Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE

Martin Dobiaš

Navigačný systém pre automobil

Ústav formální a aplikované lingvistiky

Vedoucí bakalářské práce: Mgr. Pavel Machek, ÚFAL
Studijní program: Informatika

2007

Chcel by som poďakovať všetkým vývojárom projektu Quantum GIS za ich spoluprácu. Špeciálne ďakujem Timovi Suttonovi, ktorý mi ochotne venoval mnoho času a pomáhal pri vývoji Quantum GIS.

Prehlasujem, že som svoju bakalársku prácu napísal samostatne a výhradne s použitím citovaných prameňov. Súhlasím s požičiavaním práce a jej zverejňovaním.

V Prahe dňa 30.5.2007 Martin Dobiaš

Obsah

Názov práce: Navigačný systém pre automobil
Autor: Martin Dobiaš
Katedra (ústav): Ústav formální a aplikované lingvistiky
Vedúci bakalárskej práce: Mgr. Pavel Machek
E-mail vedúceho: pavel@ucw.cz

Abstrakt: Prvá časť práce je zameraná na prípravu prostredia pre vývoj navigačného programu. Na vykresľovanie mapy a ďalšie operácie s geografickými dátami sa používajú knižnice programu Quantum GIS. Popisujú sa úpravy v aplikačnom rozhraní a architektúre a napojenie knižníc Quantum GIS na jazyk Python, v ktorom je navigačný systém napísaný. Druhá časť sa zaoberá vývojom software pre navigáciu áut po cestnej sieti. Navigačný program podľa zadaného štartu a cieľu vyhľadá v cestnej sieti vhodnú trasu. Do úvahy sa berú typy ciest, jednosmerky, zákazy odbočenia a prípadne ďalšie obmedzenia. V prípade, že sa k počítaču pripojí GPS prijímač, zobrazujú sa navigačné informácie podľa aktuálnej polohy vozidla.

Kľúčové slová: Quantum GIS, GPS, navigácia, Python

Title: Navigation system for vehicles
Author: Martin Dobiaš
Department: Institute of Formal and Applied Linguistics
Supervisor: Mgr. Pavel Machek
Supervisor's e-mail address: pavel@ucw.cz

Abstract: First part of the work is devoted to preparation of environment for building a navigation software. For map rendering and other geospatial data manipulation we use libraries of Quantum GIS. The work comprised also library refactoring and creation of bindings for Python language, that's also the language used for development of the navigation system. Second part presents principles and implementation of navigation in a roadmap. Software searches for an optimal route based on given start and destination. It takes in account different road types, one-way streets, turn restrictions and/or other restrictions. In case a GPS receiver is connected to the computer, it displays navigation hints based on current position of the vehicle.

Keywords: Quantum GIS, GPS, navigation, Python

Kapitola 1

Úvod

Hlavným cieľom tejto práce bolo vytvoriť jednoduchý navigačný systém, ktorý dokáže vyhľadať trasu v cestnej sieti za dodržania všetkých predpisov dopravy. Program by mal byť schopný pomocou pripojeného GPS prijímača navigovať vodiča vozidla po navrhnutej trase až do cieľa.

Táto funkcia je veľmi užitočná v prípade, že prechádzame miestami, ktoré nepoznáme dobre a hrozí, že niekde zle odbočíme – to má za následok zdržanie a samozrejme zbytočne najazdené kilometre.

Komerčné systémy na navigáciu

V súčasnej dobe existuje viacero komerčných programov pre osobné počítače, ktoré umožňujú vyhľadávanie optimálnej trasy a navigáciou po nej, napr. Microsoft AutoRoute. Ich masovému rozšíreniu však bráni praktická stránka použitia: pri ich použití na navigovanie po cestách je potrebné vlastniť notebook, ten je však stále relatívne príliš veľký, má pomerne komplikované ovládanie, pomalý štart a veľkú spotrebu. Jednoducho nie je na tento účel príliš vhodný.

Postupom času s príchodom stále kvalitnejších a výkonejších GPS navigátorov je potreba takéhoto software menšia. Vďaka svojim malým rozmerom a jednoduchému ovládaniu (častokrát je použitá dotyková obrazovka) sa stávajú stále častejšie súčasťou výbavy auta. Pomocou detailných cestných máp dokážu elegantne viesť vodiča až do cieľa, pre jednoduchšie použitie sú vybavené hlasovou navigáciou, takže vodič nemusí sledovať displej.

Navigácia a free software

V oblasti *free software* nie je veľa použiteľných aplikácií na navigáciu. Najznámejší je program GpsDrive. Má vcelku prepracované uživateľské rozhranie, dokáže používať rastrové mapy ako podklad pre aktuálnu pozíciu z GPS prijímača, vie pracovať s bodmi a trasami z GPS a ďalšie funkcie. Nedokáže ale navigovať podľa cestnej siete. Zaujímavým projektom je program RoadMap, ktorý má umožňovať navigáciu podľa vektorových máp, je však v ranných štádiách vývoja, podporuje iba cestnú mapu USA a už dlhší čas na ňom nikto nepracuje.

Mapy

Aby bolo možné hľadať trasu, je pochopiteľne nutné mať k dispozícii kvalitné mapové podklady s cestnými sieťami. Okrem nutnosti mať cesty a ulice presne zanesené do mapy to zahŕňa aj existenciu doplnkových informácií o cestnej sieti a obmedzeniach v nej. V prípade, že v mape chýbajú niektoré z týchto informácií alebo sú neaktuálne, reálna hodnota mapy značne klesá, pretože vyhľadané trasy môžu byť nekorektné.

Zásadnou prekážkou vývoja free software navigačných aplikácií je práve neexistencia voľne dostupných detailných cestných máp. Napriek tomu, že štátne inštitúcie majú digitálne spracované podrobné mapy, dávajú ich k dispozícii len pod reštriktívnymi licenciami za poplatky. V rámci Európskej únie vznikli aj petície za voľný prístup k mapovým dátam. Argumentujú tým, že daňoví poplatníci by mali mať možnosť využívať dáta, ktoré štát za ich penize zozbieral. Takýto zákon platí v USA a mapy sú dostupné pre verejnosť bez akýchkoľvek poplatkov.

Situáciu sa snaží riešiť projekt OpenStreetMap [1]. Ide o vytvorenie úplne nových máp bez akýchkoľvek právnych alebo technických obmedzení. Tvorba máp prebieha vďaka dobrovoľníkom, ktorí investujú svoj čas na mapovanie v teréne pomocou GPS. Nesmú ale vychádzať zo žiadnych už existujúcich máp: v tom prípade totiž vzniká odvodené dielo podliehajúce licencii. Navyše spoločnosti niekedy zámerne pridávajú do svojich máp chyby, vďaka ktorým vedia preukázať, že iná mapa je z ich dát odvodená. Momentálne je projekt OpenStreetMap stále v ranných štádiách, pretože aj mnoho veľkých miest je zmapovaných len zbežne. Dáta môžu obsahovať informácie o jednosmerkách a ďalších obmedzeniach, nezahŕňajú však zákazy odbočenia, takže pre použitie na navigáciu po cestnej sieti sú zatiaľ nepoužiteľné. Projekt sa stále vyvíja, takže časom sa možno dočkáme dostatočne kvalitných dát.

Prezentované riešenie

Program vytvorený ako súčasť tejto bakalárskej práce poskytuje základné možnosti na sledovanie pozície z GPS prijímača a vyhľadávanie spojenia v cestnej sieti. Je však otázne, či by mohol byť aj reálne nasadený vzhľadom na niektoré chýbajúce funkcie. Program nemá za cieľ konkurovať komerčným systémom, má byť skôr vyplnením prázdneho miesta na poli free software.

Pre účely vývoja a testovania som dostal k dispozícii kvalitné mapové dáta. Časť z nich mi bolo dovolené zverejniť pre skúšanie tretími osobami a ako ukážka formátu vstupných dát pre program. Dáta pozostávajú z cestnej mapy mesta Trenčín a jeho okolia.

Podstatnú časť programu tvorí interaktívna práca s mapou. Je potrebné načítať geografické dáta, vykresliť z nich mapu, umožniť prehliadanie mapy, zobrazovať aktuálnu polohu a vyhľadané spojenie, získavať geografické súradnice z mapy. Už v rámci ročníkového projektu som sa zapojil do vývoja programu Quantum GIS, ktorý slúži na zobrazovanie a úpravy geografických dát. Do zjednodušenia práce s mapovým zobrazením tohto programu som investoval mnoho úsilia, preto som vsadil na možnosť využiť existujúci kód na prácu s mapou. Druhá kapitola popisuje projekt Quantum GIS, jeho knižnice a úpravy, ktoré boli potrebné na to, aby sa sa existujúci kód dal využívať z iných programov.

V tretej kapitole sa venujem napojeniu knižníc Quantum GIS na vyšší jazyk, konkrétne Python. Motiváciou bolo zjednodušenie prístupu ku knižniciam a zefektívnenie vývoja navigačnej aplikácie.

Štvrtá kapitola dáva prehľad o GPS systéme a rozhraniach používaných na komunikáciu s GPS prijímačmi, piata kapitola pojednáva o problematike vyhľadávania ciest v uličnej sieti. Posledná kapitola sa venuje samotnej implementácii navigačného programu.

Výsledná aplikácia dostala názov **Quantum Navigator** vhľadom na svoju spojitosť s projektom Quantum GIS. Je dostupná na internetovej stránke *http://mapserver.sk/∼wonder/qnavigator/*

Kapitola 2

Quantum GIS

Quantum GIS (QGIS) je geografický informačný systém (GIS), ktorý beží na Linuxe, Mac OS X, Windows a ďalších variantách Unixu. Podporuje množstvo vektorových a rastrových formátov geografických dát a dá sa použiť na ich prehliadanie a úpravy. Pomocou prídavných modulov (pluginov) je možné rozšíriť program o ďalšie funkcie. Podrobnejší prehľad možností je na stránke projektu [2].

Celý software je licencovaný pod GNU General Public License. Táto licencia umožňuje spúšťať program za akýmkoľvek účelom a šíriť kópie, taktiež je možné študovať, ako program funguje a meniť ho (tzn. musí byť dodávaný so zdrojovými kódmi).

Projekt vznikol približne v polovici roku 2002, keď ho začal písať Gary Sherman za účelom vytvorenia jednoduchého prehliadača geografických databáz PostGIS (rozšírenie pre PostgreSql). Zdrojové kódy uverejnil na serveri SourceForge a približne od začiatku roku 2004 začali prispievať do projektu ďalší vývojári. V čase písania práce sa nachádzal vo verzii 0.8, ktorá vyšla 29. decembra 2006, a pripravuje sa vydanie verzie 0.9. Tá bude obsahovať novinky, o ktorých sa zmieňujem v tejto práci.

Zaujímavá je možnosť napojiť QGIS pomocou pluginu na GRASS GIS. Tento software sa začal vyvíjať už v priebehu 80-tych rokov v laboratóriach americkej armády, od roku 1997 je vyvíjaný medzinárodnou komunitou vývojárov. Je to systém s veľkým množstvom funkcií na úpravy a analýzu geografických dát. Jeho slabina však spočíva v uživateľskom rozhraní – je komplikované a neprehľadné, veľa funkcií sa ovláda z príkazového riadku. Spojenie z Quantum GIS rieši tento problém a ponúka pohodlné prostredie na prácu s dátami.

2.1 Používané knižnice

Quantum GIS využíva pre svoj beh a uživateľské rozhranie open source edíciu knižnice Qt od firmy Trolltech. Hlavné výhody tejto knižnice sú:

- **podpora rôznych platforiem** – aplikácie môžu bežať na rôznych operačných systémoch bez nutnosti úpravy zdrojových kódov, pretože poskytuje platformovo nezávislý prístup k službám systému: práca so súbormi, vlákna, synchronizačné primitíva apod. Na Windows a Mac OS X aplikácie môžu využívať natívny vzhľad grafických prvkov.
- **dobre navrhnuté API** – aplikačné rozhranie je jednoduché a kvalitne zdokumentované s množstvom príkladov. V súčasnosti pozostáva z viac ako 400 tried v C++.
- **jednoduchý návrh uživateľského rozhrania** – dodávaný nástroj Designer umožňuje rýchlo navrhovať aj komplikované formuláre a dialógy.
- **možnosť lokalizácie** – podporuje znakovú sadu Unicode a vďaka nástroju Linguist sa dajú preložiť všetky texty aplikácie, ktoré boli označené ako vhodné na preloženie.
- **podpora programovacích jazykov** – Qt je napísané v C++, existujú napojenia na ďalšie jazyky – Ruby, Python, PHP, Java atď.
- **slobodná licencia** – od verzie 4 je open source edícia Qt licencovaná na všetkých platformách pod GNU GPL.

Knižnica Qt je používaná v množstve slobodného aj komerčného software. V slobodnom software je známa pre použitie v rámci desktopového prostredia KDE, ktoré pozostáva z desiatok programov. Známe komerčné programy využívajúce Qt sú napríklad prehliadač Opera, Google Earth, Skype a Adobe Reader.

Ďalšie závislosti Quantum GISu sú:

- **GDAL/OGR** – knižnica na čítanie a zápis rôznych rastrových a vektorových dát
- **Sqlite** – jednoduchá databáza, používa sa na uloženie zoznamu a parametrov projekcií
- **GEOS** – knižnica na prácu s geometriou

2.2 Architektúra

V čase môjho zapojenia sa do projektu Quantum GIS boli jednotlivé časti programu silne prepojené a vytvárali jeden monolit. Zatiaľ čo pre samotnú aplikáciu to nebol veľký problém, nebolo možné využiť kód v iných programoch. Aby sa mohol využiť prvok na zobrazenie mapy (spolu s ďalšími funkciami) bolo nutné pôvodné zdrojové kódy rozdeliť do niekoľkých komponent.

Najvýhodnejšie rozdelenie sa ukázalo byť nasledovné:

- CORE knižnica – obsahuje základné triedy potrebné na prácu s geografickými dátami
- GUI knižnica – obsahuje znovu využiteľné prvky uživateľského rozhrania
- aplikácia – obsahuje ostatné triedy, u ktorých sa neočakáva, že by sa používali mimo aplikácie Quantum GIS

Motiváciou na takéto rozdelenie sú dva prípady použitia knižníc:

- konzolový nástroj alebo serverová aplikácia – využíva CORE knižnicu na čítanie alebo úpravy geografických dát prípadne renderovanie mapy. Nevyžaduje bežiaci X server.
- grafická aplikácia – využíva knižnice CORE aj GUI, používa mapové zobrazenie a umožňuje tak uživateľovi prehliadanie alebo úpravy geografických dát.

Na wiki stránke [3] je uvedený prehľad hlavných uskutočnených úprav a refaktorizácií. Najdôležitejšie bolo odstrániť niektoré závislosti z tried určených pre CORE knižnicu. Slúži aj ako prehľad všetkých tried v knižniciach a v programe, triedy sú podľa účelu zaradené do kategórií.

Rozdelenie do knižníc bol dosť zdĺhavý proces vzhľadom na množstvo tried, ktoré bolo potrebné prispôsobiť. Metodiku rozdeľovania a refaktorizácií som zvolil nasledovne: pre každú triedu predpokladám, že nie je potrebná v knižnici. V prípade, že existuje využitie triedy mimo QGIS, môže sa zaradiť CORE alebo GUI knižnice. Podmienkou je, že nemá závislosti na vyššej vrstve, tzn. triedy v GUI knižnici sa nesmú odkazovať na žiadnu z tried v aplikácii, navyše triedy v CORE knižnici nesmú používať ani triedy z GUI

knižnice. Ak takéto závislosti existujú, musia sa najprv odstrániť a až potom sa trieda môže presunúť do príslušnej knižnice.

Najviac práce bolo s triedami pre CORE knižnicu, ktoré často mali priamu väzbu na uživateľské rozhranie, napríklad trieda reprezentujúca vektorovú vrstvu riadila prácu s dialógom tabuľky atribútov atď. Podobne prvky z GUI knižnice priamo používali iné triedy špecifické pre samotnú aplikáciu – tieto väzby bolo tiež nutné odstrániť.

Projekt QGIS vznikal – ostatne ako mnoho ďalšieho slobodného software – bez nejakej konkrétnej koncepcie. Neorganizovaným pridávaním ďalších funkcií častokrát trpí architektúra, pretože ak si programátor nie je istý ako implementovať nejakú novú funkcionalitu, zvolí väčšinou cestu s najmenším množstvom práce. Niektoré funkcie sa potom veľmi komplikovane rozširujú alebo upravujú a objavujú sa nepríjemné chyby. Preto postupom času pribúda snaha zaviesť do projektu väčšiu organizovanosť vývoja. Uvažuje sa hlavne o vývoji vo vetvách repozitára, pridávaní testov a ich automatizovaného spúšťania. S organizáciou môže pomôcť aj formalizovanie pridávania novej funkcionality, kedy sa nová funkcia najprv prediskutuje z uživateľského a vývojárskeho pohľadu a až následne implementuje, aby sa zmenšila šanca použitia zlých konceptov.

2.3 CORE knižnica

Poskytuje všetky základné prostriedky na prácu s geografickými dátami. Nasleduje popis hlavných oblastí, ktoré knižnica pokrýva.

Práca s geometriami

Základom je trieda `QgsGeometry`, ktorá zaobaluje funkcie knižnice GEOS. V oblasti GIS sa rozlišuje niekoľko druhov geometrií: body, čiary, n-uholníky. Existujú aj ich viac-komponentové (*multipart*) varinaty, kedy sa v jednej geometrii nachádza viac bodov, čiar alebo n-uholníkov. Tie majú význam v prípade, ak sa nejaký objekt nedá zakresliť ako jednoduchá geometria – napríklad ostrovný štát na politickej mape sveta.

Podpora projekcií

Každá mapová vrstva nesie informáciu o tom, aký súradnicový systém je použitý. Rozdeľujeme ich na geografické a projektované. Pre geografické súrad-

nice sa používa geografická šírka a výška. Geografické súradnicové systémy sa medzi sebou mierne líšia (hlavnou a vedľajšiou poloosou elipsoidu), takže súradnice s rovnakou hodnotou označujú iné miesto v rôznych systémoch – rozdiel môže byť až niekoľko metrov. Najčastejšie používaný je americký systém WGS 84 a to aj vďaka rozšíreniu GPS, kde sa používa práve tento systém. V Českej republike a na Slovensku je dosť rozšírený systém S-JTSK.

Projektované systémy vznikajú projekciou geografických súradníc do roviny. Existuje viac základných druhov projekcií (kužeľová, valcová, polárna atď.) a tie sa ešte líšia svojimi parametrami. Projektovaných systémov je veľké množstvo, keďže veľa projekcií je vhodných len na nejaké územie, na ktorom je skreslenie minimálne.

Quantum GIS dokáže načítať informácie o použitom súradnicovom systéme z mapovej vrstvy a v prípade potreby ho projektovať na nejaký iný systém za behu. Súradnicové systémy sa ukladajú v triede `QgsSpatialRefSys` a na transformáciu medzi nimi slúži trieda `QgsCoordinateTransform`.

Data providery

Poskytujú spoločné rozhranie pre prístup k dátam z rôznych zdrojov. Rozšírenie Quantum GIS o podporu nového formátu teda spočíva vo vytvorení nového data provideru. To zahŕňa len implementáciu rozhrania provideru. Pri štarte aplikácie sa zistí zoznam dostupných providerov a uloží do registra (`QgsProviderRegistry`). Momentálne podporované providery sú pre knižnicu OGR (podporuje širokú škálu formátov), PostGIS, GPX súbory, CSV súbory a internetové služby WMS a WFS.

Rozhranie je plne funkčné zatiaľ len pre vektorové dáta.

Vektorové nástroje

Základom vektorovej funkcionality je trieda `QgsVectorLayer`. Stará sa o prístup k dátam cez data provider, vykresľovanie vrstvy, editáciu a ďalšie možnosti. Pri iterácii po jednotlivých prvkoch mapovej vrstvy sa vracia inštancia triedy `QgsFeature`. V nej je referencia na geometriu (`QgsGeometry`) a mapu atribútov, tie sú variantného typu `QVariant`, pretože môžu obsahovať čísla alebo text, v budúcnosti sa môžu pridať ďalšie typy.

Editácia prebieha tak, že vrstva sa najprv prepne do editačného módu. Následne sa môžu volať funkcie, ktoré menia geometriu alebo atribúty prvkov mapy, prípadne prvky odstraňujú alebo pridávajú. Po editácii je možné všetky prevedené operácie uložiť do provideru alebo ich zrušiť.

Rastrové nástroje

Prácu s rastrovými mapami zaobaluje trieda `QgsRasterLayer`. Umožňuje zobrazovanie vrstvy s rôznym rozložením farieb, nepodporuje však export do iných formátov ani reprojekciu do iného súradnicového systému.

Renderovanie mapy

Trieda `QgsMapRender` riadi renderovanie mapy. Je využívaná v grafickom rozhraní aplikácie, dá sa však použiť aj na export mapy do obrázku ľubovolnej veľkosti, PDF súboru, tlačiarne alebo iného formátu podporovaného knižnicou Qt. Umožňuje tak vykreslovať mapy aj bez grafického rozhrania, napríklad vytvoriť mapový server, ktorý bude podľa požiadaviek posielať vykreslené mapy.

Každá vektorová vrstva má referenciu na objekt odvodený od triedy `QgsRenderer`, ktorý určuje spôsob, akým sa jednotlivé prvky vrstvy vykreslia. Štandartne sa všetky prvky kreslia rovnako, dá sa ale rozlíšiť ich podľa niektorého atribútu. Klasifikáciu je možné nastaviť do niekoľkých tried, plynulý prechod z jednej farby do inej alebo pre každú hodnotu atribútu použiť unikátnu farbu. Pri rastrových vrstvách možnosti zobrazenia záležia na tom, či vrstva obsahuje len jednu zložku farby alebo ide o farebný obrázok.

Vektorové vrstvy navyše môžu mať popisky, ktoré sa kreslia až na konci, aby nemohlo dojsť k prekrytiu inými vrstvami. Popisovanie je ale zatiaľ veľmi jednoduché, nerieši sa detekcia kolízie popiskov ani ich automatické natáčanie podľa čiary.

2.4 GUI knižnica

Najdôležitejšou komponentou tejto knižnice je prvok uživateľského rozhrania na zobrazovanie mapy, v žargóne QGIS označovaný ako *map canvas*. Dá sa jednoducho rozširovať o nové funkcie vďaka podpore mapových nástrojov (*map tools*) a mapových prvkov (*map canvas items*).

Map canvas využíva funkcionalitu *graphics view*, ktorá je dostupná v Qt od verzie 4.2. Táto technológia pozostáva z troch typov objektov:

- scéna (`QGraphicsScene`) – kontajner pre prvky scény
- prvky (`QGraphicsItem` a odvodené)
- pohľad (`QGraphicsView`) – konkrétny pohľad na scénu

Každý prvok scény má svoju pozíciu, veľkosť a vlastný spôsob, ako sa vykreslí.

Všetky prvky map canvas sú potomkami `QgsMapCanvasItem`, čo je trieda odvodená od QGraphicsItem a pridáva doplňujúce funkcie na prevod medzi súradnicami a podobne. Výnimkou je prvok, ktorý zobrazuje vykreslenú mapu (`QgsMapCanvasMap`). Ten má špeciálne chovanie a nie je potomkom `QgsMapCanvasItem`. V rámci GUI knižnice sa nachádzajú základné a často používané prvky `QgsRubberBand` (vykreslovanie čiar a n-uholníkov) a `QgsVertexMarker` (značka pre nejaký bod).

Vytváranie nových prvkov je jednoduché, v odvodenej triede stačí implementovať funkciu vracajúcu veľkosť prvku a funkciu, ktorá ho vykresluje. Qt rozhoduje kedy je potrebné prvok alebo celú scénu znovu vykresliť. Vždy, keď dôjde k zmene pohľadu na mapu (napríklad pri posunutí alebo zmene mierky), prvky majú možnosť reagovať na túto zmenu a upraviť svoju aktuálnu pozíciu.

Mapové nástroje sú určené na interakciu map canvas s uživateľom – vždy je aktívny jeden nástroj, ktorému sa predávajú udalosti z map canvas (stlačenie a pustenie tlačítka myši, pohyb myši). Nástroj na udalosti reaguje podľa potreby. Môže vytvárať nové prvky, ktoré potrebuje k svojej činnosti. Napríklad nástroj na meranie vzdialeností vytvára prvok `QgsRubberBand`, ktorý slúži na zobrazovanie vytýčenej cesty alebo plochy na meranie. Pri aktivácii nového nástroja sa pôvodný deaktivuje a prestane dostávať udalosti z map canvas.

V GUI knižnici je aj sada základných nástrojov na posúvanie mapy (`QgsMapToolPan`) a na približovanie a odďaľovanie mapy (`QgsMapToolZoom`). V samotnej aplikácii sú ešte ďalšie nástroje, napríklad na vytváranie a úpravu vektorových vrstiev, identifikáciu a meranie vzdialeností.

Kapitola 3

Napojenie na jazyk Python

3.1 Úvod do problematiky

Napojením jazyka (anglicky *language binding*) na knižnicu sa myslí rozhranie umožňujúce využívať služby knižnice v tomto programovacom jazyku. Hlavnou motiváciou na vytváranie napojení ku knižniciam býva znovuvyužite kódu namiesto toho, aby sa knižnica implementovala v inom jazyku. Kód alebo modul, ktorý vytvára napojenie, nazývame *wrapper*.

Mnoho softwarových projektov sa vyvíja v programovacích jazykoch C a C++. Zdrojové kódy v týchto jazykoch sa kompilujú do inštrukcií procesoru a preto umožňujú rýchly beh programov. Nevýhodou zostáva, že programovanie v nich je komplikovanejšie a komplexnejšie.

Na druhej strane sú vysokoúrovňové jazyky, v ktorých sa dá vyvíjať software omnoho rýchlejšie. Medzi najpopulárnejšie jazyky z tejto kategórie patria Python, Ruby, Perl, JavaScript alebo PHP. Poskytujú vyššiu abstrakciu kódu, čo znamená aj to, že kód býva ľahšie prenositeľný medzi rôznymi platfomami. Majú dobrú podporu pre prácu s reťazcami, dynamickými a asociatívnymi poliami. Vysokoúrovňové jazyky bývajú interpretované (priamo čítané a spúšťané interpreterom bez kompilácie) alebo kompilované (preložené do optimalizovaného medzikódu). Tieto jazyky sú často dynamické, čo znamená, že počas behu programu môžu vykonávať činnosti, ktoré sú dovolené v iných jazykoch len počas kompilácie, napríklad pridávať ďalší kód, rozširovať alebo meniť definície objektov.

Daňou za vyššie popísané výhody je rýchlosť prevádzania programu. Pre porovnanie: réžia pri volaní funkcie v C alebo C++ je niekoľko inštrukcií procesoru, zatiaľ čo v Pythone sa najprv funkcia vyhľadáva v asociatívnom

poli podľa názvu. Problém s rýchlosťou behu sa dá čiastočne eliminovať tým, že výpočtovo náročné rutiny sa napíšu v nižšom jazyku, napríklad v C. Takýmto spojením vzniká zaujímavý kompromis, kedy je možné väčšinu kódu programovať vo vyššom jazyku (čo je zvyčajne jednoduchšie), zatiaľ čo nestrácame moc na rýchlosti, keďže sa náročnejšie operácie obvykle vykonávajú v rýchlejšom, kompilovanom jazyku.

V prípade, že chceme jazyk napojiť na nejaké funkcie, ku ktorým normálne nemá prístup (napríklad komunikácia s hardware) alebo využiť v jazyku nejakú existujúcu knižnicu, hovoríme o rozširovaní (*extending*) jazyka.

Niektoré vysokoúrovňové majú ďalší zaujímavý rys: môžu vykonávať príkazy v interaktívnom interpreteri tak, ako sa priebežne zadávajú z klávesnice alebo spúšťajú z iného jazyka. V prípade, že sa interpreter zabuduje do softwarového systému, hovorí sa o vkladaní (*embedding*). Vloženie interpreteru prináša viacero výhod – možnosť upravovať alebo pridávať funkcionalitu bez nutnosti dodávať alebo kompilovať zdrojové kódy. Hodí sa aj na automatizáciu niektorých úloh: namiesto ručného vykonávania série úkonov sa skriptom proces až mnohonásobne zrýchli a zmenší sa riziko vnesenia chyby.

Napojenie knižnice na iný jazyk môže byť na úrovni medziprocesovej komunikácie, kedy software v rôznych jazykoch komunikuje pomocou soketov alebo iných prostriedkov. Takto je síce možné prepojiť software v prakticky ktoromkoľvek jazyku, táto technika však zďaleka nie je optimálna.

Efektívnejšie sa dajú jazyky napojiť v prípade, že jeden z nich má rozhranie na druhý jazyk. Vysokoúrovňové jazyky obvykle obsahujú aplikačné rozhranie (API) na spojenie s jazykmi C/C++. Vďaka týmto rozhraniam môžu byť relatívne jednoducho rozšírené o nové funkcie alebo použité ako skriptovací jazyk.

Pre vyššie uvedené výhody sa začalo aj v rámci vývojárskeho tímu Quantum GIS uvažovať o možnosti podpory pre nejaký skriptovací jazyk. V prípade Quantum GIS bolo dôvodov niekoľko:

- možnosť vytvárať prídavné moduly (*pluginy*) pre QGIS.
- možnosť vytvárať samostatné programy využívajúce knižnice QGIS.
- možnosť naskriptovania niektorých úloh v programe, umožňujúc pohodlnejšiu a rýchlejšiu prácu.

Vytváranie prídavných modulov a samostatných aplikácií založených na knižiaciach QGIS fungovalo už predtým za použitia jazyka C++, no použitie C++ nedávalo takú flexibilitu.

3.2 Výber jazyka

Výber jazyka, ktorý sa použije na napojenie, je kľúčový. Každý jazyk má svoje špecifiká a od toho sa odvíja aj fakt, že pre každý jazyk sa napojenie na C++ rieši iným spôsobom.

Do úvahy pripadali jazyky Python a Ruby, iné jazyky v sfére skriptovacích jazykov nemajú takú veľkú obľubu medzi programátormi. Tieto dva jazyky majú mnoho spoločných konceptov a na internete možno nájsť množstvo stránok, ktoré sa ich snažia podľa rôznych kritérií porovnať. Záverečné hodnotenia sa rôznia podľa toho, akým detailom autor prikladá väčšiu váhu. Faktom ostáva, že obidva jazyky sú kvalitné a na tento účel sa výborne hodia.

Nakoniec sme sa pre Quantum GIS rozhodli použiť jazyk Python, keďže tento jazyk je medzi vývojármi Quantum GIS najpopulárnejší.

3.3 Nástroje

Možností, ako vytvoriť napojenie na jazyk Python [4] je viacero. Líšia sa rôznou mierou flexibility a jednoduchosti použitia. Nedá sa jednoznačne určiť, že ktorá možnosť je lepšia a ktorá horšia, výber záleží od požiadaviek aplikácie. V nasledujúcich sekciách uvádzam prehľad dostupných nástrojov.

Python API

Python obsahuje API [5], ktoré dáva prístup k interpreteru na viacerých úrovniach. Rozširenie jazyka je viacmenej priamočiary proces, vloženie Pythonu do aplikácií je o niečo komplikovanejšie. Väčšina funkcií API je použiteľná pri rozširovaní aj vkladaní Pythonu.

Základným prvkom je typ `PyObject*`, ktorý mnoho funkcií berie ako parameter alebo vracia. Ide o pointer na ľubovoľný objekt v Pythone, samotný datový typ `PyObject` ostáva nepriehľadný. Takmer všetky objekty v Pythone existujú v halde, jedinou výnimkou sú typové objekty (`PyTypeObject`), ktoré sú obyčajne statické objekty. To znamená, že pri použití API sa objekty nikdy nedeklarujú ako automatické alebo statické premenné.

Každý objekt má typ a počet referencií. Pre všetky známe typy objektov existuje makro, ktoré overí, či je inštanciou tohto typu.

Počítanie referencií

Počítanie referencií je spôsob, ktorý Python používa na správu pamäte. Vďaka počítadlu vieme o každom objekte povedať, na koľkých rôznych miestach je naň odkaz. Keď sa počítadlo dekrementuje na nulu, nikto sa už na objekt neodkazuje a môže sa zmazať. Spolu s vymazaním sa taktiež zníži o jedna počet referencií na všetky objekty, na ktoré sa pôvodný objekt odkazoval. Manipulácia s počítadlom referencií sa robí vždy explicitne pomocou makier `Py_INCREF` a `Py_DECREF`. Inkrementácia je jednoduchá operácia, pri dekrementácii je však nutná ďalšia réžia, ktorá sa stará o deštrukciu objektu v prípade, že už neexistujú ďalšie referencie.

Nie je ale nutné zvyšovať počet referencií pre každý pointer, ktorý na objekt ukazuje. Ak existuje aspoň jedna referencia na objekt, nie je potrebné pre lokálny pointer zvyšovať a znovu znižovať počet referencií objektu, keďže sa ich efekt ruší. Chybou ale je, keď sa pointer uloží bez zvýšenia počítadla na dlhší čas, počas ktorého môže dôjsť k zníženiu počtu referencií a dealokácii objektu.

V Pythone neexistuje vlastníctvo objektov – všetky objekty sú zdieľané. Namiesto toho hovoríme o vlastníctve referencií. Vlastniť referenciu znamená byť zodpovedný za volanie `Py_DECREF` v prípade, že sa skončila práca s objektom. Ak funkcia predáva vlastníctvo referencie volajúcemu, hovorí sa o novej referencii. Ak k predaniu vlastníctva nedôjde, hovoríme o požičanej referencii. Keď volaná funkcia preberá vlastníctvo, označuje sa to ako ukradnutie referencie.

Príklad konštrukcie dvojice `(1, "dva")`:

```
PyObject* t = PyTuple_New(2);
PyTuple_SetItem(t, 0, PyInt_FromLong(1L));
PyTuple_SetItem(t, 1, PyString_FromString("dva"));
```

Funkcie `PyTuple_New`, `PyInt_FromLong` a `PyString_FromString` vracajú nové referencie, funkcia `PyTuple_SetItem` kradne referenciu objektu ktorý sa do n-tice vkladá. Na konci vlastníme iba referenciu na požadovanú dvojicu.

Vytvorenie extenzií

Vytvorenie jednoduchého modulu pre Python [6] je vhodné ukázať na príklade. Nasledujúci kód ukazuje, ako vytvoriť modul s jednou funkciu, ktorá očakáva dva číselné argumenty, ktoré sčíta:

```
#include <Python.h>

static PyObject* simple_func(PyObject *self, PyObject *args)
{
  PyObject *a, *b;
  if (!PyArg_UnpackTuple(args, "func", 2, 2, &a, &b))
    return NULL;
  return PyNumber_Add(a, b);
}

static PyMethodDef simple_methods[] = {
  {"func",simple_func,METH_VARARGS,"Return sum of a and b."},
  {NULL, NULL}
};

PyMODINIT_FUNC initsimple(void)
{
  Py_InitModule3("simple", simple_methods,
          "This module is just a simple example module.");
}
```

Modul sa skompiluje pomocou pythonovského nástroja `distutils`. Výsledkom je zdieľaná knižnica simple.so (resp. simple.pyd na Windows). Keď sa z Pythonu zavolá príkaz `import simple`, načíta sa táto knižnica a zavolá sa jej inicializačná funkcia `initsimple`. Funkcia `Py_InitModule3` prijíma ako parametre názov modulu, pole metód modulu a dokumentačný reťazec. Metódy sa definujú štruktúrou `PyMethodDef`, v ktorej sa určí názov v Pythone, pointer na implementáciu v C, konvencia volania (najčastejšie `METH_VARARGS`) a dokumentačný reťazec. Pole definícií metód sa ukončuje prázdnou štruktúrou.

Veľká časť funkcií volaných z Pythonu vyzerá nasledovne:

```
static PyObject* function(PyObject* self, PyObject* args)
{ /* ... */ }
```

Pre názvy funkcií sa používa konvencia `<modul>_<funkcia>`. V argumente `self` sa predáva inštancia triedy alebo `NULL` ak je to metóda modulu alebo nie je volaná na žiadnej inštancii, v `args` sa nachádzajú všetky parametre uložené v n-tici.

Telá implementácií metód majú obvykle podobnú formu a skladáju sa z troch častí:

1. prevod parametrov z objektov Pythonu na ekvivalenty v C,
2. výpočet,
3. prevod výsledku z C do Pythonu.

Python API umožňuje taktiež vytvárať nové typy objektov. Sú reprezentované štruktúrou `PyTypeObject`, ktorá obsahuje meno typu, veľkosť objektu, príznaky, dokumentačný reťazec, pole metód (štruktúry `PyMethodDef`) a pole dát (štruktúry `PyMemberDef`). Okrem toho obsahuje pointre na funkcie, ktoré odkazujú na implementácie špeciálnych operácií ako inicializácia, prevod na iný typ, iterácia po objekte, porovnanie s iným objektom a ďalšie.

Zhrnutie

Vytvárať napojenie knižnice na Python pomocou týchto prostriedkov je väčšinou príliš pracné vzhľadom na potencionálne veľké množstvo tried a funkcií, ktoré treba napojiť. Ďalším problémom môže byť zanesenie chýb pri prevodoch objektov medzi Pythonom a C/C++, hlavne nesprávne počítanie referencií môže byť zdrojom únikov pamäti (ak sa objekt neuvoľní) alebo spôsobovať predčasné uvoľnenie pamäti – ešte horšia možnosť, kedy môže dochádzať k dosť náhodným chybám.

Keďže napojenie tried a funkcií hotovej knižnice na Python vyžaduje značné množstvo opakujúceho sa kódu (pri prevode objektov medzi Pythonom a C/C++), je logické zaoberať sa možnou automatizáciou tohoto procesu. Časom vzniklo pre Python viacero nástrojov, ktoré vytvorenie napojenia značne zjednodušujú. Je vhodné použiť tieto nástroje namiesto priameho použitia Python API, po zoznámení s nimi je rýchlosť výroby napojení mnohokrát vyššia.

SWIG

Simplified Wrapper and Interface Generator (SWIG) je nástroj umožňujúci napojiť knižnice v C a C++ na radu vyšších programovacích jazykov – okrem Pythonu podporuje napríklad aj Perl, PHP, Tcl a Ruby. Jedná sa o najpopulárnejší nástroj tohto typu vďaka rozsiahlej dokumentácii, dlhej

histórii (vznikol v r. 1995) a množstvu podporovaných jazykov. Jeho snahou je vytvárať wrappery s minimálnym úsilím uživateľa.

SWIG ako svoj vstup používa tzv. *interface* súbory (obvykle s príponou `.i` alebo `.swg`), ktoré definujú rozhranie pre napojenie. Tieto súbory obsahujú deklarácie v C/C++ podľa normy ANSI, SWIG implementuje (takmer kompletný) parser C a C++, vie pracovať aj s makrami preprocesoru. Súbory môžu navyše obsahovať špeciálne direktívy pre SWIG. Podľa interface súboru sa vygeneruje zdrojový kód wrapperu pre určený jazyk. Kompiláciou dostaneme modul daného jazyka – tým je napojenie hotové.

Uveďme si príklad interface súboru, na ktorom popíšem základy použitia SWIG:

```
%module mymodule
%{
#include "myheader.h"
%}

int foo;
int bar(int x);
```

Všetky direktívy pre SWIG začínajú znakom percenta. Direktíva `%module` určuje meno výsledného modulu, táto direktíva sa musí nachádzať na začiatku každého interface súboru. Všetko, čo sa nachádza medzi značkami %{ a %} je bez zmeny kopírované do výsledného wrapperu – SWIG nijak neparsuje ani neinterpretuje túto oblasť.

Nasledujúcim príkazom sa vygeneruje wrapper pre jazyk Python:

```
swig -python example.i
```

V prípade, že ide o C++ kód, musí sa použiť pri sputení SWIG ešte parameter `-c++`.

Generovanie kódu wrappera je dosť priamočiare pre jednoduché číselné typy. Pri reťazcoch a poliach sa situácia komplikuje, pretože nie je možné bez znalosti sémantiky rozhodnúť o ukazateli, na aký typ v inom jazyku by mal byť prevedený.

SIP

Tento nástroj sa začal vyvíjať v roku 1998 za účelom vytvorenia napojenia knižnice Qt na Python (PyQt), je však dosť univerzálny a umožňuje generovať pythonovské wrappery pre akékoľvek knižnice v C a C++ [7]. Je menej

rozšírený ako SWIG ale svoju úlohu zvláda dobre. Tým, že je určený len pre Python, snaží sa integrovať s týmto jazykom čo najtesnejšie.

SIP vychádza z podobného vstupu ako SWIG. Má zjednodušený parser C/C++, ktorý má navyše svoje špecifiká. Ide predovšetkým o podporu direktív, ktoré podobne ako SWIGu začínajú znakom percenta na začiatku riadku. Ďalším špecifikom sú anotácie, ktoré upravujú chovanie generátora kódu. Vstupné súbory majú obyčajne príponu `.sip` a na prvý pohľad pripomínajú bežné hlavičkové súbory. SIP nepodporuje preprocesor jazyka C, na podmienené generovanie má však svoje vlastné direktívy.

Uveďme príklad vstupného súboru pre SIP:

```
%Module word 0

class Word {

%TypeHeaderCode
#include <word.h>
%End

public:
    Word(const char *w);
    char *reverse() const;
};
```

Direktíva `%Module` definuje názov výsledného modulu a jeho verziu. Nasleduje deklarácia triedy `Word`, ktorú chceme použiť v Pythone. Do jej deklarácie sa pridali direktívy `%TypeHeaderCode` a `%End` – kód medzi nimi sa bez zmeny vloží do generovaného súboru pre túto triedu. Väčšinou sa na tomto mieste iba vkladá hlavičkový súbor triedy.

Wrapper sa môže vygenerovať a skompilovať ručne (ako u SWIG), dá sa však využiť kompilovací systém SIPu. V Pythone sa napíše jednoduchý skript, ktorý určí závislosti a systém sa postará o generovanie a správnu kompiláciu. Výhodou je, že táto metóda je nezávislá na platforme a kompilátore, takže odpadajú prípadné problémy s portovaním na iné systémy.

Boost.Python

Boost je známa sada C++ knižníc poskytujúca množstvo užitočného kódu, niektoré jej časti sa časom dostávajú do ďalších vydaní štandartu C++.

Súčasťou tejto sady je aj knižnica Boost.Python, ktorej funkciou je integrácia medzi jazykmi C++ a Pythonu.

Uveďme príklad vytvorenia wrapperu pre triedu `World` s členskými funkciami `greet` a `set`:

```
#include <boost/python.hpp>
using namespace boost::python;

BOOST_PYTHON_MODULE(hello)
{
  class_<World>("World")
    .def("greet", &World::greet)
    .def("set", &World::set)
  ;
}
```

Ako je vidieť, vyššie uvedený kód je celý napísaný v jazyku C++. Používa sa tu technika nazývaná *metaprogramovanie*, kde sa pomocou šablón a makier preprocesoru generuje zdrojový kód. V tomto prípade sa generuje kód v API jazyka Python, takže po skompilovaní dostaneme požadovaný modul pre Python.

3.4 Quantum GIS a Python

Výber generátora

Podľa prehladu dostupných nástrojov sa vďaka svojej rozšírenosti a univerzálnosti zdá byť najvhodnejší kandidát SWIG. Quantum GIS ale v celom kóde používa triedy knižnice Qt, takže aj jeho API využíva triedy z Qt. Pri použití SWIG by bolo nutné vymyslieť nejaký dostatočne kvalitný mechanizmus na prepojenie medzi PyQt a knižnicami Quantum GIS. Ďalšie problémy vznikajú pri použití mechanizmu signálov a slotov, ktorý je špecifický pre knižnicu Qt.

SIP všetky tieto problémy rieši, preto ako omnoho priamočiarejšie riešenie vyšlo použitie tohto nástroja.

Implementácia

Na implementáciu som sa podujal hlavne z dôvodu prehĺbenia znalosti tohto jazyka a taktiež štúdia jeho fungovania. Po tom, ako človek pochopí špecifiká prepojenia medzi C++ a Pythonu, je vytvorenie napojenia pomocou nástroja SIP viac-menej rutinná záležitosť. Je nutné dávať pozor na správne označenie argumentov a návratových hodnôt, kde dochádza k zmene vlastníctva objektu. V takom prípade sa pre funkcie a argumenty používajú anotácie: `Transfer`, `TransferThis`, `TransferBack` a `Factory`.

V niektorých prípadoch bolo nutné napísať explicitne zdrojový kód na prevod medzi šablónovými triedami pomocou Python API a funkciami knižnice SIP. Podobné prevody je možné nájsť aj v PyQt4, takže ich implementácia nebola príliš náročná.

Drobnou vadou na kráse je nutnosť upravovať SIP súbory pre triedy pri každej zmene rozhrania triedy. Bolo by dobré mať k dispozícii nástroj, ktorý by túto činnosť automatizoval.

V druhej fázi som podporu Pythonu zabudoval do aplikácie Quantum GIS. Vytvoril som jednoduchú konzolu, v ktorej možno spúšťať príkazy v Pythone a pracovať s danou inštanciu progamu. Pridal som aj podporu pre pythonovské prídavné moduly.

Zhodnotenie

Zo začiatku pri diskusii s ostatnými programátormi Quantum GIS som neprikladal napojeniu na Python veľkú váhu. Až časom som si uvedomil, že pri použití Pythonu na vývoj prídavného modulu pre Quantum GIS stačí napísať menej kódu a odpadajú rôzne obtiaže s kompilovaním. Vytvorenie napojenia na Python sa ukázalo byť správnym krokom a v komunite Quantum GIS začali vznikať zaujímavé projekty [8], či už prídavné moduly alebo samostatné aplikácie. Ukazuje sa, že počet ľudí ochotných programovať v Pythone je o dosť vyšší ako počet tých, ktorí vedia programovať v C++.

Výborné využitie som našiel aj pri jednoduchých nástrojoch na úpravu dát alebo pri overovaní chýb. Často je ladenie chyby v aplikácii Quantum GIS časovo náročné, pretože je potrebné pred vyvolaním chyby nutné spraviť niekoľko akcií, ktoré k chybe vedú. V prípade, že problém sa netýka iba uživateľského rozhrania, krátkym skriptom v Pythone sa dá chyba rýchlo vyvolať. Vďaka tomu sa skráti aj čas na opravenie chyby.

Kapitola 4

Lokalizácia pomocou GPS

4.1 Úvod

GPS je anglická skratka pre **Global Positioning System** [9], čo je satelitný systém zameraný na zisťovanie presnej polohy na Zemi. Tento systém bol vyvinutý a je vlastníctvom ministerstva obrany USA. Vznikol pre potreby americkej armády, je však bezplatne dostupný aj pre civilné použitie. Využitie má predovšetkým v navigácii, mapovaní, geodézii a vedeckých výskumoch.

Segmenty

Celý systém sa delí na tri segmenty:

- **vesmírny** – tvoria ho satelity na obežnej dráhe. Návrh systému predpokladá rovnomerné rozloženie 24 satelitov v šiestich rôznych obežných rovinách. Počet satelitov môže byť aj vyšší vzhľadom na ich priebežnú výmenu.

  Obežná dráha a výška sú nastavené tak, že satelity sa dostanú na rovnaké miesto nad Zemou každých 12 hodín a obiehajú po rovnakej dráhe. Existuje šesť obežných rovín, každá pre štyri satelity. Pri tejto konštelácii je zabezpečené, že z každého miesta na Zemi je viditeľných aspoň päť satelitov.

  Momentálne sa na obežnej dráhe nachádza až 30 aktívne vysielajúcich družíc. Vďaka tejto redundancii je možné chytiť signál z väčšieho počtu z nich a tým vylepšiť presnosť pozície.

- **riadiaci** – pozostáva z niekoľkých monitorovacích stredísk americkej armády rozmiestnených na rôznych miestach Zeme: dve v Tichom oceáne, po jednom v Atlantickom oceáne, Indickom oceáne a v Colorade, kde je aj hlavné ovládacie stredisko. Na základe dát zo satelitov sa robí korekcia ich hodín (na presnosť pod jednu mikrosekundu) a upravujú sa ich efemeridy.

- **uživateľský** – tvoria ho GPS príjimače.

Signály zo satelitov

Satelity sytému GPS vysielajú tri typy signálu:

- navigačná správa – obsahuje hrubé informácie o čase, stave družíc a informácie o ich obežných dráhach. Moduluje sa na frekvenciu L1 (1575.42 MHz).

- C/A kód – je to 1023 bitov dlhý pseudonáhodný kód. Satelit majú tento kód rôzny, a preto sa môžu pomocou neho jednoznačne identifikovať. Je voľne dostupný na frekvencii L1.

- P(Y) kód – presný kód, rezervovaný pre armádne účely. Kód je šifrovaný, takže je dostupný len pre špeciálne prijímače. Moduluje sa na frekvencie L1 a L2 (1227.60 MHz).

Výpočet polohy

GPS prijímač počíta pozíciu tak, že meria vzdialenosť medzi sebou a družicami v dosahu. Na vyhodnotenie pozície je nutný signál aspoň z troch satelitov. S rastúcim počtom satelitov je vyhodnotenie polohy presnejšie. Na kvalitu vyhodnotenej pozície ma vplyv aj momentálne rozmiestnenie satelitov na oblohe. Na výpočet 3D pozície je nutné použiť signál minimálne zo štyroch satelitov.

Vzdialenosť príjimača od satelitu sa vyhodnotí meraním časového posunu medzi odoslaním a prijatím signálu zo satelitu, pretože signál sa prenáša známou konštantnou rýchlosťou. Táto približná vzdialenosť sa označuje ako *pseudorange*. Poloha prijímača sa zo vzdialeností vypočíta trilateráciou. Vypočítané súradnice sa vzťahujú k súradnicovému systému WGS84.

S využitím voľne dostupného C/A kódu je možné získavať polohu s chybou približne 10 metrov, čo je pre bežné použitie dostačujúce. Chyba vzniká

rôznymi atmosférickými efektami, ale taktiež nepresnosťou vysielaného signálu a prijímačov.

Ak chceme súradnice nejakého miesta určiť presnejšie, stačí zostať na mieste dlhší čas. Vypočítaná pozícia bude okolo tej presnej oscilovať, takže pomocou štatistiky získame o niečo presnejšiu polohu.

Diferenčné GPS

V geodézii ale štandartná chyba civilných GPS prijímačov nie je postačujúca. Vyvinula sa metóda diferenčného GPS, ktorá umožňuje zvýšenie presnosti určenia pozície – výsledná chyba klesá pod 1 meter. Princíp fungovania je založený na fakte, že prijímače prijímajú signály s rovnakými nepresnosťami. Význam to má ale len v prípade, že nie sú od seba moc vzdialené, maximálne niekoľko desiatok kilometrov. Ak poznáme presnú pozíciu jedného z prijímačov (napr. je to pevná stanica), je možné vyhodnotiť chybu prijatého signálu a urobiť korekciu pozície druhého prijímača (v teréne).

GPS prijímač v teréne môže prijaté signály zaznamenávať a pozícia sa vyhodnotí až spätne pomocou korekčných dát. Druhou možnosťou je spresnené vyhodnotenie v reálnom čase, kedy sa korekčné dáta prenášajú pozemnými vysielačmi alebo po Internete.

Aplikácie

Systém GPS si našiel postupom času množstvo aplikácií. Pochopiteľné uplatnenie má predovšetkým v oblasti leteckej, lodnej či automobilovej navigácie. Pomáha v orientácii v neznámom teréne pri turistike alebo expedíciach. Pri spojení s ďalšími technológiami prináša však ešte ďalšie možnosti.

Ak sa počas pohybu v teréne priebežne ukladá aktuálna pozícia, je možné spätne vyhodnocovať štatistické údaje o rýchlosti, zastávkach, profilu trasy a podobne. Zaznamenaná trasa sa môže zobraziť na mape v počítači alebo nahrať do iného GPS prijímača, ktorý podľa nej môže navigovať. U služobných vozidiel príde vhod možnosť automaticky vytvárať knihu jázd podľa prejazdenej trasy.

Použitím vysielačky alebo mobilného telefónu sa dá zabezpečiť online sledovanie. Niektoré spoločnosti ponúkajú túto službu pre možnosť dohľadania ukradnutého vozidla. Používa sa aj v špedičných firmách alebo na monitorovnie sanitiek, hasičských a policajných áut.

Prijímače

GPS prijímače pre bežných uživateľov sa dajú rozdeliť na niekoľko druhov:

- automobilové
- turistické
- námorné
- letecké

Tieto prijímače (a navigátory) obyčajne majú displej, ich software a ovládanie je uspôsobené na použitie v danej sfére. Väčšina z nich podporuje nahrávanie digitálnych máp a navigáciu podľa nich. Časť z nich má komunikačné rozhranie na prepojenie s inými zariadeniami.

Okrem toho existuje trieda prijímačov bez displeja a ovládania, nazývajú sa *OEM moduly* a poskytujú iba komunikačné rozhranie.

Budúcnosť

GPS systém dosiahol plnej funkčnosti v polovici roku 1995. Napriek tomu sa však naďalej pracuje na vylepšeniach systému pre dosiahnutie vyššej presnosti a dostupnosti. Tento projekt sa označuje ako GPS III a mal by byť plne funkčný v roku 2013.

Okrem toho sa vyvíja pod záštitou Európskej únie a ďalších krajín systém Galileo, ktorý má byť dostupný aj v prípade, že by americká vláda zablokovala systém GPS. Okrem toho sa projekt snaží dosiahnuť vyššej presnosti určenia pozície. Podľa plánov by mal byť funkčný v roku 2011 alebo 2012.

4.2 Komunikačné rozhrania

Rozhrania GPS prijímačov na komunikáciu s inými zariadeniami sa rôznia. Klasickým spôsobom je prenos dát po sériovej linke (RS-232), u novších prijímačov sa používa USB rozhranie alebo beztrátové spojenie cez Bluetooth. Nás však zaujímajú hlavne aplikačné protokoly. Tu dominuje protokol NMEA 0183, existuje aj množstvo proprietárnych protokolov.

NMEA 0183

Ide o jednoduchý textový protokol, ktorý je súčasťou špecifikácie americkou asociáciou NMEA [10]. Špecifikácia nie je voľne dostupná, napriek tomu sa na Internete nachádzajú popisy tohto protokolu v zjednodušenej forme [11] potrebnej na čítanie dát z GPS. Protokol je totiž navrhnutý pre komunikáciu širokej škály námorných zariadení. Definuje prenos dát medzi vysielačom (v našom prípade GPS) a jedným alebo viacerými prijímačmi (napr. počítač).

Prenos informácií je rozdelený do viet, každá veta má svoj identifikátor, vďaka ktorému sa rozpozná význam hodnôt v tele vety. Veta začína znakom doláru (`$`), a končí znakom hviezdičky, dvoma znakmi pre kontrolný súčet a novým riadkom (znaky `<CR><LF>`). Kontrolný súčet je hexadecimálne číslo, ktorého hodnota sa vypočíta postupným XOR-ovaním ASCII hodnoty znakov medzi dolárom na začiatku vety a hviezdičkou. Ak hodnota nesedí, veta je neplatná.

Informácie vo vete sú oddelené znakom čiarky, prvý údaj pozostáva vždy z piatich veľkých písmen, ktoré určujú typ vety. Ostatné údaje vety obsahujú čísla alebo písmená. Uveďme krátku ukážku:

```
$GPGLL,4729.3621,N,00857.3906,E,072755.090,A*37
$GPVTG,0.000000,T,,M,0.082667,N,0.153097,K*54
$GPGSA,A,3,,,,,,,,,,,,1.9,1.1,,*38
```

Obvykle dochádza každú sekundu k odoslaniu sady viet s vyhodnotenou pozíciou, výškou, časom, polohou družíc a ďalšími informáciami. Sada viet sa u rôznych GPS líši, u niektorých je možné sadu aj interval posielania nastaviť, u iných je sada príliš malá a niektoré informácie sa vôbec neposielajú. Vety sú často redundantné, takže jedna informácia sa môže vyskytovať v sade viet viackrát. Asi každá GPS štandartne posiela vety `GPRMC` a `GPGGA`, ktoré obsahujú kompletnú informáciu o čase a pozícii. Práve tieto vety sa používajú v programoch na určenie pozície.

Komunikácia v tomto protokole je možná aj v opačnom smere, nedefinuje však mnoho operácií, ktoré moderné GPS navigátory potrebujú (sťahovanie a nahrávanie bodov, trás a podobne). GPS zariadenia častokrát používajú proprietárne rozšírenia protokolu NMEA na niektoré nastavenia.

Takmer všetky GPS prijímače vybavené komunikačným rozhraním dokážu posielať data v tomto protokole, preto je najvhodnejší kandidát na použitie pre zisťovanie polohy z GPS.

Proprietárne protokoly

Spločnosti do svojich GPS navigátorov implementujú aj proprietárne protokoly, ktoré ponúkajú širšie možnosti komunikácie so zariadením, nie sú však univerzálne.

Dobre popísaný je napríklad binárny protokol firmy GARMIN, ktorá patrí medzi hlavných dodávateľov GPS navigátorov. Okrem zdokumentovaného rozhrania na zisťovanie polohy, prenosu bodov a trás obsahuje aj niektoré nedokumentované činnosti ako nahrávanie máp alebo posielanie nameraných dát zo satelitov.

gpsd

Táto UNIX-ová služba [12] vznikla s cieľom ponúkať rovnaké rozhranie pre rôzne typy GPS. Dokáže komunikovať cez sériový port alebo USB a okrem NMEA 0183 dokáže používať aj niektoré proprietárne protokoly. Ku *gpsd* sa dá pristupovať priamo čítaním dát z TCP portu 2947 alebo pomocou dodávanej knižnice pre jazyk C, ktorá ponúka o niečo komfortnejšiu prácu.

Kapitola 5

Navigácia

Táto kapitola je venovaná úvodu vyhľadávania ciest. Popisujú sa hlavne nároky kladené na kvalitu spracovania cestnej siete a bežne používané algoritmy na vyhľadávanie. Lodná, letecká, turistická alebo iná navigácia sa vzhľadom na odlišné princípy nepreberá.

Pri vyhľadávaní sa používajú rôzne kritériá, nemusí ísť práve o najkratšiu cestu. Tá totiž často nie je optimálna, pretože môžu existovať spojenia po iných, rýchlejších cestách, ktoré sú o niečo dlhšie. Pri bežnom použití má uživateľ najčastejšie záujem o najrýchlejšiu cestu. Niekedy sa používajú aj iné kritériá ako napríklad optimalizácia cesty na minimálnu spotrebu pohonných hmôt.

5.1 Vstupné dáta

Dáta o cestnej sieti sa bežne reprezentujú ako vektorová mapa. Jednotlivé úseky ciest a ulíc tvoria samostatné prvky mapy. Geometria prvku určuje tvar cesty, atribúty poskytujú dodatočné informácie.

Vyhľadávanie trasy v cestnej sieti má viacero špecifík, preto sa očakávajú metadáta, ktoré skvalitňujú vyhľadanie trasy a navigovanie po nej. Nižšie uvádzam najdôležitejšie z nich:

Typ cesty

Bežne rozlišujeme viacero kategórií ciest: diaľnice, cesty prvej, druhej a tretej triedy a lokálne cesty. Toto delenie samozrejme môže byť aj jemnejšie. Určenie typu cesty môže značne pomôcť vyhľadávaniu. Ak sa hľadá cesta na

dlhšiu vzdialenosť, je logické používať cesty vyššej kategórie, pretože rýchlosť prejazdu cez ne je obvykle vyššia.

Rozdelenie typov ciest navyše umožňuje optimalizácie rýchlosti vyhľadania spojenia: algoritmus pri vstupe na diaľnicu môže prestať uvažovať lokálne cesty, ktoré z diaľnice odbočujú, a vyhľadávať cestu len po ďalších diaľniciach. Akonáhle sa užívateľ blíži k cieľu, povolia sa aj nižšie typy cesty. Takto sa ušetrí návšteva množstva vrcholov grafu. Cesty jednej kategórie by mali tvoriť súvislú sieť z ciest svojej kategórie a vyšších kategórií. Ináč by sa mohlo stať, že takto optimalizovaný algoritmus sa nevyužije optimálne.

Zákazy odbočenia

Okrem zaznačenia bežných zákazov odbočenia podľa dopravných značiek je často nutné označiť ďalšie zakázané manévre, hlavne pri komplikovanejších križovatkách. Na niektorých úsekoch nie je povolené otočenie auta do protismeru.

Jednosmerky

Cesty s jazdou povolenou iba jedným smerom sú spolu so zákazmi odbočenia hlavné obmedzujúce prvky pri vyhľadávaní spojenia.

Maximálna rýchlosť

Zatiaľ čo existuje kategorizácia maximálnej povolenej rýchlosti pre jazdu na diaľnici, v obci a mimo obce, niektoré úseky ciest majú maximálnu rýchlosť zvýšenú alebo zníženú.

Cesty s poplatkom

Vodič niekedy nechce použiť cesty, ktoré vyžadujú poplatky – napríklad diaľnice alebo tunely. Ak sú v dátach označené spoplatnené úseky, je možné sa im pri hľadaní cesty vyhnúť.

Obmedzenia na typ vozidla

Na niektoré úseky ciest je povolený vjazd len určitej skupine vozidiel. V navigačnom systéme býva možné vybrať si typ použitého vozidla (osobné auto, kamión, taxi, zásobovanie, záchranné vozidlo, bicykel ...), na spojenie sa budú používať iba cesty, kde je vjazd daného vozidla povolený.

Označenie ciest

Cesty by mali byť označené názvom ulice ak úsek má nejaký názov, prípadne číslom cesty (cesta môže mať aj viac číselných označení). Pre navigáciu sú užitočné ešte ďalšie informácie, napríklad označenie kruhového objazdu alebo nájazdov a výjazdov z diaľnice. Umožní to vodičovi rýchlejšie sa zorientovať v dopravnej situácii a tak aj zníži riziko nesprávneho odbočenia.

Geokódovanie

Termín geokódovanie (*geocoding*) označuje napojenie adresy alebo inej informácie na geografické súradnice. Geokódovací program dokáže z uživateľom zadanej adresy podľa svojich dát určiť jej polohu na mape.

Pri cestných sieťach je možné pre každý úsek ulice označiť rozsah uličných čísel pre ľavú a pravú stranu. Podľa rozsahu sa interpoláciou v segmente určuje poloha. Aj keď určenie polohy nemusí byť úplne presné, zvyčajne postačuje na navigáciu do správneho úseku ulice.

5.2 Reprezentácia cestnej siete

Cestná sieť sa skladá z uzlov (križovatky a koncové body) a ciest medzi uzlami. Transformovaním cestnej siete na graf sa problém prevedie na všeobecnejší problém hľadania najkratšej cesty v grafe. Ten spočíva v hľadaní cesty medzi dvoma vrcholmi, ktorá má najmenší súčet ohodnotení použitých hrán.

Obvykle uvažujeme aj jednosmerné cesty, takže tento graf býva orientovaný. Aby sme sa vyhli nutnosti použiť multigraf, je potrebné zakázať viacnásobné hrany. Toto obmedzenie je minimálne: cestná sieť sa dá ľahko upraviť tak, aby viacnásobné hrany neobsahovala: ak medzi dvoma bodmi existujú dve rôzne cesty, jedna z nich sa bude viesť cez ďalší uzol. Obdobne ak cesta vytvára kruh (začína a končí v rovnakom bode), rozdelí sa na dva úseky.

Priamočiara reprezentácia cestnej siete ako graf je uzly siete rezprezentovať ako vrcholy, úseky ciest sa reprezentujú ako hrany grafu. Po takto vytvorenej sieti sa dajú vyhľadávať spojenia bežnými algoritmami. Vznikajú však obtiaže v prípade, že potrebujeme brať do úvahy aj zákazy odbočenia alebo preferenciu či penalizáciu prechodu medzi niektorými cestami. Na

tieto obmedzenia nie sú grafové algoritmy stavané a preto táto reprezentácia na komplikovanejšie cestné siete nie je vhodná.

Alternatívou je reprezentovať jednotlivé úseky cesty ako vrcholy. Hrana medzi dvomi cestami potom znamená možnosť odbočiť z jednej cesty na druhú. Táto reprezentácia je vhodná aj pri použití zákazov odbočenia, pretože zakázaný manéver znamená, že neexistuje hrana medzi danými dvomi cestami. Daňou je o niečo väčší graf: zatiaľ čo v prvej reprezentácii sa pre jeden uzol použije len jeden vrchol grafu, v aleternatívnej reprezentácii sa pre uzol použije jedna hrana pre každý možný manéver. Takže pri križovatke štyroch ciest bez zakázov odbočenia sa použije až 12 hrán.

Algoritmy pri výpočte trasy používajú ohodnotenie hrán grafu. Toto ohodnotenie môže byť dĺžka úseku pri hľadaní najkratšej cesty, čas potrebný na prejdenie úseku pri hľadaní najrýchlejšej cesty, alebo môže ísť o komplikovanejšie ohodnotenie, ktoré zohľadňuje viacero kritérií. Dijkstrov algoritmus aj A* potrebujú pre korektné chovanie nezáporné ohodnotenie, záporné hodnoty však pri cestnej sieti nemajú význam.

Ohodnotenia môžu byť pre graf vypočítané dopredu a algoritmus podľa požadovaného typu spojenia využije vhodné ohodnotenie. O niečo flexibilnejšie riešenie je počítať ohodnotenie až pri návšteve hrany – výpočet je obvykle jednoduchý a spomalenie vyhľadávania je nízke. Pri výpočte ohodnotenia sa dajú dokonca použiť aj informácie, ktoré dopredu neboli známe, napríklad zohľadniť doterajšiu nájdenú cestu.

5.3 Algoritmy

Na problém hľadania najkratšej existuje viacero známych algoritmov. Najznámejší je Dijkstrov algoritmus, pre náš účel je ešte o niečo vhodnejší algoritmus A* (A-star). Okrem nich sa dá použiť napríklad Bellman-Fordov algoritmus, ktorý dokáže pracovať aj so zápornými ohodnoteniami hrán. Má však vyššiu asymptotickú zložitosť a záporné ohodnotenia sa nepredpokladajú. Za zmienku stojí ešte Floyd-Warshallov algoritmus, ktorý nájde najkratšiu cestu pre všetky dvojice vrcholov, ale pre tento účel sa nehodí, pretože má veľké časové aj pamäťové nároky.

Ako bolo spomenuté už vyššie, hľadanie cesty nie je vždy minimalizované práve na najkratšiu cestu. Vzľahom na intuitívnosť však používam na popis algoritmov pojmy vzdialenosť pre ohodnotenie hrany a pojem najkratšia cesta pre minimalizovaný súčet ohodnotení.

Dijkstrov algoritmus

Algoritmus si pre každý vrchol uchováva ohodnotenie najkratšej cesty, ktorú dosiaľ našiel. Na začiatku je vzdialenosť pre začiatočný vrchol nulová, pre všetky ostatné vrcholy je nastavená na nekonečno, čo znamená, že zatiaľ nepoznáme k ostatným vrcholom žiadnu cestu. Na konci algoritmu je uložená hodnota rovná cene najkratšej cesty alebo nekonečnu ak neexistuje k danému vrcholu žiadna cesta z počiatočného bodu.

V algoritme rozlišujeme vrcholy na tri druhy:

- uzatvorené – k týmto vrcholom už je známa najkratšia cesta a už sa nemôže zlepšiť
- otvorené – bola nájdená cesta k týmto vrcholom, je ale možné, že časom nájdeme kratšiu cestu
- nenavštívené – tieto vrcholy ešte neboli dosiahnuté, takže k nim nie je známa žiadna cesta

Na začiatku je počiatočný vrchol označený ako otvorený, všetky ostatné sú nenavštívené. Pri každej iterácii algoritmu sa vezme otvorený vrchol s dosiaľ najkratšou vzdialenosťou a prehlási sa za uzatvorený. Pre všetky susedné vrcholy algoritmus overí, či cesta cez práve ozavretý vrchol nie je vhodnejšia ako tá predchádzajúca. Navyše predtým nenavštívené susedné vrcholy označia ako otvorené.

Algoritmus končí keď nie sú už žiadne otvorené vrcholy. Keďže nepotrebujeme najkratšiu cestu do všetkých vrcholov, môžeme ho ukončiť už vo chvíli, keď sa daný koncový bod cesty označí ako uzavrený.

A* algoritmus

Problémom Dijkstrovho algoritmu je fakt, že nijako nedokáže využiť fakt, že vieme odhadnúť, ktorým smerom by sa malo hľadanie cesty uberať. Pri cestnej sieti so stovkami tisíc bodov je žiadúce, aby sa prehľadávanie grafu zameralo na oblasti, kade by cesta mala ísť. Tu nájde uplatnenie práve A*.

Funguje na podobnom princípe ako Dijkstrov algoritmus, používa však navyše aj heuristický odhad vzdialenosti vrcholu od koncového bodu. Pri každej iterácii sa uzavrie vrchol, u ktorého je najnižší súčet dosiaľ prejdenej vzdialenosti a odhadnutej vzdialenosti.

Pri hľadaní najkratšej cesty môžeme použiť ako odhad vzdialenosti ku koncovému bodu vzdialenosť po vzdušnej čiare. Algoritmus bude teda prednostne vyšetrovať vrcholy, ktoré sú geograficky bližšie k cieľu.

Dijkstrov algoritmus je vlastne špeciálny prípad, kedy je odhad vzdialenosti od koncového bodu pre všetky vrcholy nulový. Ak je odhad vzdialenosti menší ako skutočná vzdialenosť, A* garantuje nájednie najkratšej cesty, navštívi ale väčšie množstvo vrcholov. V prípade úplne presného odhadu algoritmus pôjde presne po najlepšej ceste až do cieľa. Ak je niekedy odhad vyšší ako skutočná vzdialenosť, nie je garantované, že nájdená cesta bude najkratšia, počet navštívených vrcholov však klesá. V extrémnom prípade keď je odhadovaná vzdialenosť omnoho vyššia ako reálne vzdialenosti, výsledná cesta záleží viac-menej iba na odhade. Použitá heuristická funkcia teda umožňuje zvoliť vhodný kompromis medzi rýchlosťou vyhľadania cesty a kvalitou výsledku.

Kapitola 6

Implementácia

V tejto kapitole sa venujem spojeniu programu Quantum GIS a Pythonu s vyhľadávaním spojenia, sledovaním pozície z GPS a navigáciou do jednej aplikácie, ktorú som nazval Quantum Navigator. Tak ako Quantum GIS je uvoľnená pod licenciou GNU GPL, takže môže poslúžiť ako príklad použitia rôznych častí QGIS API v spojení s jazykom Python. Taktiež sa pomocou pripravenej infraštruktúry dajú skúmať ďalšie možné stratégie hľadania spojenia.

Program bolo možné implementovať dvoma spôsobmi: ako prídavný modul (*plugin*) pre Quantum GIS alebo ako samostatnú aplikáciu. Prídavné moduly sa integrujú do uživateľského rozhrania QGIS, dajú sa zapínať a vypínať pomocou správcu modulov. Obvykle sa pri spustení pluginu vytvoria nové položky v menu a v paneloch nástrojov, ktorými sa spúšťajú jednotlivé funkcie modulu. Keďže navigačný systém nepotrebuje väčšinu funkcií na prácu s geografickými dátami, zvolil som variantu samostatnej aplikácie. Uživateľ tak má k dispozícii jednoduchšie a prehľadnejšie rozhranie.

Celý projekt pozostáva z troch častí, sú rozoberané v nasledujúcich sekciách.

6.1 Knižnica na vyhľadávanie spojenia

Napriek tomu, že samotná aplikácia je napísaná v jazyku Python, implementáciu grafu a vyhľadávania v ňom som sa rozhodol urobiť v C++, pretože v Pythone by celé vyhľadávanie trvalo o dosť dlhšie vzhľadom na to že sa pri jednom vyhľadaní spojenia očakávajú rádovo desaťtisíce iterácií algoritmu.

Na vyhľadávanie cesty som sa rozhodol použiť osvedčený Dijkstrov al-

goritmus, s tým, že ak by vo väčšej sieti potreboval príliš mnoho času, využijem druhý popisovný algoritmus A*. Na testovacích dátach (približne 20 tisíc cestných úsekov) však Dijkstrov algoritmus dával výsledky prakticky okamžite, preto som nakoniec algoritmus A* neimplementoval.

Ako základ implementácie som využil voľne dostupné zdrojové kódy z repozitára algoritmov University of Cantenbury [14]. Graf je reprezentovaný tak, že pre každý vrchol existujú dva spojové zoznamy: jeden pre hrany ktoré z vrcholu vychádzajú, druhý pre vchádzajúce hrany. Táto reprezentácia je pri použití Dijkstrovho algoritmu najvhodnejšia vzhľadom na rýchle prechádzanie zoznamu susedných vrcholov. Oproti tomu pri matici susednosti prejdenie všetkých susedných vrcholov vyžaduje počet krokov rovný počtu vrcholov a zaberá omnoho viac miesta, reprezentácia hrán ako dvojíc vrcholov je úplne nevhodná vzhľadom na nutnosť prejsť celý zoznam hrán.

Samotná implementácia Dijkstrovho algoritmu na ukladanie množiny otvorených vrcholov používa Fibonacciho haldu, ktorá vyberie minimálny prvok amortizovane v logaritmickom čase, pridanie nového prvku a operácia *decrease key* sú amortizovane konštantné. Ak si počet vrcholov označíme N a počet hrán M, na beh algoritmu potrebujeme N-krát získať a odstrániť najmenší prvok haldy a M-krát pridať prvok alebo urobiť operáciu *decrease key*. Asymptotický čas pri použití Fibonacciho haldy teda je $O(M + N \log N)$.

Algoritmus pri ozavretí cieľového vrcholu končí, keďže nepotrebujeme zistiť najkratšiu vzdialenosť do každého vrcholu. Pre každý navštívený vrchol sa okrem hodnoty cesty pamätá aj vrchol, z ktorého najlepšia cesta vedie. Vďaka tomu je možné na konci zrekonštruovať celú cestu od počiatočného po koncový vrchol.

Na ohodnotenie hrán sa používajú triedy odvodené od `CostCalculator`. Tie musia implementovať funkciu `getCost()`, ktorá podľa dostupných informácií ohodnotí hranu podľa svojej stratégie. Sú implementované tri základné triedy pre ohodnocovanie hrán podľa typu vyhľadávanej cesty – pre najkratšiu, najrýchlejšiu a ekonomickú cestu (tá by mala kombinovať požiadavky na rýchlosť a celkovú dĺžku trasy).

Samotný výpočet cesty v grafe je jadro knižnice. Nad ním je ešte vyššia vrstva určená pre použitie v aplikácii. Je tvorená triedou `RoutingCore` a niekoľkými ďalšími pomocnými triedami. Pomocou nej aplikácia načíta vstupný súbor s reprezentáciou grafu a necháva hľadať cestu, prípadne sa dotazuje na informácie o úsekoch ciest alebo križovatkách. Koncové body sa z aplikácie predávajú ako identifikátor cesty, na ktorej sa bod nachádza, a vzdialenosť bodu od jej začiatku. Funkcia `findRoute()`, ktorá zabezpečuje vyhľadanie

spojenia, prijíma ako argumenty začiatočný a koncový bod a požadovaný typ cesty. Postará sa o napojenie týchto dvoch bodov do grafu, spustenie vyhľadávania, odpojenie koncových bodov z grafu a vrátenie výslednej cesty vo forme uzlov a úsekov ciest medzi uzlami.

Vzľadom na nutnosť podpory zákazov odbočenia je cestná sieť prevedená do grafu tak, že každý úsek cestnej siete sa reprezentuje dvoma vrcholmi – pre každý smer jeden. Vrchol pre normálny smer má pozíciu na začiatku úseku, vrchol pre opačný smer má pozíciu na konci úseku. Tieto pozície sa nepoužívajú v samotnom prehľadávaní grafu, majú význam iba pri rekonštrukcii nájdenej cesty. Hrany medzi vrcholmi značia, že sa z jednej cesty dá prejsť na inú. Každá hrana má odkaz na úsek cestnej siete, ktorý reprezentuje. Informácie o úsekoch sú medzi hranami zdieľané.

6.2 Import cestnej siete

Na prevod dát zo vstupného formátu do internej, efektívnejšej reprezentácie vznikol nástroj `dgbuild`. Tento program načíta cestnú sieť zo súboru typu *shapefile*, vytiahne z neho informácie podstatné pre routing, vytvorí graf a uloží ho pre ďalšie použitie. Očakávaný formát geometrie prvkov:

- každý prvok reprezentuje jeden úsek cesty medzi dvoma križovatkami alebo medzi križovatkou a slepým koncom cesty
- križovatky sa rozpoznávajú iba na koncoch úsekov ciest s rovnakými súradnicami
- kríženie ciest je vždy chápané ako mimoúrovňové v prípade, že nemajú spoločný koncový bod

Na atribúty jednotlivých prvkov sú nasledovné požiadavky:

Atribút	Význam
LINK_ID	unikátny číselný identifikátor úseku
SPD_LIMIT	rýchlostný limit pre úsek v km/h
ROUTE_LVL	úroveň cesty od 1 (lokálne ciest) do 5 (diaľnice)
ONE_WAY	má hodnotu 1 ak je cesta jednosmerná, ináč 0
TURN_RSTRS	zákazy odbočenia oddelené znakom bodkočiarky
NAME	názov cesty resp. ulice

Každý zákaz odbočenia má prvý znak **F**irst alebo **L**ast, podľa toho či zákaz platí na začiatku úseku alebo na konci, nasleduje za ním identifikátor úseku. Napríklad L17;F10 znamená, že z danej cesty sa nesmie v normálnom smere odbočiť na cestu 17 a v opačnom smere sa nesmie z tej cesty odbočiť na cestu 10.

Meno cesty sa používa na detekciu naväzujúcich úsekov jednej cesty, to sa dá využiť pri vyhľadávaní spojenia a dať preferenciu cestám bez odbočovania.

6.3 Uživateľské rozhranie

Rozhranie je tvorené mapovým zobrazením a lišty so záložkami na ľavej strane. Pri spustení aplikácie sa do mapového zobrazenia načítajú vrstvy z uloženého projektu programu Quantum GIS. Samotná aplikácia pre jednoduchosť nemá žiadne funkcie na úpravy vzhľadu mapy, dá sa však jednoducho upraviť v QGISe.

K dispozícii je niekoľko nástrojov na prehliadanie mapy. Okrem nich ešte existuje nástroj na dotazovanie cestnej siete. Po kliknutí na mapu sa nájde najbližší úsek cesty, vyznačí sa na mape spolu vyznačením začiatku (zelený krížik) a konca úseku (červený krížik) a zobrazia sa informácie používané pri vyhľadávaní cesty.

V záložke Routing je možné zvoliť si počiatočný a koncový bod trasy – buď priamo z mapy alebo vyhľadávaním v zozname ulíc. Koncové body sa označia zelenou respektíve červenou vlajkou. Akonáhle sú vybrané obidva body, aplikácia vyhľadá pomocou knižnice spomenutej vyššie trasu požadovaného typu a zobrazí ju na mape. V ľavej časti sa zobrazuje zoznam ciest, cez ktoré cesta vedie.

Pre testovanie sa dá použiť zabudovaný simulátor GPS. Ten funguje tak, že po aktivácii tlačítka "Add waypoints" sa dajú na mape naklikať body, cez ktoré má simulovane prechádzať. Je tiež možné nastaviť si rýchlosť pohybu a takisto časové zrýchlenie. Ak simulátor prejde všetky stanovené body, skončí pohyb a ohlási strátu GPS signálu. Na záložke GPS sa okrem toho zobrazujú posledne získané informácie z GPS: stav, poloha a čas. Šípka na mape ukazuje momentálnu pozíciu a smer. Ak GPS nemá signál, šípka má šedú farbu.

Nakoniec, ak sa uživateľ chce nechať navigovať z aktuálnej pozície na iné miesto na mape, môže v záložke Navigation zvoliť koncový bod. Po navrhnutí trasy program kontroluje prejazd po trase a šípkou ukazuje smer odbočenia.

Literatúra

[1] OpenStreetMap
http://www.openstreetmap.org/

[2] Quantum GIS
http://qgis.org/

[3] Quantum GIS wiki: Splitting Into Libraries
http://wiki.qgis.org/qgiswiki/Splitting_Into_Libraries

[4] Lutz M.: *Programming Python*, Third Edition, O'Reilly, 2006

[5] Guido van Rossum: Python/C API Reference Manual, 2006
http://docs.python.org/api/

[6] Guido van Rossum: Extending and Embedding the Python Interpreter, 2006
http://docs.python.org/ext/

[7] SIP Reference Guide, Riverbank Computing Ltd., 2007
http://www.riverbankcomputing.com/Docs/sip4/sipref.html

[8] Raster Plugin for QGIS
http://www.maths.lancs.ac.uk/∼rowlings/Software/RasterAlgebra/

[9] Dana, Peter H.: *Global Positioning System Overview*, The Geographer's Craft Project, Department of Geography, The University of Colorado
http://www.colorado.edu/geography/gcraft/notes/gps/gps_f.html

[10] NMEA 0183 Standard
http://www.nmea.org/pub/0183/index.html

[11] Neoficiálny zoznam viet NMEA 0183
http://gpsd.berlios.de/NMEA.txt

[12] gpsd – a GPS service daemon
http://gpsd.berlios.de/

[13] Amit J. Patel: Amit's Thoughts on Path-Finding and A-Star
http://theory.stanford.edu/∼amitp/GameProgramming/index.html

[14] Saunders S.: Shortest Path Algorithms and Priority Queues in C++
http://www.cosc.canterbury.ac.nz/research/RG/alg/spalg.html