VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

# AUTOMATICKÉ GENEROVÁNÍ VÝŠKOVÉ MAPY Z GPS ZÁZNAMŮ

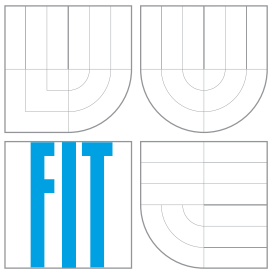
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

JAN DUŠEK

BRNO 2012

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

# AUTOMATICKÉ GENEROVÁNÍ VÝŠKOVÉ MAPY Z GPS ZÁZNAMŮ

AUTOMATIC ELEVATION MAP GENERATION FROM GPS LOGS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — JAN DUŠEK
AUTHOR

VEDOUCÍ PRÁCE — Ing. RADEK BARTOŇ
SUPERVISOR

BRNO 2012

# Abstrakt

Tato práce se zabývá automatickým generováním výškové mapy z GPS záznamů. Je používána metoda interpolace do pravidelné mřížky, konkrétně je používána Clough-Tocher metoda pro vygenerování hladkého $C^1$ povrchu. V rámci práce byl navržen a vytvořen nástroj, který umožňuje automaticky vytvořit výškovou mapu, jako zdroj slouží projekt *OpenStreetMap*.



# Abstract

This paper examines automatic elevation map generation from GPS logs. Interpolates data to regular grid, using Cloug-Tocher method for generating smooth $C^1$ surface. In this paper we propose a tool for automatic elevation map generation. We use *OpenStreetMap* as our main datasource



# Klíčová slova

Interpolace, Clough-Tocher, OpenStreetMap, GPX, výšková mapa

# Keywords

Interpolation, Clough-Tocher, OpenStreetMap, GPX, elevation map



# Citace

Jan Dušek: Automatické generování výškové mapy z GPS záznamů, bakalářská práce, Brno, FIT VUT v Brně, 2012

# Automatické generování výškové mapy z GPS záznamů

## Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Bartoně

.....
Jan Dušek
16. května 2012

## Poděkování

Rád bych poděkoval vedoucímu této práce Ing. Radku Bartoňovi za cenné připomínky a rady při vypracovávání projektu.

# Obsah

# Kapitola 1

# Úvod

V dnešní době kdy dnes každý chytrý telefon obsahuje přijímač GPS, řidiči používají stále více navigací pomocí systému GPS a sport zvaný „geocaching“ nabývá na popularitě je k dispozici stále více dat o povrchu země, který může být využit pro vytvoření digitální modelu terénu. Přesnost GPS není nekonečná, ale při dostatečném počtu kvalitních dat je možné aproximovat celkem přesné výsledky. V rámci projektu *OpenStreetMap* uživatelé nahrávají velké množství dat v podobě tras, kde se pohybovali, která jsou volně dostupná a mohou nám dát další zdroj informací o reliéfu krajiny.

Úkolem práce bylo vytvořit nástroj pro automatické generování digitálního modelu terénu z těchto tras. V první kapitole se věnujeme standardům pro ukládání záznamů trasy a tomu, jak tyto data z *OpenStreetMap* získat. V další kapitole se věnujeme interpolačním algoritmům jak z GPS tras vytvořit souvislý povrch. Dále se věnujeme návrhu nástroje a jeho implementaci. Na závěr prezentujeme výsledky experimentů navržených algoritmů na konkrétních datech z projektu *OpenStreetMap*.

# Kapitola 2

# GPS data

## 2.1 Ukládání GPS dat

Systém GPS je globální družicový polohový systém, což je služba umožňující určení polohy, používající standard WGS84, který pro vyjádření souřadnic používá zeměpisnou délku a šířku. Pro záznam či výměnu dat ze systému GPS vzniklo mnoho rozličných formátů, používaných jednotlivými výrobci GPS zařízení a software.

### 2.1.1 GPX

Postupem času se jako nejpoužívanější ukázal formát GPX[4], což je jednoduchý formát pro výměnu GPS dat mezi aplikacemi a webovými službami na internetu. Tento formát je XML schéma a je tedy podmnožinou XML, je otevřeným standardem a schéma je volně k dispozici.

V GPX je kolekce vzájemně nesouvisejících bodu reprezentována kolekcí tzv. waypointů. Vzájemně související body jsou reprezentovány buď stopami tzv. track, které reprezentují kde uživatel byl, a cestami tzv. route, které reprezentují cestu k cíli.

V GPX body mají povinné parametry pouze zeměpisnou šířku a délku, pro naší aplikaci, ale potřebujeme také nadmořskou výšku, ta je určena nepovinným elementem *ele* a je uvedena v metrech.

### 2.1.2 Konverze formátů

Jak již bylo řečeno existuje mnoho formátů pro výměnu GPS dat a implementovat jejich konverzi by bylo zbytečně složité, naštěstí existuje program *GPSBabel* [1], který umí konvertovat mezi nepřeberným množstvím různých formátů. Tento program používá příkazový řádek a lze ho tedy volat v našem nástroji automaticky a umožnit uživateli použít mnohem větší spektrum podporovaných formátů.

## 2.2 Zdroje dat

Pro naši práci je třeba získat data obsahující, jak polohu tak i nadmořskou výšku, jelikož jádrem práce je interpolace trojrozměrných bodů. Vhodným zdrojem dat se jeví projekt *OpenStreetMap* či databáze českých geodetických bodů *Dataz*.

[1] http://www.gpsbabel.org

### 2.2.1 OpenStreetMap

Projekt *OpenStreetMap* [2] je volně editovatelná mapa světa, uživatelé ji rozšiřují přidáváním GPS stop ve formátu GPX. *OpenStreetMap* používá vlastní formát zvaný OSM, který ovšem nadmořskou výšku neobsahuje, naštěstí je možno získat i zdrojové GPS stopy, které nadmořskou výšku ve většině případů obsahují.

Projekt také poskytuje *API* [1], které umožňuje získávat data ve formátu OSM či GPX z daného ohraničení, anglicky „bounding box“. *API* používá protokol *HTTP* a dotazování probíhá na server *api.openstreetmap.org* Dotaz pro získání GPX stop zní:

```
GET /api/0.6/trackpoints?bbox=left,bottom,right,top&page=pageNumber
```

kde *left, bottom, right, top* jsou souřadnice ohraničení jako maximální a minimální zeměpisná šířka a výška a *pageNumber* je číslo specifikující soubor 5000 bodů, které budou získány. Tento příkaz tedy dokáže získat maximálně 5000 bodů a pro získání všech bodů z ohraničení je tedy nutno tento parametr inkrementovat, až do doby kdy dotaz nevrátí žádné body.

Bohužel v současné verzi *API* není v OSM ani GPX přítomna elevace, ale naštěstí v GPX stopách je přítomen odkaz na zdrojové GPX stopy, obsažené url neukazuje přímo na data stopy, ale obsahuje její *id*. Když známe *id* je možné zdrojovou GPX stopu stáhnout z adresy *http://www.openstreetmap.org/trace/$i/data*, kde *$i* je *id* dané stopy.

### 2.2.2 Dataz

Databáze trigonometrických a zhušťovacích bodů[3] je projekt Českého úřadu zeměměřičského a katastrálního, kde jsou uloženy všechny trigonometrické a zhušťovací body z celého území České Republiky. Data jsou poskytována bezplatně skrze webové rozhraní, v souřadnicových systémech S-JTSK a ETRS89.

ETRS89 (Evropský terestrický referenční systém 89) je určen pro Evropu a tím se liší od WGS84, které je celosvětové, díky rychlosti pohybu kontinentů se od sebe vzdalují, na území ČR asi o 3 cm za rok. Jelikož není rozdíl tak velký, můžeme jej zanedbat. Dalším souřadnicovým systémem je S-JTSK, což je náš místní souřadnicový systém a vznikl již před dlouhou dobou, kvůli jejich odlišné povaze je transformace do WGS84 poněkud obtížnější.

Bohužel má *Dataz* jednu vadu, vyhledávání a stahování dat je ošetřeno *CAPTCHOU* a nelze tedy provést žádné automatické stahování jako v případě *OpenStreetMap*. *Dataz* jako zdroje dat používá již existují geodetické body, ty jsou zaměřeny s přesností na několik centimetrů a tak bude mít oproti *OpenStreetMap* mnohem přesnější data.

[2] http://www.openstreetmap.org
[3] http://dataz.cuzk.cz/

# Kapitola 3

# Interpolace bodů

Jádrem práce je interpolace trojrozměrných bodů jako dvourozměrný signál. Zdrojová data jsou GPS stopy, což jsou kolekce uspořádáných bodů tvořících křivku jdoucí terénem. Pro algoritmy použité v naší práci předpokládáme, že jsou tyto body neuspořádané a tvoří tzv. „oblak bodů“, anglicky „Point cloud“. Z těchto neuspořádaných bodů je nutné vytvořit povrch, tedy spojitou funkci. V této práci se zabýváme metodami interpolace do pravidelné mřížky, od jednoduché metody hledání nejbližšího souseda, po metody založené na triangulaci: lineární interpolace a *Clough-Tocherovu* metodu.

Interpolace do pravidelné mřížky je založena na vytvoření výškové mapy, [13] kdy se zvolí pravidelná mřížka a pro každý její bod se interpoluje výška bodu, na základě neuspořádaných bodů v datech. U této metody existuje několik způsobů jak se získá výška bodu a také jak zvolit pravidelnou mřížku. Z pravidelné mřížky se poté jednoduchým algoritmem vytvoří povrch.

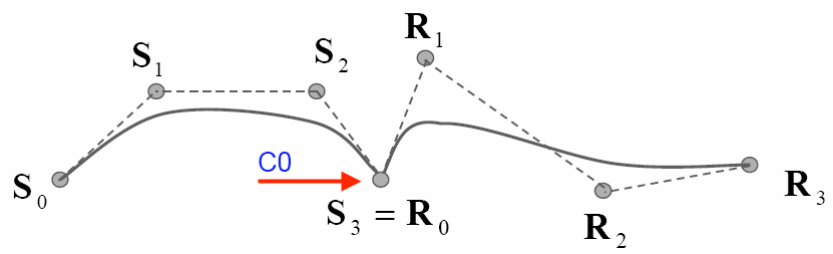
## 3.1 Spojitost

Interpolační metody jsou založeny na parametrických křivkách, a tak je možno hovořit o spojitosti i v souvislosti s interpolačními metodami. Existují dva druhy spojitosti[9]: parametrická $C^n$ a geometrická $G^n$, kde $n$ je třída spojitosti. Čím je $n$ vyšší tím je křivka „hladší“.

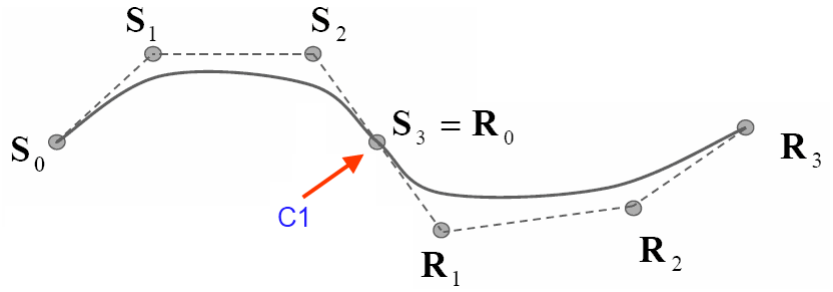
Křivka je třídy $C^n$, má-li ve všech bodech spojité derivace do řádu $n$. $C^0$ platí pokud mají segmenty totožné koncové body. $C^1$ platí pokud mají segmenty totožný tečný vektor, tj. mají totožnou první derivaci. Analogicky $C^2$ platí pokud mají segmenty totožnou druhou derivaci.

Geometrická spojitost zaručuje oproti parametrické spojitosti totožnost tečen nikoliv tečných vektorů, a tedy platí, že pokud je křivka $C^n$ pak je i $G^n$, naopak to ale neplatí tj. $G^n$ nemusí znamenat, že křivka je i $C^n$. Taktéž platí, že dosáhnout $G^n$ je jednodušší než $C^n$.

Na obrázcích 3.1 a 3.2 vidíme příklady spojitosti třídy $C^0$ respektive $C^1$. Můžeme tedy vidět, že spojitost $C^0$ není hladká, bude tedy vhodnější použít interpolaci s třídou spojitosti minimálně $C^1$, která generuje hladké povrchy.




Obrázek 3.1: Ukázka křivky se spojitostí $C^0$




Obrázek 3.2: Ukázka křivky se spojitostí $C^1$

## 3.2 Volba pravidelné mřížky

Aby algoritmus dával dobré výsledky, je potřeba, aby byla vygenerovaná mřížka kvalitní, a například nepřesahovala do míst, kde není dostatek dat pro interpolaci, či aby nebyla příliš hustá, to by pak algoritmus měl větší časovou náročnost, ale žádnou přesnost navíc. Pokud by byla mřížka zase příliš řídká, algoritmus by pak generoval nepřesné výsledky.

Vygenerování mřížky probíhá jednodušše tak, že se určí kroky $x_{\mathsf{step}}$ a $y_{\mathsf{step}}$ na ose $x$ respektive $y$ a mřížka se pak výpočítá:

$$x_i = x_{\mathsf{step}} * i + x_{\mathsf{start}} \; i \in \mathbb{N}0 \leq i \leq x_{\mathsf{size}} \tag{3.1}$$

$$y_i = y_{\mathsf{step}} * i + y_{\mathsf{start}} \; i \in \mathbb{N}0 \leq i \leq y_{\mathsf{size}} \tag{3.2}$$

kde $x_{\mathsf{start}}$ je počáteční souřadnice na ose $x$, $y_{\mathsf{start}}$ je počáteční souřadnice na ose $y$, $x_{\mathsf{size}}$ je počet kroků na ose $x$, $y_{\mathsf{size}}$ je počet kroků na ose $y$.

Aby jsme vygenerovali mřížku v daném ohraničení a s daným počtem bodů, určující hustotu mřížky, potřebujeme vypočítat velikost jedné buňky tak, aby byla přibližně čtvercová a počet buňek byl přibližně stejný jako daný počet. Výpočet velikosti mřížky v buňkách vychází ze soustavy rovnic

$$\frac{x_{\mathsf{range}}}{x_{\mathsf{size}}} = \frac{y_{\mathsf{range}}}{y_{\mathsf{size}}} \tag{3.3}$$

$$x_{\mathsf{size}} * y_{\mathsf{size}} = D_{\mathsf{size}} \tag{3.4}$$

kde $x_{\mathsf{size}}$ je počet buněk na ose $x$, $y_{\mathsf{size}}$ je počet buněk na ose $y$, $D_{\mathsf{size}}$ požadovaný počet bodů, $y_{\mathsf{range}}$ je rozdíl maximální a minimální $y$ souřadnice a $x_{\mathsf{range}}$ je rozdíl maximální a

minimální $x$ souřadnice. Potřebujeme tedy získat $x_{\mathsf{size}}$ a $y_{\mathsf{size}}$, po vyjádření pak

$$y_{\mathsf{size}} = \sqrt{\frac{x_{\mathsf{range}} * y_{\mathsf{range}}}{D_{\mathsf{size}}}} \tag{3.5}$$

$$x_{\mathsf{size}} = \frac{D_{\mathsf{size}}}{y_{\mathsf{size}}} \tag{3.6}$$

Nyní již stačí dopočítat kroky

$$y_{\mathsf{step}} = \frac{y_{\mathsf{range}}}{y_{\mathsf{size}}} \tag{3.7}$$

$$x_{\mathsf{step}} = \frac{x_{\mathsf{range}}}{x_{\mathsf{size}}} \tag{3.8}$$

A teď už máme všechny hodnoty potřebné pro vygenerování mřížky podle výše uvedeného vzorce.

Pro různá vstupní data se hodí různé velikosti mřížky a tedy neexistuje nějaký univerzální způsob jak vhodně mřížku zvolit. Jako ne vždy optimální, ale univerzální postup použijeme ten, kdy se mřížka vygeneruje v nejmenším ohraničení, které obsahuje všechny body vstupních dat, tedy jeho hranice jsou nejnižší respektive nejvýšší hodnoty na ose $x$ respektive $y$, obsažené ve vstupních datech, a s počtem bodů podobným jako je ve vstupních datech.

## 3.3 Hledání nejbližšího souseda

Nejjednodušší formou interpolace bodu mřížky je ten, že se jeho výška získá jako výška jeho nejbližšího souseda. Nejprimitivnějším algoritmem je pro každý bod mřížky projít všechny body v datech spočítat jejich vzdálenosti k bodu a zvolit ten nejbližší, ale časová složitost $O(n^2)$ je pro větší množství bodů neúnosná.

Pro více bodů je vhodnější použít sofistikovanější metodu, nabízí se použití Voroného diagramu [11], což je metoda rozdělení prostoru podle množiny bodů $M$ kdy každému bodu $b$ z $M$ se přidělí oblast $O_b$, tak aby vzdálenost všech bodů v $O_b$ k $b$ byla menší než vzdálenost k jakémukoliv jinému bodu v $M$. Nejbližší soused pro bod $x$ v oblasti $O_b$ je tedy bod $b$. Pro nalezení nejbližšího souseda je možné použít metodu[15], která umí prohledat Voroného diagram s časovou složitostí $O(\log n)$.

Tato interpolační metoda se hodí spíše do jiných oblastí použití, než interpolace terénu, kupříkladu v počítačové grafice, je totiž velmi rychlý.

## 3.4 Metody založené na triangulaci

Tyto metody pracují v základě ve dvou krocích: Nejdříve se triangulují vstupní data, a poté se interpoluje v rámci každého trojúhelníku. Tyto metody jsou lokální, pracují uvnitř trojúhelníku, avšak aby dosáhly lepší spojitosti výsledného povrchu berou často do úvahy i okolní trojúhelníky.

### 3.4.1 Delaunayova triangulace

Vstupní data jsou neuspořádané body a pro další zpracování je třeba v těchto bodech vytvořit pravidelnou trojúhelníkovou síť. [11] Pro množinu $n$ bodů $P$ platí, že uvnitř kružnice opsané libovolnému trojúhelníku neleží žádný jiný bod množiny $P$.

Delaunayova triangulace má velice blízko k voroného digramu a z hlediska teorie grafů je jeho duálním grafem. Voronův diagram [11] je dekompozice prostoru, určeného vzdálenostmi k množině objektů v prostoru. Tyto objekty jsou nazývány tzv. generátory a ke každému objektu je přiřazena voroného buňka, což je množina bodů, jejichž vzdálenost od objektu není větší než vzdálenost od ostatních objektů.

Pro výpočet triangulace existuje mnoho metod my, ale použijeme metodu publikovanou v [8], která vypočítává triangulaci na základě výpočtu konvexního obalu. Tato metoda je použita v programu *Qhull* [1], který se používá v mnoha matematických software například *Matlab*, *GNU Octave* či *Mathematica*.

### 3.4.2 Hledání trojúhelníku obsahující bod

Při interpolaci metodami založenými na triangulaci je časově nejnáročnejší výpočet, hledání trojúhelníku obsahující bod, který chceme interpolovat. Naivní algoritmus lineárního prohledávání, se složitostí $O(n^2)$ je pro více bodů zcela neefektivní, je tedy nezbytné použít nějakou formu optimalizace. Pro obecné triangulace je možno použít algoritmy pro rozdělení prostoru jako $k$-d stromy. V naší práci, ale používáme Delaunayovu triangulaci a pro ní můžeme použít jiný algoritmus, který využívá datové struktury vytvořené triangulací pro efektivní vyhledání trojúhelníku, navíc není potřeba pracně vytvářet žádné další datové struktury jako v případě $k$-d stromů.

Pro vyhledání trojúhelníku v Delaunayově triangulaci použijeme základní algoritmus popsaný v [17] jako „walking method“. Idea je následující: Mějme Delaunayovu triangulaci $\mathcal{D}$ množiny $X$ z $n$ bodů v $\mathbb{R}^d$ a hledaný bod $q$. Aby jsme jsme našli simplex v $\mathcal{D}$ obsahující bod $d$, začneme s libovolným simplexem v $\mathcal{D}$ a poté se posunume na sousedící simplex ve směru k bodu $q$. K tomu aby byla tato metoda efektivní je třeba, aby interní reprezentace $\mathcal{D}$ umožňovala konstantní přístup k sousedním simplexům.

Tato metoda je konečná pouze pro Delaunayovu triangulaci, pro ostatní triangulace může skončit v nekonečném cyklu.

Listing 3.1: Walking method pseudokód

```
P = hledany_bod;
Ti = nahodny_trojuhelnik();
while (true) {
    if (bod_v_trojuhelniku(P, Ti))
        return Ti;
    else {
        Pi = bod_uprostred_trojuhelniku(Ti);
        Ei = najdi_hranu(Ti, Pi - P);
        Ti = prilehajici_trojuhelnik(Ei);
    }
}
```

### 3.4.3 Lineární interpolace

U této metody se výška bodu v mřížce zvolí lineární interpolací sousedících bodů ve vstupních datech. Postup je takový, že se na vstupní body použije Delaunayova triangulace a poté se pro každý bod v mřížce získá trojúhelník, ve kterém bod leží, a poté se jeho výška

[1] http://www.qhull.org

spočítá lineární interpolací výšek bodů trojúhelníku. U této metody je interpolace definována pouze pro body mřížky nacházející se uvnitř konvexního obalu vstupních dat, protože bod z mřížky musí ležet v nějakém trojúhelníku vzniklém delaunayovou triangulací.

Mějme trojúhelník $T$ s vrcholy $V_1, V_2, V_3$ pak lze libovolný bod $B$ v trojúhelníku zapsat jako

$$B = w_1V_1 + w_2V_2 + w_3V_3 \tag{3.9}$$

kde $w_1, w_2, w_3$ jsou koeficienty udávající váhu jednotlivým bodům trojúhelníku. Tyto váhy se nazývají barycentrické souřadnice [21]. Pro výpočet souřadnic bodu $B$ musíme převést souřadnice bodu $B$ do barycentrických souřadnic a ty pak dosadit do výše zmíněné rovnice. Pro převod můžeme využít toho, že souřadnice jsou homogenní a platí

$$w_1 + w_2 + w_3 = 1 \tag{3.10}$$

můžeme tedy jednu dopočítat ze zbylých dvou

$$w_3 = 1 - w_1 - w_2 \tag{3.11}$$

dále dle [2] pro $B$ se souřadnicemi $(x, y)$

$$x = w_1x_1 + w_2x_2 + w_3x_3 \tag{3.12}$$
$$y = w_1y_1 + w_2y_2 + w_3y_3 \tag{3.13}$$

zavedeme substituci 3.11 z čehož dostaneme

$$w_1(x_1 - x_3) + w_2(x_2 - x_3) + x_3 - x = 0 \tag{3.14}$$
$$w_1(y_1 - y_3) + w_2(y_2 - y_3) + y_3 - y = 0 \tag{3.15}$$

což je lineární transformace, která může být zapsána jako

$$\mathbf{T} \cdot w = B - V_3 \tag{3.16}$$

kde $w$ ke vektor barycentrických souřadnic, $B$ je vektor kartézských souřadnic a $\mathbf{T}$ je matice

$$\mathbf{T} = \begin{pmatrix} x_1 - x_3 & x_2 - x_3 \\ y_1 - y_3 & y_2 - y_3 \end{pmatrix} \tag{3.17}$$

jelikož $V_1 - V_3$ a $V_2 - V_3$ jsou lineárně nezávislé vektory, můžeme výpočítat inverzní matici k $\mathbf{T}$

$$\begin{pmatrix} w_1 \\ w_2 \end{pmatrix} = \mathbf{T}^{-1}(B - V_3) \tag{3.18}$$

Je tedy vidět, že převod do barycentrických souřadnic jsme zredukovali na problém výpočtu inverzní matice k $\mathbf{T}$, což je v našem případě $2 \times 2$ matice jednoduché. Výsledný vzorec pro výpočet barycentrických souřadnic je následující

$$w_1 = \frac{(y_2 - y_3)(x - x_3) + (x_3 - x_2)(y - y_3)}{\mathsf{det}(T)} \tag{3.19}$$
$$w_2 = \frac{(y_3 - y_1)(x - x_3) + (x_1 - x_3)(y - y_3)}{\mathsf{det}(T)} \tag{3.20}$$
$$w_3 = 1 - w_1 - w_2 \tag{3.21}$$

Barycentrické souřadnice se mohou použít také k testu zda bod leží v trojúhelníku, bod se převede do barycentrických souřadnic testovaného trojúhelníku. Pro každý bod uvnitř a na hraně trojúhelníku platí $0 \leq w_i \leq 1$ pro $i = 0, 1, 2$, což by šlo využít při hledání trojúhelníku obsahující bod, ale námi použitá metoda toto nevyžaduje viz. 3.4.2.

Lineární interpolace pracuje pouze uvnitř trojúhelníku a tudíš nevytváří hladké povrchy, její třída spojitosti je pouze $C^0$, není to tedy úplně ideální interpolační funkce, ale jednoduchá a tudíš velice rychlá.

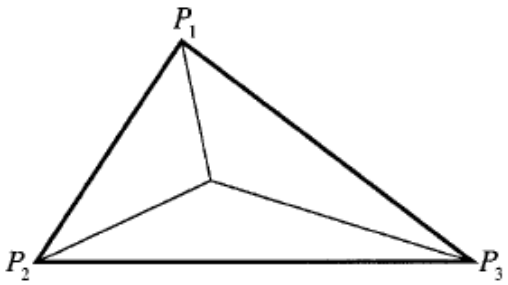
### 3.4.4 Clough-Tocherova metoda

[7] Tato metoda se snaží dosáhnout spojitosti třídy $C^1$, Pro každý bod vstupních dat potřebuje nejen jeho pozici a výšku, ale také gradient což jsou v našem případě buď $x$ $y$ parciální derivace nebo tečná rovina či normálový vektor. Existuje mnoho metod na výpočet gradientů v neuspořádaných datech, jejích porovnání a vyhodnocení lze nalézt v [19]. Tato metoda interpoluje pomocí bikubických polynómů[2]

Aby dosáhla spojitosti třídy $C^1$ vyžaduje pro každý trojúhelní, výšku a gradient (tzn. dvě parciální derivace) v každém vrcholu a jednu normální derivaci ve středu všech hran. Cílem je stanovit všech těchto dvanáct kritérií (tři pro každý vrchol a jedna pro každou hranu), bohužel to není možné, jelikož bikubický polynóm je definován pouze deseti koeficienty, a tedy nám zbyly ještě dvě kritéria nesplněna.

$$F(x, y) = a_1x^3 + a_2x^2y + a_3xy^2 + a_4y^3 + a_5x^2 + a_6xy + a_7y^2 + a_8x + a_9y + a_{10} \quad (3.22)$$

Existuje, ale způsob jak docílit splnění všech kritérií, rozdělíme každý trojúhelník na tři *minitrojúhelníky* se společným bodem v těžišti původního (obrázek 3.3) a vypočítáním bikubických polynómů v každém z nich.




Obrázek 3.3: [7] Při použití Clough-Tocher metody je každý trojúhelník rozdělen na tři menší.

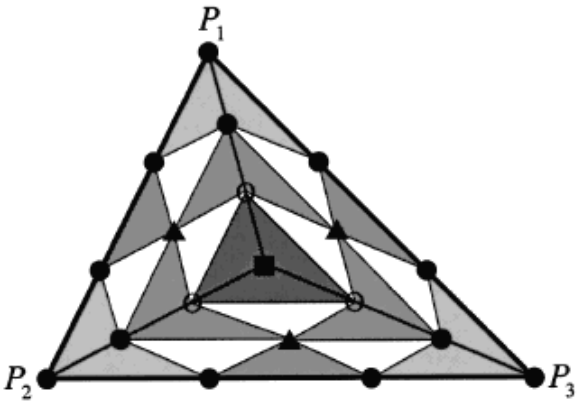
Každou bikubickou plochu rozdělíme na tři podplochy, což jsou kubické Beziérovy plochy nad každým *minitrojúhelníkem*. Tyto Beziérovy plochy jsou zobecněním Beziérových křivek ve 2D. Použijeme trojúhelníkové Beziérovy plochy, jež jsou definovány pomocí Bernsteinových polynómů, proto se nazývají Bernstein-Beziérovy polynómy. Kubický Bernstein-Beziér polynom je definován jako:

$$\begin{aligned} p(u, v, w) =& b_{3,0,0}u^3 + 3b_{2,1,0}u^2v + 3b_{1,2,0}uv^2 + b_{0,3,0}v^3 + 3b_{0,2,1}v^2w + 3b_{0,1,2}uw^2 + b_{0,0,3}w^3 \\ &+ 3b_{1,0,2}uw^2 + 3b_{2,0,1}u^2w + 6b_{1,1,1}uvw \end{aligned} \quad (3.23)$$

[2] polynom 3. stupně s dvěma proměnými

kde $u$, $v$ a $w$ jsou barycentrické souřadnice uvnitř trojúhelníku $P_1P_2P_3$, a koeficienty $b_{i,j,k}$ jsou tzv. Beziérovy souřadnice. Tento polynom má, ale také pouze deset koeficientů, a to nám nestačí.

Beziérovy souřadnice tvoří uvnitř každého *minirojúhelníku*, triangulační sít a dělí tak každou hranu podtrojúhelníku na třetiny. Tuto síť nazveme *kontrolní síť*. Tyto další malé trojúhelníky, které jsou tvořeny *kontrolní sítí*, nazveme *mikrotrojúhelníky*. Kritéria, která jsme stanovili pro spojitost třídy $C^1$ mohou být interpretována jako 2D zobecnění kritérií použitých pro zajištění hladkosti 1D Beziérovi křivky: Jako musí být v 1D, části přímky na obou stranách kolineární, musí být všechny sousední *mikrotrojúhelníky* ležící na obou stranách *minitrojúhelníku* koplanární[3]. Z toho také vyplývá, že jsou koplanární také *mikrotrojúhelníky* obklopující vrchol *minitrojúhelníku*. To lze vidět na obrázku 3.4, kde je každý šedivý pár *mikrotrojúhelníků* koplanární.




Obrázek 3.4: [7] Pohled na původní trojúhelník, s jeho třemi *minitrojúhelníky* a 19-ti Beziérovy souřadnicemi (kontrolními body). Každý *minitrojúhelník* se skládá z devíti *mikrotrojúhelníků*. *mikrotrojúhelníky*, které mají společnou hranu a leží ve stejné rovině, jsou znázorněny šedivě.

Polohy deseti Beziérových souřadnic, uvnitř každého *minitrojúhelníku* jsou plně určeny *kontrolní sítí*: jsou ve vrcholech *minitrojúhelníku*, ve 1/3 a 2/3 každé hrany a uprostřed. Výšky Beziérových souřadnic se určují z našich původních vstupů: Výšky a gradientu každého bodu $P_i$ a sklonu normály v bodech uprosřed hran. Výšky Beziérový souřadnic na 3.4 označených „●“, závisí přímo na zdrojových datech vrcholů $P_1$, $P_2$ a $P_3$: například výšky bodů v $\frac{1}{3}$ a $\frac{2}{3}$ hrany $P_1P_2$, jsou určeny tečnou rovninou v $P_1$ respektive $P_2$. Souřadnice označené „▲“ mohou být určeny z derivace normály v bodech uprostřed hran. Souřadnice označené „◯“ určíme pomocí tří přilehlých ● a ▲, které byly již vypočítány. A nakonec prostřední bod „■“, je určen třemi ◯, jelikož centrální *mikrotrojúhelníky* kolem ■ jsou koplanární.

Přesné vzorce pro výpočet Beziérových souřadnic $b_{i,j,k}$ jsou uvedeny v [14]. Jakmile tyto hodnoty známe můžeme je dosadit zpět do rovnice 3.23, tím získáme Beziérovu plochu pro daný *mikrotrojúhelník*, vypočtením ploch pro všechny trojúhleníky získáme $C^1$ spojitý povrch, definovaný přes naše vstupní neuspořádaná data.

[3] ležící v jedné rovině

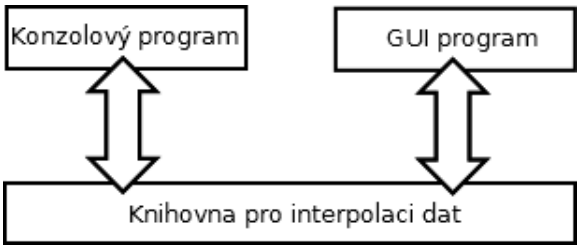
# Kapitola 4

# Návrh nástroje

Jedním z cílů této práce bylo navrhnout a implementovat nástroj pro interpolaci GPS dat. Nástroj musí umět načíst data ve formě gps stop, interpolovat je zvoleným interpolačním algoritmem, uložit do souboru, případně i zobrazit.

## 4.1 Architektura nástroje

Jelikož nástroj má i grafický výstup a zároveň je vhodné, aby si uchoval jednoduché ovládání a možnost začlenit ho do skriptů, bude nástroj obsahovat, jak konzolové tak grafické rozhraní. Aby nedocházelo k duplicitě kódu umístíme jádro programu do knihovny, která bude zajišťovat základní funkce a programy pro uživatelská rozhraní budou tuto knihovnu pouze používat, toto rozdělení je znázorněno na obrázku 4.1.




Obrázek 4.1: Základní architektura nástroje

## 4.2 Vstupní data

Jak již bylo výše uvedeno, vstupní data jsou ve formě GPS stop, uložených v souborech. Nástroj samozřejmě k činnosti potřebuje získat vstupní data, a nabízí uživately několik způsobů jak mu je dodat. Uživatel může přidávat jednotlivé souboru, nebo pokud jich má velké množství, může uživatel pouze zvolit složku a nástroj automaticky nahraje všechny dostupné soubory ze zvolené složky.

Ne vždy má uživatel soubory na lokálním disku a hodilo by se mu, nahrát soubory ze vzdáleného úložiště, nástroj tedy umožňuje uživateli zadat url, kde se soubor nachází, a nástroj soubor automaticky stáhne a nahraje.

Nástroj také podporuje získávání souborů z projektu *OpenStreetMap*, viz 2.2.1. Uživatel zadá „bounding box“ a nástroj automaticky stáhne a nahraje všechny dostupné GPS stopy, které se nacházeji v daném „bounding boxu“.

## 4.3 Výstup

Výstupem nástroje jsou elevační data v pravidelné mřížce a ty musejí být uloženy v nějakém formátu. V zásadě máme dvě možnosti a to binární formáty a textové formáty. Binární formáty mají tu výhodu, že jsou prostorově méně náročné, ale jsou obecně méně přenositelné a nejsou lehce čitelné člověkem. Textový formát se hodí pro případ, kdy chce uživatel rychle něco zkontrolovat, kupříkladu zda se dobře nahráli vstupní data. Obě možnosti mají své výhody a nevýhody, takže budeme umožňovat použít obě možnosti.

### 4.3.1 Výšková mapa

Binárním výstupem je výšková mapa, což je rastrový obrázek sloužící k uložení výškových dat. Hodnota pixelu vyjadřuje výšku příslušného bodu, jehož poloha je určena polohou pixelu. Výšková mapa lze uložit do prakticky libovolného rastrového formátu, který nepoužívá ztrátovou kompresi dat a umožňuje uložit požadovanou přesnost.

Výška může být uložena jako celé číslo, to pak máme možné hodnoty rozloženy rovnoměrně, ale v menším intervalu. Často se toto používá tam, kde chceme, aby byly body pouze v určitých výškových hladinách, nebo chceme menší prostorovou náročnost a nevadí nám menší rozsah hodnot. Výšku můžeme také uložit jako číslo s plovoucí řádovou čárkou, většinou ve formátu *IEEE 754*, o velikosti 32 či 64 bitů, tím získáme mnohem větší rozsah možných hodnot.

Nahraná data jsou v plovoucí řádové čárce, a také výstup interpolačních funkcí je v plovoucí řádové čárce, proto je přirozené, že náš výstup bude také v plovoucí řádové čárce, transformace do celých čísel by byla netriviální a hrozila by také by jsme ztratili přesnost.

#### PNG

*Portable Network Graphics*[20] je rastrový formát dat, používající bezztrátovou kompresi dat. Byl vyvinut jako náhrada za formát *GIF*, který byl zatížen patenty[3], původně bylo *PNG* často vykládáno jako rekurzivní zkratka pro *PNG Not GIF*. Tento formát podporuje volitelnou barevnou hloubku, průhlednost skze alpha kanál, prokládání pixelů, barevné profily a další. Je jednoduchý, standardizovaný a díky bezztrátové kompresi, také malý bez ztráty kvality, proto si rychle získal popularitu hlavně v prostředí internetu.

*PNG* podporuje tři základní typy obrázků: Obrázky ve stupních šedi mají pouze jeden jediný kanál o šířce 1 až 16 bitů. Dalším typem jsou klasické „truecolor“ obrázky, umožňující uložit téměr všechny barvy, co oko rozezná. *PNG* používá additivní barevný model *RGB*, skládající se ze tří barev červené(R), zelené(G) a modré(B), každá se šířkou 8 nebo 16 bitů. Také lze přidat alpha kanál umožňující použít průhlednost, pixel je tedy uložen ve 32 respektive 64 bitech. Posledním typem jsou obrázky obsahující barevnou paletu, kdy pixel neobsahuje přímo barvu, ale index do barvové palety. Tato paleta pak obsahuje seznam všech barev co se mouhou v obrázku použít. Výhodou je značná úspora místa pokud si vystačíme s 256 barvami.

Pro výškovou mapu je *PNG* vhodným formátem, jelikož umožňuje uložit, až 64 bitů na pixel, poskytuje bezztrátovou kompresi a je široce znám a používán. *PNG* neumožňuje větší

obrázky ve stupni šedi než 16 bitů, musíme tedy použít „truecolor“ s alpha kanálem, tím můžeme dosáhnout, až 64 bitů.

#### GeoTIFF

*TIFF* je velmi flexibilní rastrový obrazový formát, skrze takzvané „tagy“ lze ovládat data a strukturu souboru. [18] *GeoTIFF* značí soubory *TIFF*, které obsahují popis geografických dat vložený jako „tagy“ uvnitř *TIFF* souboru. Je to formát metadat, který dovoluje asociovat geografická data s obrazovými daty.

*TIFF* dovoluje ukládat hodnoty pixelů jako čísla v plovoucí řádové čárce. Při ukládání elevačních dat tedy nastavíme, že chceme čísla v plovoucí řádové čárce s jedním kanálem o šířce 32 či 64 bitů. Posléze nastavíme *GeoTIFF* „tagy“ dle [1], kde je uveden příklad „tagů“ pro výškovou mapu, jen změníme „tagy“ *ModelTiepointTag* a *ModelPixelScale*, tak jak jsme si vygenerovali pravidelnou mřížku do které jsme interpolovali.

### 4.3.2 JSON

*JavaScript Object Notation*, definován v [10], je jednoduchý textový formát pro výměnu dat. Syntaxí je založen na *JavaScriptu*, ale není na něm nijak závislý. Používá se k serializaci strukturovaných dat. Je dobře čitelný lidem, a oproti jeho hlavnímu konkurentovi *XML* je „lehčí“, tedy ekvivalentní zápis zabírá méně prostoru, což se nám velmi hodí jelikož exportované soubory budou velké. Je vytvořen ze dvou struktur: Kolekce páru klíč/hodnota, známá též jako *objekt* a ze seznamu hodnot, pro který se jmenuje *pole*.

*JSON* použijeme jak na export výškových dat, tak i dalších dat o průběhu interpolace, jako je proběhlá triangulace, pokud proběhla, či vstupních dat pro kontrolu zda byla dobře nahrána.

## 4.4 Technologie

Nástroj má obsahovat grafické uživatelské rozhraní, výkonově kritickou část zpracovávání dat a také má být nástroj multiplatformí, volba tedy padla na jazyk *C++* a framework *Qt*, který poskytuje kvalitní prostředky nejen pro grafické uživatelské rozhraní, ale třeba i pro komunikaci po síti či spouštění dalších procesů.

### 4.4.1 Qt

*Qt*[2] je multiplatformní aplikační framework, původně vyvíjený firmou *Trolltech*, pro vytváření aplikací s grafickým uživatelským rozraním tak i s čistě konzolovým rozhraním. *Qt* byl původně framework pro grafické aplikace, ale velmi se rozrostrl a nyní podporuje např. přístup k *SQL* databázím, parsování *XML*, vícevláknové programování, komunikace po síti, unifikované API pro přístup k souborům a mnohé další.

Framework je opensource a původně do verze 3.0 byl šířen pod licencí *QPL*, tato licence, ale nebyla kompatibilní s *GPL*, a byla odmítána několika linuxovými distribucemi, což zabraňovalo frameworku v masivnějším rozšíření. Od verze 3.0 byla uvolněna pod *GPL* a již nic nebránilo ji používat v otevřeném softwaru, ale nemohla být použita s proprietárním software, to se změnilo, když byla firma *Trolltech* koupena *Nokií* a knihovna byla uvolněna

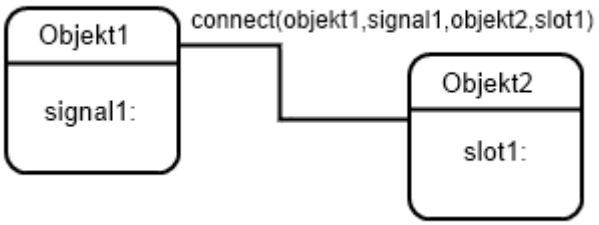
[1] http://www.remotesensing.org/geotiff/spec/geotiff3.html#3.2.3.
[2] http://qt.nokia.com

pod licencí *LGPL*, která již toto v případě dynamického linkování umožňuje. *Qt* může být za poplatek licencováno také pod komerční licencí, která dovoluje modifikovat samotné *Qt*, aniž by bylo nutné dodávat zdrojové kódy a také poskytuje nárok na technickou podporu.

### Signály a sloty

Ústřední vlastností frameworku *Qt* je systém signálů a slotů [5], který slouží ke komunikace mezi objekty. Při programování aplikací s grafickým uživatelským rozhraním dochází k situací, že když se změní stav jednoho widgetu, druhý si o tom přeje být vyrozumněn. Většina frameworku používá ke komunikaci tzv. „callbacky“. Prvnímu objektu, druhý objekt, který si přeje být upozorněn na zprávu, předá ukazatel na funkci „callback“, která se při výskytu události zavolá a tak je zajištěno, že se druhý objekt dozví o události v prvním objektu. „Callbacky“ mají, ale dva zásadní problémy: nejsou typově bezpečné tj. nevíme zda-li první objekt zavolal funkci se správnými argumenty a zadruhé první objekt musí vědět jaký „callback“ má zavolat.

V *Qt* se používá alternativní systém signálů a slotů, kdy signál je vyslán, když nastane událost, a slot je funkce, která se zavolá pro konkrétní signál. Tento systém je typově bezpečný, jelikož kompilátor může zkontrolovat zda jsou signatury signalu a slotu ekvivalentní. A taktéž objekt, který vysílá signál nic neví o slotu, který se zavolá. Ke slotu pak může být připojeno libovolné množství signálů a k signál může být připojen k libovolnému množství slotů. Díky tomuto, je pak objekt opravdu izolován od prostředí a splňuje ideu zapouzdření. Na obrázku 4.2 vidíme základní schéma, kdy máme dva objekty se signálem a slotem, a ty propojí až třetí strana funkcí *connect*.




Obrázek 4.2: Schéma propojení signálů a slotů

### Meta-Object System

Aby *Qt* zajistilo funkčnost signálů a slotů, bylo vytvořeno toto rozšíření jazyka *C++*, které je založeno na třech pilířích: Třída `QObject` ze, které dědí všechny třídy, co chtějí využít toto rozšíření. Dalším pilířem je makro `Q_OBJECT`, které se umístí do privátní sekce třídy a tak zapne podporu pro toto rozšíření. Posledním pilířem je utilita *moc*, která dodá každé podtřídě `QObject` nutný kód pro implementaci toho rozšíření.

Utilita *moc* je vlastně preprocesor, který ve zdrojových souborech jazyka *C++* vyhledá deklarace tříd s makrem *Q_OBJECT* a pro ně vygeneruje další zdrojový *C++* soubor, který obsahuje kód pro *meta-object system*.

K signálům a slotům přibyla typová introspekce, která není závislá na *RTTI* poskytovém *C++* kompilátorem. Poskytuje například vlastní verzi `dynamic_cast` nazvanou `qobject_cast`,

která narozdíl od původní verze funguje přes hranice dynamických knihoven.

### 4.4.2 VTK

*Visualization Toolkit*[3] je open source toolkit pro 3D grafiku, zpracovávání obrazků a vizualizaci vědeckých dat. Její výhodou je integrace s frameworkem *Qt*, skrze widget `QVTKWidget`, kam toolkit vykresluje data, umožňuje nám to mít v jednom okně jak výstup *VTK*, tak prvky grafického uživatelského rozhraní.

Toolkit používáme pro vizualizaci výsledků našeho nástroje a lze tak rychle zhodnotit jak byla interpolace úspěšná. Taktéž si lze vizualizovat, mezikrok interpolace, triangulaci vstupních dat nebo i samotná vstupní data. *VTK* toho umí i daleko více, zvládne také delaunayovu triangulaci, pro kterou jsme museli, ale použít jinou knihovnu, jelikož některé interpolační metody se potřebují například dotazovat na sousední trojúhelníky, a to bohužel ve *VTK* nelze provést.

### 4.4.3 Qhull

*Qhull*[4] je matematický software pro počítání konvexní obálky, delaunayovi triangulace, voroného diagramů a dalších. *Qhull* poskytuje rozhraní pro jazyk *C* v podobě knihovny *libqhull*, její rozhraní je, ale poněkud nezvyklé, silně se podobá volání programu, kdy se nejdříve vytvoří řetezec s parametry, knihovna se „spustí“ a volající si převezme výsledek. Uživatelskou přívětivost naštěstí zlepšuje obálka v jazyce *C++*, knihovna *libqhullcpp* neobsahuje ale vše co její sestra.

[3] http://www.vtk.org
[4] http://www.qhull.org

# Kapitola 5

# Implementace nástroje

## 5.1 Build systém

Kompilátory jsou složitý software, a jako takový poskytují spoustu možností a voleb, jak a s čím je spouštět, proto již od nepaměti vyvojáři používali skripty, které zjednodušovali a automatizovaly každodení aktivity jako: překlad zdrojových souborů, spouštění testů a vytváření dokumentace. Nejpoužívanějším nástrojem se v unixovém prostředí časem stal *Make*. Ten je ale stále příliš neohrabaný a vytvořit *makefile* přenositelný mezi různými systémy, je příliš složité. Vznikly tak systémy, které soubory *makefile* generují na míru stroji, kde se sestavení programu provádí. Mezi tyto nejpoužívanější systémy patří *GNU Automake*, *CMake* či *qmake*.

### 5.1.1 CMake

*CMake*[1] je open source multiplatformní build systém, který dokáže generovat nativní nástroje pro sestavování programů, jako je *Make*, *Microsoft Visual Studio* či *Apple Xcode*. V každém adresáři je umístěn soubor *CMakeLists.txt*, který obsahuje popis toho co se má v daném adresáři vykonat. Proces sestavování tedy probíhá ve dvou fázích: Nejdříve se z konfiguračních souborů *CMakeLists.txt* vygenerují standardní konfigurační soubory a v další fázi se k sestavení programu použije nativní sestavovací nástroj pro zvolenou platformu.

*CMake* je velmi flexibilní, dokáže se poprat s téměř jakoukoli adresářovou strukturou a nečiní mu problém pokud projekt používá několik knihoven. Externí knihovny jsou vyhledávány pomocí tzv. modulů, mnoho jich je obsaženo už ve standardní distribuce, pro některé méně používané knihovny je ale potřeba si takový modul dopsat. *CMake* se během času stal velmi oblíbeným, používá ho například *VTK*, *KDE*, *LLVM* a mnoho dalších.

V našem projektu použijeme schéma s podsložkami, máme složku `lib` s interpolační knihovnou, složku `gui` s grafickým rozhraním a složku `cli`, s rozhraním příkazové řádky. V hlavní složce se nachází konfigurace zasahující celý projekt, např. parametry kompilátoru. V podsložkách máme konfiguraci specifickou pro jednotlivou složku, jako import externích knihoven pomocí `find_package`, volání kompilátoru pomocí `add_executable`, a linkeru `target_link_libraries`.

Jelikož *Qt*, je poněkud komplexnější, má několik generátorů kódu, které je potřeba zavolat, před samotnou kompilací. V [6] je ukázán způsob jak použít *CMake* s *Qt*. V ukázce kódu 5.1 vidíme základní kostru konfiguračního souboru `CMakeLists.txt` pro použití s *Qt*,

[1] http://www.cmake.org

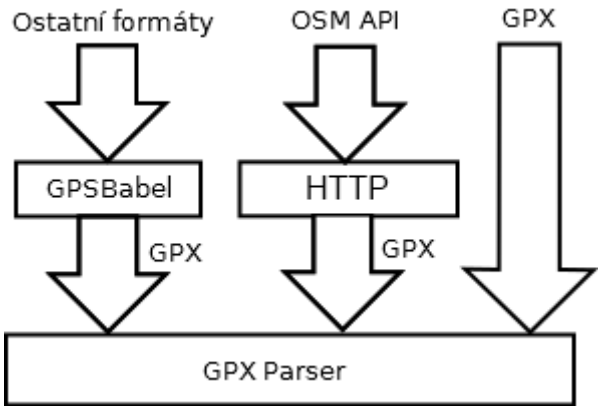
důležitými makry jsou `qt4_wrap_cpp` a `qt4_wrap_ui`, které volají generátory `moc` respektive `uic`.

Listing 5.1: Příklad CMakeLists.txt podporující Qt

```
project(Pokus)

find_package(Qt4 REQUIRED)            # nalezneme Qt
include(${QT_USE_FILE})               # importujeme makra
add_definitions((${QT_DEFINITIONS}) # definice pro preprocesor

# zdrojové soubory
set(Pokus_SOURCES main.cpp mainwindow.cpp)
# hlavičkové soubory pro moc
set(Pokus_HEADERS mainwindow.h)
# soubory s˜návrhem grafického rozhraní
set(Pokus_UIS mainwindow.ui)

# spustíme moc(meta object compiler)
qt4_wrap_cpp(Pokus_HEADERS_MOC ${Pokus_HEADERS})
# spustíme uic(user interface compiler)
qt4_wrap_ui(Pokus_HEADERS_UIS ${Pokus_UIS}

# voláme kompiláror
add_executable(pokus ${Pokus_SOURCES} ${Pokus_HEADERS_MOC} ${Pokus_HEADERS_UIS})
# linker
target_link_libraries(pokus ${QT_LIBRARIES})
```

## 5.2 Vstup

Architektura vstupu je zobrazena na obrázku 5.1, kde lze vidět, že hlavním modulem je GPX parser, který extrahuje data z GPX souboru. Uživatel může nahrávat GPX soubory které jdou přímo do parseru, ale uživatel může také nahrát soubory jiných formátů a ty se pak programem *GPSBabel* soubory překonvertují do GPX souborů, které posléze putují to parseru. Také lze parsovat vzdálené soubory, či použít *OpenStreetMap API*, které komunikuje skrze *Http*.




Obrázek 5.1: Architektura vstupu

### 5.2.1 GPX parser

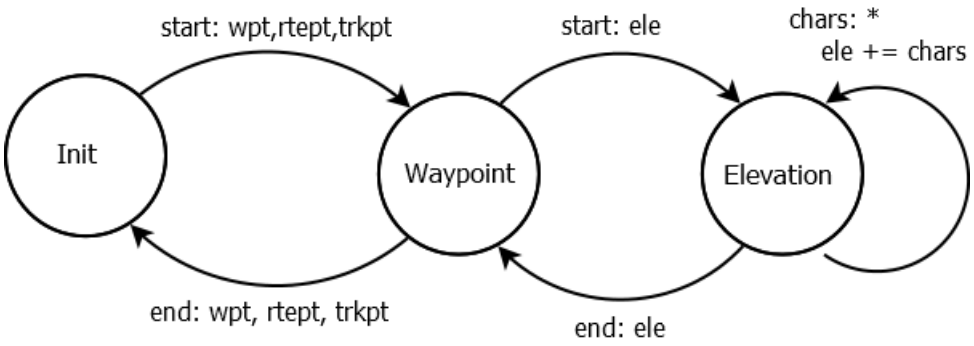
GPX formát je podmnožinou XML a parsuje se pomocí XML parseru. *Qt* obsahuje podporu pro parsování XML, třídou `QXmlStreamReader`, což je rychlý paměťově nenáročný parser podobně jako tradiční *SAX*, ale narozdíl od něj poskytuje mnohem příjemnější *API*. *StAX*, jak se toto *API* nazývá, se liší způsobem jak z parseru získáváme data, *SAX* Vám data dodává voláním „callbacků“, které mu předáte, kdežto v případě *StAX* získáváte data voláním metod parseru. Což je mnohem přímočařejší a řádově čitelnější.

Velkou výhodou *StAX* je možnost vybudovat parser využívající metodu rekurzivního sestupu [12], tj. můžete si kód parseru rozdělit do několika funkcí, kdy každá se stará o parsování jedné úrovně, když pak skončí vrátí se kontrola zpět funkci o úroveň níže, která opět pokud už má hotovo skonší a tak to pokračuje až je celý soubor zpracován. Stav parseru je pak uložen ve formě posloupnosti volání funkcí na programovém zásobníku.

`QXmlStreamReader` je inkrementální parser, dokáže se tedy vyrovnat se situací, kdy dokument není kompletní, protože je přijímán po částech. V případě, že parseru dojdou data, dříve než narazí na konec souboru, skončí s chybou `PrematureEndOfDocumentError`. Až dojdou nová data parser se z chyby zotaví a normálně pokračuje v parsování dál. V našem případě může být GPX soubor čten ze sítě, takže i náš GPX parser musí být inkrementální. Tedy umět začít tam kde přestal.

Zpracování GPX souboru obstarává třída `GpxFileParser`, což je jednoduchá obálka nad `QXmlStreamReader`. Má pouze jednu metodu `parse`, která parsuje soubor jak nejdále to lze a vrátí všechny waypointy, co se jí povedlo načíst.

Při čtení GPX souboru ctíme zásadu maximální benevolence a nijak nekontrolujeme, zda je dodržen přesný formát dat a čteme jen to co je potřeba. V GPX jsou pro nás zajímavé pouze elementy waypointů `wpt, rtpet, trkpt`. Tyto elementy jsou složené a obsahují řadu údajů jako čas, elevaci atp. V našem případě nás zajímá pouze elevace a ostatní přeskakujeme. Elevace je uložena v elementu `ele`, což je listový prvek a obsahuje už pouze text s hodnotou elevace. Na obrázku 5.2 je zobrazen stavový diagram znázorňující postup zpracovávání GPX souboru.




Obrázek 5.2: Stavový diagram parseru

Jelikož budou GPX soubory čteny i ze sítě musí být parser inkrementální, nelze tedy použít metodu rekurzivního sestupu jak je uvedeno výše, parser totiž metodou `parse` zpracovává pouze část souboru, když dojdou data skončí a vrátí kontrolu volajícímu, aby dodal další data, tím ale ztratíme tu posloupnost volání funkcí programu, ve které jsme měli uložen „kontext“ kde se v souboru nacházíme. Uložit si programový zásobník je ve vyšších programovacích jazycích nemožné, proto si vytvoříme vlastní zásobník, kam si budeme ukládat

jména elementů ve kterých jsme zanořeni.

### 5.2.2 GPSBabel

*GPSBabel* je program pro konverzi mezi formáty ukládajícími gps data, my ho použijeme na převod do *GPX*, ze kterého posléze načteme obsažená data. Bohužel neexistuje k programu žádná knihovna a musíme program spouštět skrze příkazovou řádku. Jeho předpis je:

```
gpsbabel -i IFORMAT -f IFILE -o OFORMAT -F OFILE
```

kdy `IFORMAT` je vstupní formát dat, `IFILE` je jméno souboru, ze kterého se budou číst vstupní data, `OFORMAT` je formát výstupu a `OFILE` je jméno souboru kam se zapíší data.

V *Qt* je třída `QProcess`, která slouží pro spouštění externích programů a komunikaci s nimi. Dokáže propojit standardní výstup a chybový výstup s naším programem, díky tomu nemusíme používat žádné dočasné soubory, stačí když *GPSBabelu* řekneme, aby výstup posílal na standardní výstup. Finální příkaz tedy vypadá:

```
gpsbabel -i IFORMAT -f IFILE -o gpx -F -
```

kdy `IFORMAT` a `IFILE` musí dodat uživatel, výstupní formát je `gpx`, který pošleme `GpxFileParseru` a za `OFILE` jsme nahradili `-`, což udává *GPSBabelu*, že má výstup produkovat na standardní výstup.

Pro čtení výstupu programů obsahuje `QProcess` jak synchronní tak asynchronní *API*. *GPSBabel* může produkovat velké množství množství dat, které se bude nějakou chvíli zpracovávat proto využijeme inkrementálnosti *GPX* parseru a budeme číst hned jak to bude možné, tímto také budeme šetřit pamět, výstup externího programu se totiž ukládá do interního bufferu, kde by na nás musela data zbytečně čekat. Musíme tedy použít asynchronní *API*, které používá signály a sloty.

Máme třídu `WaypointFileReader`, která je takovou obálkou nad *GPSBabelem*, má taktéž asynchronní *API*, spouští se metodou `start`, která má jako parametry cestu ke vstupnímu souboru a jeho formát, až skončí vyšle signál `finished`, v případě chyby pak vyšle signál `error` s chybovou hláškou jako parametrem. Po skončení si lze načtené waypointy vybrat metodou `waypoints`.

Když jsou v *GPSBabelu* k dispozici data ke čtení, `QProcess` vyšle signál `readyRead` a když proces skončí vyšle signál `finished`. V naší třídě `WaypointFileReader` si vytvoříme dva interní sloty ke kterým připojíme oba výše zmíněné signály. Ve slotu připojeném k signálu `readyRead` zpracujeme dostupná data `GpxFileParserem`, výsledky si ukládáme do instanční proměnné a po skončení procesu tam budeme mít všechny nahranné waypointy. Až dojde signál `finished` uklidíme a vyšleme vlastní signál `finished`. Při úklidu si musíme dát pozor na to, abychom instanci `QProcessu`, která signál vyslala, nezrušili ihned pomocí operátoru `delete`, ale provedeme zrušení odložené metodou `deleteLater`, pokud totiž `QProcess` v metodě která vyslala signál přistupuje později ještě k `this` způsobilo by to přístup do již zrušeného objektu a pád aplikace.

### 5.2.3 OpenStreetMap

V podkapitole 2.2.1, jsme si ukázali, jak použít *API OpenStreetMap*. Postup je poněkud složitější, takže si to rozebereme postupně.

Prvně musíme nejdříve umět zpracovat GPX soubor, přicházející ze sítě. Postup je velmi podobný jako u *GPSBabel*, akorát místo `QProcess` se použije `QNetworkReply`, což je

odpověd na dotaz poslaný `QNetworkManageru`. `QNetworkReply` stejně jako `QProcess`, dědí třídu `QIODevice`, a poskytuje stejné asynchronní *API*, takže postup čtení dat je stejný. Pro zpracování souboru z url byla vytvořena třída `GpxFileDownloader`, která poskytuje stejné *API* jako `WaypointFileReader`, obě třídy totiž implementují stejnou abstraktní třídu `WaypointLoader`.

Jeden dotaz pro *OpenStreetMap API* vrací stránky o velikosti maximálně 5000 bodů, musíme se tedy dotazovat po jednotlivých stránkách. Stránka pak neobsahuje přímo požadovaná data, nýbrž v tagu `url` je obsažena adresa, kde se nacházejí požadované zdrojové stopy. Zpracování stránky obstarává třída `OsmPageLoader`, která pomocí `QXmlStreamReader`, vyhledá tagy `url`, získá z nich id stopy a stáhne pomocí `GpxFileDownloaderu`. Třída nejdříve z opovědi získá id všech stop, jež jsou ve stánce, a poté je postupně stáhne.

Všechno toto zastřešuje třída `OsmLoader`, která pro zadaný „bounding box“, zavolá `OsmPageLoader` na všechny stránky. Když v dané stránce nejsou již žádné body, vrátí *OSM API* prázdnou stránku, což je validní GPX soubor ve, kterém nejsou žádné body. `OsmLoader` načítá stránky od první stránky s indexem 0, až do chvíle kdy `OsmPageLoader` ohlásí prázdnou stránku.

## 5.3 Výstup

Z hlediska implementace lze výstup rozdělit na tři případy: v *JSON*, *PNG* a *GeoTIFF*.

### 5.3.1 PNG

Pro *PNG* je to jednoduché, *Qt* obsahuje třídu `QImage`, která podporuje ukládání obrázků v *PNG*.

Listing 5.2: Uložení png

```
QImage im(data, width, height, QImage::Format_ARGB32);
if (im.save(fileName, "PNG")) {
    // ... Úspěch ...
} else {
    // ... Neúspěch ...
}
```

Její použití je vidět v ukázce 5.2, kde proměnná `data`, obsahuje ukazatel na obrazová data, `width` je výška obrázku v pixelech, `height` je šířka obrázku v pixelech a `fileName` je cesta k souboru kam se obrázek uloží, platí, že pokud soubor neexistuje je vytvořen. Poslední parametr konstruktoru `QImage` určuje formát obrázku, nastavíme ho na hodnotu `QImage::Format_ARGB32`, která značí použití čtyř kanálů každý po osmi bitech, celkem pak 32 bitů, kam můžeme uložit, jedno číslo v plovoucí řádové čárce s jednoduchou přesností.

### 5.3.2 GeoTIFF

Pro ukládání *GeoTIFF* obrázku potřebujeme dvě knihovny, první která se stará o samotný *TIFF* a druhou která do *TIFF* dokáže uložit tagy *GeoTIFFu*. Těmito knihovnami jsou *libtiff* a *libgeotiff*. Obě jsou napsány v jazyce *C*, naštěstí to tak nevadí, vytváření obrázku je nízkoúrovňová operace sama o sobě a tento nízkoúrovňový přístup není nijak na překážku.

Postup uložení je následující: Otevřeme soubor pro zápis funkcí `TIFFOpen`, poté nastavíme tagy velikosti obrázku, počet kanálů, šířku kanálu, orientaci a formát kanálu, ten nastavíme na čísla v plovoucí řádové čárce ve standardu *IEEE 754*. Tagy *GeoTIFF* jsou privátní

a v základu je *libtiff* odmítné zapsat, musíme je registrovat funkcí `TIFFMergeFieldInfo`. Nyní již můžeme přistoupit k zápisu tagu *GeoTIFF*, inicializujeme *libgeotiff* handlerem z *libtiff* funkcí `GTIFNew`, pak můžeme přistoupit k zápisu tagů funkcí `GTIFKeySet`, tato funkce však přímo tagy nezapisuje pouze je zapíše do interní struktury, která se pak zapíše funkcí `GTIFWriteKeys`. Po zapsání všech tagů, uložíme po řádcích i data obrázku.

### 5.3.3 JSON

Parser *JSON* formátu se dostane do *Qt* až v nadcházející verzi 5 [16]. Přesto je možno s *JSON* pracovat, tak že použijeme javascriptový engine, který *Qt* obsahuje, ale je to zbytečně pracné a výpočetně náročné. Využijeme knihovnu *QJson* [2], která poskytuje velmi příjemné *API*, jak pro čtení tak zápis. Knihovna reprezentuje *JSON* data jako objekty typu `QVariantMap`. *JSON* pole jsou reprezentována instancemi `QVariantList` a *JSON* objekty, instancemi `QVariantMap`.

Exportujeme zdrojové waypointy, triangulaci a interpolované body. O to se stará třída `InterpolationData`, v metodě `serialize` převede výše zmíněná data do `QVariantMap`. Třídy trojúhelníku i body mají také definovanou metodu `serialize`, která převede konkrétní objekt trojúhelníku či bodu na `QVariant`. `InterpolationData` pole těchto objektů převede na `QVariant` a spojí do jednoho `QVariantMap`.

## 5.4 Grafické uživatelské rozhraní

Vedle standardní rozhraní příkazové řádky poskytuje nástroj, také grafické uživatelské rozhraní, které je obecně pro počítačové uživatele přirozenější na ovládání.

Na jeho návrh byl použit nástroj *Qt Designer*, což je standardní návrhář grafických uživatelských rozhraní pro *Qt*. Vytváří `*.ui` soubor, který xml elementy popisuje navrhnuté *GUI*. Při překladu zdrojových souborů, pak utilita `uic` z těchto souborů vytvoří *C++* soubor, kde se jednotlivé widgety vytvářejí a nastavují se jim parametry, jaké jsme si navolili v návrháři.

### 5.4.1 Hlavní okno

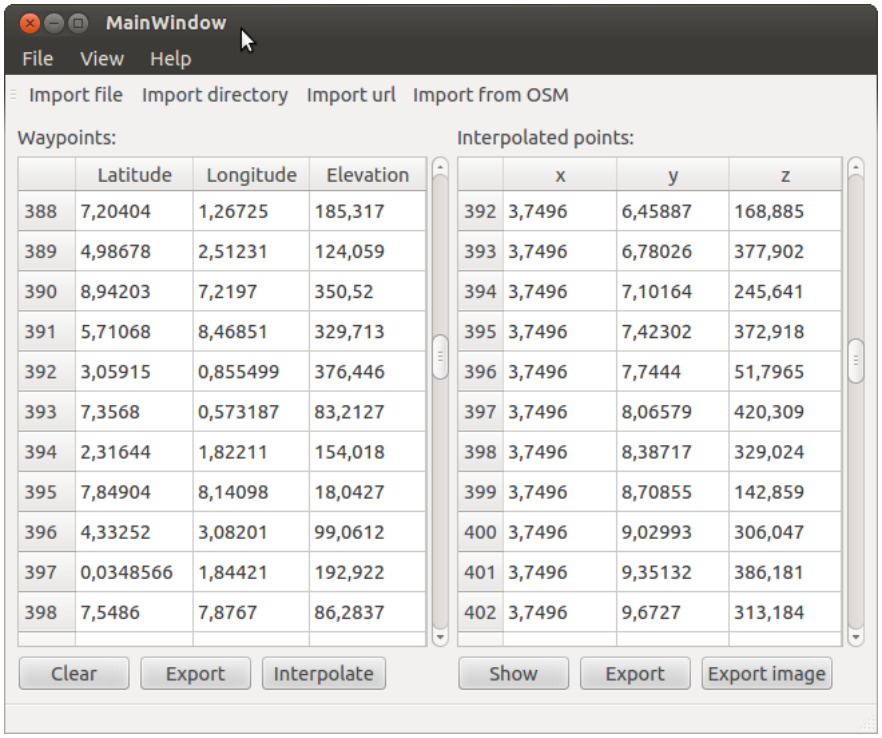
Na obrázku 5.3 je snímek hlavního okna, kde lze vidět základní rozvržení uživatelského rozhraní: Okno je rozděleno svisle na dvě části, ta vlevo obsahuje nahraná vstupní data z GPX souborů a ta vpravo obsahuje interpolovaná data. Import dat je možný několika způsoby, ty jsou uvedeny v menu ve složce *File* a v panelu nástrojů, umístěném v horní části okna. Ve spodní části ona se pak nachází stavový řádek, kterým program informuje o dění např. o dokončení interpolace. V každé části je na spodní hraně několik tlačítek, díky kterým je možno s daty manipulovat: načtená data se dají smazat, exportovat do *Json* a interpolovat. Interpolovaná data mohou být zobrazena nebo exportována.

### 5.4.2 Importování dat

Pro import dat jsou možné celkem čtyři akce:

Na obrázku 5.4, je vidět dialog pro výběr souboru k importu. Zvolí se cesta, kdy je možno tlačítkem *Browse* vyvolat systémový dialog pro výběr souboru, a jeho formát. Typ formátu se volí vybráním z nabízených možností v rozbalovacím seznamu. Pokud se nepodařilo

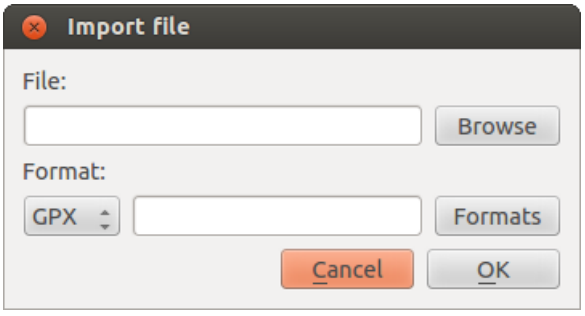
[2] http://qjson.sourceforge.net




Obrázek 5.3: Snímek hlavního okna uživatelského rozhraní

nalést *GPSBabel*, jsou zde pouze dvě možnosti, GPX a *Json*, které program dokáže načíst bez podpory *GPSBabelu*. Vedle rozbalovacího seznamu je ještě přítomno textové pole, kam se zapisují další parametry formátu pro *GPSBabel*, a tlačítko, jež otevře webový prohlížeč se stránkou popisující podporované formáty *GPSBabelu*.

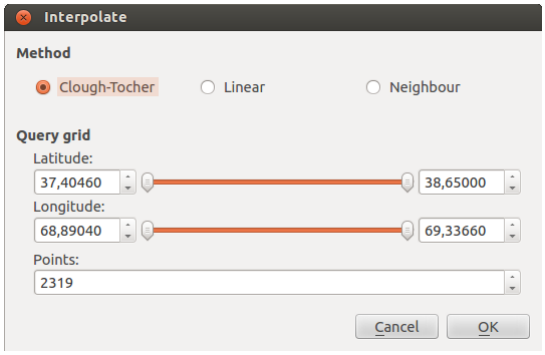
Import složky není zajímavý, pouze se otevře standardní systémový dialog pro zvolení složky. Dále při importu z url, se otevře jednoduchý dialog kam, uživatel zadá url, poté je o postupu informován dialogem s indikátorem průbehu, anglicky „progress bar“. Při importu z *OpenStreetMap* uživatel v jednoduchém dialogu zadá požadovaný „bounding box“ a pak je o postupu opět informován prostřednictvím dialogu s indikátorem průběhu.




Obrázek 5.4: Snímek dialogu pro import souboru

### 5.4.3 Interpolace

Na obrázku 5.5 vidíme dialog pro interpolaci. Ten obsahuje dvě sekce: Výběr metody interpolace pomocí přepínačů. Přepínače patří do jedné sekce a *Qt* nám zajistí, že zvolen bude moci být pouze jeden. Druhou sekcí je výběr velikosti mřížky do které se bude interpolovat. Volí se počet bodů v mřížce a „bounding box“. Ten se volí pomocí dvojitého šoupátka, kdy hraničními hodnotami jsou minimální respektive maximální souřadnice v dané ose. *Qt* standardně obsahuje pouze jednoduché šoupátko, naštěstí v knihovně *Qxt*[3] se takové šoupátko nachází. Po potvrzení dialogu je uživatel informován o postupu interpolace pomocí dialogu s indikátorem průběhu.




Obrázek 5.5: Snímek dialogu pro interpolaci

### 5.4.4 Vizualizace

Vizualizační okno z převážné většiny zaplňuje *QVTKWidget*, kam knihovna *VTK* vykresluje předaná data. Dále jsou v okně přítomny, zaškrtávací tlačítka, kterými se zapíná a vypíná vykreslování, původních dat, triangulace, vyinterpolovaných bodů a trojúhelníkové sítě vyinterpolovaných bodů, a šoupátko pro změnu měřítka na ose z.

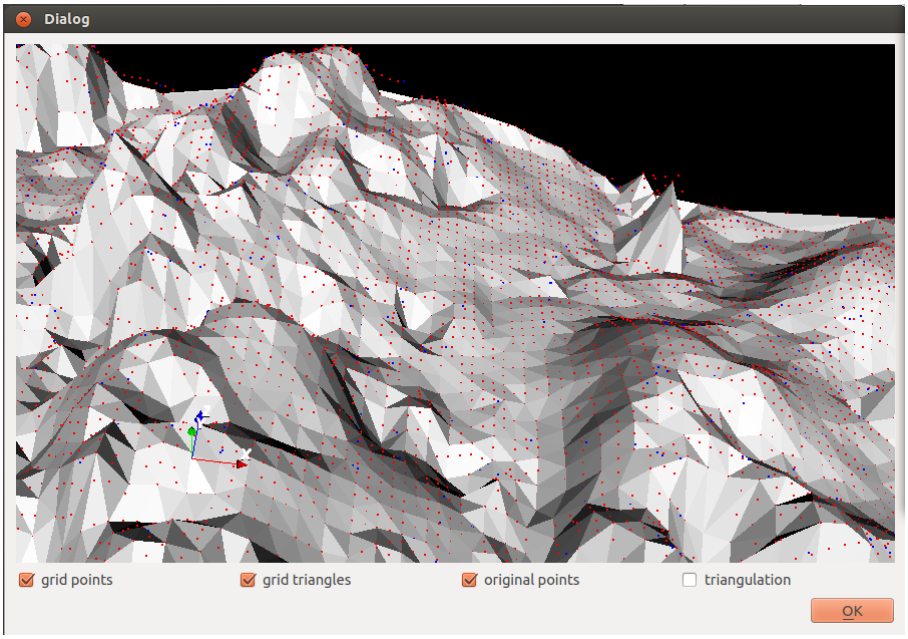
Na snímku 5.6, je vidět vizualizátor se zobrazenou interpolací. Modré tečky jsou původní vstupní data, červené jsou nově vyinterpolované body.

## 5.5 Interpolace

Po dlouhých peripetiích, jsme nakonec pro interpolace použili knihovnu *SciPy*[4], což je jedna z mála knihoven, která má implementovánu *Clough-Tocherovu* interpolaci. Dalšími co jsme našli byl *Quantum GIS*, což je geografický informační systém a ani není jasné zda by šla interpolace nějak jednoduše použít, poslední knihovnou co jsme našli je *libMesh*, u ní ale nebylo jasné jak se používá, třída jež zřejmě interpolaci provádí nebyla nijak komentována.

Knihovna *SciPy* je napsána v pythonu, což na první pohled značí jistou nekompatibilitu, jelikož my jsme za implementační jazyk zvolili C++, naštěstí python má k C velmi blízko, a poskytuje *API* jak spouštět z programu v C/C++ interpret pythonu. Pro *Clough-Tocherovu* interpolaci obsahuje *SciPy* třídu `CloughTocher2DInterpolator`, je to tzv. funktor, tj. objekt s přetíženým operátorem volání funkce. V konstruktoru se mu předají vstupní data,

[3] http://dev.libqxt.org/libqxt/wiki/Home
[4] http://www.scipy.org/




Obrázek 5.6: Snímek vizualizátoru

on vytvoří $C^1$ spojitý povrch definovaný Beziérovými plochami, a pak se ho dotazujeme na hodnoty bodů v pravidelné mřížce. Interně pro triangulaci používá *qhull*, bohužel *API* není úplně ideálně navrženo a nelze například místo spočítání triangulace v konstuktoru, použít triangulaci vlastní, například už vypočtenou jiným interpolátorem. Alespoň je umožněno k triangulaci přistoupit skrze nedokumentovanou proměnou `tri`, jinak by jsme museli zcela zbytečně triangulaci počítat znovu.

Při praktickém testování jsme narazili na situaci, kdy *Clough-Tocherova* metoda generuje extrémní hodnoty, aby jsme omezili dopady takovéto situace, tak pokud *Clough-Tocher* vygeneruje hodnotu mimo minimální a maximální hodnotu ve zdrojových datech, tak použijeme lineární interpolaci, tím sice narušíme $C^1$ spojitost, ale ponechání takto extrémní hodnoty by nám narušilo výstupní data.

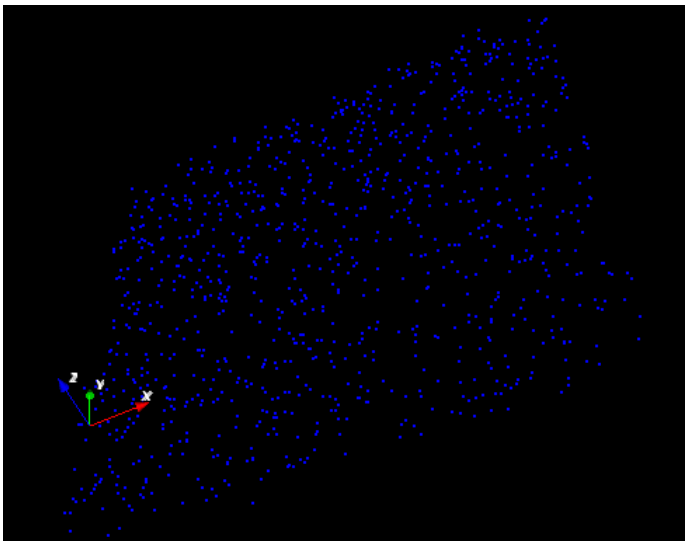
# Kapitola 6

# Testování

Testování probíhalo na třech sadách dat: Soubor dat vytvořený generátorem náhodných čísel s uniformním rozložením, GPX stopy získané z *OpenStreetMap* a data z databáze *Dataz*. Výstupy testování v podobě obrázku vygenerovaného povrchu se nalézají v příloze. Modré tečky v obrázcích jsou původní body ze vstupních dat.

## 6.1 Náhodná data

Byl vygenerován soubor s 1000 náhodnými body, na obrázku 6.1 můžete vidět jejich zobrazení. Každou interpolaci jsme otestovali pro stejný počet bodů jako vstupní soubor, a pro desetinásobek tj. 1000 a 10 000 bodů.




Obrázek 6.1: Soubor náhodně vygenerovaných 3d bodů

První a nejjednodušší interpolací je hledání nejbližšího souseda, v příloze na obrázku A.1, je vidět, že pro malý počet bodů si stojí obstojně, ale při větším počtu bodů vytváří sedliny. Lineární interpolace (obrázek v příloze A.2) je na tom pro malý počet bodů pobodně jako hledání nejbližšího souseda, ale při větším už je to znatelně lepší, stále ovšem to není ideální a mohlo by to být hladší. Metoda *Clough-Tocher* dopadla nejlépe, pro malý

počet bodů vykazuje také stejné výsledky jako ostatní metody, ale při vícero bodech je jednoznačně nejhladší.

## 6.2 OpenStreetMap

*OpenStreetMap* se zdál být dobrým zdrojem dat, jak je ale vidět, pro navržené a implementované interpolační metody, není *OpenStreetMap* vhodným zdrojem: Bodů je sice hodně, ale dat málo, například v „bounding boxu“ o straně několik kilometrů je milión bodů ale všechny jsou v jedné přímce, to pak toho moc nezmůžeme.

Testování probíhalo na stopách z „bounding boxu“ $-0.5, 51.4, 0.7, 52.1$. Zde oproti předchozímu testu, zcela pohožela metoda *Clough-Tocher*, zřejmně při koncentraci bodů do linií trpí, zaokrouhlovacími chybami.

## 6.3 Dataz

Testování probíhalo na asi 2319 bodech, které se nacházejí v okruhu 25 km od města Lanškroun, souřadnice byli v S-JTSK, do WGS84 jsme je převedli pomocí:

```
cs2cs -f "%.10f" +proj=krovak +ellps=bessel +nadgrids=czech +to +proj=longlat
% +datum=WGS84
```

což je doprovodný program pro knihovnu *Proj4*.

# Kapitola 7

# Závěr

Cílem práce bylo navrhnout a implementovat nástroj pro automatické generování výškové mapy, pomocí interpolace jednotlivých bodů. Nejdříve byli shromážděny informace o datech z GPS a o jejich možných zdrojích, poté jsme se věnovali studiju několika interpolačních algoritmů, které se dají použít pro vytvoření povrchu z neuspořádaných dat. Následně jsme se zabývali návrhem nástoje, kde jsme se seznámili s použitými technologiemi a formátem výstupu výškové mapy. Nakonec jsme nástroj naimplementovali a otestovali na třech sadách dat, náhodně generovaných, z *OpenStreetMap* a z databáze *Dataz*.

Tato práce jistě není dost rozsáhlá, aby pokryla celé široké spektrum interpolačních metod, zaměřila se jen na ty nejzákladnější, ve snaze je aplikovat na data z projektu *OpenStreetMap*, bohužel, jak ukazují testy ne zcela uspěla, stejně však vznikl nástroj, který integruje nahrávání GPX stop z různých zdrojů, interpolace a vizualizaci výsledku.

Do budoucna by se mohl nástroj přepsat do jazyka Python ve kterém je i knihovna *SciPy*, volání Pythonu z C, je oproti psaní přímo v Pythonu pomalé, neproduktivní a náchylné na chyby. Mohlo by se také zaměřit na další interpolace, či před samotnou interpolací se pokusit odstanit očividně chybné body, které jsou např. o mnoho výše než ostatní body. Celkově se pokusit při interpolaci zkonsolidovat počet bodů na přijatelnou mez.

# Literatura


[1] *API v0.6 - OpenStreetMap Wiki* [online]. [cit. 1.2.2012]. Dostupné na: <http://wiki.openstreetmap.org/wiki/API_v0.6>.

[2] *Barycentric coordinate system (mathematics)* [online]. [cit. 1.5.2012]. Dostupné na: <http://en.wikipedia.org/wiki/Barycentric_coordinate_system_(mathematics)>.

[3] *GIF is NOW finally free - for real, with a final Unisys joke* [online]. [cit. 10.5.2012]. Dostupné na: <http://www.freesoftwaremagazine.com/node/1772>.

[4] *GPX: the GPS Exchange Format* [online]. [cit. 1.2.2012]. Dostupné na: <http://www.topografix.com/gpx.asp>.

[5] *Signals & Slots* [online]. [cit. 10.5.2012]. Dostupné na: <http://qt-project.org/doc/qt-4.8/signalsandslots.html>.

[6] *Using CMake to Build Qt Projects* [online]. [cit. 13.5.2012]. Dostupné na: <http://qt-project.org/quarterly/view/using_cmake_to_build_qt_projects>.

[7] Amidror, I. Scattered data interpolation methods for electronic imaging systems: a survey. *Journal of Electronic Imaging*. Duben 2002, roč. 11. S. 157–176. Dostupné na: <http://diwww.epfl.ch/w3lsp/publications/other/sdimfeisas.pdf>.

[8] Barber, C. B., Dobkin, D. P. a Hudanpaa, H. The Quickhull algorithm for convex hulls. *ACM Transactions on Mathematical Software*. 1996, roč. 22, č. 4. S. 469–483.

[9] Bémová, K. *Křivky v počítačové grafice* [online]. [cit. 10.5.2012]. Dostupné na: <http://cgg.mff.cuni.cz/ pepca/ref/krivkyBemova.pdf>.

[10] Crockford, D. *The application/json Media Type for JavaScript Object Notation (JSON)* [online]. jul 2006 [cit. 11.5.2012]. Dostupné na: <http://tools.ietf.org/html/rfc4627>.

[11] De Berg, M., Cheong, O., Van Kreveld, M. et al. *Computational Geometry: Algorithms and Applications*. 3. vyd. [b.m.]: Springer-Verlag, 2008. 386 s. ISBN 978-3-540-77973-5.

[12] Češka, M., Hruška, T. a Beneš, M. *Překladače* [online]. [cit. 12.5.2012]. Dostupné na: <http://www.fit.vutbr.cz/ meduna/fjp/skripta.pdf>.

[13] Fabio, R. From point cloud to surface: the modeling and visualization problem. *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*. 2003, roč. 34. Dostupné na: <http://www.pf.igp.ethz.ch/tarasp_workshop/papers/remondin.pdf>.

[14] Farin, G. E. *Curves and Surfaces for Computer-Aided Geometric Design: A Practical Code*. 4th. Orlando, FL, USA: Academic Press, Inc., 1996. ISBN 0122490541.

[15] Kirkpatrick, D. Optimal seach in planar subdivisions. *SIAM J. on Computing*. 1983, roč. 12. Dostupné na: <http://www.cs.princeton.edu/courses/archive/fall05/cos528/handouts/Optimal%20Search%20In%20Planar.pdf>.

[16] Knoll, L. *Qt 5 Alpha* [online]. [cit. 13.5.2012]. Dostupné na: <http://labs.qt.nokia.com/2012/04/03/qt-5-alpha/>.

[17] Mücke, E. P., Saias, I. a Zhu, B. Fast randomized point location without preprocessing in two- and three-dimensional Delaunay triangulations. In *Proceedings of the twelfth annual symposium on Computational geometry*. New York, NY, USA: ACM, 1996. S. 274–283. SCG '96. Dostupné na: <http://doi.acm.org/10.1145/237218.237396>. ISBN 0-89791-804-5.

[18] Ruth, M. *GeoTIFF FAQ* [online]. [cit. 10.5.2012]. Dostupné na: <http://www.remotesensing.org/geotiff/faq.html>.

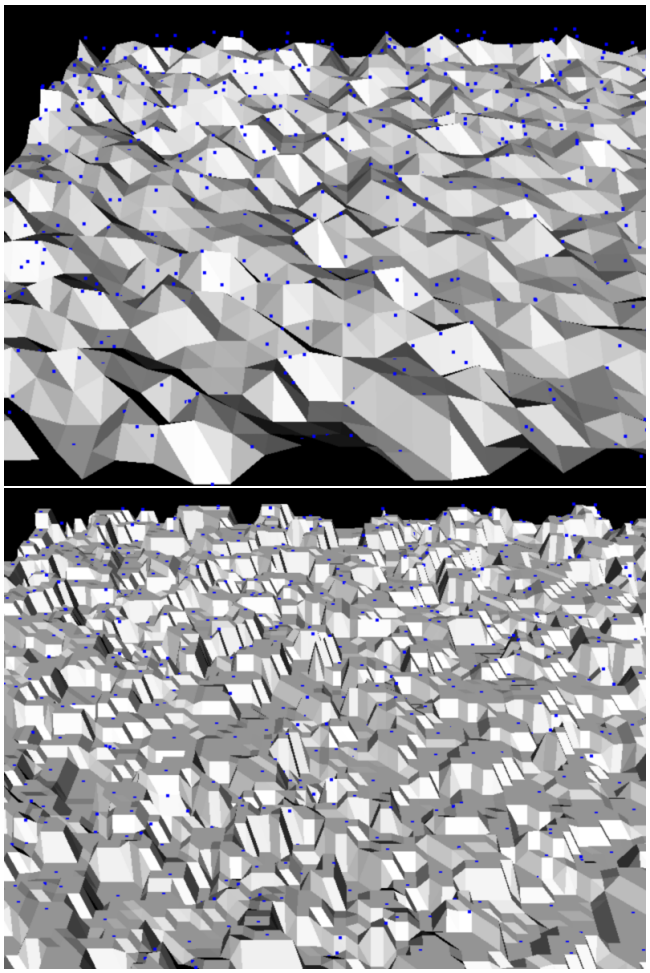
[19] Stead, S. E. Estimation of gradients from scattered data. *The Rocky Mountain Journal of Mathematics*. 1984, roč. 14. S. 265–280. Dostupné na: <http://projecteuclid.org/euclid.rmjm/1250127676>.

[20] Tišnovský, P. *PNG is Not GIF* [online]. [cit. 10.5.2012]. Dostupné na: <http://www.root.cz/clanky/png-is-not-gif/>.

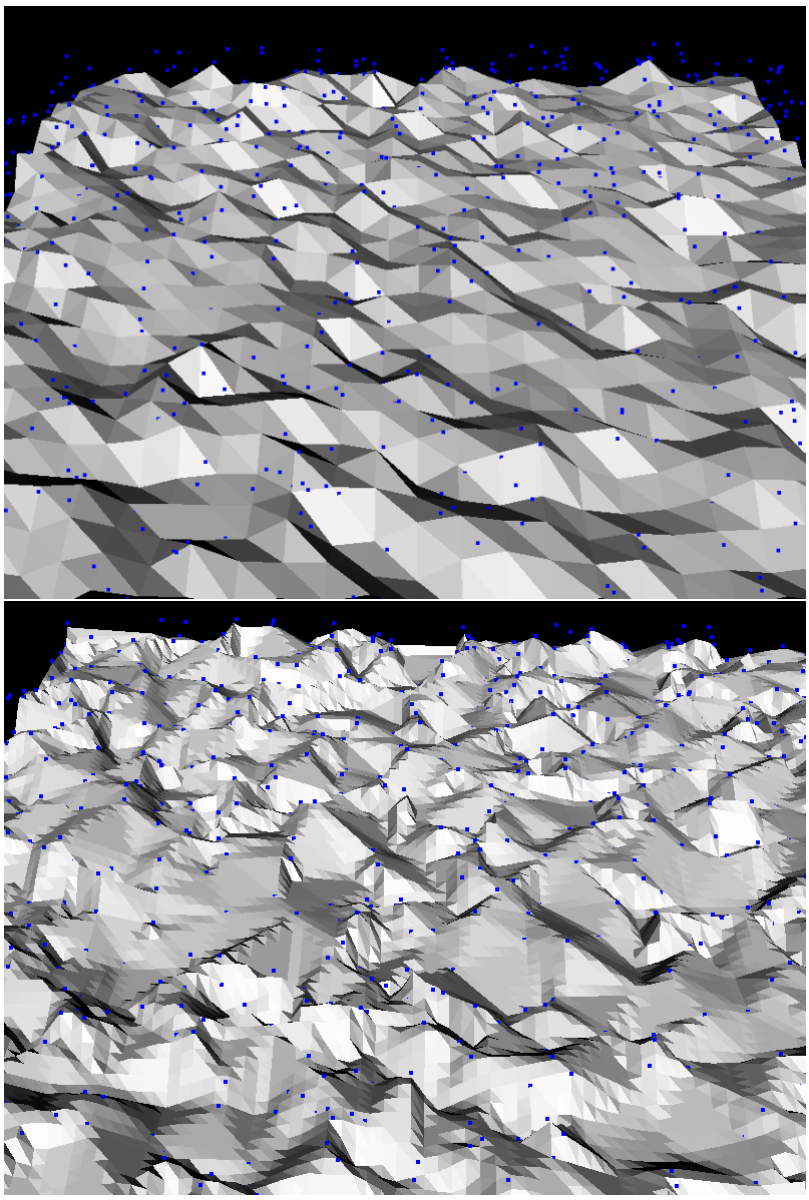
[21] Weisstein, E. W. *Barycentric Coordinates – from Wolfram MathWorld* [online]. [cit. 1.2.2012]. Dostupné na: <http://mathworld.wolfram.com/BarycentricCoordinates.html>.

# Příloha A

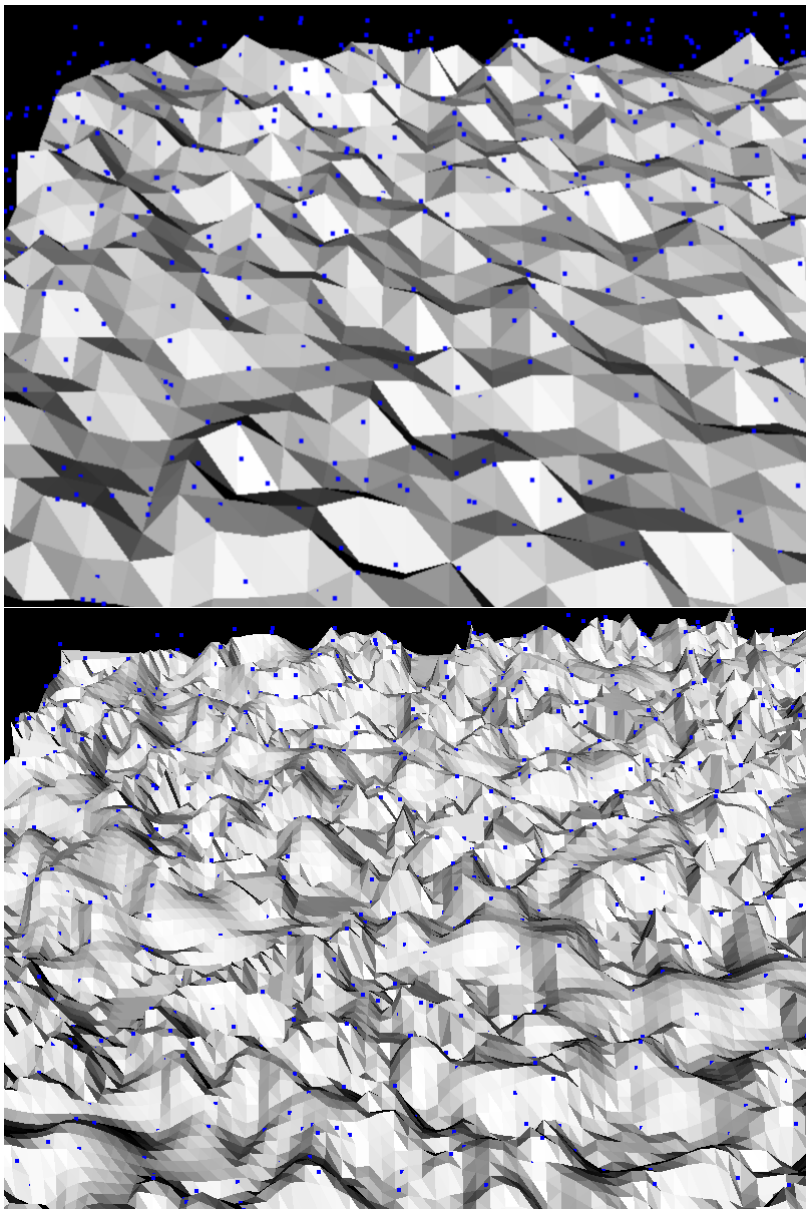
# Testovací výstupy



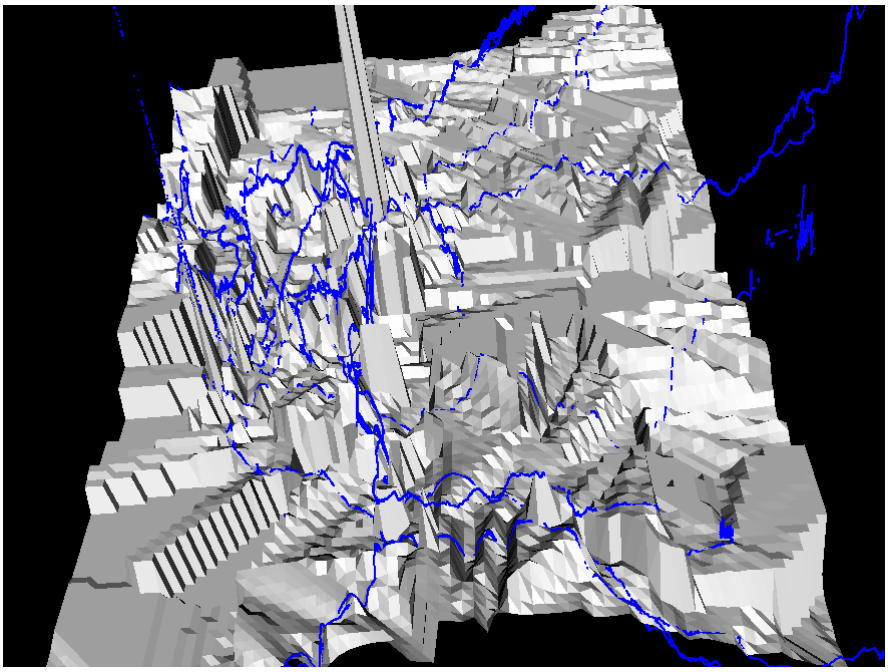
Obrázek A.1: Interpolace pomocí nejbližšího souseda pro 1000 respektive 10 000 bodů



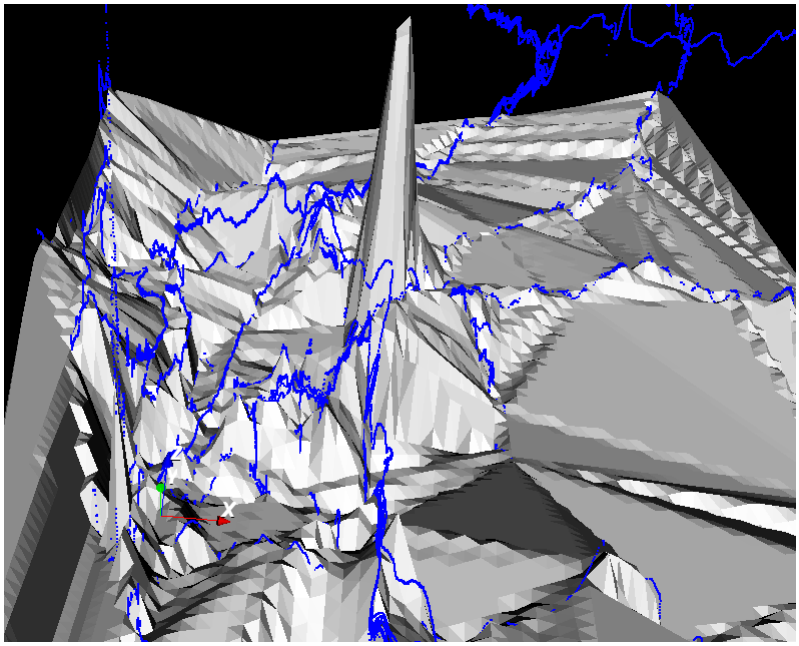
Obrázek A.2: Lineární interpolace pro 1000 respektive 10 000 bodů



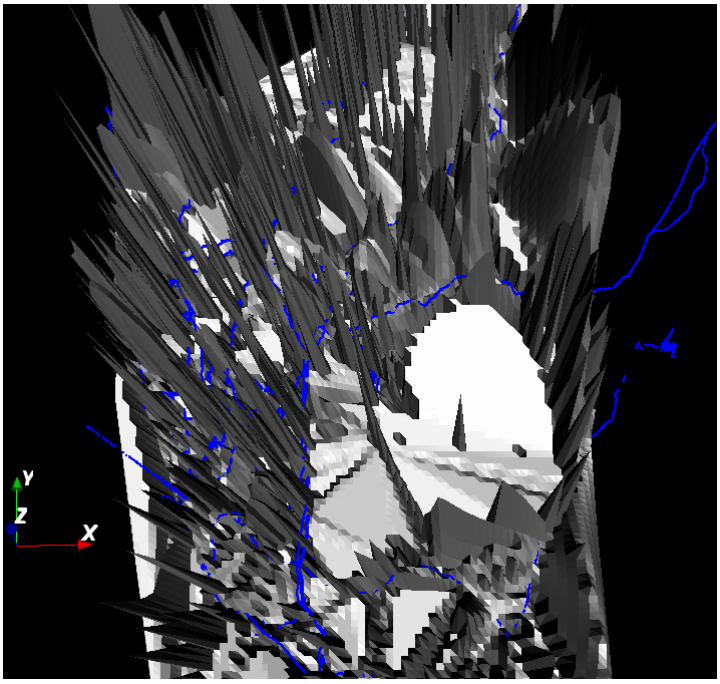
Obrázek A.3: *Clough-Tocher* interpolace pro 1000 respektive 10 000 bodů



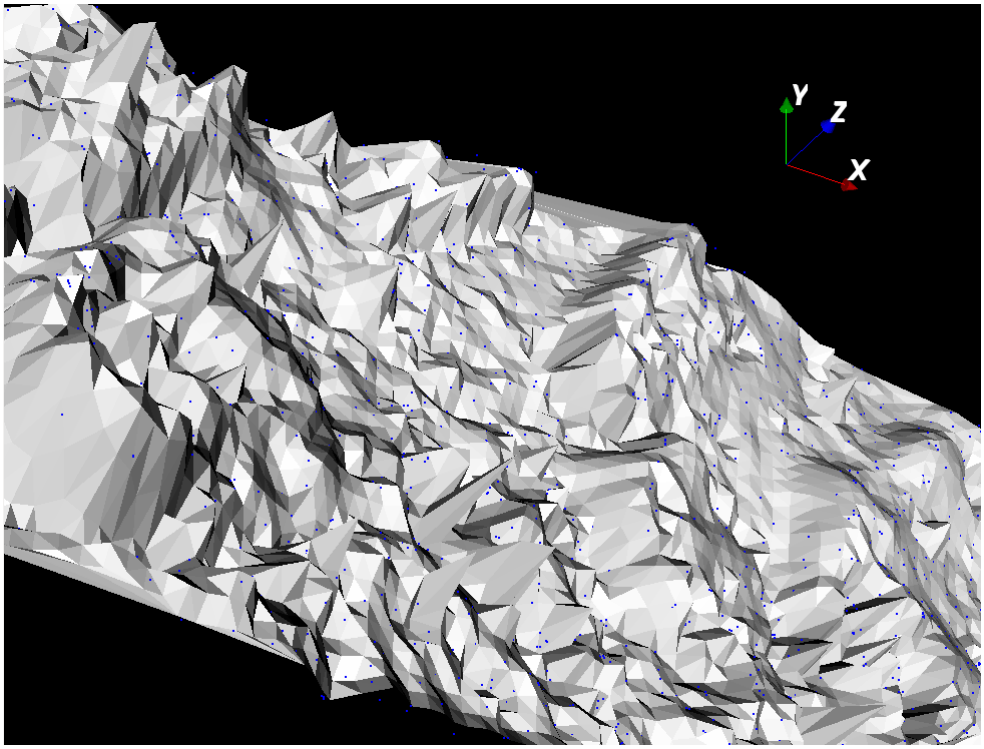
Obrázek A.4: Interpolace hledáním nejbližšího souseda pro data z *OpenStreetMap*



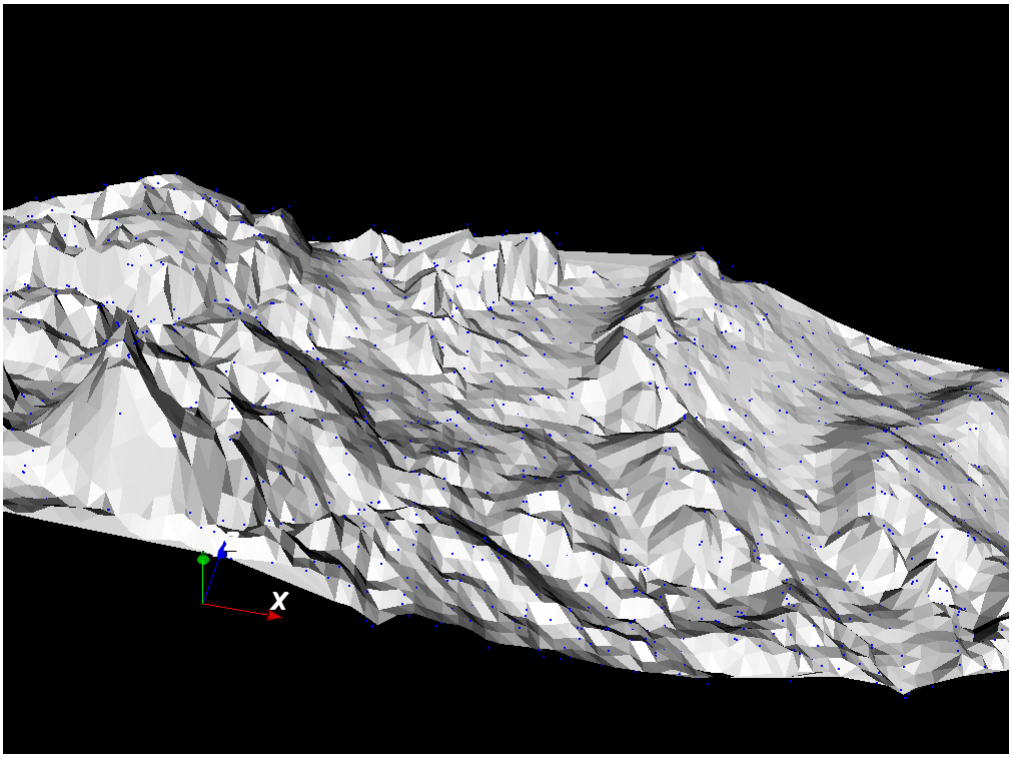
Obrázek A.5: Lineární interpolace pro data z *OpenStreetMap*



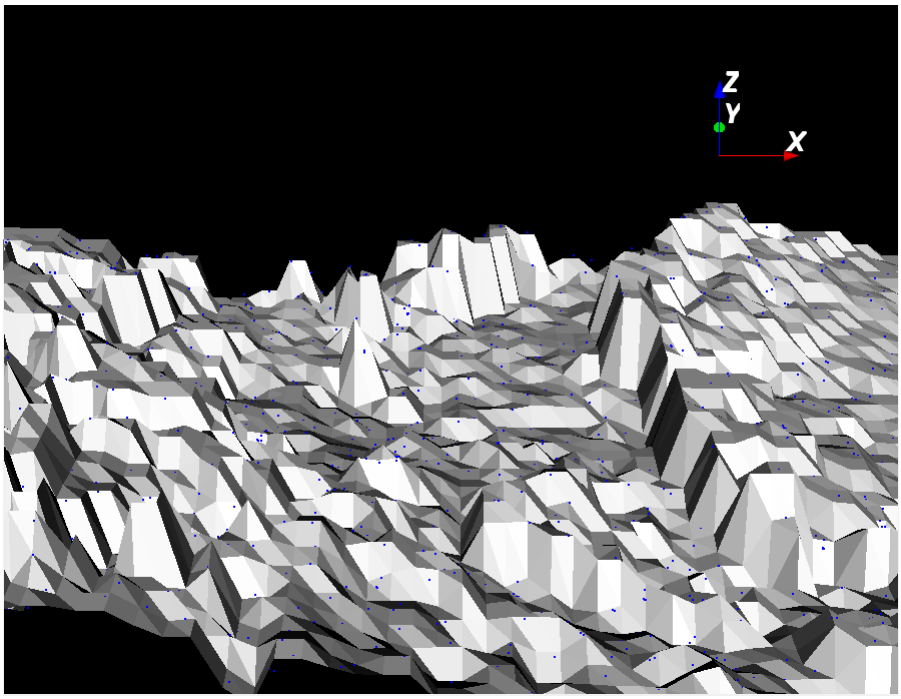

Obrázek A.6: *Clough-Tocher* interpolace pro data z *OpenStreetMap*




Obrázek A.7: *Clough-Tocher* interpolace pro data z databáze *Dataz*




Obrázek A.8: Lineární interpolace pro data z databáze *Dataz*




Obrázek A.9: Metoda hledání nejbližšího souseda pro data z databáze *Dataz*

# Příloha B

# Obsah CD

1. Zdrojové soubory programu a této práce.
2. Zdrojové soubory této práce ve formátu LaTeX.
3. Výsledný text této práce ve formátu PDF.
4. Testovací data.