Prírodovedecká fakulta Univerzity Karlovej v Prahe

Katedra aplikovanej geoinformatiky a kartografie

Algoritmy počítačovej kartografie

Zadanie č. 1: Geometrické vyhľadávanie bodov

Bc. Filip Kradijan Seider

Bc. Peter Dúbrava

2023/2024

Zadanie

Vstup: Súvislá polygónová mapa n polygónov {P1, ..., Pn}, analyzovaný bod q.

Výstup: Pi, q ∈ Pi.

Nad polygónovou mapou implementujete algoritmus priesečníka lúča pre geometrické vyhľadanie polygónu, ktorý obsahuje zadaný bod q.

Nájdený polygón graficky zvýraznite vhodným spôsobom (napríklad vyplnením, šrafovaním, blikaním). Grafické rozhranie vytvorte s využitím frameworku QT.

Pre generovanie nekonvexných polygónov môžete navrhnúť vlastný algoritmus alebo použiť existujúce geografické dáta (napríklad mapa európskych štátov).

Polygóny budú načítané z textového súboru vo vami zvolenom formáte. Pre reprezentáciu dát jednotlivých polygónov použite model špagiet.

Hodnotenie:

Úloha	**Body**
Detekcia polohy bodu s rozlíšením, či sa nachádza v polygóne alebo mimo neho *(Ray Algorithm)*	10
Ošetrenie singulárnych prípadov pri *Winding Number Algorithm*: bod leží na hrane polygónu	+5
Analýza polohy bodu (vo vnútri/mimo polygónu) metódou *Winding Number Algorithm*	+10
Rýchle vyhľadávanie potencionálnych polygónov (bod vo vnútri *min-max box*)	+10
Spolu	**35**

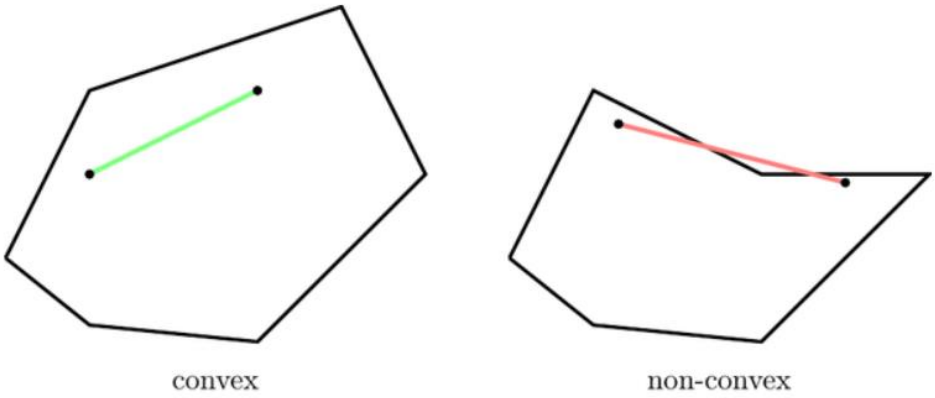
Problematika

V počítačovej geometrii je jedným z fundamentálnych problémov určiť vzťah medzi polohou dvoch objektov, v našom prípade bodu a polygónu v 2D. Kvôli tomu boli vyvinuté rôzne metódy na čo najefektívnejšie riešenie tohto problému (Ajmera, 2024). Tento problém je možné vyriešiť buď využitím techniky prevedenia problému na vzťah bodu a mnohouholníka (pomalšia), či pomocou plošného delenia roviny fungujúcej na princípe binárneho stromu (rýchlejšia) (Bayer. 2024).

Bayer (2008) rozlišuje tri situácie, ku ktorým môže dôjsť pri vzájomná polohe medzi bodom a polygónom:

- bod q leží priamo v uzavretej oblasti mnohouholníka P
- bod q je na hranici mnohouholníka P
- bod q sa nachádza mimo mnohouholníka P

Mnohouholníky môžeme rozdeliť na konvexné a nekonvexné, čo hrá dôležitú úlohu pre určovanie správnej metódy na analýzu problému. Rozdiel medzi týmito mnohouholníkmi spočíva v tom, že pri konvexných je možné spojiť dva ľubovoľné body priamkou bez toho, aby táto priamka opustila uzavretú oblasť (obr. 1), v ktorej sa tieto body nachádzajú (Fialová, 2020). Tieto typy mnohouholníkov je možné odlíšiť aj podľa veľkosti vnútorných uhlov, kedy konvexný mnohouholník má všetky uhly menšie ako 180° a pri nekonvexnom mnohouholníku existuje aspoň jeden uhol väčší ako 180°.



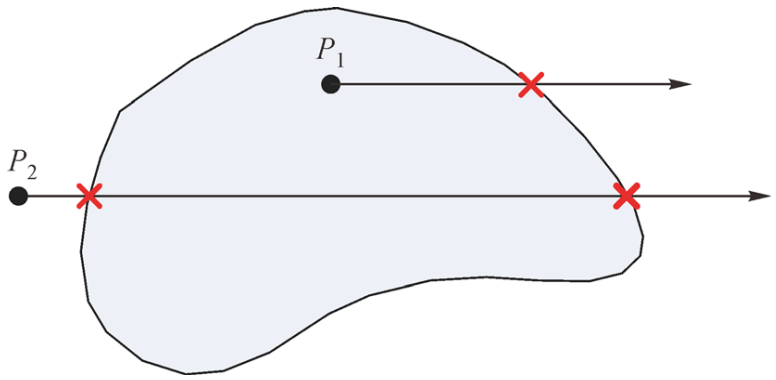
Obr. 1: Rozdiel medzi konvexným a nekonvexným polygónom (zdroj:Inginious, 2024)

Vzťah medzi konvexným mnohouholníkom a bodom je možné vyjadriť za pomoci metódy *Half-plane test*, ktorá testuje polohu bodu voči každej hrane mnohouholníka. Táto metóda je využívaná pre vytváranie trojuholníkovej siete okolo bodov. Ďalšou metódou je *Ray Crossing,* ktorú je využívané zovšeobecniť aj pri nekonvexných mnohouholníkoch (Bayer, 2024).

Pri nekonvexných mnohouholníkoch sa spomínajú hlavne dve metódy pre určenie vzájomnej polohy bodu a polygónu. Jedná sa už o spomínanú metódu *Ray Crossing* a metódu *Winding Number.* Tieto metódy majú rovnakú zložitosť, ale každá má inú východu. *Ray Crossing* je až dvadsaťkrát rýchlejšia, ale nedokáže dobre ošetriť singulárne prípady (Bayer 2024, Rourke 1998).

Ray Crossing Algorithm

*Ray Crossing Algorithm***,** alebo aj lúčový algoritmus, je jednoduchý a efektívny spôsob na zistenie, či sa daný bod nachádza vo vnútri alebo mimo mnohouholníka. Tento algoritmus overuje, koľkokrát sa „lúč“ (polpriamka), začínajúci v testovacom bode a smerujúci do nekonečna v ľubovoľnom pevne určenom smere, pretína s hranami mnohouholníka. Výnimkou je prípad, keď daný bod leží priamo na hranici mnohouholníka. V takomto prípade platí, že ak je počet priesečníkov je párny, tak sa nachádza vonku, a vo vnútri ak je nepárny počet priesečníkov (Wang, 2016). Tento algoritmus však vždy nepodáva správne výsledky. Pokiaľ sa bod nachádza presne na hrane polygónu, alebo pokiaľ bod leží v tesnej blízko hranici polygónu, môže dôjsť pri výpočte k chybám. Tieto chyby sú spôsobené dátovým typom *floating point*, ku ktorým dochádza pri zaokrúhľovaní. Môže sa tam stať, že takéto body po zaokrúhlení na určitý počet miest vo vnútri polygónu (Kačmařík, 2023). Ďalšou nevýhodou tohto algoritmu je lineárna časová náročnosť, ktorá sa prejavuje pri zložitejších polygónoch (Galetzka, 2017).



Obr. 2: Ray Crossing algoritmus (zdroj: Wang, 2016)

Podstata:

Pokiaľ máme v rovine (R^2) bod q a uzavretú oblasť O (polygón), ktorej hrany sú tvorené množinou bodov $P = \{p_1, \dots p_n, p_1\}$ môžeme uvažovať vodorovnou priamkou r prechádzajúcou bodom q,

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

Počet priesečníkov (k) priamky r s uzavretou oblasťou O nám určuje polohu bodu (q) voči polygónu,

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

Pri tejto metóde je potrené dať pozor sa situácie, kedy lúč $r(q)$ prechádza vrcholom polygónu alebo je kolineárny (leží na rovnakej priamke) s jednou alebo viacerými stranami polygónu. Pokiaľ prechádza vrcholom, tak hrozí detekcia dvoch priesečníkov a prípade kolinearity nie je možný výpočet priesečníkov. V takom prípade sa polpriamka nahradzuje úsečkou, ktorej koncový bod určuje min-max box. Keď bol q leží na hrane ∂, tak nie sme schopný určiť jeho stav. Je preto potrebné modifikovať algoritmus a redukovať priamku r na dve polpriamky s opačnou orientáciou (r_1 ľavostranná a r_2 pravostranná) a budeme počet ich priesečníkov s polygónom udržovať. Ďalej je potrebné zadať lokálny súradnicový systém s osami x', y' v počiatkom bode q, pričom obe polpriamky budú totožné s osou x' a $y' = 0$ (Bayer, 2024).

Okrem redukcie priamky je potrebné redukovať ku bodu q aj body polygónu $p_i=[x_i, y_i]$

$$x'_i = x_i - x_q,$$

$$y'_i = y_i - y_q.$$

Pokiaľ existuje priesečník M $[x'_m, y'_m = 0]$ medzi segmentom (hrana) uzavretej oblasti O a jednej z polpriamok (x'/y') na osy x', tak je tento priesečník možné zistiť ako

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

Pre existenciu tohto priesečníka *M* však platia podmienky v závislosti, či sa pretína s ľavostrannou (r_1) alebo pravostrannou (r_2) polpriamkou. Pre ľavú (t_l) a pravú (t_r) polovinu tak platia nasledujúce vzťahy, ktoré môžu nadobúdať hodnoty *True* alebo *False.*

$$t_l = y_i < y_q \neq y_{i-1} < y_q,$$

$$t_r = y_i > y_q \neq y_{i-1} > y_q.$$

Implementujeme len tie priesečníky, pri ktorých lúč nadobúda hodnoty *True* a zároveň hodnota x'_m pri ľavostrannom je menšia ako 0 a pri pravostrannom musí byť väčšia ako 0. To znamená, že tento priesečník bude započítaný len v takom prípade, kedy počiatočný a koncový bol pretnutého segmentu v inej pol rovine:

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

Vylepšenie:

Ako bolo už spomenuté vyššie, tak tento algoritmus má viacero nedostatkov, ktoré je potrebné dodatočne vyriešiť.

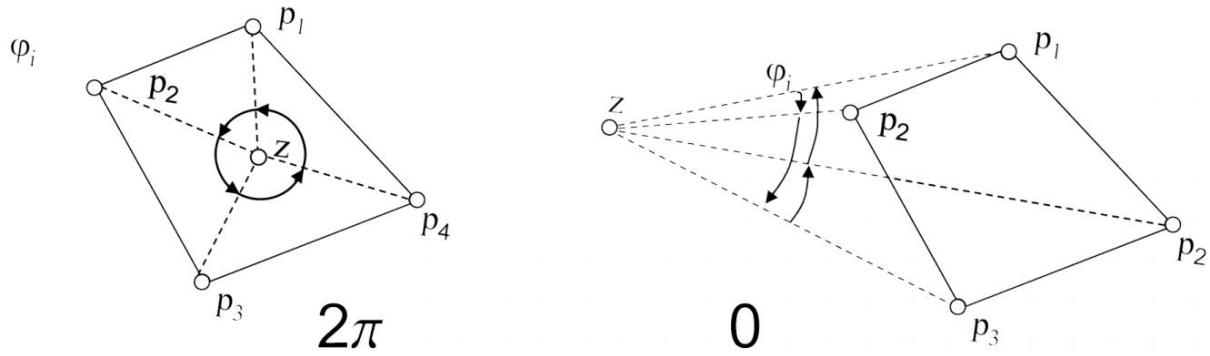
Pokiaľ bod *q* leží priamo na jednom z bodov polygónu, tak sa prehlasuje, že patrí segmentu uzavretej oblasti *O* ***(q ∈ ∂O).***

Pri prechode priamky *r*(*q*) cez jeden z vrcholov uzavretej oblasti *O*, tak nastáva už spomínané zaznamenanie dvojitého priesečníku, keďže vrchol polygónu pozostáva z jedného koncového a druhého počiatočného bodu segmentu. Problém sa vyrieši započítaním iba jedného priesečníku v tomto bode. Na základe nasledujúceho vzťahu $\boldsymbol{y_{i+1} - y_i = 0}$ tak bude platiť, že body p_{i+1} a p_i budú tvoriť horizontálnu hranu a znemožnia tak výpočet priesečníku podľa pôvodných vzorcov.

Pokiaľ sa bod p_i nachádza veľmi blízko priamky *r*(*q*), tak môže byť chybne priradený do spodnej alebo vrchnej pol roviny oblasti *O.* Tento problém sa vyrieši zavedením prahovej hodnoty, pre ktorú platí $\boldsymbol{\varepsilon \geq |y_i - y_q|}$.

Winding Number Algoritmu

Winding Number Algorithm, alebo aj metóda ovíjania, funguje na základe sčítania a odčítania uhlov (ω). Tieto uhly sa zisťujú medzi dvomi bodmi a pivotom. Pre zjednodušenie tejto metódy je možné si predstaviť pivot ako pozorovateľa, ktorý si chce postupne prehliadnuť všetky body. Pri tomto úkone je potrebné ukončiť otáčanie na počiatočnom bode. Pokiaľ sa otáča v protismere hodinových ručičiek, tak sú hodnoty uhlov kladné a v opačnom prípade je tento uhol záporný (Rourke, 1998). Pokiaľ je súčet uhlov v radiánoch rovný 2π, tak sa bod nachádza vo vnútri polygónu. Ak sa bod nachádza mimo polygónu, tak súčet uhlov je menší ako 2π (obr. 3).



Obr. 3: Winding Number algoritmus

Podstata:

Pokiaľ máme v rovine (R^2) bod *q* a uzavretú oblasť *O* (polygón), ktorej hrany sú tvorené množinou bodov P = $\{p_1, \ldots p_n, p\}$, tak pre súčet všetkých uhlov, ktoré opíše sprievodič v polygóne vzťah,

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

Pre výpočet celkového uhla je najprv potrebné analyzovať vzájomnú polohu bodu *q* ku priamke, ktorá je definovaná dvomi po sebe nasledujúcimi vrcholmi p_i a p_{i+1}. Analýza môže prebehnúť za pomoci *Half-plane testu*, prípadne za využitia vzťahu pre výpočet determinantu matice zloženej z vektorov:

$$\vec{p} = p_{i-1} - p_1,$$

$$\vec{s} = q - p_1.$$

Od hodnoty determinantu záleží, či sa daný bod leží priamo na priamke (*det = 0*), vľavo od priamky (*det > 0*), prípadne napravo od nej (*det < 0*). Na výpočet determinantu je použitý nasledujúci vzťah:

$$det = (p_x * s_y) - (s_x - p_y).$$

V závislosti na determinante sa budú pripočítavať (v protismere hodinových ručičiek) alebo odčítavať (v smere hodinových ručičiek) uhly. Je preto potrebné prechádzať všetky hrany v polygóne. Pokiaľ je hodnota determinantu kladná, tak uhol budeme pripočítavať a v prípade záporného determinantu sa bude uhol odčítavať. Nato, aby sme zistili uhly, tak potrebujeme vytvoriť vektory $\vec{u} = (u_x, u_y)$ *a* $\vec{v} = (v_x, v_y)$ medzi bodom *q* a vrcholmi polygónu p_i a p_{i+1},

$$\vec{u} = p_i - q,$$

$$\vec{v} = p_{i-1} - q.$$

Veľkosť uhla je vypočítaná pomocou nasledujúcich vzťahov, pričom zisťovaný uhol je potrebné počítať cez absolútnu hodnotu, keďže o tom či bude kladný alebo záporný rozhoduje hodnota determinantu,

$$cos(\omega) = \frac{\vec{u}.\vec{v}}{\|\vec{u}\| \,.\, \|\vec{v}\|},$$

$$|\omega| = arccos\left(\frac{\vec{u}.\vec{v}}{\|\vec{u}\| \,.\, \|\vec{v}\|}\right).$$

Po sčítaní všetkých uhlov tak môžu nastať dve situácie. Bod sa bude nachádzať v polygóne pokiaľ je celkový súčet uhlov hodnotu 2π a mimo polygónu sa bude nachádzať pokiaľ je táto hodnota menšia ako 2π.

Vylepšenie:

Vyššie popísaný algoritmus dokáže opísať vzájomný vzťah medzi polygónom a bodom, pokiaľ sa tento bod nachádza mimo polygónu, alebo priamo v ňom. Môže však nastať aj situácia, kedy sa bude bod nachádzať priamo na hrane, prípadne priamo v jednom z vrcholov. V prvom prípade je možné zistiť vzájomnú polohu za pomoci *det=0*, alebo uhla $\omega = \pi$. Pokiaľ je ale bod *q* zároveň bodom polygónu, tak ich vektorová norma sa rovná nule (výnimka pri výpočte).

Implementovanie

Po priblížení problematiky a vysvetlení metód na jej riešenie je ďalším krokom v tomto zadaní tvorba užívateľského prostredia, v ktorom bude možné demonštrovať funkčnosť metód na určenie vzťahu medzi bodom a polygónom.

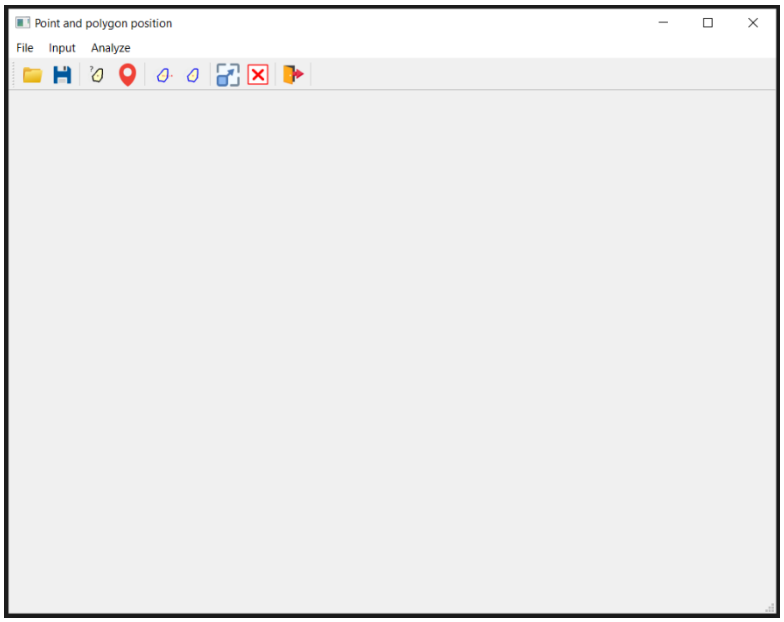
Vstupné dáta

Dáta vstupujúce do našej aplikácie boli z viacerých zdrojov. Okrem importovania polygónov v Křovákovom zobrazení (EPSG:5514) pomocou textového súboru, GeoJSON/JSON (obr. 4) bolo možné vytvoriť polygón aj za pomoci klikania myši na *widget*. V prípade bodu nebolo možné importovať jeho súradnice zo súboru, ale podobne ako pri polygóne, bolo ho možné interaktívne vyznačiť na *widget*, prípadne zadať presné súradnice bodu. Pokiaľ boli dáta získané interaktívne (klikaním), tak mali dátový typ *integer* a v prípade zadávania presných súradníc bodu, či nahrávania polygónov zo súboru sa jednalo o typ *float*.

Obr. 4: Ukážka vstupných dát v GeoJSON

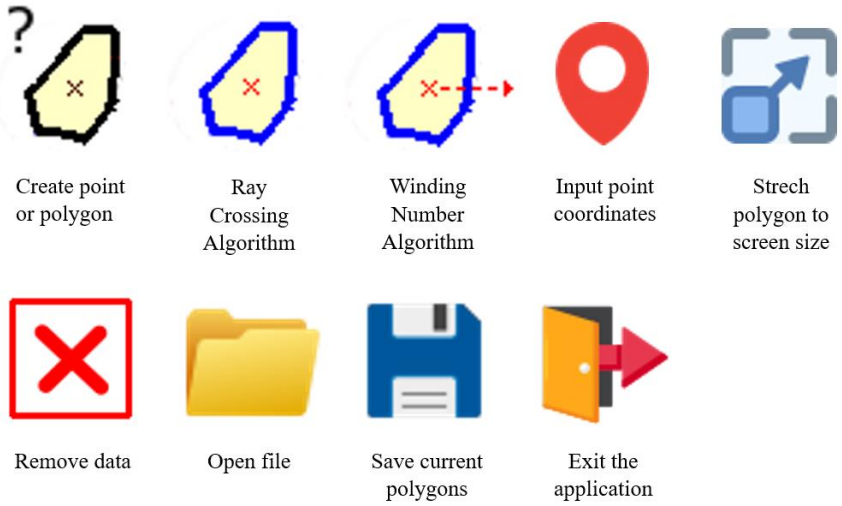
Aplikácia

Grafické užívateľské prostredie (GUI) aplikácie (obr. 5) pozostávalo z okna, lišty a panelu obsahujúceho ikony, ktoré bolo vytvorené za pomoci knižnice QT (v.4.6.1) a jeho funkcionalita bolo ďalej doplňovaná za pomoci programovacieho jazyka Python 3.11. Vďaka tomu bolo možné pre jednotlivé ikony pridať aj ich funkcie.



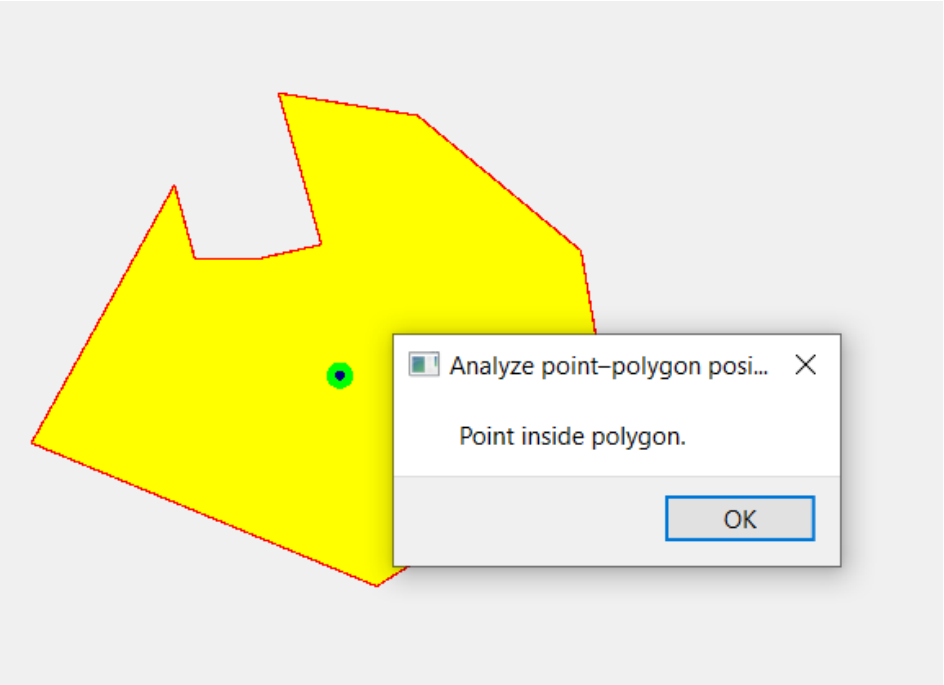
Obr. 5: Grafické prostredie aplikácie

Celkovo bolo použitých 9 ikon (obr. 6), ktorých úlohou je urýchliť prácu používateľa, vďaka čomu nie je potrebné prechádzať lištu a hľadať jednotlivé procesy, ktoré by sme pri väčšej komplexnosti aplikácie mohli prehliadnuť.



Obr. 6: Funkcionalita ikon

Aplikácia dáva používateľovi na výber z dvoch analytických metód, ktoré sa spustia hneď po kliknutí na príslušnú ikonu. Ich výsledkom je *pop-up* (obr. 7) okno s textom uvádzajúcim vzájomnú polohu medzi bodom a polygónom.



Obr. 7: Výsledok analýzy zvolenou metódou

Výstupné dáta

Polygón vykreslené v aplikácii je možné exportovať do formátu GeoJSON, pričom pixelové súradnice sa uložia ako priestorové. Pokiaľ boli dáta nahrané zo súboru, tak ich následným uložím získame okrem pôvodných priestorových súradníc aj novovytvorené pixelové súradnice.

Dokumentácia

Trieda UiMainWindow

Trieda *UiMainWindow* je zodpovedná za celé GUI, ktoré pozostáva z okna, widget a lišty, na ktorej sa nachádzajú ikonky pre zjednodušené ovládanie. Nachádzajú sa tu okrem toho prepojenia jednotlivých ikon s ich funkcionalitami v iných triedach. Funkcie *setupUi (), retranslateUI ( )* boli vytvorené automaticky QT.

pointPolygonClick ( ) *- zmena medzi polygónom a bodom pri kreslení do widget.*

clearClick ( ) *- odstránenie všetkých objektov z widget a obnovuje vychádzajúcu polohu bodu.*

exitClick ( ) *- ukončenie aplikácie.*

rayClick ( ) *- použitie metódy Ray Crossing na analýzu bodu (prepojenie na triedu Algorithms)*

windingClick ( ) *- využitie metódy Winding Number na analýzu bodu (prepojenie na triedu Algorithms).*

pntCoordClick ( ) *- možnosť vytvoriť bod zadaním presných súradníc v dátovom type float.*

rescaleClick ( ) *- prispôsobia veľkosti dáta ku rozlohe widget (prepojené na triedu Algorithms).*

saveClick ( ) *- uloženie aktuálnych polygónov z widget vo formáte GeoJSON.*

openClic ( ) - zadanie súboru pre načítanie dát

Trieda Algorithms

Metódy zastúpené v triede *Algorithms* vykonávajú geometrických operácií s polygónmi, testovanie, či bod leží vnútri polygónu, detekciu priesečníkov medzi líniami a polygónmi a transformáciu súradníc bodov. Taktiež umožňuje spracovanie polygónov, ako je zmena ich orientácie a zistenie ich minimálneho a maximálneho rozsahu. Celkovo táto trieda slúži na analýzu a manipuláciu s geometrickými dátami.

rayCrossingAll (q:QPointF, inpt_pols) *- kontroluje, či sa daný bod q nachádza na všetkých zadaných polygónoch inpt_pols pomocou metódy rayCrossingSingle.*

rayCrossingSingle (q: QPointF, pol:QPolygonF) *- testuje, či sa daný bod q nachádza na konkrétnom polygóne pol pomocou algoritmu Ray Crossing.*

windingNumAll (q:QPointF, inpt_pols) *- kontroluje, či sa daný bod q nachádza na všetkých zadaných polygónoch inpt_pols pomocou metódy windingNumSingle.*

windingNumSingle (q: QPointF, pol:QPolygonF) - *testuje, či sa daný bod q nachádza na konkrétnom polygóne pol pomocou algoritmu Winding Number.*

get2_LineAngle (p1: QPointF, p2: QPointF, p3: QPointF, p4: QPointF,) - *počíta uhol medzi dvoma úsečkami*

minMaxBox (polygon: QPolygonF) - *počíta minimálny a maximálny rozsah (bounding box) polygónu*

sjtsk2Pixel (point: list, width: int, height: int, x_min, x_max, y_min, y_max) - *prevádza body zo súradníc S-JTSK na pixely.*

rescale (polygons, w:int, h:int) - *prepočítava a upravuje polygóny na zvolenú veľkosť widget.*

updateMinMaxBox (polygons) - *metóda aktualizuje minimálny a maximálny rozsah polygónov.*

selectInsideBox (q, inpt_pols) - *vyberá polygóny, ktoré obsahujú daný bod.*

setCCW_Orientation (polygons) - *nastavuje orientáciu polygónov proti smeru hodinových ručičiek.*

Trieda Draw

Metódy v tejto triede sú zodpovedné za interakciu používateľa s grafickým užívateľským rozhraním a manipuláciu s vykresľovaním a analýzou objektov. Taktiež obsahuje funkcie typu *getter* na získanie položiek.

mousePressEvent (e: QMouseEvent) - *spracováva udalosti stlačenia myši na widget, čím sa pridáva/presúva bod, ktorý môže reprezentovať aj vertex polygónu.*

setPolygons (polygons) - *obnovuje polygóny na základe poskytnutého slovníka.*

paintEvent (e: QPaintEvent) - *vykresľuje polygón a bod na widget.*

switchDrawing ( ) - *prepína medzi pridaním vrcholu polygónu (vertex) a bodu.*

messageBox (title, text) - *zobrazuje dialógové okno s upozornením odpovedajúcemu textu v parametroch*

clearData ( ) - *vymazáva údaje o polygónoch a bode z widget.*

Trieda IO

Funkcie v tejto triede spracovávajú operácie vstupu a výstupu dát polygónov, vrátane načítania zo súborov JSON, spracovania dát a ich opätovného uloženia do súborov JSON.

readPnts (pnt_list: list) - *číta body zo zoznamu súradníc. Odstráni duplicitné body a vráti zoznam jedinečných bodov.*

loadPolygons (file: str, width: int, height: int) - *načíta dáta polygonov z JSON súboru, extrahuje súradnice, odstráni duplicitné body pomocou readPnts, transformuje súradnice na pixely a vráti slovník obsahujúci načítané polygóny.*

savePolygons (file: str, polygons) - *Táto funkcia uloží dáta polygonov do JSON súboru. pripraví dáta polygónov na uloženie vo vyžadovanom formáte a potom ich zapíše do špecifikovaného súboru.*

Trieda UiDialog

Táto trieda slúži na definovanie užívateľského rozhrania (UI) pre dialógové okno v aplikácii. V rámci tejto triedy sú definované rôzne prvky užívateľského rozhrania, ako sú označenia (*labels*) alebo tlačidlá (*buttons*).

setCoordsAccesp ( ) - *Táto metóda sa volá po stlačení tlačidla "Apply" (push_button). Získava hodnoty zadané v poliach na zadanie X a Y súradníc a ukladá ich do atribútov x a y dialógového okna. Potom volá metódu accept na zatvorenie dialógového okna.*

getCoords ( ) - *metóda vráti zadané X a Y súradnice z dialógového okna. Používa sa na získanie súradníc po zatvorení dialógového okna.*

Záver

Témou tohto zadania bol problém geometrického vyhľadávania bodov. Pre riešenie tohto problému boli použité dve metódy, konkrétne *Ray Crossing* a *Winding Number*. Funkčnosť týchto algoritmov bola otestovaná pri určovaní vzájomnej polohy bodu a polygónu. Okrem toho boli ošetrené singulárne prípady pri metóde *Winding Number* a riešenie problému za využitia *min-max* boxu. Celá analýza prebiehala v nami aplikácii, ktorú sme vytvorili za pomoci knižnice *QT* a programovacieho jazyka *Python.*

Naša aplikácia bola vylepšená o funkciu *rescale*, ktorá umožňovala čo maximálne pokrytie polygónov na *widget* (pokiaľ polygón zaberal iba malú časť *widget*, tak sa zväčšil na čo najväčšiu plochu). Funkcia *rescale* prenásobila pixelové súradnice a nenačítavala nové absolútne súradnice, vďaka čomu bol táto operácia kompatibilná ako s načítanými, tak aj vytvorenými polygónmi. Ďalšie možné vylepšenie aplikácie by spočívalo v nahrávaní bodu zo súboru, či vytvorení polygónu zadávaním presných súradníc bodu.

Kvôli možným odchýlkam spôsobených nepresnými operáciami s dátovým typom *float* bola použitá prahová hodnota, ktorá predstavovala štvrtú odmocninu najmenšieho kladného desatinného čísla reprezentovaného *sys.float_into-epsilon*.

Pri algoritme *Winding Number* nefungovala správne funkcia na duplikovanie prvého bodu a jeho následné pridanie za posledný. Počiatočný bod polygónu bol pridávaný na koniec aj napriek tomu, že sa tam už nachádzal. Kvôli tomu bola implementácia metódy *Winding Number* náročnejšia.

Použitá literatúra

- AJMERA, G. 2024. Exploring Algorithms to Determine Points Inside or Outside a Polygon. [online] dostupné na: https://medium.com/@girishajmera/exploring-algorithms-to-determine-points-inside-or-outside-a-polygon-038952946f87 [cit. 23-03-2024]

- BAYER, T. 2008. Algoritmy v digitální kartografii. Praha: Nakladatelství Karolinum, 2008. 252s. ISBN: 978-80-246-1499-1

- BAYER, T. Katedra aplikované geoinformatiky a kartografie. Přírodověděcká fakulta UK, Albertov 6, Praha. 2024. *Point Location Problem*. Prednáška z predmetu: Algoritmy počítačovej kartografie [cit. 13-03-2023]

- FIALOVA, J. 2020. Prechádzka po svete geometrie, Trnava: a Pedagogická fakulta Trnavskej univerzity, 2020. 121s. ISBN 978-80-568-0327-1

- GALETZKA, M. AND GLAUNER, P. 2017. A Simple and Correct Even-Odd Algorithm for the Point-in-Polygon Problem for Complex Polygons. In *Proceedings of the 12th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISIGRAPP 2017) - GRAPP*; ISBN 978-989-758-224-0; ISSN 2184-4321, SciTePress, pages 175-178.
DOI: 10.5220/0006040801750178

- KAČMEŘÍK, M. 2024. Algoritmizace prostorových úloh: Bod v Polygonu. [online] dostupné na: https://homel.vsb.cz/~kac072/apu/2_2.php?str=2_2 [cit. 23-03-2024]

- ROURKE, O. J. 1998. Computational Geometry in C. Cambridge: Cambridge University Press, 1998. 376s. ISBN: 9780521640107

- WANG, Y., & BENSON, D. J. 2016. Geometrically constrained isogeometric parameterized level-set based topology optimization via trimmed elements. Frontiers of Mechanical Engineering, 11(4), 328–343. doi:10.1007/s11465-016-0403-0

Ďalšie zdroje

Okresy ČR -. Map Service, Kuttelwascher, R. 2012. ArcČR 500

Ikona pin pre zvolenie bodu - https://uxwing.com/map-pin-icon/

Ikona resize - https://openclipart.org/detail/304895/resize-icon

Ikona diskety pre ukladanie - https://www.veryicon.com/icons/miscellaneous/admin-dashboard-flat-multicolor/save-32.html