GIS tutoriály

Jan Gaura

18. března 2021

Obsah

První tutoriál 5

Druhý tutoriál 9

Třetí tutoriál 13

Čtvrtý tutoriál 17

Pátý tutoriál 19

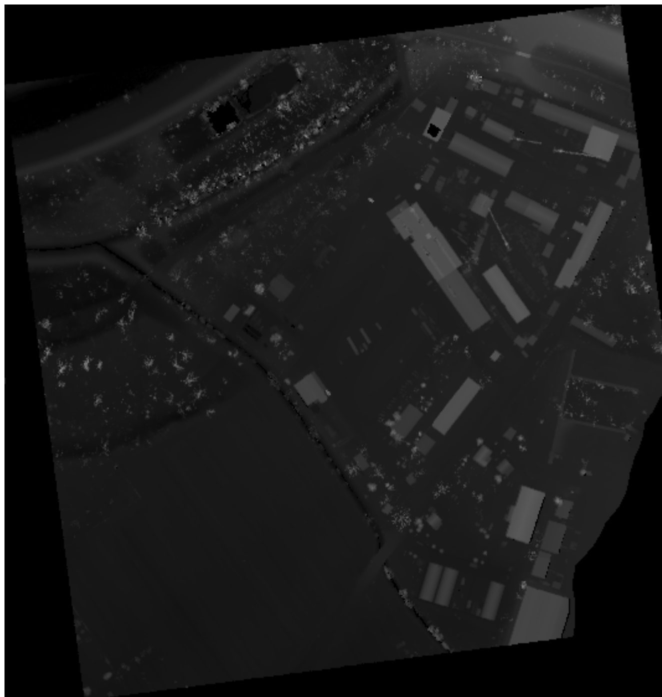
Šestý tutoriál 23

Sedmý tutoriál 25

Osmý tutoriál - Navigace 27

První tutoriál

Na tomto tutorálu provedema základní práci s LIDARovými daty, jež jsou uložena v binárním souboru. Drobnou ochutnávkou naší práce je obrázek 1.



Obrázek 1: Výšková mapa vytvořená z LIDARových dat.

Seznámení s projektem

Projekt je Vám k dispozici ve formě tzv. templatu, tedy kódu, který je možno postupně upravovat a vylepšovat jeko funkcionalitu. Funkce **`main`** kontroluje předané parametry, jednímž je i jméno souboru s LIDARovými daty. Dále pak volá funkci **`process_lidar`**. Ta volá většinu podstatných funkcí pro náš projekt.

Poskytnutá LIDARová data

LIDARová data, která jsou Vám k dispozici obsahují přes 4 miliony naskenovaných paprsků, které jsou z okolí města Karviné. Jedná o obdélník o stranách cca 535 a 560 metrů. Data jsou v souřadnicovém referenčním systému S-JTSK.

Tento systém má souřadnice vždy záporné a x hodnota je vždy větší než y, tj. $x > y$. Nadmořská výška je pak v metrech tak, jak jsme zvyklí.

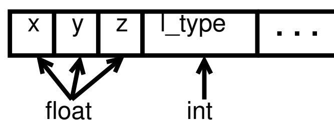
Zadání

V tomto tutorálu si zjistíme rozsahy souřadnic a nadmořské výšky v datovém souboru, tedy minimální a maximální hodnoty na osách x, y a z. Toto implementujte ve funkci **get_min_max**, která má nísledující hlavičku

```
void get_min_max( const char *filename,
                  float *a_min_x, float *a_max_x,
                  float *a_min_y, float *a_max_y,
                  float *a_min_z, float *a_max_z )
```

kde **filename** je název binárního souboru s daty, který budete číst, další parametry jsou pointery na minimální a maximální hodnoty, které získáte a předáte zpět tak, aby je mohla přečíst volající funkce **process_lidar**.

Způsob zjištění daných rozsahů je jednoduchou úlohou o zjištění minima a maxima v poli s tím, že my procházíme soubor. Tento soubor je binární a jeho struktura je zobrazena na obrázku 2. V souboru jsou tedy ke každému nasnímanému LIDARovému paprsku uloženy 4 položky (3x datového typu **float** a 1x typu **int**). Pro zjištění minim a maxim není důležitá poslední položka, která je v obrázku pojmenována jako **l_type**. Je však nutné ji při čtení souboru číst, jinak by nám data přestala dávat smysl. V souboru je uloženo přes 4 milióny LIDARových paprsků.



Obrázek 2: Struktura souboru s LIDARovými daty. První 3 položky jsou vždy typu **float**, pak 1 položka typu **int**. Toto se pak opakuje až do konce souboru.

Z výše uvedeného tedy plyne, že je nutné postupně projít celý soubor a hledat minimální a maximální čísla. Získaná čísla pak vrátíte v pointerech, které jsou předány funkci **get_min_max** a jejichž názvy jsou samopopisující. Pozor dávejte na to, že data mají zápornou x a y souřadnici.

Správný výsledek Vámi implementované funkce by měl být stejný jako v následujícím výpisu.

```
min x: -453462.187500, max x: -452927.250000
min y: -1101354.750000, max y: -1100794.750000
min z: 219.990005, max z: 260.131989
delta x: 534.937500
delta y: 560.000000
delta z: 40.141983
```

Potřebné funkce z jazyka C

Pro práci se souborem budete potřebovat následující funkce ze standardní knihovny jazyka C, ale můžete použít i jiný způsob načtení dat:

- **`fopen`**
- **`fclose`**
- **`feof`**
- **`fread`**

Hodí se také operator **`sizeof`** pro určení velikosti datových typů.

Druhý tutoriál

V druhém tutoriálu si již vykreslíme zmiňovanou výškovou mapu z předchozí kapitoly.

Zadání

V předchozím tutoriálu jsme si zjistili minimální a maximální hodnoty souřadnic v souboru. Jejich jednoduchým odečtením získáme velikost dat v metrech. Proto si můžeme vyrobit obraz, který bude nabývat právě těchto rozměrů. Pro správné zaokrouhlení použijte metodu **cvRound**. Vytvoření obrázku pak bude vypadat následovně a je téměř celé v poskytnutém projektu:

```
heightmap_8uc1_img = cv::Mat( cvSize( cvRound( delta_x + 0.5f ),
                                      cvRound( delta_y + 0.5f ) ),
                              CV_8UC1 );
```

Argument **cvSize** nám udává velikost obrazu, **CV_DEPTH_8UC1** hloubku obrazu v bitech na jeden obrazový kanál a počet kanálů v obraze (**C1**). Pro odstíny šedi je to 1 kanál (**C1**), pro RGB jsou to 3 kanály (**C3**). **CV_DEPTH_8UC1** znamená použití 8 bitů na jeden kanál, tedy hodnoty 0 až 255.

Budeme kreslit do obrazu s jedním odstínem šedi. Toto provedeme ve funkci **fill_image**, která má následující předpis:

```
void fill_image( const char *filename,
                 cv::Mat & heightmap_8uc1_img,
                 float min_x, float max_x,
                 float min_y, float max_y,
                 float min_z, float max_z );
```

Vstupem funkce je opět jméno binárního souboru (**filename**), výsledný obrázek (**heightmap_8uc1_img**) a minimální a maximální souřadnice (**min**, **max**), které jsme získali z minulého tutoriálu.

Problém však je, že soubor obsahuje cca 4 miliony LIDARových paprsků, ale obraz má cca 250 000 pixelů. Z toho tedy plyne, že více LIDARových paprsků dopadne do jednoho pixelu. Pro každý pixel si tedy spočítáme průměrnou výšku z LIDARových paprsků, které na daný pixel dopadly. Budeme tedy potřebovat dvě pole:

- jedno na průběžný součet výšek, které dopadají na daný pixel: **float *sum_height**;
- druhé na uložení počtu LIDARových paprsků, které dopadly na daný pixel: **int *sum_height_count**.

Opět nezapomeňme, že souřadnice jsou v systému S-JTSK jsou záporné a nemůžeme je tedy přímo použít pro adresování v poli.

Po výpočtu průměrné výšky v každém pixelu je třeba hodnoty zapsat do obrázku. Toto provedeme pomocí následujícího volání:

```
heightmap_8uc1_img.at<uchar>( y, x ) = value;
```

Proměnná **value** je typu **uchar** (**typedef** pro **unsigned char**) a může obsahovat hodnoty 0 až 255[1].

Z hodnot **min_z** a **max_z** víme, že nadmořské výšky neodpovídají takto definovanému rozsahu. Proto je nutné všechny průměrné nadmořské výšky nejprve normalizovat do rozsahu $\langle 0, 255 \rangle$. Takto normalizované hodnoty již můžeme zapsat do obrazu tak, že projdeme všechny pixely a zapíšeme korespondující hodnotu z pole průměrných výšek:

```
for ( y = 0; y < heightmap_8uc1_img->height; y++ ) {
    for ( x = 0; x < heightmap_8uc1_img->width; x++ ) {
        heightmap_8uc1_img.at<uchar>( y, x ) = normalized_value;
    }
}
```

Předchozí kód je spíše schématický, ale dostatečně demostruje, jak je možné projít celý obraz a zapsat do něj hodnoty.

Pro úplnost ještě ukažme, jak bude vypadat podstatná část funkce **process_lidar**, aby nám zobrazila výsledek. Nekopírujte ji do svého zdrojového kódu, je tam již obsažena a stačí ji jen na správných místech odkomentovat.

```
heightmap_8uc1_img = cv::Mat( cvSize(
                              cvRound( delta_x + 0.5f ),
                              cvRound( delta_y + 0.5f ) ),
                            CV_8UC1 );
heightmap_show_8uc3_img = cv::Mat( cvSize(
                                   cvRound( delta_x + 0.5f ),
                                   cvRound( delta_y + 0.5f ) ),
                                 CV_8UC3 );

create_windows( heightmap_8uc1_img.cols, heightmap_8uc1_img.rows );
mouse_probe = new MouseProbe( heightmap_8uc1_img,
                              heightmap_show_8uc3_img,
                              edgemap_8uc1_img );

cv::setMouseCallback( STEP1_WIN_NAME, mouse_probe_handler, mouse_probe );
cv::setMouseCallback( STEP2_WIN_NAME, mouse_probe_handler, mouse_probe );
```

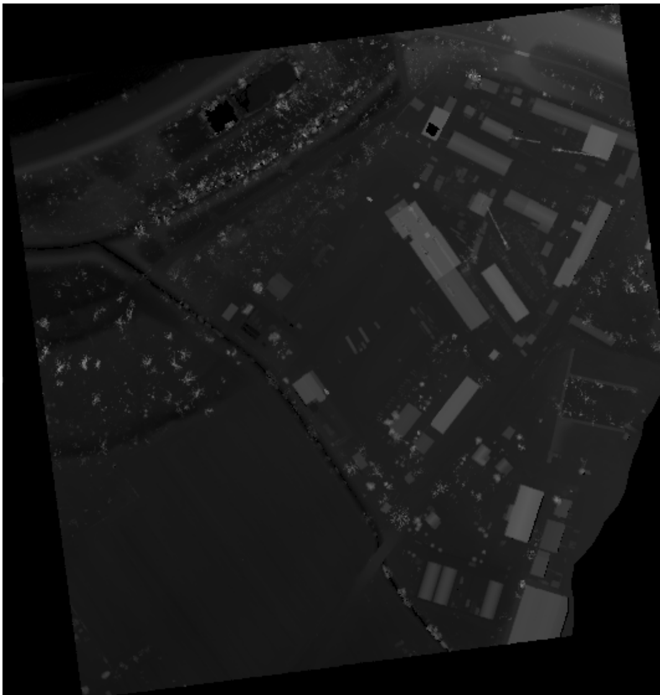
[1] 0 reprezentuje černou barvu a 255 barvu bílou.

```
fill_image( bin_filename, heightmap_8uc1_img, min_x, max_x, min_y, max_y, min_z, max_z );
cv::cvtColor( heightmap_8uc1_img, heightmap_show_8uc3_img, CV_GRAY2RGB );

cv::imwrite( img_filename, heightmap_8uc1_img );

// wait here for user input using (mouse clicking)
while ( 1 ) {
    cv::imshow( STEP1_WIN_NAME, heightmap_show_8uc3_img );
    int key = cv::waitKey( 10 );
    if ( key == 'q' ) {
        break;
    }
}
```

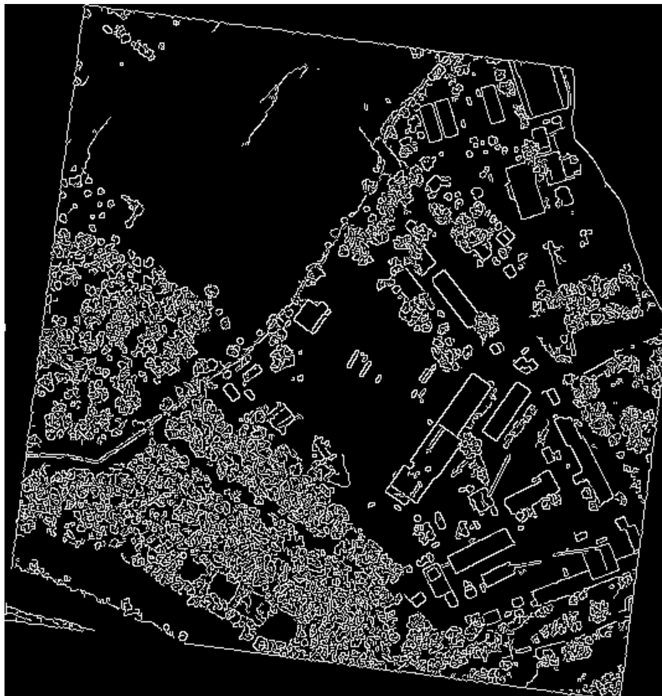
Výsledný obraz pak vypadá tak, jako na obrázku 3.



Obrázek 3: Výšková mapa vytvořená z LIDARových dat.

Třetí tutoriál

V minulém tutoriálu jsme obdrželi výškovou mapu povrchu. Pokud na takovouto mapu budeme aplikovat detektor hran (např. Cannyho detektor hran), obržíme hranový obrázek 10.



Obrázek 4: Cannyho detektor hran aplikovaný nad výškovou mapou.

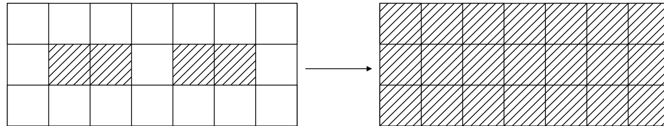
V tomto tutoriálu naimplmentujeme algoritmy dilatace a eroze na hranách proto, abychom mohli později manuálně extrahovat objekty z mapy (převážně budovy).

Před aplikací operace dilatace a eroze je ještě třeba obraz hran binarizovat. To provedeme tak, že všechny hodnoty pixelů v obraze hran, které jsou větší než hodnota 128 nastavíme na hodnotu 255. V opačném případě nastavíme hodnotu na 0. Výsledný obraz bude vypadat podobně jako po aplikaci detektoru hran, jsou však v něm odstraněny „nejasné stavy".

Operace dilatace a eroze jsou morfologickými operacemi. Cílem

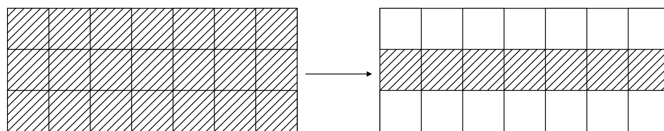
těchto operací je „nafouknutí" nebo „zhuštění" hran v obraze. Toto můžeme schématicky vidět na obrázku 5 pro dilataci a na obrázku 6 pro erozi.

Operace dilatace pracuje tak, že aplikuje masku zadaného rozměru (příklad je pro masku 3×3). Maska je aplikována na každý pixel obrazu (pro jednoduchost vynecháváme okraje obrazu). Pokaždé, když je středový pixel masky nad obrazovým pixelem, jehož hodnota je rovna 255, je celá maska ve výsledném obraze vyplněna hodnotou 255.



Obrázek 5: Operace dilatace.

Operace eroze je inverzní k operaci dilatace. Pokud se centrální pixel masky nachází na hodnotě 255 a nějaký jiný pixel masky má hodnotu jinou, než středový pixel, je středový pixel nastaven na hodnotu 0.



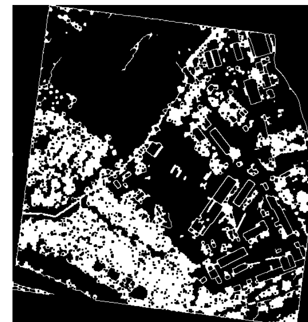
Obrázek 6: Operace eroze.

Nezapomeňte, že tyto operace nelze implementovat na jednom obraze (tzv. in place), ale výstup operace musí být uložen do nového obrazu.

Výsledný obraz po aplikaci operace dilatace a eroze na původní obraz 10 je na obrázku 7.

Obrázek 7: Hrany po aplikaci operace dilatace a eroze.

Čtvrtý tutoriál



Obrázek 8: Hrany po aplikaci operace dilatace a eroze.

Nyní chceme extrahovat jednotlivé objekty z obrazu. To budeme provádět na obraze, který je dán výstupem po operacích dilatace a eroze z minulého tutoriálu, což můžete vidět opět na obrázku 8.

V minulém tutoriálu jsme zacelili všechny díry v hranách objektů (zajímají nás budovy). Nyní chceme tyto budovy v obraze detekovat. Vzhledem ke složitosti daného problému se uhýlíme k jednoduššímu řešení s manuální selekcí budov.

V principu chceme do výškové mapy nebo do mapy hran kliknout myší v místě budovy. Daná budova se nám pak vybarví zvolenou barvou.

Nejprve se podívejme, jak obsloužíme ono kliknutí. Ve funkci **`process_lidar`** máme volání

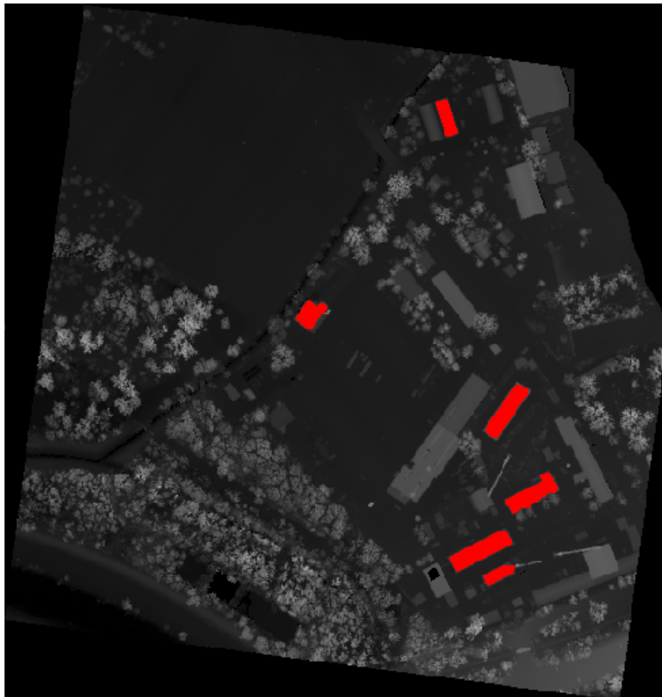
```
cv::setMouseCallback( STEP1_WIN_NAME, mouse_probe_handler, mouse_probe );
cv::setMouseCallback( STEP2_WIN_NAME, mouse_probe_handler, mouse_probe );
```

Ta zajistí, že se při kliknutí myší v daném okně zavolá funkce **`mouse_probe_handler`**. V ní se pak zavolá funkce **`flood_fill`**. Tato funkce provede tzv. semínkové vyplňování okolí zadané souřadnice. Algoritmus je rekurzivní a tak se vyplní první pixel pod kurzorem myši a dále se postupuje do okolních bodů (nahoru, dolů, doleva, doprava). Vyplňování proveďte na hranovém obraze. Budeme chtít vyplnit pixely stejné barvy, jako je barva pixelu, na který jsme klikli myší. Pokud narazíte na pixel jiné barvy (hrana budovy), rekurze se ukončí.

Algoritmus je také pospsán na Wikipedii anglicky[2].

Příklad výsledku můžete vidět na následujícím obrázku 9.

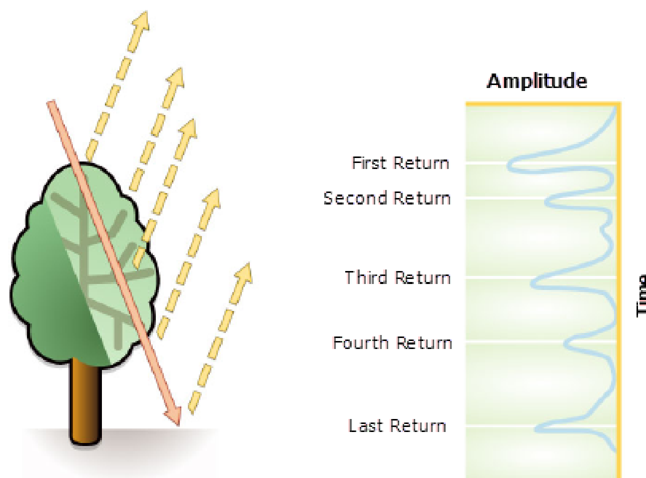
[2] `https://en.wikipedia.org/wiki/Flood_fill`
Osobně považuji anglickou Wikipedii za vynikající zdroj informací, pokud je článek dobře zpracován. Do svých diplomových prací ji však neuvádějte, protože může některé členy komise dosti rozlítit).



Obrázek 9: Příklad extrahovaných objektů.

Pátý tutoriál

V dnešním tutoriálu se podíváme na tajemnou položku v datech a tou je **`l_type`**. Jedná se o počítadlo odrazu lidarového paprsku na daném místě. Více odrazů může vzniknout tak, že paprsku stojí v cestě nějaký polopropustný předmět. Jednoduchým příkladem takového objektu je strom. Ukázka, jak takový paprsek prolétá stromem, kde se postupně odráží a nakonec se odrazí od země je na následujícím obrázku:

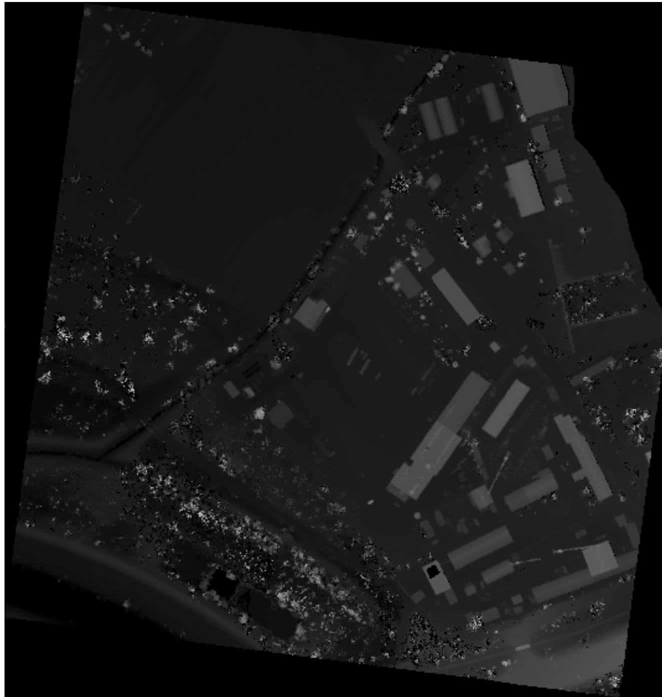


Obrázek 10: Schématická ukázka odrazů paprsku při průchodu stromem (`https://desktop.arcgis.com/en/arcmap/10.3/manage-data/las-dataset/what-is-lidar-data-.htm`).

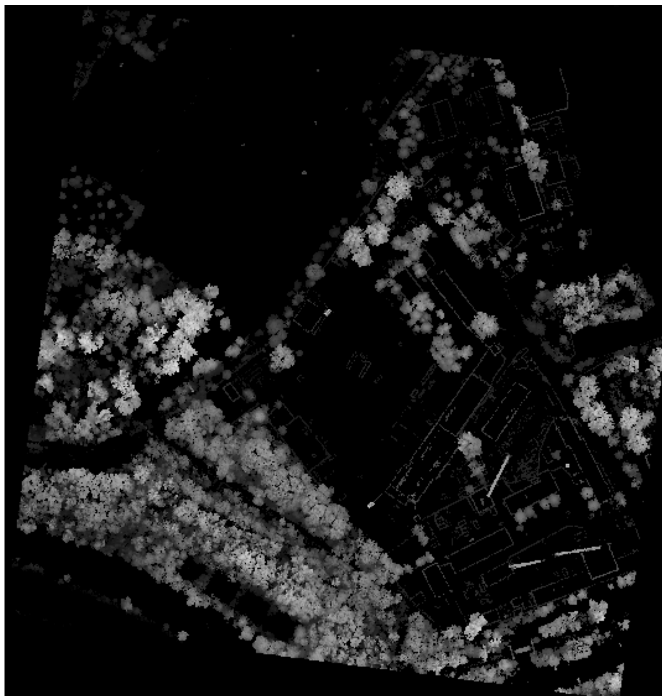
V našich datech máme pouze tři odrazy (0, 1, 2). Typ odrazu je specifikován položkou **`l_type`** na obrázku 2. Datovým typ položky je **`int`**. Ke každému paprsku tak můžeme určit, jedná-li se o odraz přímý (obrázek 11), odraz z vegetace (obrázek 12) nebo odraz od povrchu pod vegetací (obrázek 13).

Vaším úkolem je vytvořit pro každý typ lidarového paprsku jednu výškovou mapu tak, jak jsou zobrazeny na obrázcích 11, 12 a 13. Každý lidarový odraz tak vlastně tvoří jednu vrstvu . Berte tedy v potaz, že každá vrstva může mít jiný rozsah z-ové souřadnice a je proto nutné normalizovat každou vrstvu zvlášť.

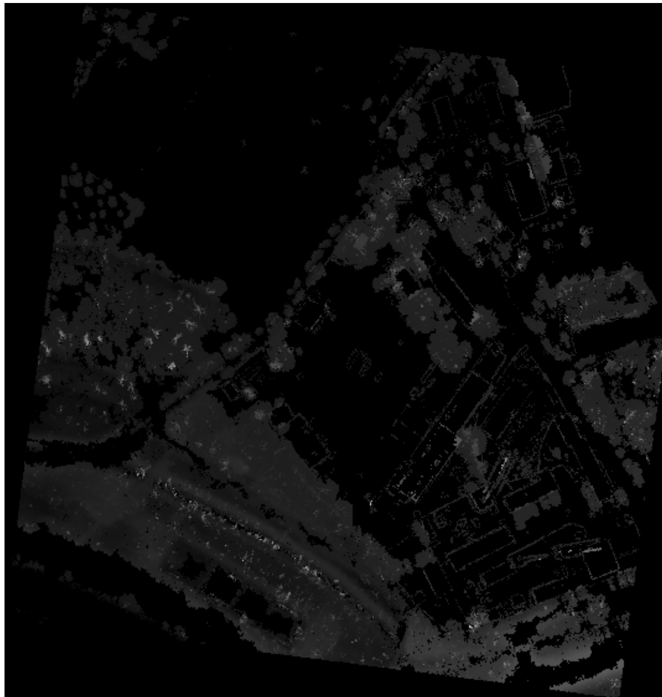
S vrstvami jsme pracovali v GRASS GIS na prvním cvičení



Obrázek 11: Výšková mapa prvního odrazu (`l_type == 0`).



Obrázek 12: Výšková mapa vegetace (`l_type == 1`).



Obrázek 13: Výšková mapa povrchu pod vegetací (**`l_type == 2`**).

Šestý tutoriál

V tomto tutoriálu chceme zobrazit námi vytvořený výškový obrázek v kontextu mapy. Chceme do mapy vložit obrázek a překrýt jím povrch ve správném místě jako na obrázku 14.

Obrázek 14: Výsledné zobrazení výškové mapy v kontextu okolí v mapě `https://leafletjs.com`.

Vzhledem k licenční politice Google Maps budeme muset využít nějaký jiný nástroj. Velmi dobrou alternativou se tak jeví mapy Leaflet.js[3]. Budete si muset přečíst alespoň základní dokumentaci k vrstvě **`ImageOvetlay`**[4].

[3] `https://leafletjs.com/`

[4] `https://leafletjs.com/reference-1.6.0.html#imageoverlay`

Zobrazení lidarových dat v Leafletjs

Obraz s lidarovými daty zobrazte v Leafletjs pomocí připraveného templatu[5]. Do tohoto souboru budete muset vložit několik údajů, kterými jsou:

[5] `http://mrl.cs.vsb.cz/people/gaura/gis/index_stub.html`

Your access token

Token pro práci s Leaflet mapami. Získáte jej po registraci na stránce projektu. Existuje také možnost použít mapy bez tokenu, ale tuto možnost prozkoumejte samostatně.

URL with your image

URL obrázku s výškovou mapou povrchu. Obrázek můžete nahrát např. na server **`homel.vsb.cz`** do adresáře **`public_html`** ve Vašem domovském adresáři (přihlášení jako do pošty). Dávejte pozor na cachování obrázku v případě, že byste jej chtěli změnit. Spíše jej uložte pod novým názvem, než čekat, až cache propadne.

Your coordinates

Zjistěte bounding box dat (BBox). Ten ohraničuje naše data ze všech stran a jsou to ony minimální a maximálni souřadnice na osách x a y. Naše data jsou však v systému S-JTSK. Leaflet mapy, stejně jako i jiné celosvětové mapy, pracují v systému WGS84. Je proto nutné tyto souřadnice transformovat z S-JTSK do WGS84. Tuto transformaci provedeme pomocí knohovny **`PROJ.4`**. V linuxu se tato knihovna dá nainstalovat jako balíček **`libproj-dev`**.

Stáhněte si example z knihovny **`PROJ.4`**[6].

Vytvořte si dvě projekce (wgs84 a s-jtsk) s následujícím init stringem:

```
wgs84: +proj=longlat +ellps=WGS84 +datum=WGS84 +no_defs
s-jtsk: +proj=krovak +ellps=bessel +towgs84=570.8,85.7,462.8,4.998,1.587,5.261,3.56
```

Proveďte transformaci okrajových souřadnic dat (**`min_x`**, **`min_y`**) a (**`max_x`**, **`max_y`**).

Vložte transformované souřadnice do připraveného templatu. Nemusíte generovat žádny soubor programově.

Když pak načtete upravený template soubor, měl by se vám zobrazit podobný výsledek jako na obrázku 14.

[6] `https://proj.org/development/migration.html`

Sedmý tutoriál

Na cvičení jsem zaslechl, že byste chtěli programovat radeji v Pythonu, nežli v jazyce C. Budiž Vám tedy vyhověno.

V dnešním tutoriálu budeme pracovat s vektorovými daty a to prostřednictvím dat z projektu OpenStreetMap[7]. Pro práci jsem Vám předpřipravil výřez z Ostravy - Poruby, který neleznete na příslušném odkaze[8].

[7] `https://www.openstreetmap.org/`

[8] `http://mrl.cs.vsb.cz/people/gaura/gis/poruba.osm`

Vaším úkolem je z OSM souboru vyextrahovat zastávky MHD a zobrazit je ve slepé mapě. OSM soubor je v podstatě XML soubor s daty projektu OpenStreetMap. Dále v mapě zakreslety všechny silnice klasifikované jako **`secondary`** a **`residential`**. Vykreslete je různými barvami. Zastávky MHD vykreslete kruhem nějaké barvy. Výběr barev nechávám na Vás. Vzhledem k zakřivení referenčního systému WGS 84, ale poměrně malé ploše mapy můžete k vykreslení přistoupit jako ke kreslení do kartézské soustavy souřadnic s posunutím.

Jak extrahovat zastávky

Níže je uveden příklad pro zastávku Poruba U nemocnice vedle naší fakulty. Všechny zastávky mají v uzlu **`node`** poduzel **`tag`**, který má atribut **`k`** nastaven na **`public_transport`** a hodnotu **`v`** nastavenu na **`stop_position`**. Souřadnice zastávky je uložena v atributech **`lat`** a **`lon`** uzlu **`node`**.

```
<node id="1512535002" visible="true" version="4"
      changeset="47933008"
      timestamp="2017-04-19T10:14:27Z" user="Hlav"
      uid="100163" lat="49.8306687" lon="18.1597174">
 <tag k="bus" v="yes"/>
 <tag k="highway" v="bus_stop"/>
 <tag k="name" v="Poruba U nemocnice"/>
 <tag k="public_transport" v="stop_position"/>
</node>
```

Jak extrahovat cesty

Níže je uveden příklad pro ulici Dvorní naproti VŠB přes ulici 17. listopadu. Cesta je tvořena uzlem **`way`**, který obsahuje poduzel **`tag`**, který má atribut **`k`** nastaven na **`highway`** s hodnotou **`v`** nastavenou na **`residential`** (jedna ze dvou cest, které chceme extrahovat). Takováto cesta obsahuje poduzly **`nd`**, které obsahují atribut **`ref`**. Pomocí tohoto artibutu reference se dostanete na jednotlivé uzly se souřadnicemi podobně jako je tomu u zastávky. Pak již stačí jen vykreslit čáru mezi jednotlivými uzly cesty do obrázku.

Příklad cesty

```
<way id="173119730" visible="true" version="1"
     changeset="12507666"
     timestamp="2012-07-27T07:22:31Z"
     user="Datin" uid="115815">
 <nd ref="347552565"/>
 <nd ref="1839509068"/>
 <nd ref="1839509067"/>
 <nd ref="249457527"/>
 <nd ref="352961533"/>
 <nd ref="347552561"/>
 <tag k="highway" v="residential"/>
 <tag k="name" v="Dvorni"/>
</way>
```

Příklad jednoto uzlu cesty

```
<node id="347552565" visible="true" version="3"
 changeset="12258717"
 timestamp="2012-07-17T06:43:12Z"
 user="Datin" uid="115815" lat="49.8333133"
 lon="18.1672890"/>
```

Osmý tutoriál - Navigace

Budeme pokračovat v použití jazyka Python. Opět budeme pracovat s vektorovými daty a to prostřednictvím dat z projektu OpenStreetMap[9]. Pro práci jsem Vám předpřipravil výřez z Ostravy - Poruby, který je stejný jako v minulém zadaní a neleznete jej na příslušném odkaze[10].

V dnešním zadání chceme naimplementovat navigaci v silniční sití. Můžeme si představit, že vytváříme subsystém pro aplikace jako Google Maps apod. Navigace nebo také výpočet cesty je úloha již dobře známa a existuje pro ni řada algoritmů. Vašim zadáním je jeden z těchto algoritmů naimplementovat.

Prvním algoritmem pro hledání cesty je Dijkstrův algoritmus[11]. Ten je schopen nalézt cestu v grafu. Silniční síť je takovým grafem a proto jej můžeme snadno použít pro naše zadání.

Dalším algoritmem je tzv. A*[12]. Rozdíl oproti Dijkstrovu algoritmu je v tom, že A* používá heuristiku pro odhad vzdálenosti do cíle. Tím je schopen vynechat značnou část prohledávaného grafu. Je však třeba mít nastavenu heuristiku dobře.[13]

Pro dnešní tutoriál naimplementujte Dijskrův algotirmus mezi dvema body v mapě. Prvním bod bude takový, který se nalézá na silnici v okolí naší budovy FEI. Druhým bodem bude křizovatka ulic *Polská* a *Vietnamská*. Můžete využit některého bodu ze zastávek, avšak ty nemusí ležet přimo na cestě a graf by tak nebyl spojitý.

Pro to, aby nám vše dobře fungovalo budeme si muset ještě přidat cesty, jež jsou tvořeny uzlem **`way`**, který obsahuje poduzel **`tag`**, který má atribut **`k`** nastaven na **`highway`** s hodnotou **`v`** nastavenou na **`primary`**. Chceme tedy použít i silnice prvních tříd.

[9] https://www.openstreetmap.org/

[10] http://mrl.cs.vsb.cz/people/gaura/gis/poruba.osm

[11] https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm

[12] https://en.wikipedia.org/wiki/A*_search_algorithm

[13] http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html