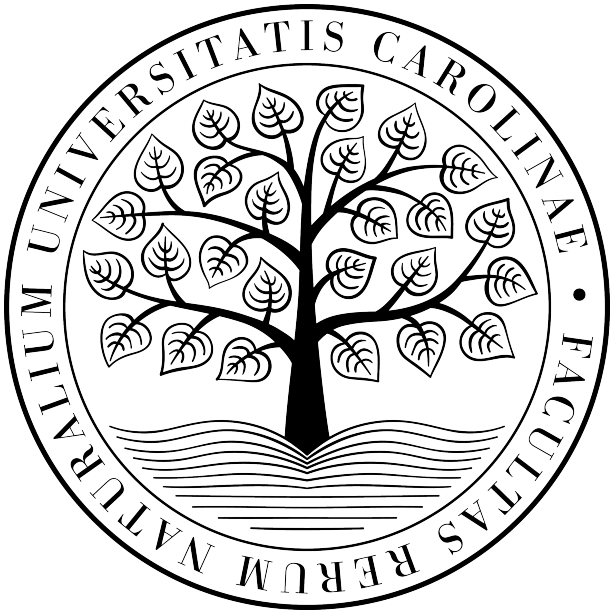
Přírodovědecká fakulta
Univerzita Karlova



Algoritmy počítačové kartografie

Úkol č. 1: Point Location Problem

Martínek David, Nováková Lucie a Zikešová Anna

1.N-GKDPZ

Praha 2024

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

V rámci tohoto programu byly řešeny následující bonusové úlohy:

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

Popis a rozbor problému

Point Location Problem

Point Location Problem neboli *point-inclusion problem* se řadí mezi nejzákladnější témata výpočetní geometrie. Využívá se při práci s mapami, databázemi a geoinformačními systémy. Hlavní otázkou je, zda bod leží uvnitř, vně, na hraně či na hranici polygonů (Zygmunt, Michalowska, Slusarski, 2023).

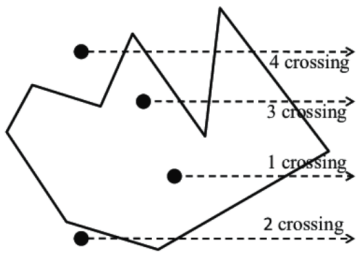
V geoinformatice je nezbytné, co nejrychlejší nalezení požadovaného polygonu na základě definování bodu. Je proto nutné nalézt efektivní řešení jak pro konvexní polygon - mnohoúhelník, jehož součet vnitřních úhlů je menší nebo roven 180 °, tak nekonvexní polygon - mnohoúhelník mající jeden vnitřní úhel větší než 180 °. Řešení problému lze nalézt dvěma technikami:

- Převedení problému na vztah bodu a mnohoúhelníku, kdy je problém převeden na úlohu opakovaného určení polohy vzhledem k mnohoúhelníku. Jedná se o jednoduchou, ale pomalou techniku řešení.
- Planární dělení roviny, kdy je rovina rozdělena na množinu pásů tzv. trapezoids, čímž vzniká *trapezoidální mapa*. Toto řešení má složitou implementaci, ale vyhledávání je rychlejší (Rourke, 2005).

Častější variantou polygonů jsou nekonvexní mnohoúhelníky. Pro řešení point location problému se staly populárními metodami: *ray crossing algoritmus* a *winding numbers algoritmus*. Oba algoritmy využívají k řešení časovou složitost O(n) (Poveda, Gould, Oliveira, 2004). Oba algoritmy jsou níže popsány a v rámci úlohy byly implementovány.

Ray Crossing Method

Ray Crossing algoritmus, který je označován také jako *Ray Algorithm*, *Ray Casting Algorithm* nebo *Even-Odd Rule Algorithm* (Bayer, 2008) je univerzálním algoritmem, který lze napasovat na jednoduché polygony s dírami i bez děr.



Obrázek 1: určení polohy bodu na základě Ray Crossing algoritmu (převzato z Yan 2012)

Hlavní myšlenkou je, že bodem q je vedena přímka r v libovolném směru, na jejímž základě se zjišťuje poloha bodu, díky počtu průsečíků k s hranami polygonu (obr. 1). Ve chvíli, kdy je počet průsečíků lichý, tak se bod nachází uvnitř polygonu. Když je počet průsečíků sudý nebo roven nule, tak bod leží mimo polygon. Když je počet průsečíků roven jedné, tak se bod nachází na hraně polygonu.

V praxi je nutné hodnotu k vydělit dvěma a zbytek po vydělení pak určuje, zda bod leží uvnitř či vně polygonu. Když je zbytek nula tak bod q leží vně polygonu, když je zbytek po dělení jedna, tak bod leží uvnitř polygonu.

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

Přímku r lze vést všemi směry a lze ji vést pod libovolným úhlem. Problém nastává v singulárních případech, tedy v případě že je přímka r totožná s hranou polygonu nebo když prochází vrcholem polygonu. Algoritmus také neumí rozlišit singulární případ, kdy leží bod q na hraně polygonu. Algoritmus lze modifikovat s redukcí q, a to tak že se vytvoří lokální souřadnicový systém s počátkem v bodě q, kdy x' náleží paprsku. Tato metoda však nerozpozná, zda bod se bod q nachází na hraně polygonu. Pro určení toho, že bod q leží na hraně polygonu je potřeba přidat druhou polopřímku r_2 s opačnou orientací. Tyto dvě polopřímky rozdělují rovinu na dvě poloroviny σ_1 a σ_2. Zároveň se udržuje počet levostranných a pravostranných průsečíků k_l a k_r.

V případě, že se počet průsečíků k_l a k_r s polopřímkami r_1 a r_2 nerovná, tak bod q leží na hraně polygonu. Ověření toho zda bod q leží na vrcholu polygonu probíhá porovnáním souřadnic bodu q s každým vrcholem vstupních dat. V případě, že jsou souřadnice vrcholu a bodu q totožné, nachází se bod q na vrcholu polygonu (Bayer, 2024).

Pseudokód Ray Crossing

Algorithm 1 *Ray Crossing*

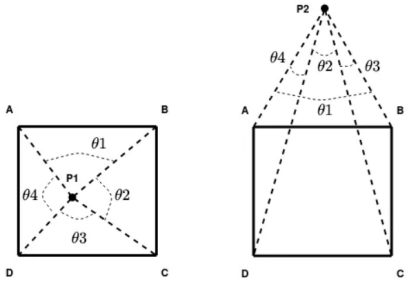
```
inicializuj k_r a k_l na 0 a počet vrcholů n
for všechny vrcholy:
    vypočti nové souřadnice x_ir, y_ir, x_i+1r, y_i+1r

    if x_ir = 0 a y_ir = 0
        bod q leží na vrcholu polygonu
    vypočti x souřadnice průsečíku x_m

    if (y_i < 0) ≠ (y_i+1 < 0) (levý paprsek)
        if x_m < 0:
            zvyš počet průsečíků k_l
    if (y_i > 0) ≠ (y_i+1 > 0) (pravý paprsek)
        if x_m > 0:
            zvyš počet průsečíků k_r

if (k_l%2) ≠ (k_r%2)
    q ∈ ∂P
if k_r%2 = 1
    q ∈ P
else p ∉ P
```

Singulární případy Ray Crossing

V Algoritmu *Ray Crossing* je potřeba ošetřit několik singulárních případů. V případě, že je bo q totožný s jesním z bodů p_i, tak se může prohlásit, že $q \in \delta P$. Pokud přímka prochází vrcholem může nastat detekce dvou průsečíků. Situaci lze vyřešit tím, že se tento vrchol započte vrchol jako průsečík pouze jednou. Bod leží na hraně za splnění podmínky, že se počet průsečíků v pravé a levé polorovině nerovná.

Winding Number Method

Dalším algoritmem pro detekci polohy bodu a polygonu je algoritmus *Winding number*. Využívá se v případě, že je polygonem nekonvexní mnohoúhelník. Algoritmus pracuje s velikostmi úhlů, sčítá je a odčítá je na základě úhlu svírajícího polohu bodu a jednotlivých úseků polygonů - přímka tvořena po sobě jdoucími body polygonu. V případě, že je součet úhlů roven 2π, tak se bod nachází uvnitř polygonu. V opačném případě bod leží vně polygonu (obr. 2) (Bayer, 2024). Jednoduše lze algoritmus vnímat tak, že v případě, že se pozorovatel dívá z bodu do všech rohů místnosti, tak se musí otočit celý dokola, tedy o úhel 2π.



Obrázek 2: určení polohy bodu na základě Winding Number algoritmu (převzato z Topiwala 2020)

Algoritmus představuje sumu Ω všech úhlů ω_1, které svírá pozorovaný bod q s vrcholy P daného mnohoúhelníku.

$$\Omega = \sum_{i=1}^{n} \omega_1$$

V případě že se součet úhlů rovná jedné neboli násobkům 2π, tak bod q leží uvnitř polygonu. Jedná se o počet oběhů. V opačném případě leží bod q vně polygonu.

$$\Omega(q, P) = \begin{cases} 1, & q \in P \\ 0, & q \notin P \end{cases},$$

Pro výpočet celkového úhlu, tedy polohy bodu q, je nutné nejprve zjistit polohu bodu q ke každé přímce p, která je určena dvěma po sobě jdoucími vrcholy p_i a $p_i + 1$, tvořící úsek polygonu.Prvně je potřeba vypočítat

směrový vektor přímky p a vektor v, který je určen bodem q a p_i

$$\vec{p} = (x_{i+1} - x_i, y_{i+1} - y_i)$$
$$\vec{v} = (x_q - x_i, y_q - y_i)$$

Tyto vektory lze přepsat do matice, kdy hodnota determinantu určuje, ve které polorovině bod q leží. Jednotlivé poloroviny od sebe odděluje přímka p.

$$d = \begin{vmatrix} q_x & q_y \\ v_x & v_y \end{vmatrix} = \begin{cases} > 0, & q \in \sigma_l \\ = 0, & q \in p \\ < 0, & q \in \sigma_r \end{cases}$$

Pokud je determinant vetší než nula, tak bod q leží v levé polorovině a úhel je orientován ve směru hodinových ručiček (CW). Když bod leží v pravé polorovině, tak je determinant naopak menší než nula a úhel je orientován v protisměru hodinových ručiček (CCW).

Výhodou *Winding Number Algoritmu* je jeho lepší ošetření v singulárních případech, oproti paprskovému algoritmu. Algoritmus je pomalejší než *Ray Crossing*. Problém nastává v případě, když je poloha bodu q stejná jako poloha jednoho poloho bodu polygonu (Bayer, 2024).

Pseudokód Winding Number

Algorithm 2 *Winding Number*

Inicializuj $\Omega = 0$, tolerance ϵ
Spočti počet vrcholů v polygonu
for opakuj pro všechny vrcholy:
    Spočti vektor u: p_i - q
    Spočti vektor v: p_{i+1} - q
    Spočti determinant vektorů u a v
    Spočti úhel mezi vektory u a v
    if determinant > 0:
        $\Omega + \omega$
    if determinant < 0:
        $\Omega + \omega$
    if determinant $= 0$ a zároveň leží q na linii mezi p_i a p_{i+1}
        q je leží na hraně
if $|\Omega - 2\pi| < \epsilon$
    q leží uvnitř polygonu
else
    q leží vně polygonu

Singulární případy Winding Number

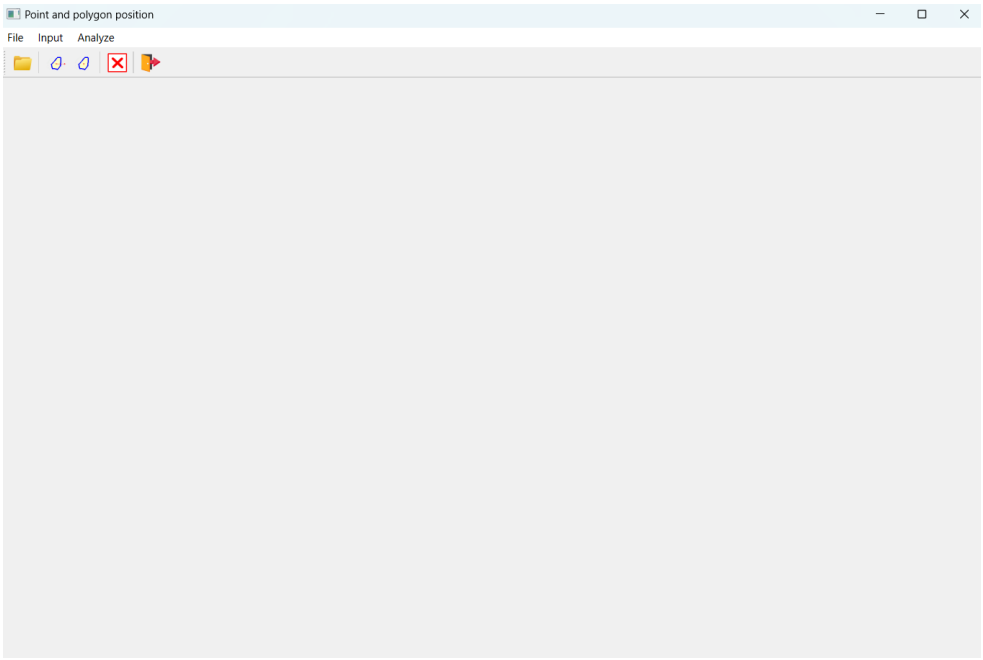
Algoritmus dokáže určit zda bod leží uvnitř či vně polygonu, ale neurčí zda bod q leží na hraně polygonu. Bod q leží na hraně, když je determinant roven nule. Dále je nutné, aby úhel ω byl roven π a ležel na linii mezi p_i a p_{i+1}. Za splnění těchto podmínek leží bod na hraně. V případě, že se $q = p_i$ bod leží na vrcholu polygonu, a tedy na hraně.

Aplikace

V rámci úlohy bylo mimo implementace samotných algoritmů analyzující polohu bodu v prostoru také vytvoření uživatelského rozhraní v aplikaci *Qt Designeru*. V aplikaci jsou pak demonstrovány oba výše zmíněné algoritmy *Winding Number* a *Ray Crossing* na vybrané polygonové mapě.

Jako vstupní data byla zvolena polygonová vrstva okresů České republiky ve formátu *shapefile*.

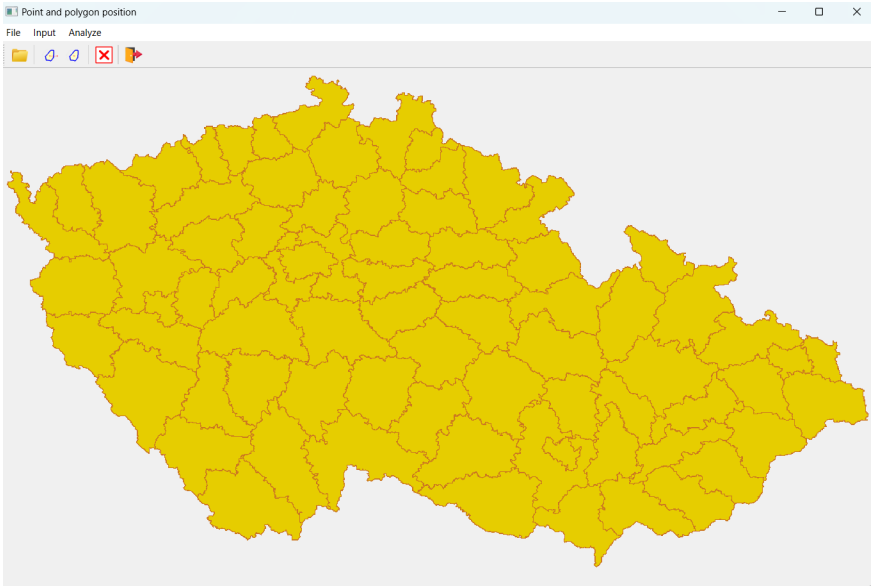
Aplikace je spuštěna přes soubor *mainform.py*. Je otevřeno okno aplikace, jejíž hlavní část tvoří prázdná plocha pro vykreslení polygonů (obr. 3). V horní části se nachází menu s jednotlivými funkcemi aplikace. V záložce *File* se nachází funkce *Open*, která načte vstupní data (obr. 4) a funkce *Exit*, která zavře celé okno. V záložce *Input* se nachází funkce *Clear*, která vymaže obsah okna, v záložce *Analyze* se nachází funkce *RayCrossing* a *WindingNumber*. Pod hlavním menu se nachází lišta s ikonami jednotlivých funkcí.



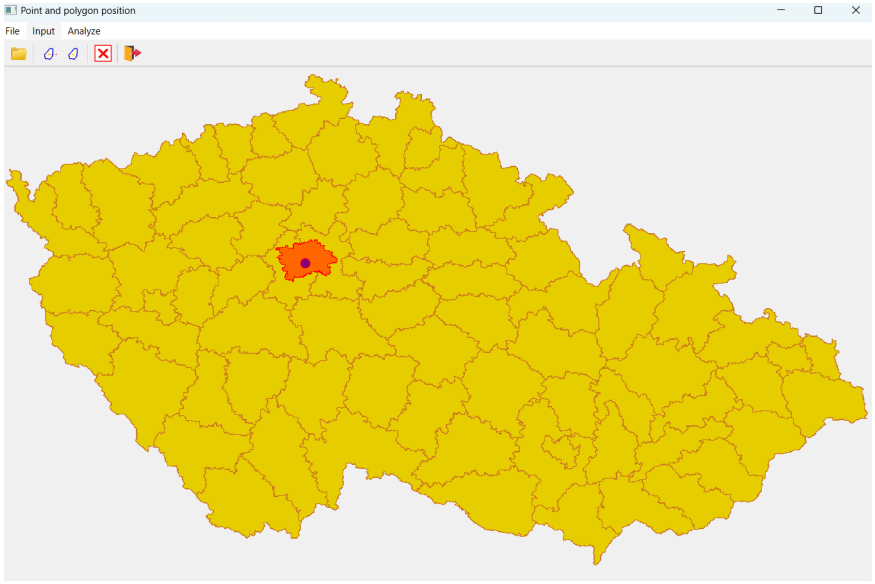
Obrázek 3: inicializace aplikace

Kliknutím do okna je vykreslen bod. Po zvolení algoritmu, který má být na analýzu polohy použit je polygon, ve kterém bod leží zvýrazněn (obr. 5). V případě umístění bodu na hranu či vrchol, jsou vykresleny všechny polygony, kterým daná hrana nebo vrchol náleží (obr. 6). Algoritmus je možné zvolit kliknutím na příslušnou

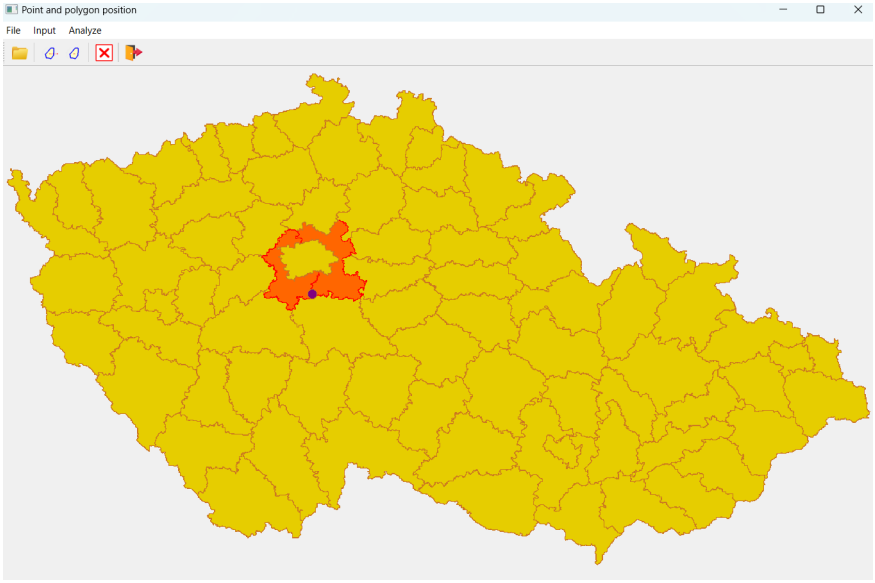
ikonu nebo vybrat z nabídky v záložce *Analyze*.



Obrázek 4: načtená data



Obrázek 5: bod leží uvnitř polygonu



Obrázek 6: bod je totožný s vrcholem více polygonů

Dokumentace programu

Program byl vytvořen v prostředí *Pycharm* v programovacím jazyce Python s využitím *Qtdesigner*. Program se skládá ze souborů *MainForm.py*, *draw.py*, a *algorithm.py*, které obsahují názvům odpovídající třídy. Ve složce *icons* je umístěno pět obrázků, které jsou využity pro ikony jednotlivých funkcí.

třída MainForm:

Třída MainForm ze souboru *MainForm.py* inicializuje okno aplikace, menu, lištu ikon a tlačítek. Současně přepíná jednotlivé funkce okna s metodami, které konají jednotlivé akce. Jde především o otevření souboru, přepínání algoritmů a provedení analýzy polohy bodu vůči polygonu atd. Část této třídy byla vygenerovaná v prostředí *Qtdesigner* a ta byla překonvertována do skriptu v pythonu. Tato třída obsahuje 6 metod.

Metody *setupUi* a *retranslateUi* byly vytvořeny automaticky na základě vytvořeného prostředí v *QtDesigner*. V rámci metody *setupUi* je definované uživatelské rozhraní napojeny jednotlivé položky menu a tlačítka na připravené metody. Níže jsou vyjmenovány implementované metody:

openClick() Zavolá metodu *getData*() ze třídy *Draw* a otevře složku se vstupními daty a po vybrání správného souboru ve formátu $shapefile$, data z $shapefile$ načte.

clearClick() Zavolá metodu *clearData*() z třídy *Draw* a dojde k vymazání okna.

rayCrossingClick() Zavolá metodu *getQ*() a *getPol*() z třídy *Draw* čímž získá aktuální polohu bodu a seznam vykreslených polygonů. Následně jsou procházeny všechny polygony a je na ně aplikována metoda *rayCrossing* z třídy *Algorithms*. Následně je vyvoláno okno, ve kterém je napsána poloha bodu q vůči polygonu či polygonům.

windingNumberClick() Zavolá metodu *getQ*() a *getPol*() z třídy *Draw* čímž získá aktuální polohu bodu a seznam vykreslených polygonů.Následně jsou procházeny všechny polygony a je na ně aplikována metoda *windingNumber* z třídy *Algorithms*. Následně je vyvoláno okno, ve kterém je napsána poloha bodu q vůči polygonu či polygonům.

třída Draw:

Třída *Draw* ze souboru *Draw.py* slouží k incializaci proměnných mající prostorovou informaci. Díky ní se načítají a vykreslují geoprostorové informace.

Inicializační metoda má dva argumenty pozice. Vykreslovaný bod q je typu $QPointF$ a počáteční souřadnice jsou nastaveny v záporných hodnotách, tak aby se bod vykresloval mimo okno. Dále dochází k inicializaci seznamu polygonů (*polygons*) a seznamu, do kterého se při každém spuštění analýzi ukládá výsledek pro jednotlivé polygony (pol_res).

mausePressEvent(e:QMauseEvent) Metoda reaguje na kliknutí myši. Na základě souřadnic kurzoru

jsou zjištěny souřadnice bodu q. A na základě tohoto zjištění dojde k překreslení okna, tak aby došlo k vykreslení nové pozice bodu.

paintEvent(e:QPaintEvent) Metoda slouží k vykreslení bodů a polygonů.

setResult() Metoda umožňuje vykreslení výsledků po analýze.

getData() Metoda načte soubor ve formátu $shapefile$ pomocí python modulu $shapefile$ a zobrazuje polygony.

setView() Metoda přepočítává souřadnice polygonů tak, aby se vešly do okna aplikace.

getQ() Metoda vrací souřadnice bodu.

getPolygons() Metoda vrací seznam vstupních polygonů.

clearData() Metoda slouží k vymazání vykreslených objektů v okně. Seznam s polygony je smazán a bod q je umístěn mimo okno, tak že jsou mu nastaveny záporné souřadnice. Nakonec je vyvoláno překreslení okna.

třída Algorithms:

Tato třída v sobě implementuje algoritmy pro analýzu polohy bodu a polygonu.

polygonFilter(q: QPointF, pol: QPolygonF) Metoda slouží k určení zda bod q leží uvnitř min-max boxu. V případě, že bod leží uvnitř min-max boxu tak metoda vrátí list polygonů, kterých se to týká. Metoda provádí rychlý počáteční filtr před použitím výpočetně náročnějších operací k určení přesného vztahu mezi bodem a polygonem.

rayCrossing(g:QPointF, pol:QPolygonF) Metoda má jako vstupní prvky analyzovaný bod q a polygon, k němuž analyzuje poloha bodu a implementuje metodu *RayCrossing* s redukcí souřadnic k bodu q a dvěma paprsky pro určení, zda bod leží na hraně. Nejprve jsou incializovány proměnné určující počet levých a pravých průsečíků a také počet vrcholů polygonu. Následně jsou procházeny všechny vrcholy polygonu a určeno zda bod leží na vrcholu polygonu. Určení toho, zda bod g leží na vrcholu proběhne pomocí redukování souřadnic k bodu q. Pokud jsou souřadnice bodu q rovny redukovaným souřadnicím prvního bodu (tyto souřadnice jsou nulové), tak bod leží na vrcholu a je vrácena hodnota 2. Následně dojde ke kontrole, zda jde o pravostranný či levostranný paprsek a k výpočtu průsečíků s hranami.V případě, že průsečík splňuje danou podmínku, tak je navýšena daná proměnná. Pokud je počet levostranných průsečíků lichý, tak bod q leží uvnitř polygonu a je vrácena hodnota 1. V situaci, že se počet levých a pravých průsečíků nerovná, bod leží na hraně a metoda vrátí hodnotu 2. V ostatních případech se vrací hodnota 0 a bod leží mimo polygon.

windingNumber(q:QPointD, pol:QPolygonF) Metoda má jako vstupní prvky analyzování bod q a polygon, k němuž analyzuje polohu bodu a implementuje metodu *Winding Number*. Nejprve jsou inici-

alizovány proměnné pro součet úhlů omega, tolerance e a počet vrcholů polygonu. Poté jsou procházeny vrcholy polygonu. Následně jsou vypočítány vektory a determinant. V případě, že je determinant roven nule, tak je zjišťováno, zda bod q leží na vrcholu, pokud jsou splněny podmínky, tak je vrácena hodnota 2. Následně je vypočítána velikost úhlu, v případě že byl determinant v předchozích krocích větší než nula, tak se úhel k sumě přičte (bod leží v levé polorovině), v opačném případě se úhel od sumy odečítá (leží v pravé polorovině). Následně je určeno zda bod q leží uvnitř či vně polygonu. Pokud je součet všech úhlů polygonu roven násobku 2π, tak bod leží uvnitř a je vrácena hodnota 1, pokud ne tak je vrácena 0.

Závěr

V této úloze byla vytvořena aplikace v softwaru *QtDesigner* za využití programovacího jazyka Python. V aplikaci je analyzována poloha bodu vůči daným polygonům, včetně singularit, na základě uživatelem zvoleného algoritmu *Ray Crossing* nebo *Winding Number*. Aplikace umožňuje načítat data obsahující prostorovou informaci pouze ve formátu $shapefile$. Vylepšení aplikace by se tedy mohlo týkat ošetření načítání dat i z jiných formátů než jenom $shapefile$.

Zdroje

přednášky z předmětu *Algoritmy počítačové kartografie*, dostupné z: https://web.natur.cuni.cz/ bayertom/ index.php/teaching/algoritmy-pocitacove-kartografie

POVEDA, J., GOULD, M., OLIVEIRA, A. (2004): A New Quick Point Location Algorithm. Dostupné z: https://www.inesc-id.pt/ficheiros/publicacoes/2211.pdf

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

TOPIWALA, A. (2020): Is the Point Inside the Polygon?. Towards Data Science. Dostupné z: https://towardsdatascience.com/is-the-point-inside-the-polygon-574b86472119

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