náhľad kamery. Tieto hodnoty závisia od hardvéru konkrétneho zariadenia. Prístup k týmto hodnotám máme v triede `android.hardware.Camera.Parameters` [4] pomocou metód `getHorizontalViewAngle` a `getVerticalViewAngle`. Uhly sú dané v stupňoch. Pre testované zariadenie Samsung Galaxy S5 Neo je horizontálny zorný uhol 62,2° a 39,4° vertikálny.

1.4.2 Vkladanie digitálnych objektov

Digitálny objekt, ktorý budeme v našej aplikácii vkladať do zosnímaného obrazu, je silueta panorámy spolu s pomenovaním zobrazených vrcholov. Postupom vytvárania obrázku siluety spolu s umiestňovaním názvov vrcholov na tejto siluete sa budeme venovať v ďalších kapitolách. V tejto časti sa budeme zaoberať len zobrazením pripravenej

siluety vo forme bitmapového obrázku na obrazovke.

Hlavnou časťou aplikácie je trieda *MainActivity*, ktorá je potomkom triedy `android.app.Activity` [30]. Dedenie od tejto triedy je nevyhnutné v každej Android aplikácii, pretože zabezpečuje spustenie programu. Všetky objekty zobrazované na obrazovke sú typu `View`. V metóde `onCreate` triedy *MainActivity* je potrebné, aby bol pomocou funkcie `setContentView` špecifikovaný objekt typu `View` alebo identifikátor XML objektu, ktorý popisuje rozloženie (angl. layout) objektov, ktoré chceme zobraziť po spustení. Trieda `ViewGroup` je potomkom triedy `View`, a na rozdiel od `View`, `ViewGroup` môže obsahovať viacero objektov typu `View`, a teda aj jej potomkov typu `ViewGroup`. [24]

V našej aplikácii potrebujeme v okne hlavnej aktivity zobraziť viac objektov, tak použijeme inštanciu triedy `ViewGroup`. Tieto triedy majú zvyčajne príponu „Layout“. Rozhodli sme sa použiť `FrameLayout`, aj keď pôvodne sme plánovali aplikovať `LinearLayout`. Hlavný rozdiel je v tom, že `LinearLayout` sa používa najmä na zarovnanie objektov vedľa seba, zatiaľ čo `FrameLayout` umožňuje umiestňovať objekty cez seba, čím poskytuje možnosť prekrývania objektov.

Na vkladanie digitálnych objektov do obrazu kamery sme vytvorili triedu *DrawView*, ktorá je podtriedou Android triedy `android.view.SurfaceView` [8], tá je potomkom `View`. Pridáme ju metódou `addView` do `FrameLayout`, čím dosiahneme vykreslenie tohto objektu na popredí.

Táto trieda poskytuje možnosť zobrazenia priehľadného kresliaceho povrchu. Prostredníctvom objektu typu `android.graphics.Canvas` poskytuje grafickú plochu, na ktorú môžeme kresliť tvary, umiestňovať text, obrázky a podobne. Týmto spôsobom sme vložili siluetu panorámy typu `BitMap` a anotáciu vrcholov. Mená vrcholov sme vložili ako text a bod označujúci vrchol pomocou geometrického tvaru kruhu. Tento objekt následne vykreslíme na obrazovku prostredníctvom triedy *DrawView*.

Kapitola 2

Orientácia a lokalizácia

V tejto kapitole sa budeme venovať senzorom v smartfónoch. Naša aplikácia potrebuje údaje zo senzorov pre výpočet orientácie a lokalizácie.

Vychádzame z odbornej literatúry [33, 35, 31] a Android manuálov [20].

2.1 Senzory v mobilných zariadeniach Android

Operačný systém Android rozdeľuje senzory do troch kategórií:

- Pohybové senzory – Senzory pohybu merajú zrýchlenie a rotáciu okolo troch osí. Do tejto kategórie patria senzory na meranie zrýchlenia, gravitačné senzory, gyroskopy a rotačné vektorové senzory.
- Senzory prostredia – Tieto senzory merajú rôzne parametre prostredia, ako je teplota a tlak vzduchu, osvetlenie a vlhkosť. Do tejto kategórie patria napríklad barometre, fotometre a teplomery. Túto kategóriu senzorov v našej aplikácií nevyužijeme.
- Polohové senzory – Senzory z tejto kategórie merajú fyzickú pozíciu zariadenia. Do tejto kategórie patria orientačné senzory a magnetometer.

Aby sme mohli zobraziť pridané informácie do obrazu z kamery na vhodné miesto, potrebujeme lokalizovať smartfón v trojrozmernej súradnicovej sústave. Chceme teda zjednotiť súradnicové systémy reálneho sveta a počítačového 3D modelu reálneho sveta. Jednoznačnú polohu bodu v tomto súradnicovom systéme určujú GPS súradnice a nadmorská výška.

Okrem lokalizácie potrebujeme poznať aj orientáciu, teda smer, kam sa pozeráme. Orientáciu určíme pomocou troch osí zariadenia. Zo senzorov budeme potrebovať akcelerometer, magnetometer a lokalizačný senzor GPS. Väčšina súčasných inteligentných zariadení má zabudovaný potrebný hardvér pre tieto senzory.

Pomocou získaných informácií zo senzorov budeme následne vedieť určiť, ktoré vrcholy je možné vidieť na obrazovke a ich približné umiestnenie.

2.2 GPS a nadmorská výška

Pre účely našej aplikácie budeme potrebovať jednoznačne určiť pozíciu zariadenia v reálnom svete. Na získanie potrebných dát potrebujeme niektorý zo satelitných navigačných systémov (GNSS – Global Navigation Satelite System). Verzií GNSS je viacero, napríklad ruská verzia GLONASS, európska verzia Galileo a v súčasnosti najrozšírenejšia verzia americkej armády – GPS. V našej aplikácií budeme používať polohový senzor GPS (angl. Global Positioning System), pretože v súčasnosti je práve GPS najpodporovanejší GNSS na mobilných zariadeniach.

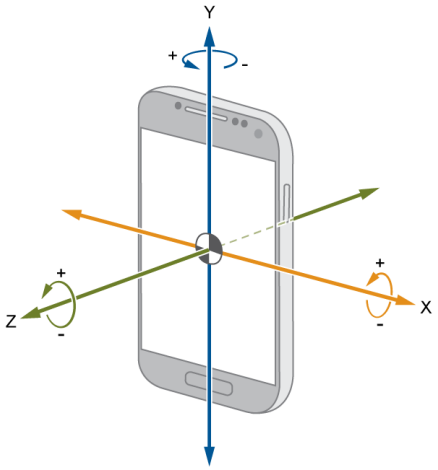
GPS je globálny lokalizačný systém, ktorý umožňuje pomocou satelitov obiehajúcich planétu Zem takmer presne identifikovať aktuálne umiestnenie smartfónu na Zemi pomocou dvoch geografických súradníc a nadmorskej výšky. Chyba merania GPS sa pohybuje približne od 10 do 15 metrov, čo je pre nás takmer zanedbateľná odchýlka. GPS nie je spoľahlivý najmä v blízkosti kovových predmetov, v budove alebo na miestach, kde je zakrytý výhľad na oblohu. Problém môže nastať aj v hustom lese alebo v úzkych vysokohorských priesmykoch, tu však nepredpokladáme využívanie aplikácie. Predpokladáme, že primárne bude aplikácia používaná mimo uzavretých priestorov na miestach s výhľadom na okolitú krajinu.

2.2.1 Implementácia

Pri implementácií tejto časti budeme používať triedy `Location`, `LocationManager` a `LocationListener`, ktoré nájdeme v Android knižnici `android.location`. Táto knižnica nám poskytuje prístup k hodnotám z lokalizačných senzorov zariadenia. Pomocou funkcií `getLatitude`, `getLongtitude` a `getAltitude` z triedy `Location` [5] vieme určiť zemepisnú šírku, zemepisnú dĺžku a nadmorskú výšku zariadenia. Geografické súradnice, zemepisnú dĺžku a šírku, nám funkcie vrátia v stupňoch a nadmorskú výšku v metroch.

2.3 Naklonenie smartfónu a kompas

Pre určenie vrcholov videných prostredníctvom kamery nám nestačí vedieť zemepisné súradnice a nadmorskú výšku. Potrebujeme tiež zistiť, ktorú svetovú stranu na obrazovke sledujeme a aký má smartfón sklon. Na zistenie týchto údajov využijeme magnetometer a akcelerometer. V tejto časti budeme používať Android knižnicu `an-`



Obr. 2.1: Osi x, y a z na mobilom zariadení. [13]

`droid.hardware` a jej triedy `Sensor`, `SensorManager`, `SensorEvent` a `SensorEventListener` [7].

2.3.1 Magnetometer

Magnetometer je hardvérový polohový senzor merajúci v μT okolité geomagnetické pole v smere osí x, y a z. Za ideálnych podmienok vieme pomocou hodnôt z magnetometra určiť svetové strany. Tento senzor však často rušia trvalo zmagnetizované predmety z magenticky tvrdých materiálov (angl. hard iron), ale aj predmety z magneticky mäkkých materiálov (angl. soft iron). Často je veľkosť magnetického poľa Zeme niekoľkonásobne menšia ako veľkosť rušivých vplyvov spôsobených zmagnetizovanými predmetmi. Z tohto dôvodu nedokážeme pomocou magnetometra zistiť správnu informáciu o orientácii zariadenia [32]. Preto je nevyhnutné použiť ďalší senzor - akcelerometer.

2.3.2 Akcelerometer

Akcelerometer je hardvérový pohybový senzor, ktorý meria zrýchlenie v m/s^2 vzhľadom na osi x, y a z (viď obrázok 2.1). Pri meraní berie do úvahy gravitačnú silu. Pomocou samotného akcelerometra nevieme určiť natočenie smartfónu. Namerané hodnoty používame na spresnenie dát z magnetometra.

Týmto senzorom sledujeme chvenie a naklonenie zariadenia.

2.3.3 Implementácia

V čase, keď senzormi zaznamenáme pohyb zariadenia, spustí sa nami vytvorená metóda *orientation* (kód 2.1), ktorá z hodnôt získaných zo senzorov vytvorí maticu rotácie pomocou funkcie `getRotationMatrix` z Android knižnice. Táto funkcia pomocou

```
1  public void orientation () {
2      float R[] = new float[9];
3      boolean haveRotation = sensorManager.getRotationMatrix(R, null,
           mAccelerometer, mGeomagnetic);
4
5      if (haveRotation) {
6          float orientation[] = new float[3];
7          sensorManager.getOrientation(R, orientation);
8
9          int z = (int) (rad2deg(orientation[0]));
10         int x = (int) (rad2deg(orientation[1]));
11         int y = (int) (rad2deg(orientation[2]));
12     }
13 }
```

Ukážka kódu 2.1: Funkcia orientation

hodnôt nameraných akcelerometrom a magnetometrom vypočíta vektor, ktorý označíme R. Vstupnými hodnotami pre túto funkciu sú prázdna matica R, ktorá je zároveň výstupnou maticou, prázdna hodnota pre maticu sklonu, ktorú nebudeme potrebovať, a dva vektory typu float dĺžky 3 - *mAccelerometer* a *mGeomagnetic*, ktoré obsahujú dáta zaznamenané senzormi akcelerometrom a magnetometrom. Výstupná matica R predstavuje otočenie zariadenia. R je identita práve vtedy, ak pre osi zariadenia platí: os X smeruje na východ, os Y na sever a obrazovka je otočená smerom hore, teda zariadenie držíme vodorovne. Funkcia `getRotationMatrix` vráti hodnotu true v prípade úspešného výpočtu. Hodnotu false v prípade zlyhania, čo nastáva napríklad pri voľnom páde zariadenia, v tomto prípade sa matica R ani matica sklonu nemodifikujú.

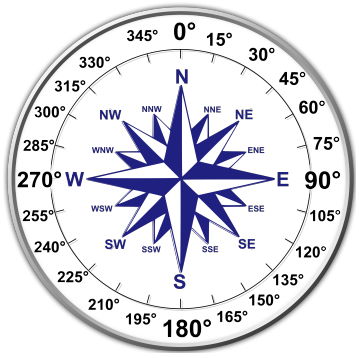
Následne sa vykoná Android funkcia `getOrientation`, ktorá pomocou matice R vypočíta vektor orientácie, ktorý je v kóde pomenovaný *orientation*. Výsledný vektor obsahuje tri uhly otočenia zariadenia okolo jednotlivých osí v radiánoch.

Význam hodnôt v matici *orientation* je nasledovný:

1. *orientation*[0] – Azimut, uhol natočenia okolo osi Z. Hodnota predstavuje uhol medzi osou Y zariadenia a severom. Hodnoty sú z intervalu [-π, π]. Ak os Y smeruje na sever, hodnota je 0, ak smeruje na juh, uhol je π. Pri otočení na východ je uhol π/2 a na západ -π/2.

2. *orientation*[1] – Uhol natočenia okolo osi X. Hodnota predstavuje uhol medzi rovinou rovnobežnou s obrazovkou zariadenia a rovinou rovnobežnou so zemou. Rozsah hodnôt je [-π, π]. Ak vrch zariadenia ukazuje smerom hore, hodnota je -π a π, ak vrch zariadenia smeruje k zemi.

3. *orientation*[2] – Uhol natočenia okolo osi Y. Táto hodnota predstavuje uhol medzi rovinou kolmou na obrazovku zariadenia a rovinou kolmou na zem. Hodnoty sú z intervalu [-π/2, π/2]. Hodnota je 0 v prípade, že je zariadenie položené



Obr. 2.2: Určenie svetovej strany podľa hodnoty danej v stupňoch. [9]

vodorovne, π nastane, ak obrazovka je otočená vpravo a -π, ak je obrazovka smerom doľava.

Pre jednoduchšiu manipuláciu prepočítame radiány na stupne pomocou funkcie *rad2deg* jednoduchou rovnicou:

Nech s je hodnota uhla v stupňoch a r je hodnota uhla v radiánoch, potom platí:

$$s = r \cdot \frac{180}{\pi} \tag{2.1}$$

Z hodnoty z získanej z matice $orientation[0]$ vieme jednoducho určiť svetové strany, a tak si vytvoriť kompas podľa obrázku 2.2. Stačí, ak v prípade zápornej hodnoty z k nej pripočítame 360°.

Zistená svetová strana je však v smere osi Y, teda ukazuje naň vrch zariadenia. Naša aplikácia je po celý čas zobrazovaná na šírku, preto, ak chceme získať hodnotu pre smer, ktorý takto snímame kamerou, potrebujeme od nameranej hodnoty odčítať 90° a takto určiť svetovú stranu.

Kapitola 3

Vytvorenie panorámy

Cieľom tejto časti našej práce je vytvorenie panorámy. Panorámu vieme vytvoriť z databázy nadmorských výšok vzhľadom na aktuálnu pozíciu smartfónu. V tejto kapitole popíšeme spôsob a implementáciu vytvorenia panorámy okolia ako počítačového modelu a popis databáz použitých na jeho vytvorenie. Vychádzať budeme z diplomovej práce [36].

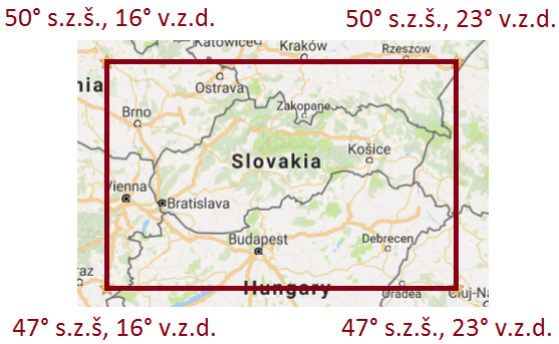
Postup tvorby siluety panorámy pozostáva z viacerých krokov. Prvým krokom je získanie obrázku 360° panorámy podľa GPS súradníc a nadmorskej výšky. Následne algoritmom na detekciu hrán získame čierno-biely obrázok siluety panorámy. Zmenou čiernej farby na priehľadnú získame potrebný výsledný obrázok. Posledným krokom je pomenovanie vrcholov na panoráme a zobrazenie správnej časti siluety na obraz kamery. Podrobnejšie si o každom tomto kroku povieme nižšie.

3.1 Databáza nadmorských výšok

Prvou databázou, ktorú použijeme, je databáza nadmorských výšok vzhľadom na zemepisné súradnice. Táto databáza bola vytvorená spoločnosťou NASA počas výskumného programu SRTM (z angl. Shuttle Radar Topography Mission) [21] a celá táto databáza je voľne dostupná [3].

Dáta sú uložené v mnohých súboroch, pričom každý súbor obsahuje údaje pre územie v rozlohe 1° zemepisnej šírky a 1° zemepisnej dĺžky. Tieto údaje vytvárajú digitálne výškové modely (angl. digital elevation model, DEM). Digitálny výškový model je mriežka, ktorá v každom políčku obsahuje nadmorskú výšku na danej zemepisnej súradnici.

Súbory sú pomenované podľa juhozápadného kraja. Napríklad, súbor s názvom N47E016 obsahuje informácie o nadmorskej výške v území od 47° severnej zemepisnej šírky a 16° východnej zemepisnej dĺžky po 48° severnej zemepisnej šírky a 17° východnej zemepisnej dĺžky.



Obr. 3.1: Rozsah databázy, ktorá pokrýva celé územie Slovenska označený červenou farbou. Hodnoty na okrajoch sú geografické súradnice vo vrcholoch vyznačeného obdĺžnika.

Dostupné rozlíšenie údajov pre naše územie sú 3 uhlové sekundy, čo je na území Slovenska približne 90m. Pre územie Ameriky sú dostupné aj údaje s rozlíšením 1 uhlová sekunda. Premena jednotiek z uhlových sekúnd na metre nie je lineárna. Jedna uhlová sekunda má na rôznych miestach sveta rôznu vzdialenosť v metroch.

Počet údajov v jednom dátovom súbore je 2001x2001 a dáta sú typu float. V pravom dolnom políčku je nadmorská výška na zemepisnej súradnici, ktorú poznáme z názvu súboru. O políčko vyššie je nadmorská výška na mieste s tou istou zemepisnou dĺžkou, ale o jednu uhlovú sekundu smerom na sever, teda približne o 90m severnejšie.

Aby sme vedeli, ktoré súbory z databázy potrebujeme použiť, potrebujeme vedieť geografické súradnice okrajových bodov Slovenska [12]. Z týchto hodnôt sme zistili, že z celosvetovej databázy potrebujeme použiť súbory N47E016 až N49E022, čo je najmenšie možné pokrytie územia Slovenska. Rozsah týchto súborov vidíme na mape zobrazenej na obrázku 3.1. Keďže však chceme, aby aplikácia fungovala aj v krajných oblastiach Slovenska, je potrebné pridať aj ďalšie súbory s dátami za hranicami. Na zabezpečenie funkčnosti aplikácie na celom území Slovenska teda potrebujeme súbory N46E015 až N50E023, čo je 123,8 MB dát v 45 súboroch.

3.2 Databáza vrcholov

Ďalšou použitou databázou je databáza vrcholov. Pre vytvorenie tejto databázy použijeme voľne dostupnú databázu na mapovom portáli Freemap Slovakia [10], ktorý patrí do komunity OpenStreetMap. OpenStreetMap [18] je projekt pre tvorbu voľne šíriteľných geografických databáz pre tvorbu máp celého sveta.

Táto databáza obsahuje informácie o všetkých geografických objektoch na území Slovenskej republiky. Okrem iných informácií obsahuje názvy vrcholov, ich zemepisné súradnice a nadmorskú výšku. My budeme v našej práci potrebovať práve tieto údaje.

Z tohto portálu sme získali dáta pre celé slovenské územie vo formáte OSM XML

```
1        <node id="242810882" lat="49.2118018" lon="19.093979">
2                <tag k="ele" v="1607"/>
3                <tag k="name" v="Stoh"/>
4                <tag k="natural" v="peak"/>
5        </node>
```

Ukážka kódu 3.1: Pôvodná syntax databázy

```
1  <peaks_list>
2          ...
3          <peak>
4                  <id>242810882</id>
5                  <lat>49.2118018</lat>
6                  <lon>19.093979</lon>
7                  <alt>1607</alt>
8                  <name>Stoh</name>
9          </peak>
10         ...
11 </peaks_list>
```

Ukážka kódu 3.2: Údaj o vrchole v našej syntaxi

(z angl. OpenStreetMap XML) [19], ktorý je pre človeka ľahko čitateľný a prehľadný. Po dekomprimovaní je síce súbor veľmi veľký, no my sme jednoduchým skriptom vybrali z celej databázy len vrcholy s ich údajmi. Takto sme zmenšili objem dát z 3,85GB na 920kB.

Vo vstupnom XML súbore nás zaujímajú len XML elementy s názvom *node*, v ktorých je element *tag* s atribútom *k* (z angl. key) "natural" a atribútom *v* (z angl. value) "peak". V ukážke 3.1 vidíme, ako vyzerá údaj o vrchole v pôvodnej databáze už len s vybranými potrebnými údajmi. Pre jednoduchšiu manipuláciu so súborom sme si upravili formát, ako vidíme v ukážke 3.2.

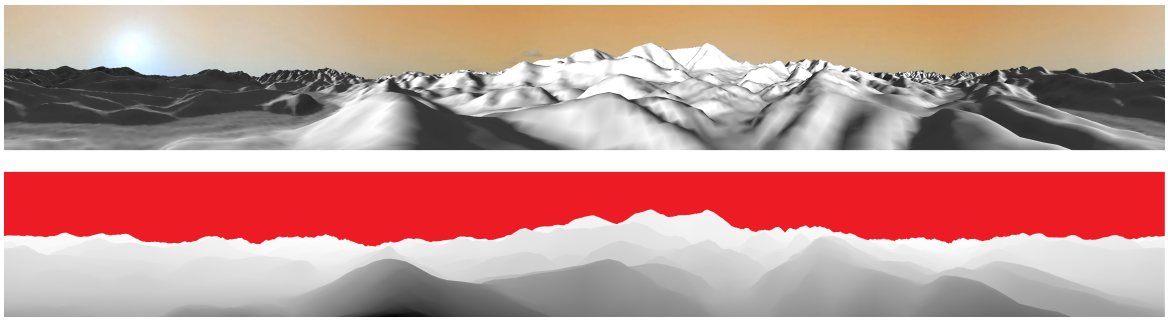
3.3 Model panorámy

Na vytvorenie počítačového dvojrozmerného modelu panorámy používame algoritmus ComoFlyer vytvorený v diplomovej práci [36]. Tento algoritmus používa knižnicu `jMonkeyEngine` [14], ktorá je určená na tvorbu profesionálnych 3D hier v jazyku Java. Knižnica je podporovaná aj pre platformu Android, ale nie všetky jej časti.

3.3.1 Algoritmus ComoFlyer

Program ComoFlyer má k dispozícii databázu nadmorských výšok. Zo vstupných hodnôt, ktorými sú geografické súradnice pozorovateľa, vyberie súbory databázy, ktoré bude potrebovať, a uloží si ich do pamäte.

Pomocou `jMonkeyEngine` knižnice vytvorí trojrozmerný model krajiny reálneho



Obr. 3.2: Panoráma z 3D modelu (hore) a vzdialenostná panoráma (dole).

sveta z dát digitálneho výškového modelu a zo zadaných súradníc. Najskôr je kamera otočená na sever s posunom o 22,5° v smere hodinových ručičiek a vytvorí akoby fotografiu modelu krajiny v zornom uhle 45°. Následne sa osemkrát otočí v smere hodinových ručičiek, aby takto postupne pospájal 360° model krajiny. Z tohto modelu vytvorí obrázok 360° panorámy v zadanej nadmorskej výške. Príklad takejto dvojrozmernej panorámy je na obrázku 3.2 hore.

Ďalším produktom je vzdialenostná panoráma. Je to obrázok panorámy v odtieňoch sivej, kde jas každého bodu určuje, v akej vzdialenosti od nás je tento bod v reálnom svete. Príklad vzdialenostnej panorámy je na obrázku 3.2 dole.

Posledným výsledným produktom tejto práce je dvojrozmerné pole typu float, ktoré je ukladané do tabuľky v csv formáte, ktorá, podobne ako vo vzdialenostnej panoráme, popisuje našu vzdialenosť od bodu zobrazeného daným pixelom v reálnom svete. V tejto tabuľke sú vzdialenosti uvedené v metroch. Rozmer matice je 6400x800, čo je zároveň aj počet pixelov panorámy.

V našej práci sme použili len časť tejto diplomovej práce. Prístup k nej máme prostredníctvom triedy *ServerComunication* zdedenej z `android.os.AsyncTask` vykonaním *execute*. Funkcia sa vykonáva len raz po spustení aplikácie, keď zariadenie zistí aktuálne GPS súradnice a nadmorskú výšku. Ďalšie spustenie je len v prípade vyžiadania užívateľom prostredníctvom menu, pri zmene GPS o viac než 0,0267° zemepisnej šírky, 0,0411° zemepisnej dĺžky, čo je v oboch prípadoch približne tri kilometre alebo pri zmene nadmorskej výšky o viac než 100 metrov. Súradnice získané zo senzorov sú vstupné hodnoty funkcie. Výstupom funkcie je vzdialenostná panoráma ako objekt typu `opencv.core.Mat` a tabuľka vzdialeností typu float[][], ktoré budeme neskôr používať.

Sú dva spôsoby získavania aktuálnej nadmorskej výšky zariadenia. Nadmorskú výšku môžeme získať zo senzorov, ako bolo spomínané v 2. kapitole alebo ju môžeme zistiť z databázy nadmorských výšok podľa geografických súradníc. V použitej diplomovej práci autor pracoval s nadmorskou výškou získanou z databázy. My sme v našej práci skúšali algoritmus pozmeniť tak, aby sme použili hodnoty získané zo senzorov, čím sme očakávali dosiahnutie väčšej presnosti. Zistili sme, že získaná hodnota nad-

morskej výšky z lokalizačného senzora na jednom bode nie je stabilná, čo spôsobuje nepresnosti. Preto sme sa rozhodli pri vytváraní počítačového modelu použiť nadmorskú výšku z databázy nadmorských výšok, a teda predpokladať, že užívateľ stojí na zemi.

Algoritmus ComoFlyer si takto získanú nadmorskú výšku dodatočne upravuje. Dôvodom potreby tejto úpravy je slabé rozlíšenie databázy nadmorských výšok. Presnosť je na území Slovenska približne 90 metrov. Tu nastáva problém. Predstavme si pozorovateľa, ktorý stojí napríklad na vrchole kopca a pohne sa 10 metrov dozadu. Predtým sledoval výhľad na okolité kopce, no teraz ich už nevidí, pretože sledovanú panorámu mu zakryje pohorie, na ktorom sa nachádza. Obrázok 3.3 zobrazuje riešenie tohto problému. Pozorovateľa potrebujeme umiestniť vyššie, nad úroveň pohoria.

3.3.2 Sieťová komunikácia

Ako sme už spomenuli vyššie, nie všetky použité knižnice sú priamo kompatibilné s Androidom. Z tohto dôvodu sme sa rozhodli, že táto časť bude implementovaná na operačnom systéme Windows a cez sieť bude komunikovať s našou aplikáciou na systéme Android.

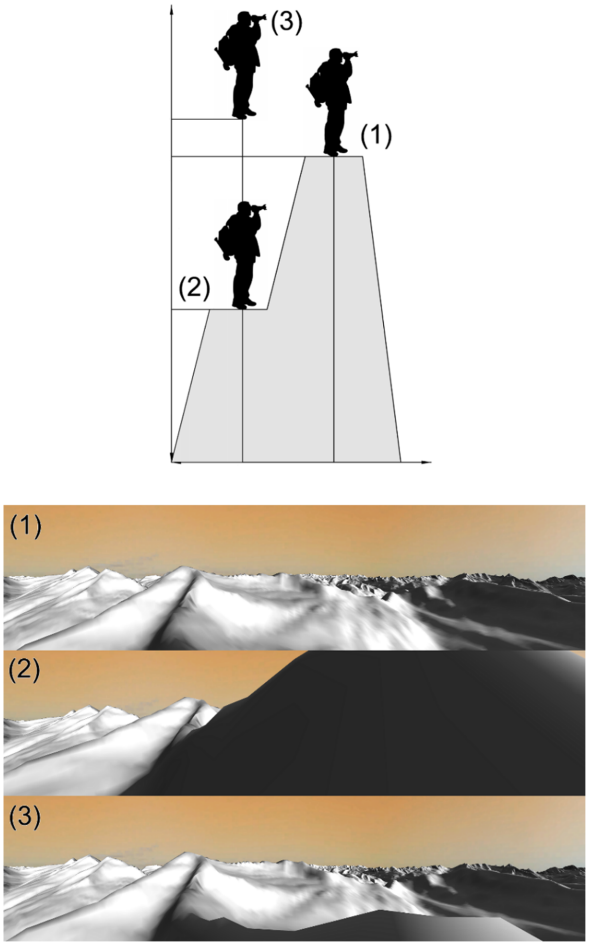
Po získaní GPS súradníc komunikujeme so serverom prostredníctvom internetu. Pošleme, ako klient, správu obsahujúcu GPS súradnice. Na serveri sa spustí služba, ktorá po každej prijatej správe od klienta spustí algoritmus ComoFlyer, ktorý ako odpoveď vráti obrázok panorámy a pole vzdialeností.

Na vytvorenie spojenia používame knižnice `java.net.Socket` a `java.net.ServerSocket`. Pri prenose údajov treba vyriešiť niekoľko problémov s typom dát.

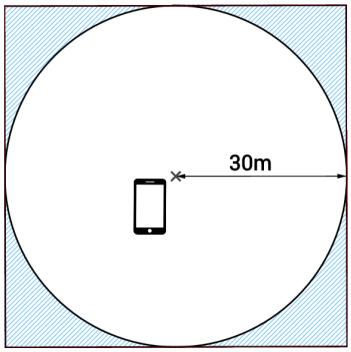
Prvým problémom je typ prenášaných dát. Klient posiela na server len text, čo je v poriadku, ale dáta posielané zo servera sú dvojrozmerné pole typu float a obrázok. Tieto dáta je potrebné posielať ako jednorozmerné pole typu byte.

Ďalším problémom je nekompatibilita typu obrázku v kóde pre Android a pre Windows. Server na systéme Windows posiela obrázok typu `java.awt.image.BufferedImage`, zatiaľ čo klient ho potrebuje prijať ako typ `opencv.core.Mat` alebo `android.graphics.Bitmap`. Medzi typmi `opencv.core.Mat` a `android.graphics.Bitmap` existuje metóda na ich konverziu `Utils.matToBitmap`, resp. `Utils.bitmapToMat`. Keďže bezprostredne po prijatí obrázka chceme na ňom vykonať detekciu hrán a tú vykonávame na objektoch typu `opencv.core.Mat`, prijaté pole bytov konvertujeme práve na tento typ funkciou `put` z triedy `opencv.core.Mat`.

Posielanie dát cez sieť aplikáciu nespomaľuje, konverzia jednotlivých typov po prijatí však nie je rýchla. Táto časť spomaľuje vytváranie panorámy. Keďže aplikácia je v jazyku Java, ak by sme chceli prijatú maticu konvertovať na dvojrozmernú, potre-



Obr. 3.3: Dôvod korekcie nadmorskej výšky použitej v algoritme ComoFlyer. Zdroj obrázka [36].
(1) Pohľad z vrcholu kopca. Prípad málokedy zodpovedajúci reálnym podmienkam. Tento prístup by fungoval, ak by bola databáza nadmorských výšok s veľmi vysokým rozlíšením.
(2) Pohľad, v ktorom pozorovateľovi zakrýva výhľad pohorie, na ktorom sa nachádza, teda ak by sme nepoužili korekciu nadmorskej výšky.
(3) Rovnaké umiestnenie pozorovateľa ako v druhom prípade, ale o niekoľko metrov vyššie, čo umožňuje pohľad ponad pohorie, ktoré v druhom prípade bránilo výhľadu.



Obr. 3.4: Z databázy vyberáme najskôr vrcholy v štvorci, následne odstraňujeme vrcholy nachádzajúce sa v modrej oblasti.

bovali by sme vytvoriť pole polí, čo vyžaduje vytváranie nových polí pre každý riadok a následné kopírovanie hodnôt do jednotlivých riadkov, čo by spomalilo vytváranie panorámy ešte viac. Z tohto dôvodu sme pole typu float ponechali jednorozmerné s nasledovným indexovaním: V pôvodnej dvojrozmernej matici sme na políčko, ktoré určuje našu vzdialenosť od bodu [x,y] na obrázku panorámy, pristupovali ako matrixA[x][y]. V jednorozmernom poli pristupujeme pomocou matrixB[y*6400+x], pričom hodnota 6400 je počet bodov v jednom riadku obrázka panorámy, a teda zároveň aj šírka pôvodného dvojrozmerného poľa.

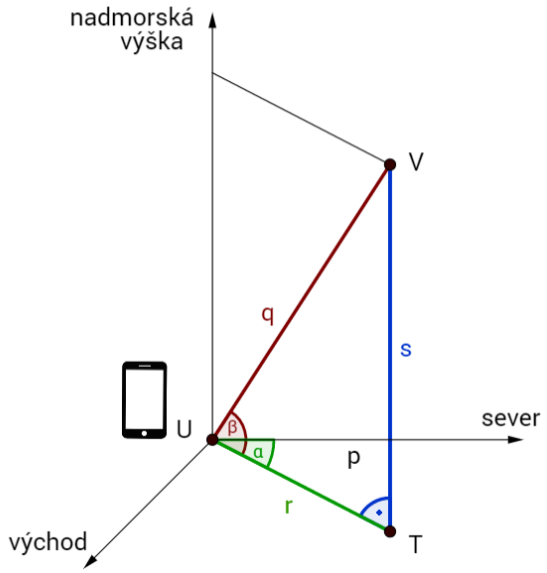
3.4 Anotácia vrcholov

K implementácii tejto časti potrebujeme informácie o vrcholoch, ktoré sú vo vyššie spomínanom XML súbore. V aplikácii budeme používať `SQLite` databázu. K databáze máme prístup prostredníctvom Android knižnice `android.database.sqlite.SQLiteDatabase`.

Pri prvom spustení aplikácie vytvoríme databázu s tabuľkou s názvom *peaks*. Do tabuľky načítame údaje o vrcholoch z XML súboru. Aby sme vedeli pomenovať vrcholy na obrázku, potrebujeme zistiť, ktoré vrcholy sú v okruhu niekoľko kilometrov od používateľa a kde na obrázku panorámy sa daný vrchol nachádza. Preto potrebujeme vypočítať pozíciu vrcholu vzhľadom na našu pozíciu v trojrozmernom priestore.

V našej aplikácii budeme zobrazovať vrcholy v okruhu do vzdialenosti 30km. Pomocou algoritmu *distance* získaného z GeoDataSource [11], na výpočet vzdialenosti daných zemepisných súradníc, zistíme, koľko stupňov zemepisnej šírky a koľko stupňov zemepisnej dĺžky predstavuje vzdialenosť 30km. Z databázy tak vyberáme dáta o vrcholoch, ktoré sa nachádzajú v štvorci 60x60km, v strede ktorého sa nachádza užívateľ. Takto získaná oblasť je znázornená štvorcom na obrázku 3.4.

Keď máme k dispozícii vrcholy v okolí, zistíme ich vzdialenosť od nás pomocou spomínanej funkcie *distance*. Keďže medzi vybranými vrcholmi sa nachádzajú aj tie,



Obr. 3.5: Vertikálny a horizontálny uhol

ktoré sú vo vzdialenosti väčšej ako 30km, môžeme ich odstrániť. Sú to vrcholy, ktoré sú v oblasti znázornenej na obrázku 3.4 modrou farbou. V prípade, že sme vrchol neodstránili, potrebujeme ho správne umiestniť. K tomu potrebujeme získať vertikálny a horizontálny uhol definovaný nižšie.

3.4.1 Horizontálny uhol

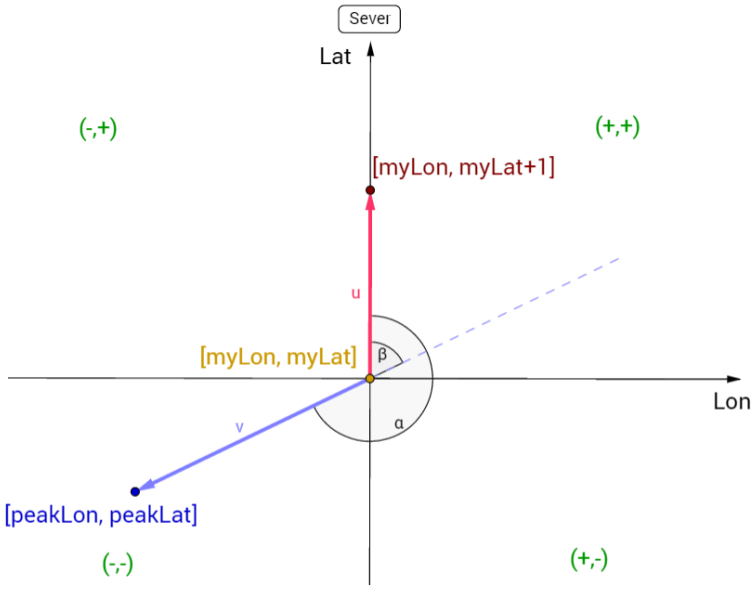
Prvý je uhol, ktorý je na obrázku 3.5 znázornený zelenou farbou. Je to uhol α medzi priamkou p smerujúcou z juhu na sever v konštantnej nadmorskej výške a priamkou r, ktorá prechádza bodom U určeným zemepisnými súradnicami používateľa a bodom T so zemepisnými súradnicami vrcholu, takisto vodorovne v nadmorskej výške určenej nadmorskou výškou smartfónu. Tento uhol budeme nazývať horizontálny uhol.

Na výpočet horizontálneho uhla α (obrázok 3.6) využijeme GPS súradnice vrcholu $[peakLon, peakLat]$ a zariadenia $[myLon, myLat]$. Nadmorskú výšku neberieme do úvahy, teda využívame dvojrozmerný priestor. Zo zadaných bodov nájdeme dva smerové vektory - vektor u smerujúci zo zariadenia do vrcholu a vektor v zo zariadenia priamo na sever. Zo vzťahu:

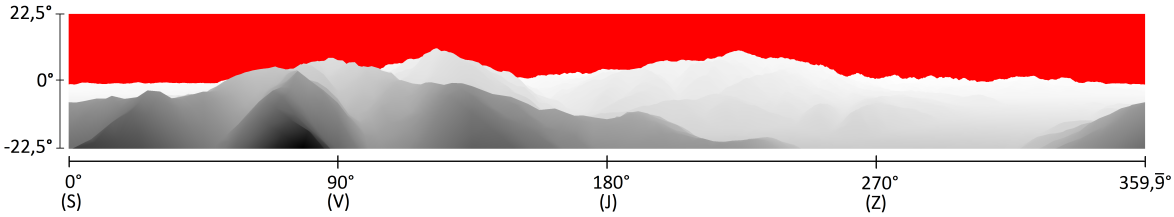
$$\cos(\alpha) = \frac{u_1 v_1 + u_2 v_2}{\sqrt{u_1^2 + u_2^2} \cdot \sqrt{v_1^2 + v_2^2}} \tag{3.1}$$

vypočítame uhol dvoch vektorov. Keďže smerový vektor u je (0,1), môžeme rovnicu zjednodušiť:

$$\cos(\alpha) = \frac{v_2}{\sqrt{v_1^2 + v_2^2}} \tag{3.2}$$



Obr. 3.6: Horizontálny uhol



Obr. 3.7: Horizontálny a vertikálny uhol na panoráme

Výsledkom je veľkosť uhla α, ktorý zvierajú tieto vektory z rozsahu 0° až 90°. Ostáva dopočítať presnú hodnotu uhla z intervalu $[0, 360)$, podľa kvadrantu, v ktorom sa vektor u nachádza. Kvadranty sú znázornené na obrázku 3.6 zelenou farbou, sú rozdelené osami súradnicovej sústavy so stredom v bode $[myLon, myLat]$. Implementácia výpočtu uhla α je v ukážke kódu 3.3. Takúto aproximáciu na dvojrozmerný priestor môžeme použiť len v predpoklade malých vzdialeností, čo je v našom prípade 30 kilometrov, kde môžeme zanedbať zaoblenie zeme. Takisto predpokladáme, že zemepisné súradnice sú z územia Slovenska.

Obrázok panorámy zobrazuje 360° pohľad okolo bodu používateľa. Uhly otočenia, ako sú znázornené na obrázku 2.2 v kapitole 2 sú totožné s horizontálnymi uhlami znázornenými na obrázku 3.7. Následne stačí prerátať uhol na pixely. Takto získame jednu, konkrétne x-ovú, súradnicu bodu na obrázku panorámy.

3.4.2 Vertikálny uhol

Druhý uhol budeme nazývať vertikálny uhol. Na obrázku 3.5 je to uhol β a je vyznačený červenou farbou. Tento uhol zvierajú priamky r, ktorá je popísaná vyššie, a q, ktorá

```
1  private double getHorizontalAngle(double myLat, double myLon, double
       peakLat, double peakLon) {
2          //smerovy vektor v (uzivatel->vrchol)
3          double v1 = peakLon-myLon;
4          double v2 = peakLat-myLat;
5
6          double result = Math.acos(v2 / (Math.sqrt((v1*v1)+(v2*v2))));
7
8          result = rad2deg(result);  // prepocet z radianov na stupne
9
10         if (v1 >= 0 && v2 >= 0) {                // ++
11             return result;
12         } else if (v1 >= 0 && v2 < 0) {          // +-
13             return 180-result;
14         } else if (v1 < 0 && v2 >= 0) {          // -+
15             return 360-result;
16         } else {                                 // --
17             return 180+result;
18         }
19 }
```

Ukážka kódu 3.3: Funkcia na výpočet horizontálneho uhla

smeruje z bodu U do bodu V. Bod V je určený zemepisnými súradnicami a nadmorskou výškou vrcholu.

Na výpočet tohto uhla sme využili skutočnosť, že po aproximácii môžeme považovať trojuholník VUT za pravouhlý. Dĺžka strany s je rozdiel nadmorských výšok a dĺžku strany r dostaneme ako výsledok funkcie distance so zadanými súradnicami bodu U a T. Keďže je trojuholník pravouhlý, goniometrickou funkciou tangens dostávame uhol β nasledovne:

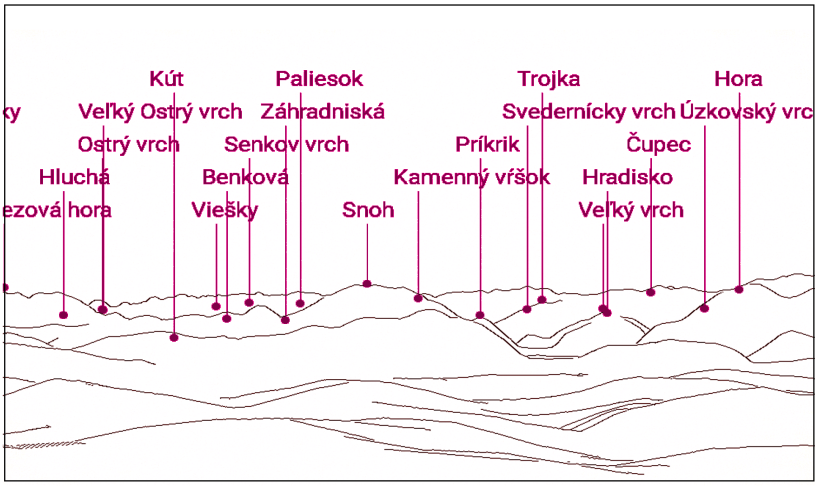
$$\beta = \arctan\left(\frac{|s|}{|r|}\right) \tag{3.3}$$

Po vypočítaní vertikálneho uhla pripočítame konštantu 112,5°, aby sa zhodovali hodnoty zo senzorov s hodnotami na panoráme zobrazenými na obrázku 3.7. Ďalej postupujeme prepočtom na pixely a analogicky ako pri horizontálnom uhle. Uhly veľkosti väčšej ako 22,5° alebo menšej ako -22,5° sa na obrázku nenachádzajú, teda ich do panorámy nezakresľujeme.

3.4.3 Prekrývanie vrcholov

Teraz, keď poznáme obidve súradnice bodu, kde sa na panoráme nachádza hľadaný vrchol, ostáva nám vyriešiť prekrývanie vrcholov, teda či na nájdenom bode sa skutočne nachádza daný vrchol alebo ho prekryl nejaký vrchol, ktorý je bližšie.

Z použitej diplomovej práce na vytvorenie modelu panorámy sme získali aj pole vzdialeností bodov na panoráme od nás. Tieto dáta sú uložené v poli typu float[][] uvádzané v metroch. Po zistení súradníc vrcholu na obrázku panorámy zistíme, aká



Obr. 3.8: Anotácia vrcholov na obrázku panorámy bez prekrývania popisov.

je vzdialenosť tohto bodu od používateľa podľa matice vzdialeností. Zároveň použijeme vypočítanú vzdialenosť používateľa a vrcholu funkciou *distance*. Túto hodnotu porovnáme s hodnotou na danom bode v matici a okolitými hodnotami v matici. Ak je rozdiel niektorej hodnoty z matice v oblasti okolia bodu menší ako konštanta, je to hľadaný vrchol.

Po vyskúšaní viacerých možností sme sa rozhodli ako okolie zvoliť štvorec rozmerov 30x30 bodov a konštantu až 5 kilometrov. Voľba tak vysokej konštanty je z dôvodu, že matica vzdialeností obsahuje často nepresné údaje. Takáto veľká konštanta na druhej strane spôsobuje, že niektoré vrcholy sa zobrazia, aj keď ich iný vrchol prekrýva a zároveň sa môže stať, že sa vrchol nezobrazí z dôvodu ešte väčšej chyby v matici.

3.4.4 Popisy vrcholov

Takto sme vytvorili zoznam vrcholov, ktoré sú v dohľade, spolu s informáciou, kde sa nachádzajú na panoráme. Na toto miesto vykreslíme bod označujúci vrchol a v jeho blízkosti text s názvom tohto vrcholu. Ak by sme názvy vrcholov písali priamo k tomuto bodu, v prípade, že sa nachádzajú príliš blízko seba na obrázku, boli by texty nečitateľné. Z tohto dôvodu zoradíme vrcholy podľa x-ovej súradnice na panoráme a anotácie susedných vrcholov umiestňujeme na y-ovej súradnici tak, aby sa neprekrývali. Obrázok 3.8 zobrazuje príklad takéhoto rozmiestnenia popisov. Z dôvodu lepšej čitateľnosti tiež prispôsobujeme veľkosť písma podľa priblíženia panorámy.

3.5 Zarovnanie reliéfu

Po zobrazení náhľadu kamery sa snažíme čo najpresnejšie odhadnúť a zakresliť hrany panorámy tak, aby zodpovedali reliéfu, ktorý vidíme cez náhľad kamery.

```
1 WindowManager wm = (WindowManager) context.getSystemService(Context.
    WINDOW_SERVICE);
2 Display display = wm.getDefaultDisplay();
3 Point size = new Point();
4 display.getSize(size);
5 int widthScreen = size.x;
6 int heightScreen = size.y;
```

Ukážka kódu 3.4: Zistenie výšky a šírky obrazovky

Zarovnávanie funguje v dvoch módoch. Predvolený mód je automatický, druhý mód je manuálny. Módy môžeme kedykoľvek zmeniť v ponuke aplikácie.

V automatickom móde sa vykonáva automatické zarovnávanie, ktoré môžeme pohybom prstov po obrazovke manuálne upravovať.

V manuálnom móde sa automatické zarovnávanie nevykonáva. Všetky pohyby s panorámou vykonáva používateľ ručne.

3.5.1 Automatické zarovnanie

Automatické zarovnávanie funguje na základe hodnôt získaných zo senzorov zariadenia. Senzorom sme sa bližšie venovali v kapitole 2. V tejto časti popíšeme ich konkrétne využitie v aplikácii.

Z hodnoty otočenia okolo osi Z získame informáciu o tom, aký je horizontálny uhol medzi obrazom sledovaným kamerou a severom. Analogicky vieme zistiť údaj o vertikálnom uhle z otočenia okolo osi Y.

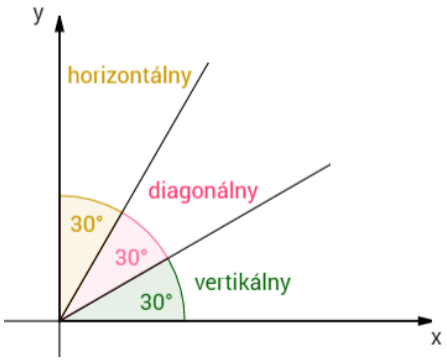
Ďalšími potrebnými informáciami sú horizontálny a vertikálny zorný uhol kamery, ktorým sme sa venovali v kapitole 1 a veľkosť, teda výška a šírka, obrazovky (kód 3.4). Zo zorných uhlov kamery a z uhlov otočenia okolo osi Y a Z vieme zistiť, ktorý výsek panorámy potrebujeme zobraziť. Následne tento výrez prispôsobíme veľkosti obrazovky a zobrazíme.

Pôvodne sme nechceli z panorámy vyrezávať časti, ale pôvodný obrázok vypočítanej panorámy zmenšovať, zväčšovať a posúvať, čím by nám na obrazovke zostala časť, ktorú sledujeme. Avšak pri zväčšovaní nám často aplikácia zlyhala kvôli nedostatku pamäte. Z tohto dôvodu sme sa rozhodli zobrazovanú panorámu vždy podľa potreby vystrihnúť.

Z dôvodu nepresnosti senzorov toto zobrazenie nemusí byť správne. Preto sme pridali možnosť manuálneho zarovnania, ktoré je v praxi použiteľnejšie ako automatické zarovnávanie.

3.5.2 Manuálne zarovnanie

Manuálne zarovnanie umožňuje ručne posúvať, zmenšovať a zväčšovať obraz panorámy podľa toho, ako skutočne vidíme krajinu pomocou náhľadu. Zmeny vykonané v ma-



Obr. 3.9: Uhly na rozlíšenie typu priblíženia

nuálnom zarovnaní sa následne budú aplikovať pri automatickom zarovnávaní.

Pohyb prstov po obrazovke zariadenia zachytíme pomocou `android.view.View.setOnTouchListener`. Pomocou Android triedy `android.view.MotionEvent` [6] vieme zaznamenať konkrétny pohyb. Z každej akcie vieme získať informáciu o pozícii prstu na obrazovke. Keď používateľ položí prvý prst na obrazovku, kód akcie je `ACTION_DOWN`, každý ďalší prst, ktorý sa dotkne obrazovky, je považovaný za akciu `ACTION_POINTER_DOWN`. Každý dotyk obrazovky má vlastný identifikátor, ktorý je platný, až kým nedvihneme prst, čo je akcia `ACTION_POINTER_UP`. Pohyb má kód `ACTION_MOVE`. V aplikácii rozlišujeme dva módy pohybu.

Prvý mód je *DRAG*, ktorý nastáva v čase, keď sa obrazovky dotýka len jeden prst. Počas tohto módu sledujeme posun prsta a podľa vzdialenosti posunu presúvame siluetu panorámy po obrazovke. Smer posunu určíme podľa zmeny súradníc. Ak je väčší rozdiel v súradniciach x, posúvame o tento rozdiel panorámu horizontálne. V opačnom prípade presúvame vertikálne. Posúvanie panorámy len horizontálne alebo len vertikálne sme sa rozhodli použiť z dôvodu presnejšej a jednoduchšej manipulácie.

V čase, keď sa dotýkajú obrazovky dva prsty naraz, zmení sa mód na *ZOOM*. V tomto móde meníme priblíženie siluety panorámy. Či ide o priblíženie alebo oddialenie, rozhoduje vzdialenosť medzi prstami pred posunom a po ňom. Ďalej potrebujeme rozlíšiť, či meníme veľkosť náhľadu len vertikálne, len horizontálne alebo súčasne horizontálne aj vertikálne. Na toto slúži funkcia na rozlíšenie typu priblíženia 3.5. Vo funkcii najprv zistíme umiestnenie prstov na obrazovke. Tieto dva body vytvárajú myslenú priamku. Vypočítaním absolutnej hodnoty uhla smernice priamky s kladnou časťou osi X vieme rozlíšiť tieto tri prípady, ako je zobrazené na obrázku 3.9.

Na obrázku 3.10 vidíme snímky obrazovky aplikácie po manuálnom zarovnaní panorámy so sledovaným reliéfom krajiny.

```
1  private int getZoomMode(MotionEvent e) {
2          float x1 = e.getX(0);
3          float y1 = e.getY(0);
4          float x2 = e.getX(1);
5          float y2 = e.getY(1);
6
7          if (x1 == x2) {
8                  return VERTICAL;
9          }
10         if (y1 == y2) {
11                 return HORIZONTAL;
12         }
13         double slope = Math.abs(rad2deg(Math.atan((y2 - y1) / (x1 - x2
               ))));
14
15         if (slope < 30) {
16                 return HORIZONTAL;
17         } else if (slope > 60) {
18                 return VERTICAL;
19         } else {
20                 return DIAGONAL;
21         }
22 }
```

Ukážka kódu 3.5: Funkcia na rozlíšenie typu priblíženia

Obr. 3.10: Panoráma po manuálnom zarovnaní.

Kapitola 4

Detekcia hrán

V tejto časti popíšeme algoritmus na detekciu hrán. Tento algoritmus budeme používať na detekciu hrán panorámy. Detekciu hrán potrebujeme vykonať, aby sme mohli na obrazovke zobraziť len siluetu počítačového modelu pred náhľadom z kamery.

Detekcia hrán je jeden zo základných a nevyhnutných krokov množstva aplikácií počítačového videnia, pretože zjednodušuje grafickú informáciu. Odstraňuje nepotrebné informácie a ponecháva len tie, ktoré sú pre tieto aplikácie užitočné. Aby sme mohli zarovnať získanú panorámu s obrazom z náhľadu kamery, budeme potrebovať vykonať detekciu hrán na panoráme. Hlavným cieľom tohto algoritmu je nájsť a identifikovať hrany pomocou nesúvislosti farieb na obrázku. Detekcia hrán slúži na zjednodušenie analýzy spracúvaných vizuálnych dát a výrazne znižuje množstvo dát, keďže z daného obrazu nám uchováva len štrukturálne informácie potrebné pre naše ďalšie výpočty. Takto môžeme jednoducho detegovať objekty.

Nie je jednoduché vytvoriť dobrý algoritmus pre detekciu. Pri ich implementácii nastáva množstvo problémov, ako sú nájdenie falošnej hrany, šum, chýbajúca hrana, nízky kontrast hraníc alebo zlé umiestnenie nájdenej hrany. Kvalita detekcie hrán závisí aj od svetelných podmienok a hustoty hrán na obrázku. Aby sme predišli nekvalitnej detekcii je dôležité použiť dobrý algoritmus a zvoliť vhodnú prahovú hodnotu.

Sú známe viaceré algoritmy na detekciu hrán. Saket Bhardwaja a Ajay Mittal vo svojom článku [29] porovnávajú viaceré významnejšie algoritmy. Pre účely našej práce sme vybrali Cannyho detekciu hrán. Tento algoritmus je v článku označený ako za jeden z najlepších. Algoritmus je jedným z najspoľahlivejších a najpresnejších súčasných algoritmov na detekciu hrán a použijeme ho z dôvodu, že je zároveň veľmi rozšírený a používaný.

4.1 Cannyho detekcia hrán

Cannyho detekcia hrán (angl. Canny edge detection) je viacfázový algoritmus na detekciu hrán, ktorý prezentoval John Canny v novembri roku 1986 v publikácií IEEE Transactions on Pattern Analysis and Machine Intelligence.

John Canny vo svojej práci uvádza tri hlavné kritéria, ktoré by mal algoritmus pre detekciu hrán spĺňať. Prvá požiadavka na výsledok algoritmu je snaha o nízku pravdepodobnosť výskytu chýb. Je zrejmé, že na výslednom obrázku nechceme, aby chýbali hrany, ktoré sa vyskytujú v pôvodnom obrázku a zároveň aby sa neobjavovali žiadne falošné hrany. Týmto maximalizujeme pomer signálu k šumu (angl. signal-to-noise ratio). Ďalšou, tiež veľmi dôležitou požiadavkou je správne umiestnenie hrany. Chceme teda dosiahnuť minimálnu vzdialenosť medzi bodom výslednej hrany a stredom skutočnej hrany. Posledné, tretie kritérium, vyžaduje, aby algoritmus zabránil viacnásobnému zobrazeniu jednej reálnej hrany. Toto kritérium Canny pridal dodatočne. Je síce podobné prvému kritériu, ale v ňom nebral do úvahy viacnásobné hrany.

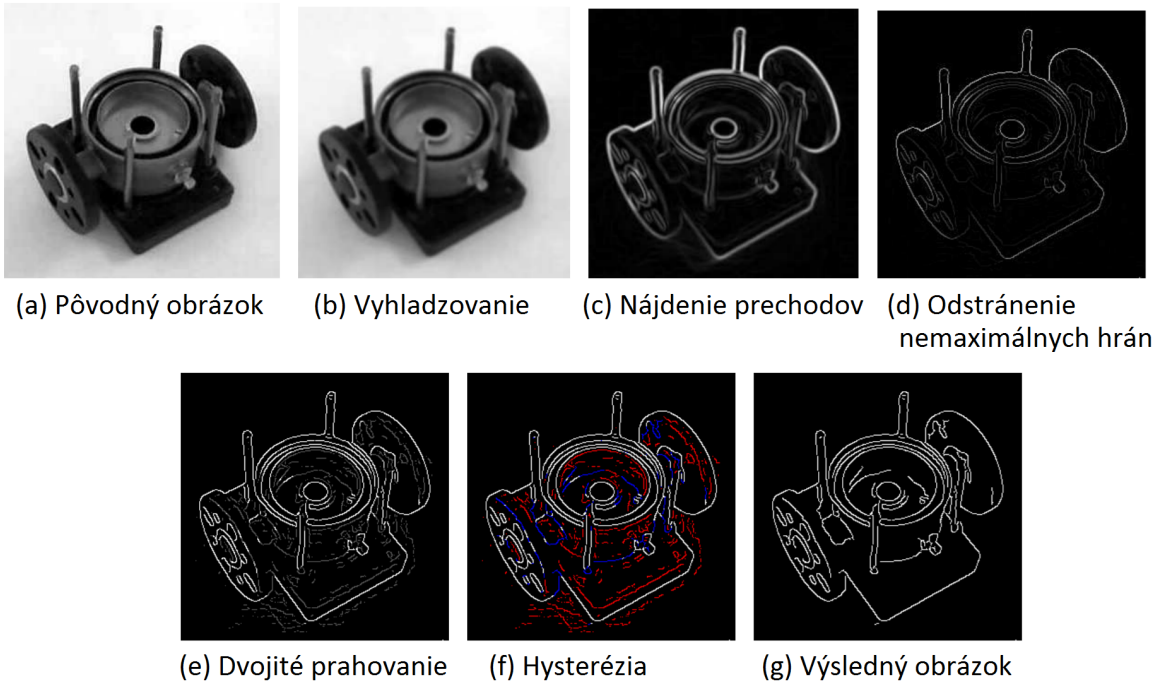
Cannyho algoritmus na detekciu hrán je zložený z postupnosti týchto piatich krokov [2, 26]:

1. Vyhladzovanie – v tomto kroku odstraňuje šum, pričom využíva Gaussovo filtrovanie. Z pôvodného obrázku tak vznikne rozmazanejší obrázok. Tento rozdiel vidíme na príklade na obrázkoch 4.1 (a) a (b).

2. Nájdenie prechodov – transformuje obrázok do odtieňov sivej a na základe zmeny farebnej intenzity nájde hrany na obrázku. Zmenu vidíme na obrázku 4.1 (c).

3. Odstraňovanie lokálne nemaximálnych hrán – týmto krokom sa odstránia body, ktoré nie sú považované za hrany. Po tomto kroku zostávajú iba tenké čiary, ktoré sú vhodnými kandidátmi na hrany. Sledovaním obrázka pred týmto krokom (obrázok 4.1 (c)) a po ňom (obrázok 4.1 (d)) sa zdá, akoby sme predtým rozmazané hrany zaostrili.

4. Dvojité prahovanie – tento krok je potrebný na to, aby sme odstránili hrany, ktoré nepotrebujeme. Sú to hrany, ktoré vytvára šum ako napríklad drsný povrch materiálu. Prahovanie vo všeobecnosti ponecháva na obrázku len hrany, ktoré sú silnejšie ako daná hodnota. Cannyho algoritmus používa dve prahové hodnoty, ktoré nám body hrán rozdelia do troch kategórií. Tieto hodnoty nazývame vysoká a nízka prahová hodnota. Body hrán, ktoré sú silnejšie ako vysoká prahová hodnota, sú označované ako silné. Tie, ktoré sú menšie ako nízka prahová hodnota, odstráni a hrany, ktoré sú medzi vysokou a nízkou prahovou hodnotou, sú označené ako slabé. Na obrázku 4.1 (e) vidíme, akú zmenu spôsobilo použitie



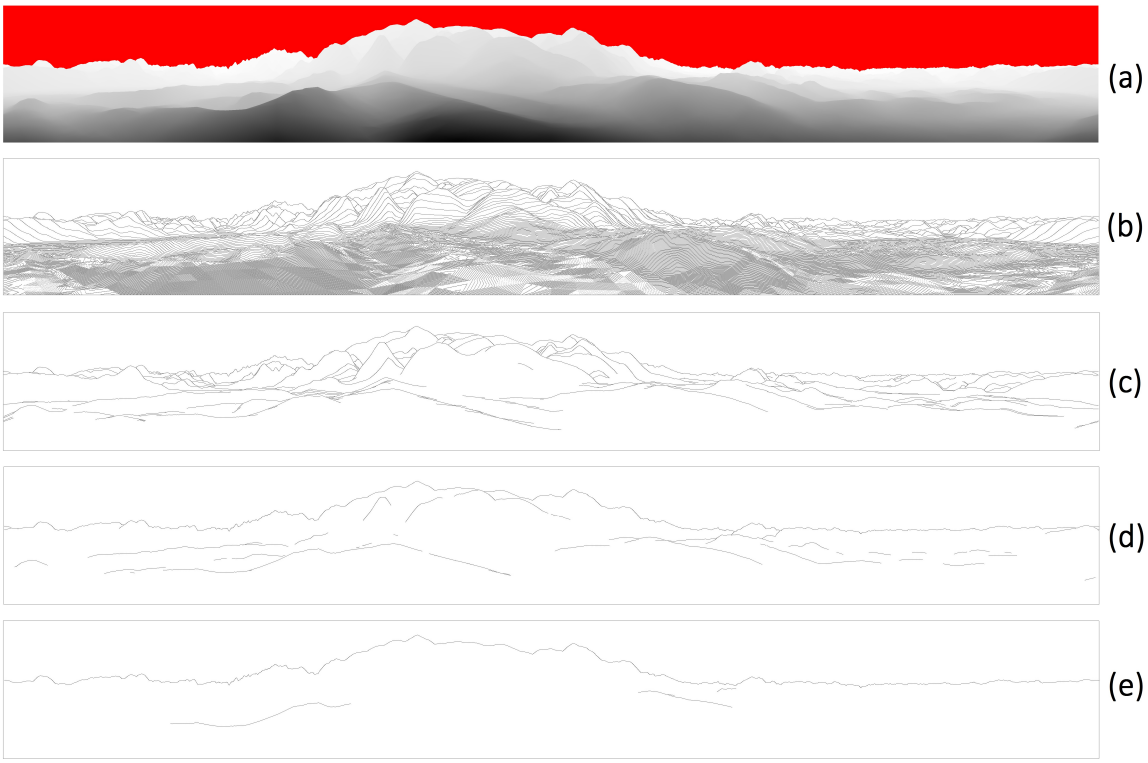
Obr. 4.1: Postup cannyho detekcie hrán. [26]

prahových hodnôt 20 a 80. Na tomto obrázku sú silné hrany biele a slabé sivé, ostatné hrany boli odstránené.

5. Hysterézia – silné hrany sú automaticky vložené do výsledného obrázku a slabé hrany sú vložené práve vtedy, ak sú spojené s nejakou silnou hranou. Na obrázku 4.1 (f) sú modrou farbou označené slabé hrany, ktoré sú spojené so silnými a červenou farbou slabé hrany, ktoré odstráni. Takto vznikne výsledná detekcia hrán, ktorú vidíme na obrázku 4.1 (g).

4.2 Implementácia

Pri implementácií tejto časti práce sme skúšali použiť rôzne voľne dostupné implementácie algoritmu v jazyku Java. Ich cieľom bolo vytvoriť detekciu hrán statického obrázku. Pri testovaní týchto algoritmov na počítači sme zistili, že pre naše účely je čas spracovania obrázku príliš dlhý. Napríklad, obrázok vzdialenostnej panorámy približnej veľkosti 213 kB sme na počítači s procesorom Intel Core i7-5600U 2,60 GHz a 8 GB RAM spracúvali približne 15 sekúnd. Pri predpoklade menej výkonných procesorov v mobilných zariadeniach sú tieto hodnoty nevyhovujúce. Potrebujeme implementáciu, ktorá by vedela detekciu vykonávať rýchlejšie. Preto sme sa rozhodli použiť knižnicu `OpenCV4Android`, ktorú sme spomínali vyššie.



Obr. 4.2: Porovnanie rôznych prahových hodnôt použitých pri Cannyho detekcii hrán. Prahové hodnoty sú uvedené v poradí nízka/vysoká prahová hodnota.
(a)Pôvodný obrázok, (b) 1/1, (c) 1/20, (d) 40/60, (e) 100/200.

4.2.1 Nastavenie prahovej hodnoty

Prahová hodnota (angl. threshold) určuje hranicu, čo už je považované za hranu a čo ešte nie. Vhodným nastavením tejto hodnoty redukujeme šum, no následkom toho môže dôjsť k skresleniu niektorých hrán. Žiaden z algoritmov si tieto hodnoty nevie nastaviť sám na ideálne hodnoty, pretože využitie algoritmu na detekciu hrán je rôzne, čo spôsobuje aj množstvo rôznych požiadaviek na tento algoritmus. Preto je potrebné nastaviť vhodnú prahovú hodnotou pre konkrétny účel. Skúšali sme rôzne nastavenia prahovej hodnoty. Na obrázku 4.2 vidíme rozdiely v zobrazení a nezobrazení detailov v rôznych nastaveniach. V algoritme Cannyho detekcie v `OpenCV` knižnici v našej práci sme použili hodnoty 1 pre nízku a 20 pre vysokú prahovú hodnotu.

Záver

V tejto práci sme vytvorili aplikáciu, ktorá do náhľadu kamery zobrazuje užívateľovi siluetu s anotáciou vrcholov. Túto siluetu si užívateľ vie sám prispôsobiť podľa reality, aby čo najlepšie opisovala reliéf sledovaný náhľadom kamery.

Spomalenie aplikácie pri vytváraní panorámy je spôsobené najmä zmenou typu dát pri prenášaní, ale aj samotou tvorbou panorámy na strane servera. Nepresnosti zobrazovania sú spôsobené vo väčšine častí algoritmu nepresnosťou senzorov, ale zároveň aj nízkym rozlíšením databázy nadmorských výšok. Pri anotácií vrcholov nám tiež robili problém nepresné hodnoty v matici vzdialeností, ktorú sme prevzali z použitej diplomovej práce.

Túto aplikáciu je možné v budúcnosti rozšíriť o viacero prvkov. Jedným z nich je automatické zarovnávanie hrán počítačového modelu s hranami vypočítanými z náhľadu kamery. Na toto vylepšenie je vhodným predpokladom knižnica `OpenCV`, ktorú sme v práci použili. Táto knižnica vie vykonávať detekciu hrán v reálnom čase. Pri tvorbe tohto rozšírenia však môžeme naraziť na problém pri veľkej odlišnosti výsledku z detekcií hrán z reálneho obrazu a z modelu. Práca [28] sa zaoberá algoritmom, ktorý by riešil tento problém špecifickým spôsobom. Autori však v tejto práci zdôrazňujú časovú zložitosť, ktorá nie je vhodná na použitie pri zarovnávaní v reálnom čase.

Ďalším možným rozšírením je doplnenie ďalších informácií o sledovaných vrcholoch, ako napríklad ich vzdialenosť od nás, informácie o trasách na daný vrchol a podobne. Pridaním ďalších databáz je možnosť vytvoriť aplikáciu, ktorá bude fungovať celosvetovo, nielen na území Slovenska.

Priamym rozšírením našej aplikácie by mohlo byť sprístupnenie niektorých nastavení užívateľovi. Takto by používateľ mohol napríklad sám nastaviť GPS súradnice v prípade nedostupnosti GPS signálu, zmeniť určovanie nadmorskej výšky zo senzora namiesto databázy pre prípad, že užívateľ sa nenachádza priamo na zemi, zväčšiť alebo zmenšiť polomer okruhu sledovaných vrcholov a podobne.

Literatúra

[1] Angle of view. [Citované 23.1.2017] Dostupné z https://en.wikipedia.org/wiki/Angle_of_view.

[2] Canny Edge Detector OpenCV. [Citované 20.4.2017] Dostupné z http://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/canny_detector/canny_detector.html.

[3] Digital Elevation Data. [Citované 26.2.2017] Dostupné z http://viewfinderpanoramas.org/dem3.html.

[4] Dokumentácia k triede Camera.Parameters. [Citované 20.1.2017] Dostupné z https://developer.android.com/reference/android/hardware/Camera.Parameters.html.

[5] Dokumentácia k triede Location. [Citované 23.1.2017] Dostupné z https://developer.android.com/reference/android/location/Location.html.

[6] Dokumentácia k triede MotionEvent. [Citované 5.4.2017] Dostupné z https://developer.android.com/reference/android/view/MotionEvent.html.

[7] Dokumentácia k triede SensorManager. [Citované 23.1.2017] Dostupné z https://developer.android.com/reference/android/hardware/SensorManager.html.

[8] Dokumentácia k triede SurfaceView. [Citované 18.4.2017] Dostupné z https://developer.android.com/reference/android/view/SurfaceView.html.

[9] Dual Compass Rose. [Citované 4.3.2017] Dostupné z https://openclipart.org/detail/247149/dual-compass-rose.

[10] Freemap Slovakia. [Citované 26.2.2017] Dostupné z http://wiki.freemap.sk.

[11] GeoDataSource. [Citované 24.3.2017] Dostupné z http://www.geodatasource.com/developers/java.

[12] Geografia Slovenska. [Citované 26.2.2017] Dostupné z https://sk.wikipedia.org/wiki/Geografia_Slovenska.

[13] Gyroscope. [Citované 3.3.2017] Dostupné z https://www.mathworks.com/help/supportpkg/android/ref/simulinkandroidsupportpackage_galaxys4_gyroscope.png.

[14] jMonkeyEngine. [Citované 25.2.2017] Dostupné z http://jmonkeyengine.org/.

[15] Nokia Lumia 1020: 41 Mpx fotoaparát pod drobnohledem. [Citované 23.1.2017] Dostupné z https://m.mobilenet.cz/clanky/nokia-lumia-1020-41mpx-fotoaparat-pod-drobnohledem-12474.

[16] OpenCV. [Citované 20.2.2017] Dostupné z http://opencv.org/about.html.

[17] OpenCV Tutorial 1: Camera Preview. [Citované 28.4.2017] Dostupné z https://docs.nvidia.com/gameworks/content/technologies/mobile/opencv_tutorial_camera_preview.htm.

[18] OpenStreetMap. [Citované 26.2.2017] Dostupné z http://wiki.freemap.sk/OpenStreetMap.

[19] OSM XML. [Citované 27.2.2017] Dostupné z http://wiki.openstreetmap.org/wiki/OSM_XML.

[20] Sensors Overview. [Citované 21.1.2017] Dostupné z https://developer.android.com/guide/topics/sensors/sensors_overview.html.

[21] Shuttle Radar Topography Mission. [Citované 26.2.2017] Dostupné z https://en.wikipedia.org/wiki/Shuttle_Radar_Topography_Mission.

[22] The Sword of Damocles. [Citované 23.1.2017] Dostupné z https://en.wikipedia.org/wiki/The_Sword_of_Damocles_(virtual_reality).

[23] <uses-feature>. [Citované 10.5.2017] Dostupné z https://developer.android.com/guide/topics/manifest/uses-feature-element.html.

[24] Vyvíjíme pro Android: Bližší pohled na pohledy – 2. díl. [Citované 18.4.2017] Dostupné z https://www.zdrojak.cz/clanky/vyvijime-pro-android-blizsi-pohled-na-pohledy-2-dil/.

[25] What is computer vision? [Citované 22.2.2017] Dostupné z http://www.bmva.org/visionoverview.

[26] 09gr820. Canny Edge Detection, 2009.

[27] Ronald T. Azuma. A survey of augmented reality. *Presence: Teleoperators and virtual environments*, 6(4):355–385, 1997.

[28] Lionel Baboud, Martin Čadík, Elmar Eisemann, and Hans-Peter Seidel. Automatic Photo-to-Terrain Alignment for the Annotation of Mountain Pictures. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR), oral presentation*, 2011.

[29] Saket Bhardwaj and Ajay Mittal. A survey on various edge detector techniques. *Procedia Technology*, 4:220–226, 2012.

[30] Nicolas Gramlich. *andbook! Android Programming.* anddev.org, 2008.

[31] Jens Grubert and Dr. Raphael Grasset. *Augmented Reality for Android Application Development.* Packt Publishing Ltd., November 2013.

[32] Bc. Branislav Kulka. Inerciální navigační jednotka. Diplomová práca, Fakulta elektrotechniky a komunikačních technologií, Vysoké učení technické v Brně, 2014.

[33] Greg Milette and Adam Stroud. *Professional Android™ Sensor Programming.* John Wiley & Sons, Inc., Indianapolis, Indiana, 2012.

[34] Peter Nohejl. Rozšířená realita pro platformu Android. Diplomová práca, Vysoké učení technické v Brně, Fakulta informačních technologií, 2011.

[35] Raghav Sood. *Pro Android augmented reality.* Apress, 2012.

[36] Lev Tverdokhlebov. Landscape Modelling for Outdoor Photography Analysis, Geolocation and Distance Estimation. Diplomová práca, Politecnico di Milano, Scuola di Ingegneria Industriale e dell'Informazione, 2015.

Príloha

CD

Zdrojové súbory aplikácie sa nachádzajú na priloženom CD.