UNIVEZITA KARLOVA
PŘÍRODOVĚDECKÁ FAKULTA

ALGORITMY POČÍTAČOVÉ KARTOGRAFIE

Geometrické vyhledávání bodu

Daniela DANČEJOVÁ
Anna KOŽÍŠKOVÁ
Praha 2023

Zadání

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

Výstup: $P_i, q \in P_i$.

Nad polygonovou mapou implementujte Ray Crossing Algorithm pro geometrické vyhledání incidujícího polygonu obsahujícího zadaný bod q.

Nalezený polygon graficky zvýrazněte vhodným způsobem (např. vyplněním, šrafovaním, 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 rozlišující 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*
Max celkem:	**35b**

Popis a rozbor problému

Často řešenou úlohou v oblasti počítačové grafiky a digitální kartografie bývá určování vzájemného vztahu polohy daného bodu a uzavřené oblasti (Bayer 2008). Řešení tohoto úkolu, známého jako *Point-in-Polygon Problem* (PIPP), poskytuje informaci o tom, zda se bod nachází uvnitř, vně nebo na hranici souvislého uzavřeného útvaru (polygonu). Možnosti řešení se pro konvexní a nekonvexní útvary liší zejména v náročnosti algoritmů potřebných pro zohlednění specifických případů polohy vrcholů jednotlivých polygonů (Rourke 2005). Bayer (2023) rozlišuje 2 základní techniky řešení PIPP:

- převedení problému na vztah bodu a mnohoúhelníku;
- planární dělení roviny.

V prvním případě jde o opakované určování polohy bodu vzhledem k mnohoúhelníku; tato technika je snadno implementovatelná, avšak pomalejší. V druhém případě je rovina rozdělena na množinu pásů či lichoběžníků, čímž vzniká *trapezoidální mapa* (Rourke 2005). Rozdělení rovin vede k rychlejšímu nalezení řešení, implementace však bývá obtížnější.

Zjištění vzájemné polohy bodu a konvexního útvaru je možné provést jednoduchými algoritmy, například testováním polohy bodu vůči každé hraně útvaru (tzv. ***Half-plane test***, složitost $O(n)$). Nacházejí uplatnění především v triangulačních algoritmech. Mnoho takových algoritmů však nelze použít samo o sobě pro nekonvexní útvary, které se často vyskytují právě v oblasti geoinformatiky a kartografie.

Existují dva základní algoritmy vhodné pro nekonvexní útvary schopny detekovat vzájemnou polohu bodu a uzavřené oblasti: ***Winding Number Algorithm*** a ***Ray Algorithm***. Oba tyto algoritmy mají časovou složitost $O(n)$, přičemž jeden z nich je výrazně rychlejší (Rourke 2005). Těmto algoritmům bude níže věnována samostatná pozornost.

Winding Number Algorithm

Winding Number algoritmus je prvním z řešení, kterým lze analyzovat polohu bodu pro nekonvexní mnohoúhelníky. V české literatuře se může označovat jako metoda *ovíjení*. Algoritmus je založen na sčítání/odečítání úhlů, který svírá bod s jednotlivými segmenty polygonu. Za předpokladu, že součet úhlů je 2π, pak se bod nachází uvnitř polygonu. Segmentem je uvažována přímka tvořená dvěma po sobě následujícími body v polygonu.

Zjednodušeně řečeno, bude-li se pozorovatel dívat z bodu do každého vrcholu v polygonu a bude postupně sčítat úhly, o které se otáčí, součet výsledného úhlu bude 360° (Bayer 2008, Žára 2004).

Podstata algoritmu

Mějme uzavřenou oblast O v $\mathbb{R}^2$, jejíž hrany tvoří množinou bodů $P = \{P_1, ..., P_n, P_1\}$ a bod q. Výsledná hodnota algoritmu Winding Number je součet všech uhlů, které opíše průvodič v polygonu.

Platí vztah:

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

Aby bylo možné spočítat celkový úhel pro daný polygon, je nejprve nutné analyzovat polohu bodu q vůči každé přímce, ktertá je definovaná dvěma po sobě následujícími vrcholy p_i a p_{i+1} jež tvoří segment v polygonu. Vzájemná poloha bodu a dané přímky se vyšetří pomocí Half-plane testu, kde mohou nastat 3 situace (Bayer 2023):

- bod q leží vpravo od přímky,
- bod q leží vlevo od přímky,
- bod q leží na přímce.

Jako testovací kritérium se použije vztah pro výpočet determinantu matice, která se skládá z vektorů $\vec{p} = (p_x, p_y)$ a $\vec{s} = (s_x, s_y)$:

$$\begin{aligned} \vec{p} &= p_{i+1} - p_i, \\ \vec{s} &= q - p_i. \end{aligned} \tag{2}$$

Vztah pro výpočet determinantu:

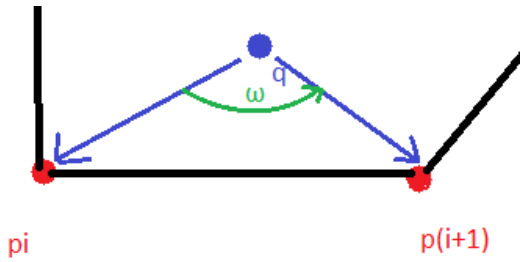
$$det = (p_x * s_y) - (s_x * p_y). \tag{3}$$

Pokud

$$det \begin{cases} < 0, & \text{bod } q \text{ leží vpravo od přímky} \\ > 0, & \text{bod } q \text{ leží vlevo od přímky} \\ = 0, & \text{bod } q \text{ leží na přímce} \end{cases} \tag{4}$$

Následně je potřeba procházet jednotlivé hrany v polygonu a sčítat, či odečítat všechny úhly $\omega_i + \omega_{i+1} + ... + \omega_n$ ve směru hodinových ručiček (nebo v opačném směru) v závislosti na výsledku determinatu. Pro výpočet úhlu ω je nutné spočíst vektory $\vec{u} = (u_x, u_y)$ a $\vec{v} = (v_x, v_y)$ spočtené jako (viz obrázek 1):

$$\begin{aligned} \vec{u} &= p_i - q, \\ \vec{v} &= p_{i+1} - q. \end{aligned} \tag{5}$$



Obrázek 1: Úhel vektorů $\vec{u}$ a $\vec{v}$

Velikost úhlu se pak spočítá podle vztahu:

$$\cos(\omega) = \frac{\vec{u} \cdot \vec{v}}{||\vec{u}|| \cdot ||\vec{v}||} \tag{6}$$

Pro následující kroky je nutné pracovat s absolutní hodnotou úhlu ω, jelikož se bude v dalším kroku rozhodovat o jeho přičtení, nebo odečtení.

$$|\omega| = \arccos\left(\frac{\vec{u} \cdot \vec{v}}{||\vec{u}|| \cdot ||\vec{v}||}\right) \tag{7}$$

Bude-li

$$det \begin{cases} < 0, & \text{pak} - = \omega \\ > 0, & \text{pak} + = \omega \end{cases} \tag{8}$$

Následně se všechny úhly sečtou podle vztahu (1) a pokud:

$$\Omega \begin{cases} = 2\pi, & q \in O, \\ < 2\pi, & q \notin O. \end{cases} \tag{9}$$

Speciální případy algoritmu

Takto výše popsaný algoritmus bude schopný detekovat bod uvnitř polygonu a mimo něj. Následující část bude věnována případu, kdy se bude analyzovaný bod q nacházet na hraně polygonu. Vztah (4) se již částečně o tomto případu zmiňuje. Bod bude ležet na hraně právě tehdy, když

$$det = 0. \tag{10}$$

Pro následnou implementaci však pouze tato podmínka nestačí, a proto se dále dá detekovat bod na hraně právě tehdy, když

$$\omega = \pi. \tag{11}$$

Výše uvedené dva vztahy ovšem nezohledňují situaci, kdy se bod bude nacházet v jednom z vrcholů množiny P. Pro ošetření této možnosti se využije podmínky:

$$\text{if } q = p_i \tag{12}$$

pak se bod nachází ve vrcholu - tedy na hraně polyognu.

Pseudokód

Na závěr této sekce jsou výše zmíněné kroky shrnuty v pseudokódu, který je implementován v metodě *windingNumberAlgorithm* v souboru *algorithms.py*.

Algorithm 1 Winding Number

Require: q : *bod*, *pol* : *polygon*
Ensure: $-1, 0, 1$
  $n \leftarrow$ *délka polygonu*
  eps $\leftarrow$ *prahová hodnota* ▷ velmi malá kladná hodnota blízká 0
  $totalAngle \leftarrow 0$ ▷ inicializace velikosti úhlu
  for všechny vrcholy v polygonu **do**
    if bod je vrchol **then**
      vrať hodnotu -1 ▷ bod leží na hraně
    end if
    Spočti vektor p: pi+1 - i
    Spočti vektor s: q - pi
    Spočti determinant z vektorů p a s
    Spočti vektor u: pi - q
    Spočti vektor v: pi+1 - q
    Spočti úhel ω z vektorů u a v
    if determinant je větší než 0 **then**
      $totalAngle \leftarrow +\omega$
    else if determinant je menší než 0 **then**
      $totalAngle \leftarrow -\omega$
    end if
    if determinant je 0 a zároveň je úhel (u a v) - π menší než eps **then**
      vrať hodnotu -1 ▷ bod leží na hraně
    end if
  end for
  if absolutní hodnota(z (abs. hodnoty totalAngle)) - 2π je menší než eps **then**
    vrať hodnotu 1 ▷ bod leží uvnitř polygonu
  else
    vrať hodnotu 0 ▷ bod leží vně polygonu
  end if

Ray Crossing Algorithm

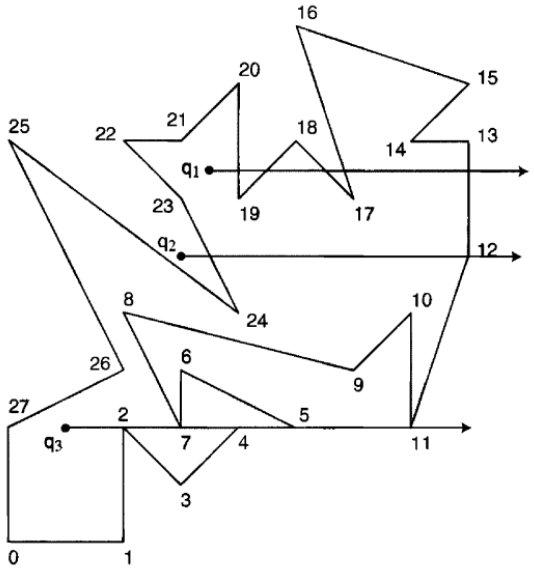
Další možné řešení PIPP pro nekonvexní útvary představuje *Ray Crossing Algorithm*, který bývá také označován jako *Ray Algorithm*, *Ray Casting Algorithm* nebo *Even-Odd Rule Algorithm*, v češtině i jako *paprskový algoritmus* (Bayer 2008). Tento algoritmus zjišťuje vzájemný vztah polohy bodu a útvaru na základě počtu průsečníků, které tvoří polopřímka vedena z daného bodu s hranami tohoto útvaru. V případě, že je tento počet lichý, uvažovaný bod se nachází uvnitř polygonu, je –li tento počet sudý nebo rovný nule, uvažovaný bod se nachází vně polygonu. V případě, že je s uvažovaným bodem totožný právě jeden průsečník, tento bod se nachází na hraně polygonu (obrázek 2).

Tento algoritmus je možné dále modifikovat tak, abychom jeho implementaci zjednodušili a zabránili vzniku problematických situací.

Podstata algoritmu

Mějme uzavřenou oblast O v $\mathbb{R}^2$, jejíž hrany tvoří množinou bodů $P = \{P_1, ..., P_n, P_1\}$ a bod q. Uvažujme vodorovnou testovací přímku r procházející bodem q takovou, že

$$r(q) : y = y_q. \tag{13}$$



Obrázek 2: Detekce polohy bodů vzhledem k polygonu za použití Ray Crossing algoritmu (převzato z Rourke 2005, s. 240)

Počet průsečníků k přímky r s oblastí O pak určuje polohu bodu q vůči O tak, že

$$k\%2 = \begin{cases} 1, & q \in O, \\ 0, & q \notin O. \end{cases} \tag{14}$$

Tato varianta algoritmu však neřeší případy, když $r(q)$ prochází vrcholem P_i nebo hranou a neumí detekovat stav, když q leží na hraně δ. Pro vylepšení je možné provést modifikaci algoritmu s redukcí ke q a rozdělením r na dvě polopřímky r_1 a r_2 s opačnou orientací: nechť je r_1 levostranná a r_2 pravostranná polopřímka. Udržujme si počet levostranných a pravostranných průsečníků k_l a k_r. Zavedeme–li si lokální souřadnicový systém s počátkem v bodě q a osami x', y' a polopřímky $r_1(q)$ a $r_2(q)$ ztotožníme s osou x' tak, že je možné je popsat rovnicí (Bayer 2023)

$$y' = 0, \tag{15}$$

můžeme provést redukci bodů $p_i = [x_i, y_i]$ ke q:

$$\begin{aligned} x'_i &= x_i - x_q, \\ y'_i &= y_i - y_q. \end{aligned} \tag{16}$$

Pokud průsečník $M = [x'_m, y'_m = 0]$ segmentu (hrany) oblasti O a osy x' (jedné z polopřímek x', y') existuje, můžeme ho určit ze vztahu

$$x'_m = \frac{x'_{i+1} y'_i - x'_i y'_{i+1}}{y'_{i+1} - y'_i}. \tag{17}$$

Podmínky existence průsečníku M s jednou z polopřímek $r_1(q)$, $r_2(q)$ udávají vztahy:

- pro levou polrovinu:

$$t_l = y_{i+1} < y_q \neq y_i < y_q, \tag{18}$$

- pro pravou polrovinu:

$$t_r = y_{i+1} > y_q \neq y_i > y_q, \tag{19}$$

kde t_l, t_r nabývají hodnoty `True` nebo `False`. Průsečník M se pak vypočte pro každou polopřímku zvlášť:

- pokud t_l = `True` $\wedge$ x'_m **<** 0, inkrementujeme k_l,
- pokud t_r = `True` $\wedge$ x'_m **>** 0, inkrementujeme k_r.

Jinak řečeno: Průsečníky pro levou a pravou část budeme započítávat v případě, leží –li počáteční a koncový bod protnutého segmentu v jiné polrovině (vrchní nebo spodní). Pak

$$q = \begin{cases} \in \delta\mathrm{O}, & k_l\%2 \neq k_r\%2, \\ \in O, & k_r\%2 = 1, \\ \notin O, & \text{jinak.} \end{cases} \tag{20}$$

Speciální případy algoritmu

Při řešení PIPP pomocí *Ray Algorithm* může docházet k případům, které je nutno dodatečne ošetřit.

Pokud je zvolený bod q totožný s jedním z bodů p_i, pak je možné prohlásit, že $q \in \delta O$.

Pokud přímka $r(q)$ prochází vrcholem oblasti O, může nastat detekce dvou průsečníků (koncový bod jednoho segmentu a počáteční bod druhého segmentu). Situaci je možné ošetřit započtením tohoto vrcholu jako průsečníku jenom jednou (Bayer 2008, Rourke 2005).

Pokud platí, že

$$y_{i+1} - y_i = 0, \tag{21}$$

pak body tvoří p_{i+1} a p_i horizontální hranu a výpočet průsečníku ze vztahu (17) nebude možný. Ve vlastní implementaci algoritmu se v tom případě pokračuje nasledující iterací.

Může nastat situace, když se bod p_i chybně zařadí do vrchní nebo spodní polroviny oblasti O, pokud tento bod leží velmi blízko testovací přímky $r(q)$. Je proto vhodné zavést prahovou hodnotu ε, která tento případ ošetří:

$$|y_i - y_q| \leq \varepsilon. \tag{22}$$

Pseudokód

Vlastní implementace *Ray Crossing Algorithm* je níže shrnuta v pseudokódu. Nachází se v metodě *rayCrossingAlgorithm* v souboru *algorithms.py*.

Algorithm 2 Ray Crossing

Require: $q : bod, pol : polygon$
Ensure: $-1, 1, 0$
  $kr \leftarrow$ *počet pravostranných průsečníků*
  $kl \leftarrow$ *počet levostranných průsečníků*
  $n \leftarrow$ *délka polygonu*
  for všechny vrcholy v polygonu **do**
    Spočti x_i, y_i
    if bod je vrchol **then** ▷ bod leží na hraně
      vrať hodnotu -1
    end if
    Spočti x_{i+1}, y_{i+1}
    if $(y_{i+1} - y_i) == 0$ **then** ▷ horizontální segment
      pokračuj novou iterací
    end if
    Spočti průsečník x'_m
    if $y_{i+1} < y_q \neq y_i < y_q$ **then** ▷ spodní segment
      if $x'_m < 0$ **then** ▷ průsečník v levé polrovině
        Inkrementuj k_l
      end if
    end if
    if $y_{i+1} > y_q \neq y_i > y_q$ **then** ▷ vrchní segment
      if $x'_m > 0$ **then** ▷ průsečník v pravé polrovině
        Inkrementuj k_r
      end if
    end if
  end for
  if $k_l\%2 \neq k_r\%2$ **then**
    vrať hodnotu -1 ▷ bod leží na hraně
  else if $k_r\%2 = 1$ **then**
    vrať hodnotu 1 ▷ bod leží uvnitř polygonu
  else
    vrať hodnotu 0 ▷ bod leží vně polygonu
  end if

Vlastní implementace

Součástí úlohy bylo kromě samotných algoritmů analyzujících polohu bodu v prostoru také vytvoření přívětivého uživatelského prostředí ve frameworku QT, ve kterém je demonstrována funkčnost obou výše zmíněných algoritmů na zvolené polygonové mapě.

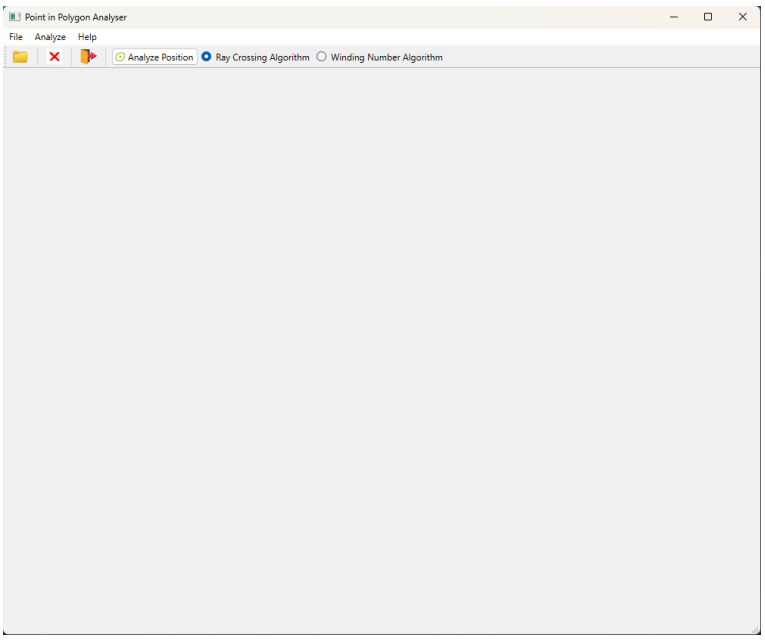
Vstupní data

Jako vstupní data byly zvoleny geografická data s kódem `EPSG:5514` (Křovákovo zobrazení). Aplikace umožňuje otevřít, zpracovat a vykreslit prostorové souřadnice pro soubory ve formátu `.JSON` a `.GEOJSON`. K aplikaci jsou přiloženy testovací data v obou formátech v adresáři */input_files/*.

Výsledkem analýzy je grafické vykreslení příslušnosti bodu k polygonu, případně k více polygonům v situaci, kdy je bod umístněn na rozhraní dvou nebo více polygonů.

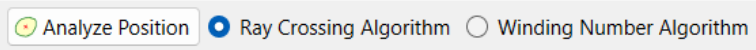
Aplikace

Grafické rozhraní aplikace (obrázek 3) bylo vytvořeno v prostředí `Qt Creator 9.0.1` a dále upravováno v prostředí programovacího jazyka `Python 3.11`. Uživateli je umožněno otevřít soubor ve vlastním adresáři a v těle aplikace kliknutím levého tlačítka myši umístnit vlastní bod q pro analýzu jeho polohy vůči vstupním datům.



Obrázek 3: Grafické rozhraní aplikace

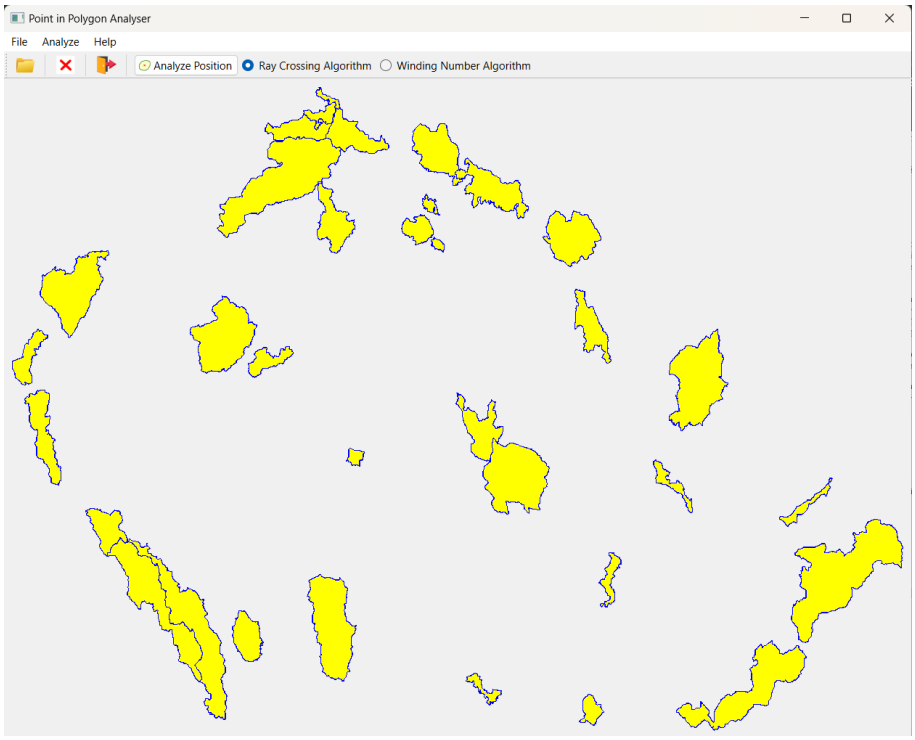
Součástí rozhraní je možnost volby matoedy analýzy mezi algoritmy *Ray Crossing* a *Winding Number* (obrázek 4). Po zvolení algoritmu a kliknutí na `Analyze Position` (případně využití klávesové zkratky `Ctrl+A`) se zvýrazní polygon, ve kterém se bod q nachází. V případě, že se bod q nachází na hraně dvou polygonů, resp. na rozhraní více polygonů, zvýrazní se všechny incidující polygony.



Obrázek 4: Panel nástrojů pro volbu algoritmů

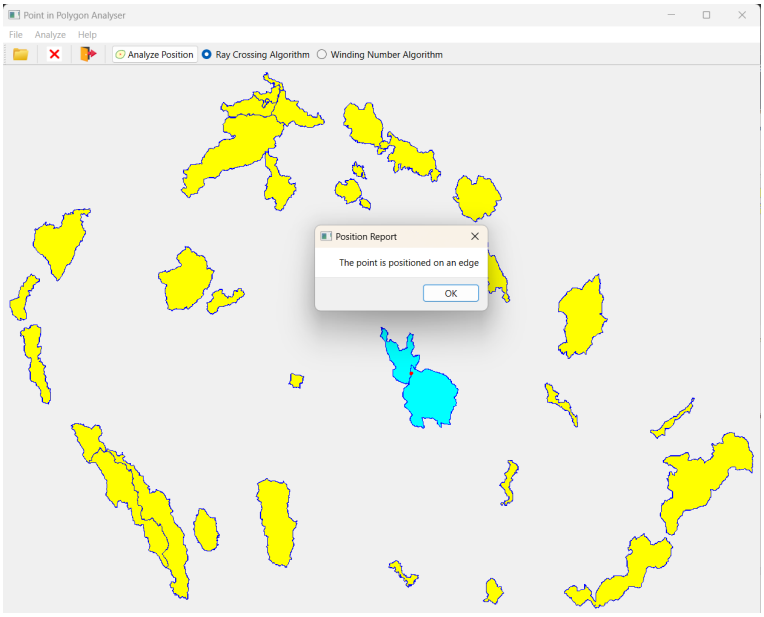
Uživatel má také možnost vstupní polygonovou vrstvu smazat a nahrát novou vrstvu. V případě otevření nového souboru se předešlá vrstva smaže a nahraje se vrstva nová.

Vstupní vrstva se nahraje tak, aby vyplnila co nejvíc prostoru v okně aplikace (obrázek 5). S měnící se velikosti okna se ale již nahraná vrstva nemění, je tedy nutno znovu otevřít soubor tak, aby se data vhodně přeškálovála vzhledem k nové velikosti okna.



Obrázek 5: Ukázka nahrané polygonové vrstvy

Polygon, ve kterém se zvolený bod může nacházet, se zabarví na modrou barvu. Pokud se zvolený bod nachází na hraně dvou polygonů nebo na vrcholu více polygonů, na tuto skutočnost upozorní vyskakovací okno (obrázek 6) a všechny tyto polygony se zabarví na modrou barvu.



Obrázek 6: Vyskakovací okno při detekci hrany

Třídy a metody

Funkční chod aplikace si vyžaduje tři povinné soubory, v kterých jsou implementovány potřebné třídy a metody: `mainform.py`, `algorithms.py` a `draw.py`.

Třída MainForm

Třída MainForm ze souboru `mainform.py` zabezpečuje inicializaci okna aplikace, vrchní lišty, panelu nástrojů, ikon a tlačítek. Zároveň přepojuje jednotlivé interaktivní položky okna s metodami, které vykonají specifické akce. Týkají se především otevření souboru, přepínání algoritmů, provedení analýzy polohy bodu vůči polygonu apod. Část této třídy byla vygenerována v prostředí `Qt Creator 9.0.1` (metody `setupUi()` a `translateUi()`). Níže jsou vyjmenovány nově implementované metody:

- `switchToRayCrossing()`

  Nastaví algoritmus pro analýzu polohy bodu na *Ray Crossing*.

- `switchToWindingNumber()`

  Nastaví algoritmus pro analýzu polohy bodu na *Winding Number*.

- `analyzePosition()`

  Provede analýzu polohy bodu q vůči polygonové mapě. Bod a polygonovou mapu (uloženou v seznamu) zavolá z třídy `Draw`. V iteraci přiřadí každému polygonu hodnotu, podle které se polygon zabarví a určí se tak jeho vztah k bodu q. Může zavolat metodu pro vyskakovací okno `onEdgePopup()` .

- `processFile()`

  Zabezpečuje otevření souboru. Samotný soubor načte do proměnné pomocí metody `openFile()` a následně zavolá metodu `clearEvent()` pro vyprázdnění okna. Pokud je vstupní `JSON` nebo `GeoJSON` nečitelný (např. nestandardní hierarchie položek v slovnících), uživatele na to upozorní vyskakovacím oknem.

- `openFile()`

  Otevře `JSON` nebo `GeoJSON` soubor a načte ho do proměnné.

- `exitClick()`

  Ukončí aplikaci.

- `clearClick()`

  Zavolá metodu `clearEvent()` z třídy `Draw`, která vyprázdní okno.

- `aboutClick()`

  Otevře repozitář s aplikací v portálu `GitHub`.

- `onEdgePopup()`

  Vytvoří vyskakovací okno v případě, že se zvolený bod nachází na hraně nebo na vrcholu polygonu.

Třída Algorithms

V této třídě jsou obsaženy metody, které byly detailně popsány v předchozí kapitole věnované teorii algoritmů Winding Number a Ray Crossing. Obsahuje metody:

- `rayCrossingAlgorithm(q, pol),`

  pro předané parametry `q` (bod) a `pol` (polygon) analyzuje polohu bodu `q` vůči polygonu `pol` pomocí algoritmu Ray Crossing

- `windingNumberAlgorithm(q, pol)`

  pro předané parametry `q` (bod) a `pol` (polygon) analyzuje polohu bodu `q` vůči polygonu `pol` pomocí algoritmu Winding Number

Třída obsahuje jednu proměnnou `default_alg`, která nastavuje metodu rayCrossingAlgorithm jako primární metodu pro analýzu bodu a polygonu.

Třída Draw

Třída Draw ze souboru `Draw.py` slouží pro inicializaci proměnných nesoucí prostorovou informaci, načítání a vykreslování geoprostorové informace. Při spuštění aplikace se inicializují proměnné pro bod, polygonový list a polygonový status:

- `self.__q,`
- `self.__polyg_list,`
- `self.polyg_status,`

Třída Draw obsahuje následující metody:

- `mousePressEvent(e:QMouseEvent)`

  Zodpovídá za změnu pozice bodu.

- `paintEvent(e:QPaintEvent)`

  Vykresluje objekty (bod a polygony) na plátno (Canvas).

- `getPoint()`

  Vrací souřadnice bodu.

- `getPolygonList()`

  Vrací seznam vstupních polygonů.

- `clearEvent()`

  Maže vyreslené objekty (bod a polygony) na plátně (Canvas).

- `findBoundingPoints(p:QPointF, xmin, ymin, xmax, ymax)`

  Nalezne minimální a maximální souřadnice pro ohraničení polygonu (tzv. bounding box).

- `resizePolygons(xmin, ymin, xmax, ymax)`

  Roztáhne vstupní data na plátno podle velikosti okna aplikace.

- `loadData(data)`

  Prochází slovník ze vstupního souboru `.JSON/.GEOJSON` a načítá geoprostorovou informaci.

Závěr

V této úloze byly představeny a implementovány algoritmy *Ray Crossing* a *Winding Number*, které jsou schopny řešit *Point-in-Polygon Problem* pro nekonvexní útvary. Jejich funkčnost byla testována pomocí vytvořené aplikace, která umožňuje provést analýzu polohy bodu vůči zvolené polygonové mapě ve formátu `JSON` nebo `GeoJSON`. Tato aplikace byla vytvořena v programovacím jazyce `Python 3.11`.

Samotnou aplikaci je možné vylepšit o přidání podpory pro jiné formáty obsahující prostorovou informaci (např. `shapefile`). Současný stav aplikace navíc neumožňuje načíst `JSON` a `GeoJSON` soubory v jiném než nestardandním formátu – je nutno zabezpečit nalezení souřadnic v různě uspořádaných slovnících a seznamech v `JSON`u a `GeoJSON`u. Další možné vylepšení je implementace dynamické měnění velikosti načteného obsahu vzhledem k velikosti okna. Vhodným doplňkem může být i možnost přiblížení/oddálení obsahu okna pro zpřesnení umístění bodu.

Vytvořená aplikace je volně dostupná přes `GitHub` na adrese `https://github.com/koziskoa/APK_2023/tree/master/point_in_polygon`.

Literatura

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

[2] BAYER, T. (2023): Point Location Problem. Přednáška pro předmět Algoritmy počítačové kartografie, Katedra aplikované geoinformatiky a kartografie. Přírodovědecká fakulta UK, dostupné *zde* (cit. 13. 3. 2023).

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

[4] ŽÁRA, J., BEDŘICH, B., SOCHOR, J., FELKEL, P. (2004): Moderní počítačová grafika. Computer Press, Brno.