ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

FAKULTA STAVEBNÍ, OBOR GEODÉZIE A KARTOGRAFIE

KATEDRA GEOMATIKY

název předmětu

ALGORITMY DIGITÁLNÍ KARTOGRAFIE A GIS

číslo úlohy	název úlohy				
1	Geometrické vyhledávání bodů				
školní rok	**studijní skup.**	**číslo zadání**	**zpracovali**	**datum**	**klasifikace**
2023/24	60	xxx	Koudelka Jan Müller Vojtěch	14.3. 2024	

Technická zpráva z úlohy 1 – Geometrické vyhledávání bodu

1. **Zadání:**
   Vstup: Souvislá polygonová mapa n polygonů {P1, ..., Pn}, analyzovaný bod q.
   Výstup: Pi, q ∈ Pi.

   Nad polygonovou mapou implementujete Ray Crossing Algorithm pro geometrické vyhledání 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.

   - **Detekce polohy bodu rozlišující stavy uvnitř, vně polygonu.**
   - Analýza polohy bodu (uvnitř/vně) metodou Winding Number Algorithm.
   - Ošetření singulárního případu u Winding Number Algorithm: bod leží na hraně polygonu.
   - Ošetření singulárního případu u Ray Crossing Algorithm: bod leží na hraně polygonu.
   - Ošetření singulárního případu u obou algoritmů: bod je totožný s vrcholem jednoho či více polygonů.
   - Zvýraznění všech polygonů pro oba výše uvedené singulární případy.
   - Rychlé vyhledávání potenciálních polygonů (bod uvnitř min-max boxu).
   - Řešení pro polygony s dírami.

2. **Popis:**
   Geometrické vyhledání bodu je častá úloha řešena v prostředí GIS. I v praktickém životě je důležité znát polohu v závislosti na ostatních objektech. Je nutné vyhledávat dané mnohoúhelníky rychle i při velikém množství dat. [1]

3. **Postup:**
   3.1. Vytvoření základního grafického rozhraní v prostředí QT:
   V prostředí QT bylo navrženo grafické rozhraní výsledné aplikace. Získaný kód byl dále upravován ve skriptovacím prostředí Visual Studio Code.

   3.2. Nahrání vytvořeného kódu do skriptovacího prostředí, propojení jednotlivých tříd, vytvoření kreslících funkcí:
   V jazyce Python byl získaný kód v prostředí Visual Studio Code upraven do podoby, aby se aplikace dala spouštět. Ve třídě *draw* byly vytvořeny kreslící funkce, které umožňují ruční vykreslení polygonu a určení polohy bodu.

   3.3. Ray Crossing algoritmus s ošetřením singulárních případů:
   Ray Crossing algoritmus byl vytvořen ve třídě *algorithms*.
   Algoritmus prochází všechny strany polygonu, na které vysílá myšlenou polopřímku směrem z bodu *q*. Úkolem algoritmu je rozhodnout, zda je bod *q* uvnitř či mimo analyzovaný polygon. Uvnitř je tehdy, když přímka s polygonem vytvoří lichý počet průsečíků. [1]
   - Procházení polygonu bod po bodu, hledání počtu průsečíků přímky zpuštěné z bodu q na danou hranu polygonu s polygonem.
   - Zkoumání počtu průsečíků v jednotlivých průchodech, pokud je počet průsečíků vždy lichý, bod leží uvnitř polygonu.
   - Pokud neleží bod ani vlevo, ani vpravo, bod leží na hraně. Pokud se to stane vícekrát, leží ve vrcholu.

1. Inicializuj $k = 0$ //Pocet prueciku
2: Opakuj pro $\forall$ body $p_i \in P$:
3: $x'_i = x_i - x_q$.
4: $y'_i = y_i - y_q$.
5: if $(y'_i > 0)\&\&(y'_{i-1} <= 0)||(y'_{i-1} > 0)\&\&(y'_i <= 0)$. //Vhodny segment
6: $x'_m = (x'_i y'_{i-1} - x'_{i-1} y'_i)/(y'_i - y'_{i-1})$. //Vhodny prusecik
7: if $(x'_m > 0)$ pak $k = k + 1$.
8: if $(k\%2) \neq 0$ pak $q \in P$
9: else $q \notin P$

Obrázek 1 – Ray Crossing algoritmus [1]

3.4. Winding Number algoritmus s ošetřením singulárních případů:

Winding Number algoritmus byl vytvořen ve třídě *algorithms*.

Algoritmus prochází jednotlivé strany polygonu a vždy vytvoří trojúhelník s daným bodem *q*.

V každém kroku počítá algoritmus úhel při vrcholu *q*. Tento úhel se nasčítává a na závěr se posuzuje velikost této sumy, která je v případě bodu uvnitř polygonu rovna 360°. [1]

- Procházení polygonu bod po bodu, výpočet úhlu sevřeného spojnicemi bodu *q* se dvěma vrcholy dané hrany polygonu.
- Pokud leží bod vlevo od hrany, úhel se přičte, pokud vpravo, úhel se odečte.
- Pokud neleží bod ani vlevo, ani vpravo, bod leží na hraně. Pokud se to stane vícekrát, leží ve vrcholu.
- Po projetí všech bodů se získaný výsledný úhel porovná s hodnotou 360° (vložena hodnota tolerance).
- Pokud je absolutní hodnota (řeší možnost opačné definice polygonu) výsledného úhlu roven 360°, leží bod uvnitř polygonu.

1: Inicializuj $\Omega = 0$, tolerance ε.
2: Opakuj pro $\forall$ trojici (p_i, q, p_{i+1}):
3: Urči polohu q vzhledem k $p = (p_i, p_{i+1})$.
4: Urči úhel $\omega_i = \angle p_i, q, p_{i+1}$.
5: If $q \in \overline{\sigma}_l$, pak $\Omega = \Omega + \omega_i$. //Bod v leve polorovine
6: else $\Omega = \Omega - \omega_i$. //Bod v prave polorovine
7: if $||\Omega| - 2\pi| < \varepsilon$, pak $q \in P$ //Test na odchylku od 2π
8: else $q \notin P$

Obrázek 2 – Winding Number algoritmus [1]

3.5. Vytvoření třídy pro automatické nahrávání polygonových vrstev ze souborů a aplikace vytvořených algoritmů na tuto situaci:

Pro testování aplikace byl vytvořen polygon obsahující několik vybraných ORP v okolí Prahy.

Aplikace umožňuje načítání souborů ve formátu *shapefile*. Uživatel může tyto soubory nahrát v prostředí samotné aplikace.

3.6. Zvýraznění vyhledaných polygonů:

Polygon obsahující zadaný bod se po spuštění přebarví. Je tomu po ošetření tak, i když bod leží na hraně polygonu, či v jeho vrcholu. V těchto případech se obarví všechny polygony sdílící tuto hranu či vrchol.

3.7. Rychlé vyhledávání pomocí min-max boxu:

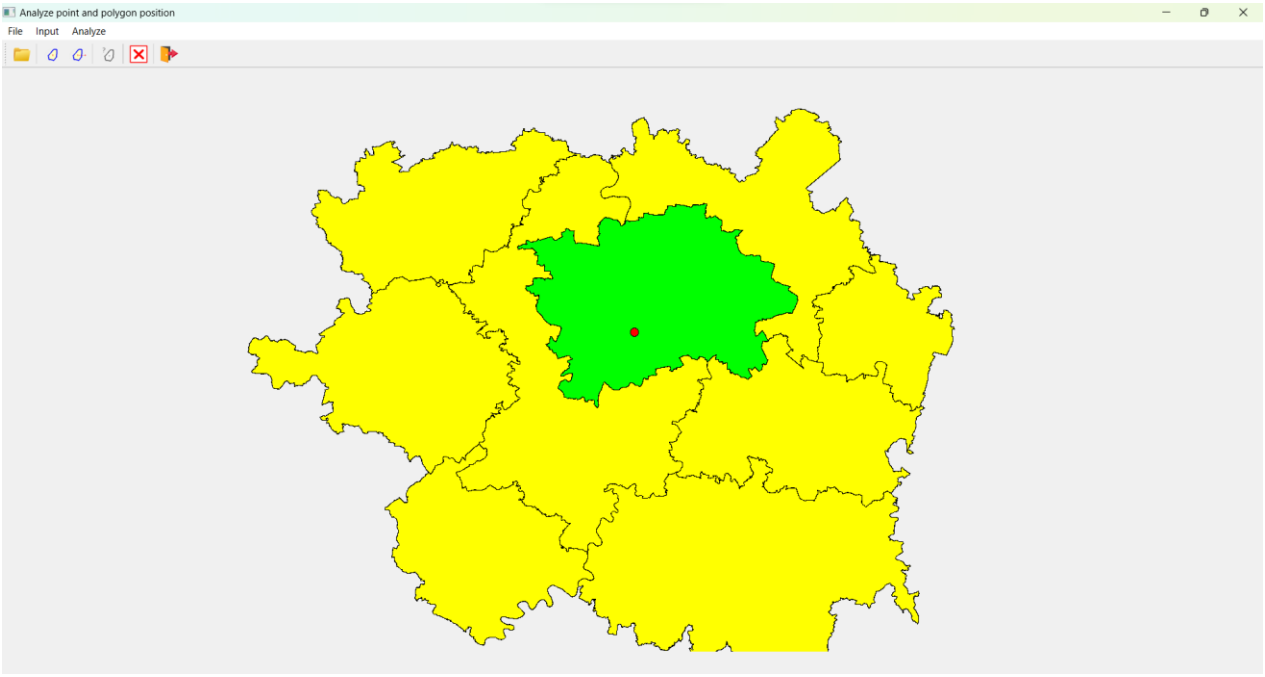
Min-max box dokáže rychleji detekovat body nacházející se ve větší vzdálenosti od vložených

polygonů. S množstvím dat, která máme k dispozici není ovšem časové zjednodušení pozorovatelné.

3.8. Řešení pro polygony s dírami:
Tento bod nebyl řešen.

4. Výsledky:

Výsledkem této úlohy je grafická aplikace pro rychlé vyhledávání bodu, respektive určování polygonu, ve kterém se bod nachází. Práce aplikace lze vidět na obrázku 3:



Obrázek 3 – ukázka aplikace

5. Závěr:

Byla vytvořena grafická aplikace umožňující určit aktuální polohu bodu vzhledem k okolním polygonům. Tato funkce může probíhat na základě dvou různých algoritmů – Ray Crossing, Winding Number. Oba algoritmy jsou také ošetřeny o singulární případy, kdy se bod nachází buď na hraně polygonu nebo na jeho vrcholu. Aplikace umožňuje ruční zadání polygonu, dokáže ovšem také nahrát vlastní polygonovou vrstvu ve formátu *shapefile* a analyzovat bod v reálném území. Hledané polygonu jsou po spuštění intuitivně obarveny.

Poslední bonusová úloha, řešení pro polygony s dírami, nebyla nakonec řešena. Hlavním důvodem byl nedostatek času.

6. Vstupní data (součást odevzdaných souborů, včetně dokumentace):

- Soubory aplikace:
  - MainForm.py
  - algorithms.py
  - draw.py
  - pio.py
- Soubor obsahující polygonovou vrstvu pro nahrání do aplikace: ORP.shp

7. Seznam literatury:

[1] BAYER, Tomáš. Point Location Problem [online]. Praha [cit. 2024-03-13]. Dostupné z: https://web.natur.cuni.cz/~bayertom/images/courses/Adk/adk3_new.pdf. Katedra aplikované geoinformatiky a kartografie. Přírodovedecká fakulta UK.

Dne: 14.3.2024 **Město:** Praha
Jméno: Jan Koudelka, Vojtěch Müller