Přírodovědecká fakulta
Univerzita Karlova

Úkol č. 1: Point Location Problem

Algoritmy počítačové kartografie

Eliška Králová, Eliška Pospěchová

1.N-GKDPZ
Praha 2025

1 Zadání

Úloha č. 1: Geometrické vyhledávaní bodu

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:

Krok	**Hodnocení**
Detekce polohy bodu rozšiřující stavy uvnitř, vně polygonu.	10b
Analýza polohy bodu (uvnitř/vně) metodou Winding Number Algorithm.	*+10b*
Ošetření singulárního případu u Winding Number Algorithm: bod leží na hraně polygonu.	*+5b*
Ošetření singulárního případu u Ray Crossing Algorithm: bod leží na hraně polygonu.	*+5b*
Ošetření singulárního případu u obou algoritmů: bod je totožný s vrcholem jednoho či více polygonů.	*+2b*
Zvýraznění všech polygonů pro oba výše uvedené singulární případy.	*+3b*
Rychlé vyhledávání potenciálních polygonů (bod uvnitř min-max boxu).	*+10b*
Řešení pro polygony s dírami.	*+10b*
*Načtení vstupních dat ze *.shp.*	*+10b*
Max celkem:	**65b**

2 Údaje o bonusových úlohách

V rámci této úlohy jsou řešeny tyto bonusové úlohy: *Analýza polohy bodu (uvnitř/vně) metodou Winding Number Algorithm* (10b), *Ošetření singulárního případu u Winding Number Algorithm: bod leží na hraně polygonu* (5b), *Ošetření singulárního případu u Ray Crossing Algorithm: bod leží na hraně polygonu* (5b), *Ošetření singulárního případu u obou algoritmů: bod je totožný s vrcholem jednoho či více polygonů.* (2b), *Zvýraznění všech polygonů pro oba výše uvedené singulární případy* (3b), *Rychlé vyhledávání potenciálních polygonů (bod uvnitř min-max boxu)* (10b), *Načtení vstupních dat ze *.shp.* (10b).

3 Popis a rozbor problému

Point Location Problem

Problém určení polohy bodu neboli *Point Location Problem* se zabývá určením, ve které oblasti (např. polygonu) se nachází bod v rozdělené rovině. Tedy zda bod leží uvnitř, vně nebo na hraně polygonu či polygonů. Prakticky se tento problém řeší například v GIS, počítačové grafice a robotice.

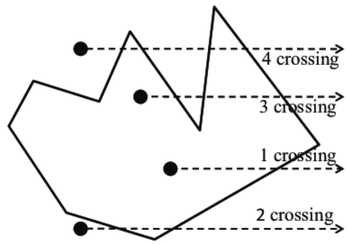
Jedním z řešení tohoto problému je převedení na vztah bodu a mnohoúhelníku (polygonu), při čemž se opakovaně určuje poloha bodu vzhledem k mnohoúhelníku. Dalším způsobem je řešení přes rozdělení roviny na množinu pásů či lichoběžníků.

Při testování polohy bodu a mnohoúhelníku lze postupovat lokálně a globálně. Lokální procedura spočívá v opakovaném určování polohy bodu vzhledem k dotazovanému mnohoúhelníku. Výsledkem je, že bod leží uvnitř, vně, nebo na hraně testovaného mnohoúhelníku. Naopak globální procedura hodnotí vztah bodu a množiny mnohoúhelníků, kdy se postupně prochází poloha bodu vůči každému mnohoúhelníku. Po provedení této procedury je určena poloha bodu jako uvnitř konkrétního mnohoúhelníku, vně všech mnohoúhelníků, nebo jako hrana/uzel dvou a více mnohoúhelníků.

Algoritmy řešící tento problém lze rozdělit do dvou kategorií dle typu mnohoúhelníku. Prvním typem mnohoúhelníků jsou konvexní mnohoúhelníky, u kterých každý úhel má velikost menši nebo rovnu 180 °. Pro tento typ mnohoúhelníků se využívají algoritmy Ray Crossing Method a Half-plane Test. Jako druhý typ jsou nekonvexní mnohoúhelníky, u nichž alespoň jeden úhel je větší než 180 °. U těchto mnohoúhelníků se používají algoritmy Ray Crossing a Winding Number.

Ray Crossing Algorithm

Principem je zakreslení polopřímky r v libovolném směru procházejícím bodem q, u níž se spočítá počet průsečíků k této přímky s hranami polygonu P (*Obrázek 1*).



Obrázek 1: Ray Crossing Algorithm (zdroj: Yan 2012)

Pokud je počet průsečíků k lichý, bod q se nachází uvnitř polygonu P. Pokud je sudý, bod se nachází vně polygonu:

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

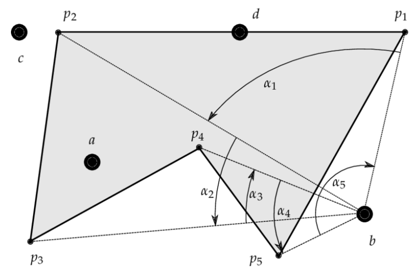
Problémovými jsou singulární případy, tedy kdy přímka r prochází vrcholem nebo hranou polygonu P a bod q ležící na hraně polygonu P. Bod ležící na hraně polygonu lze vyřešit pomocí redukce ke q. V rámci této metody se přímka r rozdělí na levostrannou r_1 a pravostrannou r_2 s opačnou orientací. Poté se spočítá počet průsečíků s polygonem pro každou stranu zvlášť, tedy k_l a k_r. Jestliže se tyto počty průsečíků vzájemně nerovnají, znamená to, že bod q leží na hraně polygonu P.

$$q = \begin{cases} \in \partial P, & k_l\%2 \neq k_r\%2 \\ \in P, & k_r\%2 = 1 \\ \notin P, & jinak \end{cases} \tag{2}$$

Winding Number Algorithm

Metoda využívá hodnotu Winding Number Ω, která je sumou všech úhlů $\omega(p_i, q, p_{i+1})$, které svírá pozorovaný bod q s vrcholy polygonu P (*Obrázek 2*). Výsledné číslo se dělí 2π, Ω tak udává počet ovinutí okolo bodu q.

Zda se úhel ω k výslednému Winding Number Ω přičte nebo odečte závisí na poloze bodu q vůči přímce $p \approx \overleftrightarrow{p_i, p_{i+1}}$. Pro výpočet polohy je třeba znát směrový vektor přímky p a vektor $\vec{v}$, který je určen body q a p_i.



Obrázek 2: Winding Number Algorithm (zdroj: Santos 2024)

$$
\begin{aligned}
\vec{p} &= (x_{i+1} - x_i, y_{i+1} - y_i) \\
\vec{v} &= (x_q - x_i, y_q - y_i)
\end{aligned}
\tag{3}
$$

Přímka p dělí rovinu σ na levou polorovinu σ_l a pravou polorovinu σ_r. Ve které z nich bod q leží zjistíme výpočtem vektorového součinu $\vec{p} \times \vec{v}$, který lze zapsat maticově a vypočítat pomocí determinantu:

$$
t = \begin{vmatrix} p_x & p_y \\ v_x & v_y \end{vmatrix} = \begin{cases} t > 0, & q \in \sigma_l \\ t = 0, & q \in p \\ t < 0, & q \in \sigma_r \end{cases}
\tag{4}
$$

Podle poloroviny, ve které bod q leží, je určeno znaménko úhlu $\omega(p_i, q, p_{i+1})$:

$$
\omega(p_i, q, p_{i+1}) = \begin{cases} +\omega(p_i, q, p_{i+1}), & q \in \sigma_l \\ -\omega(p_i, q, p_{i+1}), & q \in \sigma_r \end{cases}
\tag{5}
$$

Pro velikost úhlu $\omega(p_i, q, p_{i+1})$ platí:

$$
\cos\omega = \frac{\vec{p} \cdot \vec{v}}{\|\vec{p}\| \|\vec{v}\|}
\tag{6}
$$

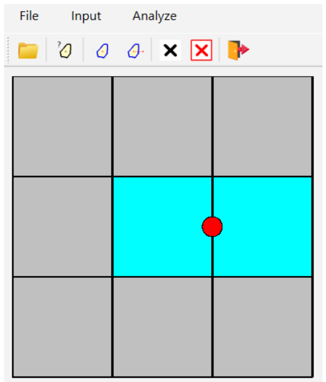
Pro výsledné Winding Number tedy platí:

$$
\Omega(q, P) = \frac{1}{2\pi} \sum_{i=1}^{n} \omega(p_i, q, p_{i+1}) = \begin{cases} 1, & q \in P \\ 0, & q \notin P \end{cases}
\tag{7}
$$

V případě, že $q \in p$, je třeba ověřit, zda se bod q nachází na hraně polygonu. V tom případě musí platit:

$$
\exists \omega(p_i, q, p_{i+1}) = \pi, \quad q \in \partial P
\tag{8}
$$

Všechny singularity byly ověřovány přes zkušební grid, jelikož je obtížné umístit bod přesně na hranu nebo vrchol polygonu. Kromě sítě byly pevně určeny také souřednice analyzovaného bodu. I tato síť byla vizualizována v aplikaci (*Obrázek 3*).



Obrázek 3: Testovací grid

4 Popisy algoritmů formálním jazykem

Pseudokód Ray Crossing Algorithm

Algorithm 1 *Ray Crossing Method*

inicializuj počet průsečíků k_r a k_l na 0 a počet vrcholů n
pro všechny vrcholy:
  vypočti souřadnice redukovaných bodů x_{ir}, y_{ir}, x_{i+1r}, y_{i+1r}

  pokud $x_{ir} = 0$ a $y_{ir} = 0$:
    bod q leží na vrcholu polygonu P
  vypočti x souřadnice průsečíku x_m
  pokud $(y_{i+1} < 0) \neq (y_i < 0)$:
    pokud $x_m < 0$:
      zvyš počet průsečíků k_l v levé polorovině
  pokud $(y_{i+1} > 0) \neq (y_i > 0)$:
    pokud $x_m > 0$:
      zvyš počet průsečíků k_r v pravé polorovině

pokud $(k_l\%2) \neq (k_r\%2)$:
  $q \in \partial P$
pokud $k_r\%2 = 1$:
  $q \in P$
jinak $p \notin P$

Pseudokód Winding Number Algorithm

Algorithm 2 *Winding Number Method*

inicializuj Winding Number Ω na 0 a počet vrcholů n
pro všechny vrcholy:
  pokud $q_x = p_x$ a $q_y = p_y$:
    bod q leží na vrcholu polygonu P
  zjisti polohu bodu q vůči přímce p
  vypočti úhel ω
  pokud $q \in p$:
    pokud $\omega < \pi + \epsilon$ a $\omega > \pi - \epsilon$:
      bod q leží na hraně polygonu P
  pokud $q \in \sigma_l$:
    $\Omega = \Omega + \omega$
  pokud $q \in \sigma_r$:
    $\Omega = \Omega - \omega$

pokud $2\pi - \epsilon < |\Omega| < 2\pi + \epsilon$:
  bod q je uvnitř polygonu P
jinak:
  bod q je vně polygonu P

5 Aplikace

Po spuštění skriptu *MainForm.py* se otevře okno aplikace. V záložce *File* jsou umístěny funkce *Open* a *Exit*. V další záložce *Input* jsou funkce *Point/Polygon*, *Clear results* a *Clear All*. Poslední záložka *Analyze* obsahuje algoritmické funkce *Winding Number* a *Ray Crossing*. Všechny tyto funkce mají ikonku nacházející se na nástrojové liště.

Po kliknutí na fukci *Open* se otevře nabídka pro vybrání požadovaného souboru ve formátu *shapefile*, který obsahuje pouze topologicky správné polygony. Jako polygonová mapa byla vybrána mapa městských částí Prahy (*Geoportál Praha 2024*). Pro umístění bodu stačí kliknout na libovolné místo v aplikaci. Pokud je však zakliknutá funkce *Point/Polygon*, lze kreslit svůj vlastní polygon, jestliže není otevřen shapefile soubor. Po umístění analyzovaného bodu a spuštění příslušné metody na zjištění polohy bodu vzhledem k polygonům, se otevře okno s informací o poloze bodu. Dále se zvýrazní polygon, jemuž náleží analyzovaný bod (*Obrázek 4*).

Kliknutím na funkci *Clear Results* se smaže analyzovaný bod a zvýrazněný polygon. Pro nahrání nebo nakreslení jiného polygonu je zapotřebí spustit funkci *Clear All*. Aplikaci lze ukončit funkcí *Exit* nebo tlačítkem "křížek"v pravém horním rohu.

Obrázek 4: Ukázka aplikace

6 Dokumentace

Program řešící *Point Lacation Problem* byl vytvořen v SW Visual Studio Code v jazyce Python a skládá se ze tří souborů, které odpovídají použitým třídám. Grafické prostředí aplikace bylo vytvořeno v SW Qt Designer.

Třída MainForm

Pomocí třídy *MainForm* se spouští celé uživatelské prostředí aplikace, které obsahuje ikonky akcí a funkcí a zajišťuje propojení s ostaními třídami. Třída dále obsahuje tyto funkce:

- **openClick** - funkce zavolá metodu *openFile*.
- **exitClick** - funkce zavolá metodu *exit*.
- **clearAllClick** - funkce zavolá metodu *clearAll*.
- **clearClick** - funkce zavolá metodu *clearRes*.
- **getRes** - funkce zavolá metodu *getQ* a třídu *Algorithms*. Funkce dále bere všechny polygony z metody *getPol*, nad kterým zavolá metody *in_min_max_box* a dále dle vstupního parametru zavolá daný algoritmus. Pro zvýraznění výsledného polygonu po provedení algoritmu je zavolána metoda *paintRes*. Nakonec funkce vypíše výsledek zjištěné polohy bodu vzhledem k polygonu.
- **windingNumberClick** - funkce zavolá funkci **getRes** se vstupním parametrem určujícím typ algoritmu (*Winding Number*).
- **rayCrossingClick** - funkce zavolá funkci **getRes** se vstupním parametrem určujícím typ algoritmu (*Ray Crossing*).
- **switchClick** - funkce zavolá metodu *switchInput*.

Třída Draw

Na začátku této třídy jsou inicializovány pomocné proměnné. Dále třída obsahuje následující funkce:

- **openFile** - funkce zobrazí systémové okno s možností nahrání souboru ve formátu *shapefile*. Vybraný shapefile přečte a pro vykreslení dat zavolá funkci **geomShapefile**.
- **geomShapefile** - funkce vykresluje shapefile, kdy zpracovává geometrii vstupních dat a upaví souřadnice dat podle velikosti plochy okna aplikace.
- **exit** - funkce ukončí aplikaci (zavře její okno).
- **clearAll** - funkce vymaže všechna vložená i nakreslená data.
- **clearRes** - funkce vymaže výsledek (analyzovaný bod a zvýrazněný polygon).

- **mousesPressEvent** - funkce zjišťuje souřadnice, na které bylo kliknuto. Tyto souřadnice jsou následně využity pro umístění bodu nebo určení vrcholu kresleného polygonu.
- **paintEvent** - funkce vykreslí polygon/y, analyzovaný bod a výsledný polygon (ve kterém leží analyzovaný bod).
- **paintRes** - funkce přiřadí výsledný analyzovaný polygon k proměnné pro zvýraznění.
- **switchInput** - funkce přepíná mezi kreslením analyzovaného bodu a polygonu.
- **getQ** - funkce vrací analyzovaný bod.
- **getPol** - funkce vrací vstupní polygony jako seznam.

Třída Algorithms

V rámci této třídy jsou nadefinovány tyto funkce:

- **point_on_vertex** - funkce kontroluje souřadnice analyzovaného bodu a souřadnice vrcholu polygonu. Jestliže se shodují, analyzovaný bod leží na vrcholu polygonu.
- **calculate_angle** - funkce počítá velikost úhlu mezi vektorem $\vec{v_1}$ (vrchol polygonu p_i, analyzovaný bod q) a vektorem $\vec{v_2}$ (vrchol polygonu p_{i+1}, q). Použité vzorce jsou vypsány v kapitole 3, přesněji se jedná o rovnice (3) a (6).
- **get_point_location** - funkce analyzuje vzájemnou polohu bodu a přímky výpočtem determinantu podle rovnice (4) v kapitole 3, který dále vstupuje do rovnice (5).
- **ray_crossing** - funkce analyzuje vzájemnou polohu bodu a polygonu metodou Ray Crossing, která je blíže popsána v kapitole 3. Vrací hodnotu 1 v případě, že se bod nachází uvnitř polygonu, hodnotu 0 v případě, že je bod vně polygonu a hodnotu -1, v případě, že je bod totožný s vrcholem polygonu nebo leží na jeho hraně. Tento princip zobrazují rovnice (1) a (2) v uvedené kapitole 3.
- **winding_number** - funkce analyzuje vzájemnou polohu bodu a polygonu metodou Winding Number, která je blíže popsána v kapitole 3. Vrací hodnotu 1 v případě, že se bod nachází uvnitř polygonu, hodnotu 0 v případě, že je bod vně polygonu a hodnotu -1, v případě, že je bod totožný s vrcholem polygonu nebo leží na jeho hraně. Využity přitom byly rovnice (7) a (8) uvedené v kapitole 3.
- **in_min_max_box** - funkce hodnotí analyzovaný bod, zda se nachází v min-max boxu všech polygonů. Pokud bod neleží v žádném min-max boxu, tak s jistotou neleží ani v žádném polygonu.

7 Závěr

V rámci řešení problému *Point Location* byly implementovány algoritmy *Ray Crossing* a *Winding Number*, včetně vybraných singularit. Analyzovaná data, na kterých mohu být spuštěny oba algoritmy, jsou vizualizovaná v aplikaci. Aplikace zároveň zobrazuje výsledek po provedení algoritmu. Tedy zda je analyzovaný bod uvnitř, vně nebo na hraně polygonu.

Jako možné vylepšení skriptu by byla možnost pracovat na vstupu i s multipolygonem nebo polygonem s dírou. Dále by šlo vylepšit kreslení polygonů, kdy nyní lze nakreslit pouze jeden polygon namísto několika polygonů.

Zdroje

BAYER, T. (2025): Point Location Problem. Přednáška pro předmět Algoritmy počítačové kartografie, Katedra aplikované geoinformatiky a kartografie, Přírodovědecká fakulta UK [cit. 8.3.2025].

DE BERG, M., VAN KREVELD, M., OVERMARS, M., SCHWARZKOPF, O.: Computational Geometry. 2000, Springer

GEOPORTAL PRAHA (2024): Data a služby - Městské části. https://geoportalpraha.cz/data-a-sluzby/3edca1d2007d46e982fa32422ed926c8_0 [cit. 8.3.2025].

HUANG, C. W., SHIH, T. Y. (1997): On the complexity of point-in-polygon algorithms. Computers & Geosciences, 23, 1, 109-118.

OPENAI (2025): ChatGPT. https://chatgpt.com [cit. 8.2.2025].

QT GROUP (2025): Qt Documentation. https://doc.qt.io/ [cit. 8.3.2025].

SANTOS, V. (2024): Analytical Solution for the Problem of Point Location in Arbitrary Planar Domains. Algorithms, 17.444, 10.3390/a17100444.

YAN, D., ZHAO, Z., NG, W. K. (2012): Monochromatic and Bichromatic Reverse Nearest Neighbor Queries on Land Surfaces. 10.1145/2396761.2396880.