UNIVERZITA KARLOVA
PŘÍRODOVĚDECKÁ FAKULTA

ALGORITMY POČÍTAČOVÉ KARTOGRAFIE

Geometrické vyhledávání bodu

Jáchym Černík, Monika Novotná

1 Zadání

Hlavní úkol: Detekce polohy bodu rozlišující stavy uvnitř, vně polygonu. Bonusové úkoly jsou zaneseny do Tabulky 1.

Bonusový úkol	**Ohodnocení**
Analýza polohy bodu (uvnitř/vně) metodou Winding Number Algorithm.	10b
Ošetření singulárního případu u Winding Number Algorithm: bod leží na hraně polygonu.	5b
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
Rychlé vyhledávání potenciálních polygonů (bod uvnitř min-max boxu).	10b
Načtení vstupních dat ze *.shp.	10b

Tabulka 1: Bonusové úkoly

2 Popis a rozbor problému

2.1 Point-in-Polygon Problem

Jedním ze základních problémů v oblasti geoinformatiky je analýza vzájemné polohy objektů. Mezi klíčové otázky se řadí určování zda se bod nachází uvnitř nebo vně zájmového území neboli polygonu. Tento problémy nazývaný jako Point-in-Polygon Problem lze řešit několika metodami. Výsledek analýzy nabízí tři varianty: bod může ležet uvnitř/vně nebo na hranici polygonu. Pro řešení se často vyžívají metody převedení problému na vztah bodu a mnohoúhelníku nebo planární dělení roviny.

V případě prvního přístupu dochází k testování každého polygonu zvlášť, což je u velké datové sady časově náročné. Planární dělení roviny je efektivnější varianta, která dělí roviny na lichoběžníky a následně vzniká trapezidální mapa.

Dalším významným faktorem, který ovlivňuje zvolený algoritmus je tvar polygonu.Pro konvexní mnohoúhelníky se problematika řeší jiným způsobem než pro mnohoúhelníky konvexní. Co se týče mnohoúhleníků konvexních, jedná se o jednodušší analýzu na kterou lze zvolit metodu Half-plane test (testování polohy bodu vzhledem ke každé straně polygonu) nebo Ray Crossing Algorithm (viz. kapitola 2.3). Co se týče nekonvexních mnohoúhelníků, jsou řešeny pomocí Ray algorithmu nebo Winding Number Algorithmu (viz. kapitola 2.2).

2.2 Winding Number Method

Winding Number Method neboli metoda ovíjení je algoritmus používaný při detekování polohy bodu zejména vůči nekonvexnímu mnohoúhelníku. Určuje kolikrát polygon obtočí daný bod. Algoritmus pracuje na principu sčítání a odčítání úhlů mezi daným bodem a vrcholy bodu při ovíjení kolem polygonu. Pokud je součet úhlů roven 2π, bod se nachází uvnitř polygonu, a pokud je součet roven 0, bod se nachází vně polygonu (Rouke 2005).

$$\Omega = \sum_{i=1}^{n} \omega_i.$$

Při aplikování Winding Number se bere v úvahu jakým směrem ovíjení probíhá. Tato informace se získává vypočtením determinantu, který pracuj s vektory vedoucími z bodu q k vrcholům hrany.

- $v_1 = (u_x, u_y)$ je vektor **od bodu q k prvnímu vrcholu hrany** p_i:

$$\mathbf{v_1} = (x_i - x_q,\, y_i - y_q)$$

- $v_2 = (v_x, v_y)$ je vektor **od bodu q ke druhému vrcholu hrany** p_{i+1}:

$$\mathbf{v_2} = (x_{i+1} - x_q,\, y_{i+1} - y_q)$$

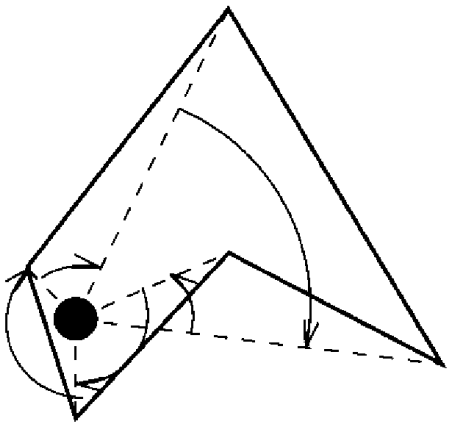
- **Determinant** (ekvivalent 2D cross product):

$$\det(\mathbf{v_1}, \mathbf{v_2}) = (x_i - x_q)(y_{i+1} - y_q) - (y_i - y_q)(x_{i+1} - x_q)$$

  Determinant odpovídá $|\mathbf{v_1}||\mathbf{v_2}| \sin \Omega_i$

$$\begin{aligned} \det < 0 \quad &\Rightarrow \quad \text{jedná se o otáčení proti směru hodinových ručiček} \\ \det = 0 \quad &\Rightarrow \quad \text{vektory jsou kolineární (žádná rotace nebo 180°),} \\ \det > 0 \quad &\Rightarrow \quad \text{jedná se o otáčení po směru hodinových ručiček} \end{aligned}$$

Díky této informaci algoritmus určí, zda má být úhel odečítán nebo přičítán. Jednotlivé otáčky v rámci metody Winding Number jsou ilustrovány na Obrázku 1, kde jsou zároveň zobrazeny vektory spojující bod q s každým vrcholem, pro který se úhly počítají.



Obrázek 1: Určení polohy bodu v polygonu metodou Winding Number (Heckbert 1994, s. 27)

Velikost úhlu Ω_i mezi vektory $\mathbf{v_1}$ a $\mathbf{v_2}$ lze vypočítat pomocí funkce atan2 (ZENVA 2023). Pro tento výpočet nejprve určíme skalární součin a determinant, který v dvourozměrném prostoru slouží jako ekvivalent k vektorovému součinu:

- **Skalární součin**:

$$\mathbf{v_1} \cdot \mathbf{v_2} = (x_i - x_q)(x_{i+1} - x_q) + (y_i - y_q)(y_{i+1} - y_q)$$

  Tento součin odpovídá $|\mathbf{v_1}||\mathbf{v_2}| \cos \theta_i$.

Celkový úhel mezi těmito vektory získáme pomocí funkce atan2:

$$\Omega_i = \text{atan2}\Big(\det(\mathbf{v_1}, \mathbf{v_2}),\, \mathbf{v_1} \cdot \mathbf{v_2}\Big)$$

Protože atan2 je kvadrantově korektní, nemusíme explicitně určovat, zda je úhel otočen ve směru hodinových ručiček (CW) nebo proti nim (CCW).A tudíž tedy stačí úhly sečíst a následně aplikovat rozhodovací pravidlo.

Rozhodovací kritérium:

- Pokud je $|\Omega_i| \approx 2\pi$ (resp. -2π), bod q se nachází uvnitř polygonu.
- Pokud je $|\Omega_i| \approx 0$, bod q se nachází vně polygonu.

2.3 Ray Crossing Algorithm

Jedná se o jeden z nejpoužívanějších algoritmů pro určování polohy bodu vůči polygonu. Algoritmus je rozšířen díky své efektivitě a možnosti využití pro konvexní i nekonvexní polygony. Metoda je založena na vodorovné přímce, která vychází z bodu q a protíná hrany polygonu (viz. Obrázek 2), přičemž právě počet protnutí definuje polohu bodu q.

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

Zda přímka opravdu protíná body je testováno pomocí dvou podmínek:

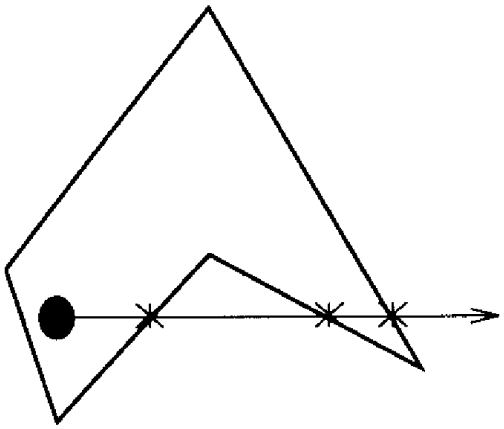
$$p_1.y \leq q.y < p_2.y \quad \text{nebo} \quad p_2.y \leq q.y < p_1.y$$

Pokud platí podmínka, následuje výpočet souřadnice x průsečíku pomocí lineární interpolace:

$$x_{\text{intersect}} = p_1.x + \frac{(q.y - p_1.y) \cdot (p_2.x - p_1.x)}{p_2.y - p_1.y}$$

Výsledkem je lichá nebo sudá hodnota. Pokud je k liché, bod q se nachází uvnitř polygonu. Pokud je k sudé, bod je vně polygonu (Bayer 2008).

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



Obrázek 2: Určení polohy bodu v polygonu metodou Ray Crossing (Heckbert 1994, s. 27)

3 Popisy algoritmu formálním jazykem

Implementce Algoritmů, *Ray Crossing* a *Winding number* jsou stručně schrnuty v pseudokódech v podkapitoláxh 3.1 a 3.2. Jedná se o funkce které lze najít v souboru *Algorithms.py*. Samotné algoritmy jsou následně iterované pro každý polygon souboru skrz *winding_number_pol()* a *ray_crossing_pols()*

3.1 Pseudokód pro Winding Number Algorithm

Algorithm 1 Winding Number

Require: q : bod, pol : polygon
Ensure: $\{-1, 1, 0\}$

$n \leftarrow$ počet vrcholů polygonu
$eps \leftarrow$ prahová hodnota blízká nule
$totalAngle \leftarrow 0$ ▷ inicializace velikosti úhlu
if bod q je vně min-max boxu **then** ▷ implementace funkce *get_bounding_box()*
    return 0 ▷ bod q leží definitivně mimo polygon
end if
for $i \leftarrow 0$ to $n - 1$ **do**
    $p_1 \leftarrow$ vrchol i
    $p_2 \leftarrow$ vrchol $(i + 1) \bmod n$
    if odchylky obou souřadnic mezi p_1 a q jsou menší než ϵ **then**
        return -1 ▷ bod q přesně odpovídá poloze vrcholu
    end if
    $v_1 \leftarrow (p_1.x() - q.x(),\ p_1.y() - q.y())$
    $v_2 \leftarrow (p_2.x() - q.x(),\ p_2.y() - q.y())$
    $determinant \leftarrow v_1[0] \cdot v_2[1] - v_1[1] \cdot v_2[0]$
    $skalar \leftarrow v_1[0] \cdot v_2[0] + v_1[1] \cdot v_2[1]$
    if $(\, determinant = 0\,)$ **and** $(\, skalar < eps\,)$ **then**
        return -1 ▷ bod q leží na hraně
    end if
    $totalAngle \mathrel{+}= \text{atan2}(determinant, skalar)$ ▷ (přičíst úhel ω mezi v_1 a v_2)
end for
if $\left|\, totalAngle - 2\pi \right| < eps$ **then**
    return 1 ▷ bod q leží uvnitř polygonu
else
    return 0 ▷ bod q leží vně polygonu
end if

3.2 Pseudokód pro algortimus Ray Crossing Algorithm

Algorithm 2 Ray Crossing

Require: q : *bod*, *pol* : *polygon*
Ensure: $\{-1, 1, 0\}$
1: $k \leftarrow$ počet průsečníků
2: $n \leftarrow$ počet vrcholů polygonu
3: $eps \leftarrow$ prahová hodnota ▷ velmi malá kladná hodnota blízká 0
4: **if** bod q je vně min-max boxu **then**
5: **return** 0 ▷ bod q leží definitivně mimo polygon
6: **end if**
7: **for** všechny vrcholy v polygonu **do**
8: $p_1 \leftarrow$ vrchol i
9: $p_2 \leftarrow$ vrchol $(i + 1) \bmod n$
10: **if** odchylky obou souřadnic mezi p_1 a q jsou menší než ϵ **then**
11: **return** -1 ▷ bod q přesně odpovídá poloze vrcholu
12: **end if**
13: $v_1 \leftarrow (p_1.x() - q.x(),\ p_1.y() - q.y())$
14: $v_2 \leftarrow (p_2.x() - q.x(),\ p_2.y() - q.y())$
15: $determinant \leftarrow v_1[0] \cdot v_2[1] - v_1[1] \cdot v_2[0]$
16: $skalar \leftarrow v_1[0] \cdot v_2[0] + v_1[1] \cdot v_2[1]$
17: **if** $(\, determinant = 0\,)$ **and** $(\, skalar < eps\,)$ **then**
18: **return** -1 ▷ bod q leží na hraně
19: **end if**
20: **if** $q.y()$ se nachází mezi souřadnicemi $p_1.y, p_2.y$ **and** $q.x() <$ interpX$(q.y(), p_1, p_2)$ **then**
21: $k \leftarrow k + 1$ ▷ $q.x()$ je menší a tudíž je intercept na pravo; přičítá se průsečík
22: **end if**
23: **end for**
24: **if** $k \bmod 2 = 1$ **then** ▷ Použijeme pravidlo sudá-lichá (even-odd rule)
25: **return** 1 ▷ bod q leží uvnitř polygonu
26: **else**
27: **return** 0 ▷ bod q leží vně polygonu
28: **end if**

4 Problematické situace a jejich rozbor (tj. simplexy) + ošetření těchto situací v kódu

4.1 Singulární případy

4.1.1 Bod na hraně polygonu

V kapitole 2.2 a 2.3 bylo popsáno jak lze pomocí Winding Number Method a Ray Crossing Algorithm zjistit zda, se bod q nachází uvnitř nebo vně polygonu. Otázkou však zůstává, jak budou algoritmy fungovat v případě, že bod q leží na hraně polygonu. V této chvíli je situace opět řešena determinantem.

V případě Winding Number Method se jedná o již zmíněný postup, který pracuje s vektory mezi bodem q a vrcholy dané hrany polygonu a jejich vzájemný skalární a vektorový součet (viz. kapitola 2.2).

kde:

- $v_1 = (u_x, u_y)$ je vektor **od bodu q k prvnímu vrcholu hrany** p_i:

$$\mathbf{v_1} = (x_i - x_q,\ y_i - y_q)$$

- $v_2 = (v_x, v_y)$ je vektor **od bodu q ke druhému vrcholu hrany** p_{i+1}:

$$\mathbf{v_2} = (x_{i+1} - x_q,\ y_{i+1} - y_q)$$

V obou případech se jedná o shodné řešení, pokud vektorový součet(det) = 0, pak jsou vektory v_1 a v_2 kolineární. Ve výsledku to znamená, že vektory v_1 a v_2 buď míří stejným směremem

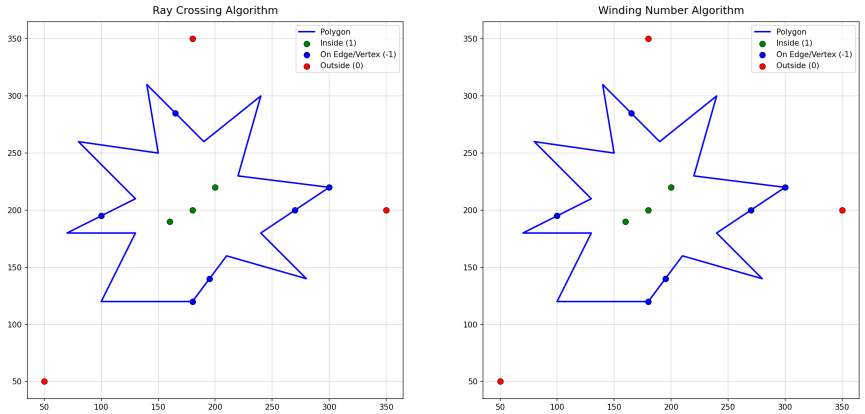
nebo od sebe. Abychom potvrdili, že se bod q vskutku nachází na hraně tak vypočteme skalární součin daných vektorů a v případě, že je skalární součin menší než nula tak tím potvrdíme, že dané vektory míří od sebe a tudíž je bod na hraně. Situaci lze předvést v této online aplikaci (PothOnProgramming, 2018). Tento způsob detekce bodu na hraně byl implementován v obou algorithmech. V případě funkce *winding_number*() se počítá s proměnnými, které se využívají pro výpočet celkového "widing number", v *ray_crossing*() jsou proměnné determinantu a skalárního součtu vypočítané pouze pro účel detekování bodů na hraně. Využitý postup lze vidět v pseudokódu.

Algorithm 3 Ověření bodu na hraně

```
cross ← v_1x · v_2y − v_1y · v_2x
dot ← v_1x · v_2x + v_1y · v_2y
if |cross| < ϵ and dot ≤ 0 then
    return −1
end if
```

4.1.2 Bod na vrcholu polygonu

Algoritmus porovnává souřadnice bodu a vrcholu, pokud je rozdíl souřadnic nulový nebo se velmi blíží k 0 bod q je označen za bod ležící na vrcholu polygonu.



Obrázek 3: Porovnání algoritmů Ray Crossing a Winding Number v obecných a singulárních případech)

Pro otestování singularních případů byl vytvořen individuální skript s názvem `test_edge_detection` pro testování detekce bodů na hraně, kde byla importována stejná třída `Algorithms()` jako v hlavní aplikaci a následně využita pro detekci polohy bodů na syntetických datech. Výsledky byly zobrazeny s využitím `matplotlib`.

V případě detekce bodu na hraně či vrcholu byl postup v obou algoritmech stejný (viz kapitola 4), nicméně samotný problém rozlišení, zda je bod uvnitř či vně polygonu, byl řešen odlišně. Jak lze vidět z Obrázku 3, oba algoritmy v testovacím skriptu obstály při určení polohy bodů.

5 Vstupní data, formát vstupních dat

Jako vstupní data byla použita shp vrstva obsahující ORP. Vrstva s polygony byla stažena z geoportálu ČÚZK v rámci geodatabáze RUIÁN.

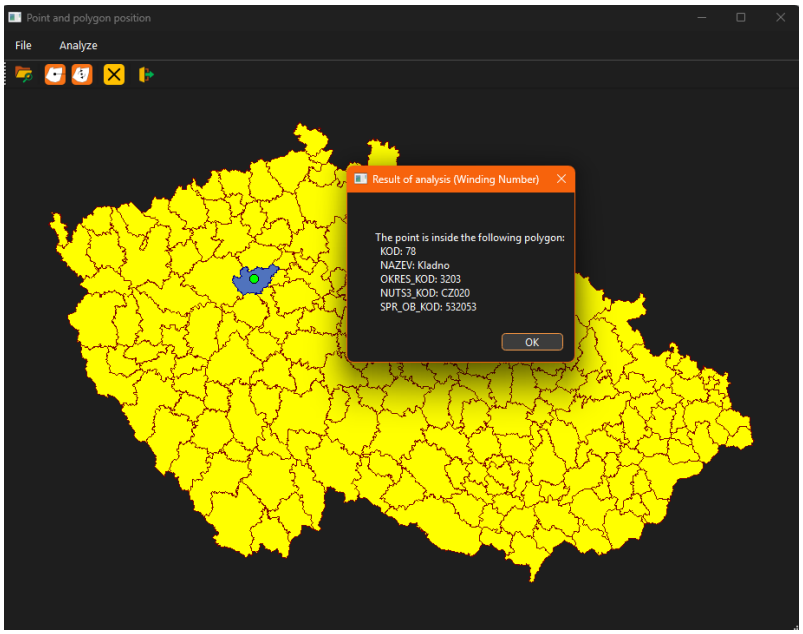
5.1 Implementace nástrojů pro nahrání dat

Pro účely načtení geodat ze souborů formátu shapefile byla využita knihovna GeoPandas. Nejprve se pomocí funkce `read_file` přečtou potřebné soubory shp a výsledná kolekce geografických entit je uložena do struktury `GeoDataFrame`. Z této struktury lze následně získat jak samotnou geometrii (v této aplikace pouze Polygon nebo Multipolygon), tak atributy spojené s každou geografickou entitou. Aby bylo možné obsah zobrazit v grafickém uživatelském rozhraní, je pro každou geometrii vypočítáno celkové ohraničení a souřadnice se následně transformují do pixelového systému

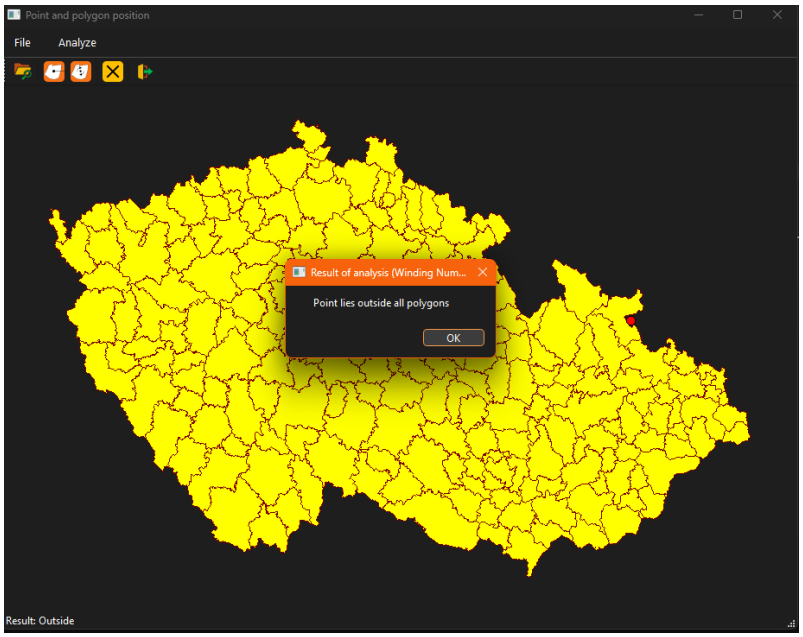
daného vykreslovacího plátna. Tato koordinátová transformace zahrnuje normalizaci podle rozsahu mapových dat (hodnoty `min_x`, `max_x`, `min_y`, `max_y`) a poté škálování do rozměrů cílového okna, přičemž se bere v potaz i inverze osy `y`.

6 Výstupní data a printscreen vytvořené aplikace

Grafická podoba aplikace byla vytvořena v prostředí Qt Designer. Při vývoji byly nakonfigurovány ovládací prvky a přidány doplňující ikony. Obrázky pro inkony byly vytvořeny v prostředí Microsoft PowerPoint a uloženy ve formátu PNG, přičemž byla zajištěna kompatibilita s Qt Creatorem. Obrázek 4 znázorňuje aplikaci s variantou bodu q v polygonu a obrázek 5 znázorňuje aplikaci s bodem q mimo polygon. Pro nový test polohy bodu q lze kliknout kamkoliv v prostředí aplikace, následně se zvolí ikona pro Winding Number Method nebo Ray Crossing Algorithm a aplikace zobrazí polohu bodu.



Obrázek 4: Bod leží uvnitř polygonu



Obrázek 5: Bod leží vně polygonu

6.1 Interakce uživatele a analýza polohy bodu

1. **Inicializace:** Uživatel zahájí pokládání bodu prvním kliknutím kdekoliv na mapě.
2. **Načtení dat:** Následně může kliknout na tlačítko `Open` v horní liště nebo zvolit v menu `File` volbu načtení souboru ve formátu Shapefile (`.shp`). Po úspěšném načtení se zobrazí polygony v žluté barvě s červeným okrajem.
3. **Umístění bodu:** Kliknutím do okna aplikace se umístí bod, který se zobrazí jako bílý kroužek.
4. **Analýza:**
   - Pro analýzu polohy bodu vůči polygonům lze využít metodu *Ray Crossing Algorithm* nebo *Winding Number Algorithm* (volitelné jak z menu `Analyze`, tak z nástrojové lišty).
   - Po provedení analýzy se bod zbarví podle výsledku:
     - **Zelená** — bod se nachází uvnitř polygonu,
     - **Červená** — bod je mimo polygon,
     - **Modrá** — bod leží na hraně nebo vrcholu polygonu.
5. **Výstup analýzy:** Aplikace zobrazí dialogové okno s detailními informacemi o výsledku včetně atributů polygonů, ve kterých se bod nachází. Současně se polygon obsahující bod vybarví modře.
6. **Nová analýza:** Pro spuštění nové analýzy stačí kliknout na jiné místo v okně, což efektivně vymaže předchozí analýzu. Pomocí tlačítka `Clear data` lze také vymazat všechna data a začít znovu.

7 Dokumentace: popis tříd, datových položek a jednotlivých metod

7.1 Třída Algorithms

Třída Algorithm slouží ke kontrole polohy bodu. Implemetnace dvou základních algoritmů Winding Number Method a Ray Crossing Algorithm detekuje zda se bod nachází uvnitř, vně nebo na hraně polygonu. Třída umožňuje práci s více polygony najednou. Pracuje s pomocnou metodou min-max boxu, který otestuje zda bod q spadá do obdélníka ohraničující polygon (viz. kapitola 8.1.1). Následně jsou procházeny všechny polygony a algoritmus detekuje polohu bodu.

Seznam funkcí:

`ray_crossing_pols(self, q: QPointF, polygons):` Iteruje ray crossing algoritmus nad n-ticí polygonů, tak aby byla daná analýza provedena pro každý polygon v souboru .shp zvlášť.

`winding_number_pols(self, q: QPointF, polygons):` Iteruje winding number algoritmus nad n-ticí polygonů, tak aby byla daná analýza provedena pro každý polygon v souboru .shp zvlášť.

`get_bounding_box(self, polygon):` Vypočítává minimální a maximální hodnoty polygonu pro vytvoření ohraničujícího obdélníku, pro rychlé vyhledávání.

`ray_crossing(self, q: QPointF, pol: QPolygonF):` Aplikuje Ray Crossing algoritmus na daný polygon pro vyhodnocení polohy bodu.

`winding_number(self, q: QPointF, pol: QPolygonF):` Aplikuje Winding Number algoritmus na daný polygon pro vyhodnocení polohy bodu.

7.1.1 Min-max Box

Algorithm 4 Get Bounding Box

Require: *polygon*
Ensure: $(x_{\min}, y_{\min}, x_{\max}, y_{\max})$
if *polygon* is empty **then**
  return $(0, 0, 0, 0)$
end if
$x_{\min} \leftarrow polygon[0].x(), \quad x_{\max} \leftarrow polygon[0].x()$
$y_{\min} \leftarrow polygon[0].y(), \quad y_{\max} \leftarrow polygon[0].y()$
for each vertex v in $polygon[1:]$ **do**
  $x_{\min} \leftarrow \min(x_{\min}, v.x())$
  $x_{\max} \leftarrow \max(x_{\max}, v.x())$
  $y_{\min} \leftarrow \min(y_{\min}, v.y())$
  $y_{\max} \leftarrow \max(y_{\max}, v.y())$
end for
return $(x_{\min}, y_{\min}, x_{\max}, y_{\max})$

Min-Max Box je nejmenší obdélník, který kompletně obklopuje daný polygon. Jeho konstrukce je založena na souřadnicích vrcholů polygonu. Nejprve se vybere počáteční vrchol, který slouží jako referenční hodnota pro stanovení minimálních a maximálních souřadnic. Při průchodu všemi vrcholy se tyto hodnoty průběžně aktualizují – pokud narazíme na souřadnici, která je menší než dosavadní minimum nebo větší než dosavadní maximum, aktualizujeme příslušnou hodnotu.

Výsledné minimální a maximální hodnoty určují čtyři vrcholy obdélníku, který se nazývá Min-Max Box. Tento box slouží také jako rychlé filtrování – pokud je bod umístěn mimo Min-Max Box, nemusí se dále testovat, zda je uvnitř polygonu, čímž se výrazně zrychluje vyhledávání potenciálních polygonů, kde by se analyzovaný bod mohl nacházet. Funkce z Algoritmu 3, popisuje tento proces, funkce je implementována jako tzv. "Helper"funkce předcházející složitějším funkcím pro vyhledávání.

7.2 Třída Draw

Třída draw slouží pro načítání, zpracování a vizualizaci dat, díky kombinaci grafického vykreslování a analýze prostorových vztahů. Metoda loadData načítá data z externího souboru. Následně funkce resizeData nastaví velikost dat tak, aby se data vešla do widgetu podle jeho aktuálních rozměrů. Vakreslené polygonů a interaktivních prvků je zajištěno funkcí paintEvent, metoda zároveň překresluje zvýrazněné polygony a označuje barevně body na základě jejich polohy vůči polygonu. Pro bod uvnitř polygonu je zvolena zelená barva, pro bod mimo polygon je zvolena červená barva a pro bod ležící na hraně nebo vrcholu polygonu je zvolena barva oranžová viz. obrázek 3. Pro vytvoření nového bodu je definována funkce mousePressEvent, která odstraní původní bod a zaznamená souřadnice nového bodu. Grafické zvýraznění analýzy probíhá na základě funkcí setResult a setContainigPolygons. Pro resetování výsledku slouží funkce clearAll, jedná se o reset celé analýzy a dat, což umožňuje vložit nová data nebo provést novou analýzu.

Třída Draw zajišťuje načítání, zpracování a vizualizaci dat, přičemž kombinuje grafické vykreslování s analýzou prostorových vztahů. Nejprve metoda `loadData` načte data z externího souboru a funkce `resizeData` přizpůsobí jejich velikost aktuálním rozměrům widgetu.

Vykreslení polygonů a interaktivních prvků realizuje funkce `paintEvent`, která zároveň překresluje zvýrazněné polygony a barevně odliší body – zeleně pro vnitřní, červeně pro vnější a modře pro body na hraně či vrcholu (viz obrázek 3). Pro vytvoření nového bodu slouží funkce `mousePressEvent`, která odstraní původní bod a zaznamená souřadnice nového. Grafické zvýraznění analýzy pak zajišťují funkce `setResult` a `setContainingPolygons`, přičemž funkce `clearAll` resetuje výsledky a data pro novou analýzu.

Seznam funkcí:

`loadData(self, filename=None)`: Načte data ze zadaného souboru. a najde bbox

`resizeData(self)`: Přizpůsobí rozměry dat k aktuálnímu widgetu qt.

`add_polygon(self, geom, width, height, x_scale, y_scale, attributes=None)`: Přidá polygon s danými parametry do vizualizace.

`mousePressEvent(self, e: QMouseEvent)`: Zaznamená nové souřadnice bodu a odstraní původní.

`paintEvent(self, e: QPaintEvent)`: Vykreslí polygony, přičemž aktualizuje zvýraznění v případě že je polygon zvýrazněný po analýze

`getQ(self)`: Vrátí aktuální bod pro analýzu. Bod vybarví dle atrbutu výsledku analýzy

`setResult(self, result)`: nastaví výsledek analýzy pro vizualizaci.

`setContainingPolygons(self, polygons)`: určí a zvýrazní polygony obsahující daný bod.

`__attrs_match(self, attrs1, attrs2)`: porovná atributy jednotlivých prvků pro ověření shody.

`getContainingPolygons(self)`: Vrátí polygon, který obsahuje zadaný bod.

`clearResults(self)`: Vymaže výsledky aktuální analýzy.

`clearAll(self)`: Resetuje data a výsledky pro novou analýzu.

`getPols(self)`: Vrátí všechny polygony.

7.3 Třída MainForm

V této třídě je vytvořeno okno se samotnou aplikací ve které celá analýza probíhá. Program je vytvořen s pomocí knihovny PyQt6, což zajišťuje otevření okna s ovládacími prvky. Hlavní částí je plátno, které slouží pro zobrazení polygonů a bodů. Nástrojová lišta obsahuje prvky pro tvorbu analýzy.

Seznam funkcí:

`setupUi(self, MainWindow)`: Inicializuje hlavní okno a všechny jeho komponenty.

`openClick(self)`: Načte data ze souboru a aktualizuje stavový řádek s výsledkem.

`resizeDisplay(self)`: Přizpůsobí velikost dat aktuálnímu widgetu a překreslí obsah.

`clearClick(self)`: Vymaže veškerá data a výsledky z plátna.

`_display_analysis_result(self, result_value, containing_polygon, algorithm_name)`: Zobrazí výsledky analýzy v dialogovém okně se stručným přehledem s atributy daného polygonu.

`rayCrossingClick(self)`: Spustí Ray Crossing algoritmus a aktualizuje vizualizaci na základě jeho výsledku.

`windingNumberClick(self)`: Spustí Winding Number algoritmus a aktualizuje vizualizaci podle získaného výsledku.

`retranslateUi(self, MainWindow)`: Nastaví texty a titulky okna podle aktuálního jazyka.

8 Závěr, možné či neřešené problémy, náměty na vylepšení

8.1 Shrnutí výsledků

V této práci byl analyzován Point-in-Polygon problém pomocí dvou algoritmů: *Winding Number Method* a *Ray Crossing Algorithm*. Oba algoritmy se ukázaly jako velmi účinné při řešení dané problematiky a jejich funkčnost byla ověřena testováním v aplikaci vytvořené v rámci úkolu.

8.2 Omezení současné implementace

Současná implementace vykazuje několik nedostatků. Například při optimalizaci hledání polohy bodu, zejména na vrstvě ORP, algoritmus momentálně iteruje skrz celý soubor, čímž se chová jako brute force řešení. Přestože funkce *get_bounding_box()* (viz kapitola 7.1.1) v mnoha případech významně urychlila výpočet, existují situace, kdy tento přístup není dostačující. Nadále, jenom *ray_crossing()*, by zvládla správně detekovat body v dírách a polygonech uvnitř děr s polygonu.

8.3 Náměty na vylepšení

- **Optimalizace vyhledávání:** Zvážit implementaci metod jako *Divide and Conquer* či jiné přístupy, které by snížily nutnost iterace přes všechny prvky.
- **Využití bounding boxu:** Rozšířit využití bounding boxu na celý shapefile (SHP) pro efektivnější lokalizaci vzdálených bodů.
- **Polygony s díry a "nested"polygony** Vybudování nástroje pro detekování bodů v díře polygonu.

Uznání

Při přípravě této práce byla využita asistence umělé inteligence, konkrétně nástroje ChatGPT a Claude, které pomohly s úpravou syntaxe, řešením chyb a při konvertování vlastního pseudokódu do programovacího jazyka Python.

Reference

[1] ANTROPIC (2025). *Claude*. https://www.claude.ai.

[2] BAYER, T. (2008). *Algoritmy v digitální kartografii*. Nakladatelství Karolinum, Praha.

[3] BAYER, T. (2025). *Point Location Problem*. Přednáška pro předmět Algoritmy počítačové kartografie, Katedra aplikované geoinformatiky a kartografie, Přírodovědecká fakulta UK. Dostupné z: https://web.natur.cuni.cz/~bayertom/images/courses/Adk/adk3_new.pdf [cit. 12. 3. 2025].

[4] CUZK (2025). *Vybraná data RÚIAN poskytovaná pro stát ve formátu SHP, ORP*. https://geoportal.cuzk.cz.

[5] MATH IS FUN. (n.d.). Vectors dot product. Retrieved March 16, 2025, from https://www.mathsisfun.com/algebra/vectors-dot-product.html

[6] MATH IS FUN. (n.d.). Vectors cross product. Retrieved March 16, 2025, from https://www.mathsisfun.com/algebra/vectors-cross-product.html

[7] OPENAI (2025). *ChatGPT*. https://www.chat.com.

[8] POTHONPROGRAMING (2018). *Get The Angle Between 2D Vectors*. https://www.youtube.com/watch?v=_VuZZ9_58Wg.

[9] HECKBERT, P.S. (ed.) (1994). *Graphics Gems IV. The Graphics Gems Series*. San Francisco: Morgan Kaufmann. ISBN-0-12-336155-9.

[10] ROURKE, O. J. (2005): Computational Geometry in C. Cambridge University Press, Cambridge.

[11] ZENVA (2023). *Atan2 C++ Tutorial – Complete Guide*. GameDev Academy. https://gamedevacademy.org/atan2-c-tutorial-complete-guide/?hl=en-US#Calculating_angle_between_two_vectors.