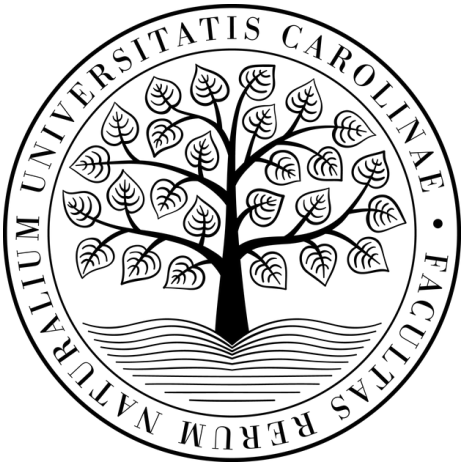
Algoritmy počítačové kartografie

Geometrické vyhledávání bodu



Horák Pavel
Tomášková Eliška

Přírodovědecká fakulta
Univerzita Karlova

březen 2024

Zadání

Vstup: Souvislá polygonová mapa n polygonů $\{P_1, ..., P_n\}$, analyzovaný bod q.

Výstup: $P_i, q \in P_i$.

Nad polygonovou mapou implementujete Ray Crossing Algorithm pro geometrické vyhledávání incidujícího polygonu obsahujícího zadaný bod q.

Nalezený polygon graficky zvýrazněte vhodným způsobem (např. vyplněním, šrafováním, blikáním). Grafické rozhraní vytvořte s využitím frameworku QT.

Pro generování nekonvexních polygonů můžete navrhnout vlastní algoritmus či použít existující geografická data (např. mapa evropských států).

Polygony budou načítány z textového souboru ve Vámi zvoleném formátu. Pro datovou reprezentaci jednotlivých polygonů použijte špagetový model.

Hodnocení

V rámci této úlohy byly řešeny následující kroky:

Krok	**hodnocení**
Detekce polohy bodu rozšiřující stavy uvnitř, vně polygonu.	10 b.
Analýza polohy bodu (uvnitř/vně) metodou Winding Number Algorithm.	*+10 b.*
Ošetření singulárního případu u Winding Number Algorithm: bod leží na hraně polygonu.	*+ 5 b.*
Ošetření singulárního případu u Ray Crossing Algorithm: bod leží na hraně polygonu.	*+ 5 b.*
Ošetření singulárního případu obou algoritmů: bod je totožný s vrcholem jednoho či více polygonů.	*+ 2 b.*
Zvýraznění všech polygonů pro oba výše uvedené singulární případy.	*+ 3 b.*
Celkem.	35 b.

1 Popis a rozbor problému

1.1 Point Location Problem

Point Location Problem je jedním z nejdůležitějších problémů počítačových aplikací, ve kterých jsou využity geometrické struktury. V rovině je zadána množina n bodů, jež tvoří vrcholy mnohoúhelníků Pj, a dále je určen bod q. Cílem problému je najít takový mnohoúhelník, který obsahuje zmíněný bod q. V praktickém životě člověka jde tedy také o velice důležitou věc, neboť často lidé potřebují znát svoji polohu vzhledem k různým objektům. Point Location Problem bývá často řešen v GIS.

Zmíněný problém z hlediska automatizace není vůbec snadný, neboť datasety mohou být tvořeny stovkami tisíc i miliony mnohoúhelníků, tudíž jde pak o časově velice náročné algoritmy. Je potřeba nalézt datové struktury a algoritmy, jež umožňují problém řešit efektivně pro konvexní i nekonvexní mnohoúhelníky. Existují dva základní algoritmy, které jsou schopny určit vzájemnou polohu bodu a uzavřených mnohoúhelníků a zároveň jejichž použití je vhodné pro nekonvexní mnohoúhelníky. Jde o *Winding Number Algorithm* a *Ray Crossing Algorithm*, kterým jsou věnovány následující řádky.

1.2 Winding Number Algorithm

Winding Number Algorithm se používá k analýze polohy bodu pro nekonvexní mnohoúhelníky. Jinak se tomuto algoritmu říká také Metoda ovíjení. Princip spočívá v tom, že pozorovatel stojí v bodě q. Pokud bod q náleží mnohoúhelníku a pozorovatel chce vidět všechny rohy daného objektu, tak se musí otočit o úhel 360 stupňů. Pakliže však bod q danému mnohoúhelníku nenáleží, pak je výše zmíněný úhel menší než 360 stupňů.

1.2.1 Princip algoritmu

Výpočet *Winding Number* probíhá tak, že se sečtou všechny úhly ω_1, jež svírá pozorovaný bod q s vrcholy daného polygonu P.

$$\Omega(q, P) = \begin{cases} 1, q \in P \\ 0, q \notin P \end{cases}$$

Abychom zjistili polohu bodu q vzhledem k přímce $p \approx \overleftrightarrow{P_i P_{i+1}}$, potřebujeme nejprve spočítat směrový vektor této přímky q a dále také vektor v, který spojuje q s p_i.

$$\vec{p} = (x_{i+1} - x_i, y_{i+1} - y_i)$$
$$\vec{v} = (x_q - x_i, y_q - y_i)$$

Přepišme tyto vektory do matice A a následněme spočtěme její determinant. Výsledný determi-

nant nám určí, zda se pozorovaný bod nachází v levé či pravé polorovině.

$$A = \begin{pmatrix} q_x & q_y \\ v_x & v_y \end{pmatrix}$$

$$det(A) = (q_x * v_y) - (q_y * v_x)$$

Pokud se bod q nachází v levé polorovině (σ_l), pak je úhel orientován ve směru hodinových ručiček. Avšak u pravé poloviny (σ_r) je to přesně naopak a výsledná hodnota Ω vychází záporná. Výsledek je následně uváděn jako celkový počet oběhů, tedy násobky 2π.

$$\det \begin{cases} < 0, & \text{bod } q \text{ leží vpravo od přímky } (\sigma_r) \\ > 0, & \text{bod } q \text{ leží vlevo od přímky } (\sigma_l) \\ = 0, & \text{bod } q \text{ leží na přímce} \end{cases}$$

Výhodou tohoto algoritmu je efektivita, neboť pro výpočet stačí pouze znát souřadnice bodu a vrcholů polygonu. Není tedy třeba provádět žádné složité geometrické operace. Nevýhodou Winding Number může být citlivost na orientaci hran. Pokud jsou totiž hrany polygonu zadané ve špatném směru, algoritmus může poskytnout chybné výsledky.

1.2.2 Pseudokód metody Winding Number

Algorithm 1 *Winding Number*

```
Ω = 0, tolerance ε                          ▷ Inicializace sumy úhlu a tolerance
for všechny vrcholy v polygonu:
    if bod je vrcholem:
        return -1                           ▷ Bod leží na hraně
    spočti determinant
    urči úhel ω mezi q a p_i a p_{i+1}
    if determinant > 0:
        Ω += ω                              ▷ Bod leží v levé polorovině
    if determinant < 0:
        Ω -= ω                              ▷ Bod leží v pravé polorovině
    if determinant == 0 and ω > π − ε :
        return -1                           ▷ Bod leží na hraně
if |Ω| − 2π < ε:
    return 1                                ▷ Bod leží uvnitř polygonu
return 0                                    ▷ Bod leží vně polygonu
```

1.3 Ray Crossing Method

Ray Crossing Method, často označovaný také jako *Ray Casting Algorithm*, v češtině *paprskový algoritmus* představuje snadnou metodu řešení *point in polygon* problému pro nekonvexní mnohoúhelníky. Metoda určuje vzájemný vztah polygonu a zkoumaného bodu na základě počtu průsečíků, které protíná polopřímka (paprsek) vedená daným bodem. Pokud je počet průsečíků lichý, tak zkoumaný bod náleží danému mnohoúhelníku. Avšak pokud byl zjištěn sudý počet průsečíků, tak bod leží mimo zkoumanou oblast. U tohoto algoritmu ale musíme dávat pozor na singulární případy, kdy dochází k problematickým situacím.

1.3.1 Princip algoritmu

Nejprve vyberme bod pro který chceme zjistit, zda se nachází uvnitř nebo vně polygonu P. Tento bod označme jako q. Bodem q je vedena polopřímka r *(ray)*, přičemž platí:

$$r(q) : y = y_q$$

Poté jednoduše vypočteme počet průsečíků - tedy kolikrát paprsek protne hrany polygonu. Pokud je počet průsečíků lichý, bod q leží uvnitř polygonu; pokud je sudý, leží vně, tedy:

$$k\%2 = \begin{cases} 1, & q \in P \\ 0, & q \notin P \end{cases}$$

Princip algoritmu Ray Crossing je poměrně snadný, problém však nastává ve chvíli, prochází-li přímka r některým z vrcholů polygonu, nebo je-li totožná s nějakou jeho hranou. Pro tyto situace je možné využít modifikaci algoritmu Ray Crossing s redukcí ke q. Při této úpravě dochází k posunutí počátku do bodu q. Aby byl algoritmus schopen rozpoznat, zda se bod q nachází na hrané polygonu, je přímka r dále rozdělena na dvě polopřímky, přičemž jedna má orientaci levostrannou r_1 a druhá pravostrannou r_2. V situaci, kdy se počet průsečíků k_l a k_r s polopřímkami r_1 a r_2 neshoduje, znamená to, že bod q leží na hraně polygonu P. Bod q je porovnáván se souřadnicemi všech vrcholů polygonu. Jsou-li souřadnice bodu q totožné se souřadnicemi nějakého z vrcholů, leží bod q na vrcholu mnohoúhelníku P. I při této variantě metody *Ray Crossing* platí, že je-li počet průsečíků k_r nacházejících se na polopřímce r_1 lichý, bod leží uvnitř polygonu P.

1.3.2 Pseudokód metody Ray Crossing

Algorithm 2 *Ray Crossing*

```
1: k_l = 0, k_r = 0                          ▷ Inicializace průsečíků pro levou a pravou polorovinu
2: for všechny vrcholy v polygonu:
3:     if bod je vrcholem:
4:         return -1                          ▷ Bod leží na hraně
5:     if y_i == y_{i+1}:                     ▷ Kontrola vodorovných hran
6:         continue
7:     vypočti x souřadnice průsečíku x_m
8:     if (y_i < 0) ≠ (y_{i+1} < 0) and x_m < 0:   ▷ Levostranný paprsek
9:         inkrementuj počet průsečíků k_l
10:    if (y_i > 0) ≠ (y_{i+1} > 0) and x_m > 0:   ▷ Pravostranný paprsek
11:        inkrementuj počet průsečíků k_r
12: if k_r%2 = 1:
13:    return: 1                              ▷ Bod leží uvnitř polygonu
14: if (k_l + k_r)%2 ≠ 0:
15:    return -1                              ▷ Bod leží na hraně
16: return 0                                  ▷ Bod leží vně polygonu
```

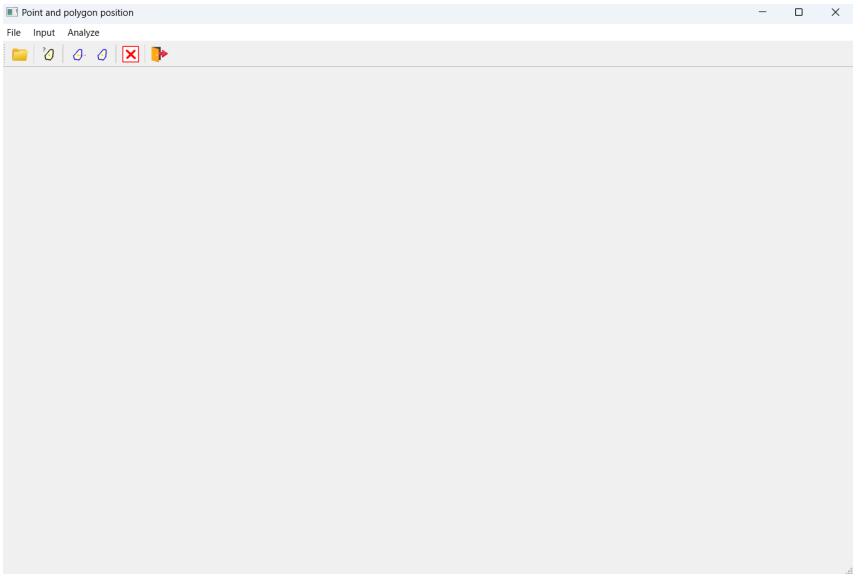
2 Vlastní aplikace

2.1 Vstupní data

Jako vstupní data byla vybrána polygonová vrstva městských částí Prahy. Tato data jsou v Křovákově zobrazení. Vzniklá aplikace umožňuje práci se soubory ve formátu shapefile. U aplikace jsou uloženy námi vybraná testovací data ve zmíněném formátu.

2.2 Grafické rozhraní aplikace

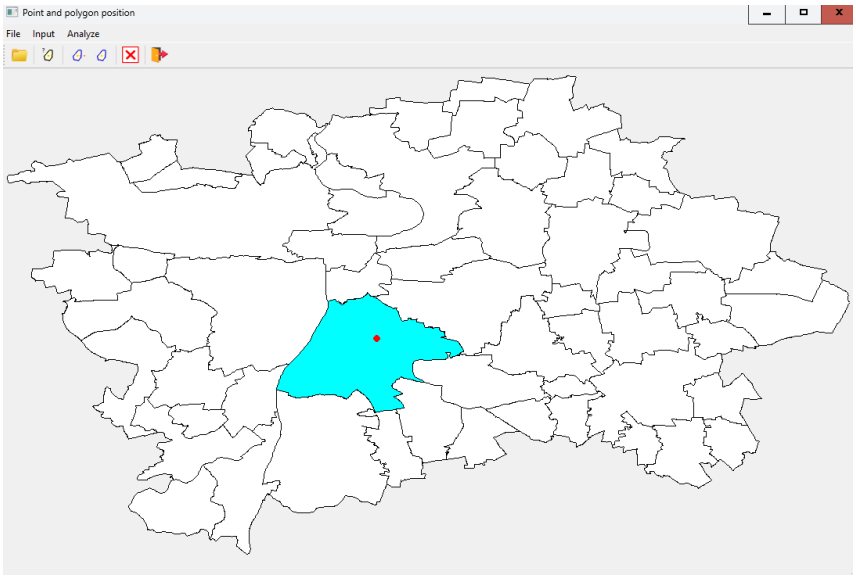
Grafické rozhraní aplikace bylo vytvořeno v prostředí *Qt creator*. Po spuštění aplikace se otevře okno, v jehož horní části se nachází menu s jednotlivými funkcemi. V záložce *File* si uživatel může vybrat z dvou různých funkcí. Po stisknutí *Open* se otevře nové okno, ve kterém je možné vybrat data ve formátu *shapefile*, která jsou následně vykreslena na rozsáhlou volnou plochu v aplikaci. Pokud uživatel stiskne *Exit*, bude aplikace uzavřena. Grafické rozhraní aplikace je znázorněna na obrázku 1.



Obrázek 1: Grafické rozhraní aplikace

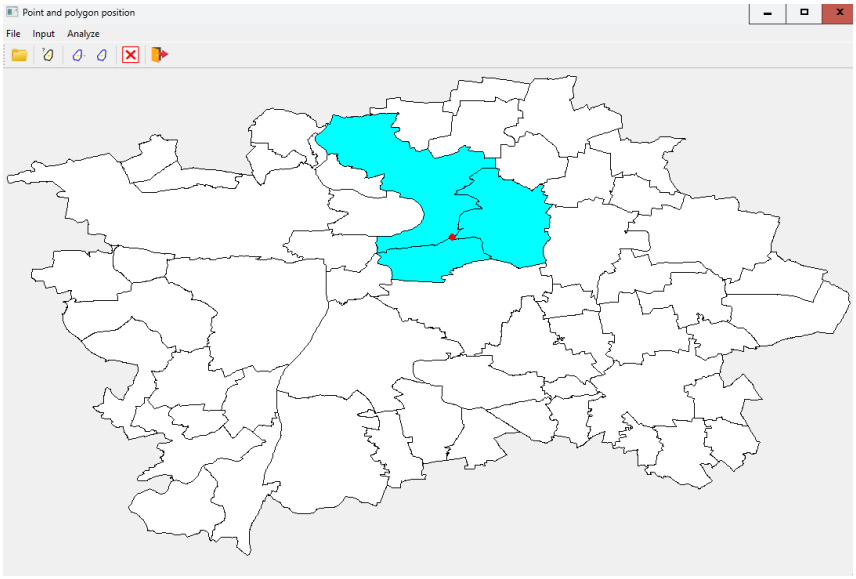
Pokud chce uživatel nakreslit bod, stačí pouze kliknout levým nebo pravým tlačítkem myši na volnou plochu a červenou barvou se vykreslí bod. V záložce *Input* si může uživatel vybrat funkci *Clear*, pomocí které jsou smazána všechna nahraná data (tedy polygony i bod).

Další záložkou naší aplikace je *Analyze*, kde jsou v nabídce funkce *Ray Crossing Algorithm* a *Winding Number Algorithm*. Zde si uživatel může zvolit pomocí kterého algoritmu chce analyzovat polohu svého vytvořeného bodu vůči polygonové vrstvě. Po stisknutí tlačítka zvoleného algoritmu se modrou barvou označí polygon, ve kterém daný bod leží. Tento jev je znázorněn na následujícím obrázku.



Obrázek 2: Bod ležící uvnitř polygonu

Dále bylo také ošetřeno, aby při poloze bodu na hraně či vrcholu polygonu byly označeny všechny polygony, na jejichž hranicích bod leží. Zmíněná situace je znázorněna na obrázku 3, kde bod leží na hraně třech polygonů.



Obrázek 3: Bod ležící na hraně více polygonů

Na horním panelu jsou také umístěny ikony, pomocí kterých může uživatel rychleji vyvolat požadovanou funkci. Tyto ikony odpovídají funkcím, ketré jsou v jednotlivých záložkách nad nimi. Konkrétně jde tedy zleva o následující ikony: *Open file*, *Input point*, *Ray Crossing Algorithm*, *Winding Number Algorithm*, *Clear data* a *Close application*.

2.3 Dokumentace programu

Program byl vytvořen v prostředí *Visual Studio Code* v programovacím jazyce Python. Součástí programu jsou následující soubory: *mainform.py, draw.py, algorithms.py*. Složka *icons* obsahuje obrázky s ikonami nezbytné pro přehledné fungovaní aplikace.

Třída mainform
Třída *mainform* je určena k nastavení uživatelského rozhraní aplikace a propojení s metodami definovanými v ostatních třídách. Konkrétně tato třída zabezpečuje inicializaci okna aplikace, panelu nástrojů, vrchní lišty a dále také jednotlivých ikon a tlačítek. Celkem je v *mainform* obsaženo sedm metod, z nichž dvě (`setupUi(MainWindow)` a `retranslateUi(MainWindow)`) vznikly automaticky po vytvoření prostředí v QtCreator. Dále jsou zde následující nové metody:

- `openClick()`: otevírá soubor ve formátu shapefile a načítá ho do proměnné
- `handleClick(algorithm)`: analyzuje polohu bodu q vůči polygonu
- `windingNumberClick()`: nastavuje Winding Number jako algoritmus pro analýzu polohy bodu
- `rayCrossingClick()`: nastavuje Ray Crossing jako algoritmus pro analýzu polohy bodu
- `clearClick()`: maže nahrané polygony a bod

Třída draw
Třída *draw* zajišťuje grafické rozhraní, slouží tedy k vykreslení prostorových dat a ovládání aplikace. V této třídě jsou obsaženy následující metody:

- `mousePressEvent(e:QMouseEvent)`: odečítá aktuální souřadnice kurzoru myši
- `paintEvent(e:QPaintEvent)`: vykresluje body a polygony na plátno
- `getQ()`: vrací souřadnice bodu
- `getPolygons()`: vrací seznam vstupních polygonů
- `getPol()`: vrací analyzovaný polygon (analyzované polygony)
- `clearPol()`: odstraňuje veškeré kresby na plátně
- `setHighlightedPolygons(indices)`: nastavuje seznam indexů polygonů, které mají být zvýrazněny
- `loadDataAndRescalePolygons(width, height)`: načítá prostorová data a roztahuje je v závislosti na velikosti grafického okna

Třída algorithms
Třída *algorithms* obsahuje metody, které jsou detailně popsány v předchozí kapitole, která se detaině věnuje použitým algoritmům. Tato třída je tedy tvořena následujícími metodami:

- `rayCrossing (q, pol)`: pomocí algoritmu Ray Crossing analyzuje polohu bodu q vůči polygonu pol
- `windingNumber (q, pol)`: pomocí algoritmu Winding Number analyzuje polohu bodu q vůči polygonu pol

3 Závěr

V této úloze byla vytvořena aplikace, která na základě uživatelem zvoleného algoritmu analyzuje polohu bodu vůči polygonům. Konkrétně si uživatel může vybrat mezi *Ray Crossing* a *Winding Number*, což jsou algoritmy schopné řešit problém bodu a polygonu pro nekonvexní mnohoúhelníky. Aplikace dokáže také vyřešit singularity, tedy že vybraný bod leží na hraně či vrcholu polygonu.

Možným zlepšením aplikace by mohla úprava, aby byla schopna vyřešit i úlohu pro polygony s dírami. Dalším vylepšením by určitě bylo nahrání souboru v jiném formátu, neboť současná aplikace dokáže nahrát pouze data ve formátu *shapefile*. Užitečné by bylo, kdyby do aplikace šla nahrát data například formátů *JSON* nebo *GeoJSON*. Uživatelé by také určitě uvítali, kdyby si v aplikaci mohli přibližovat a oddalovat, což by jim umožnilo přesnější umístění bodu. Tato vlastnost by aplikaci také výrazně vylepšila.

4 Zdroje

BAYER, T. (2024): Point Location Problem. Prezentace pro předmět Algoritmy počítačové kartografie, dostupné z: https://web.natur.cuni.cz/ bayertom/index.php/teaching/algoritmy-pocitacove-kartografie (cit. 21. 3. 2024).

HECKBERT, P. S. (1994). Graphics Gems IV. Morgan Kaufmann.

HUANG, C. W., SHIH, T. Y. (1997): On the complexity of Point-in-Polygon Algorithms. Computers and Geosciences, 23 (1), 109-118.

ROURKE, O. J. (1998): Computational Geometry in C. Cambridge University Press, Cambridge.