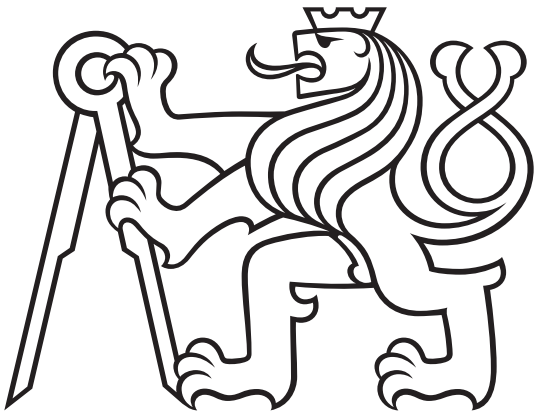
České vysoké učení technické v Praze
Fakulta stavební
program geodézie a kartografie
obor geomatika



geometrické vyhledávání bodu

Autoři: Bc. Linda Kladivová, Bc. Jana Špererová

Předmět: Algoritmy digitální kartografie a GIS

Úloha č. 1: Geometrické vyhledávání 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 následující algoritmy pro geometrické vyhledávání:

- Ray Crossing Algorithm (varianta s posunem těžiště polygonu).
- Winding Number Algorithm.

Nalezený polygon obsahující zadaný bod q 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:

Krok	**Hodnocení**
Detekce polohy bodu rozlišující stavy uvnitř, vně na hranici polygonu.	10b
Ošetření singulárního případu u Winding Number Algorithm: bod leží na hraně polygonu.	*+2b*
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.	*+2b*
Algoritmus pro automatické generování nekonvexních polygonů.	*+5b*
Max celkem:	**21b**

Čas zpracování: 2 týdny.

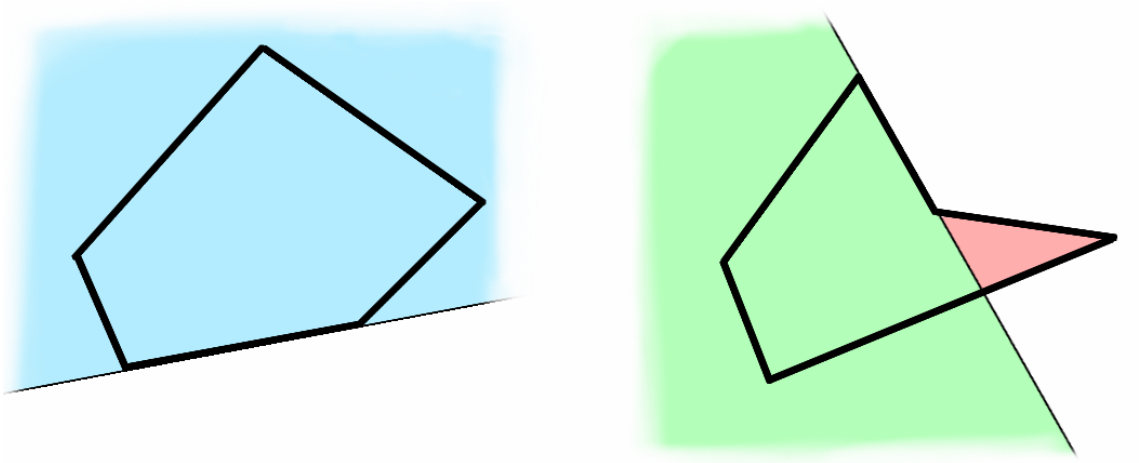
Obsah

Seznam obrázků

1 Popis a rozbor problému

Formulace problému je velmi přímočará. V rovině je dána množina bodů, které tvoří vrcholy mnohoúhelníků, a je dán bod q. Cílem je vyřešit tzv. Point Location Problem (PLP), tedy najít takový mnohoúhelník P, který bod q obsahuje. Tento problém je velmi významný, jelikož velice často potřebujeme znát svoji polohu vůči jiným objektům. Přístupů řešení tohoto problému je hned několik a závisí na tvaru polygonu, který může být ve speciálním případě konvexní, ale nejčastěji bývá nekonvexní.

Mnohoúhelník, který je konvexní, splňuje podmínku, že všechny jeho vnitřní úhly jsou konvexní, tedy musí být menší nebo rovny 180°. Dále platí, že všechny body na úsečkách, jejichž krajní body leží uvnitř polygonu, jsou také uvnitř polygonu. Další vlastností je, že všechny strany konvexního mnohoúhelníku leží v tzv. opěrné polorovině. U nekonvexního typu toto neplatí, což je zřejmé z obrázku 1.



Obrázek 1: Srovnání konvexního a konkávního mnohoúhelníku [2]

Výsledkem testu polohy bodu a mnohoúhelníku může být několik různých vzájemných poloh:

- bod q leží uvnitř testovaného polygonu P,
- bod q leží mimo testovaný polygon P,
- bod q leží na hraně testovaného polygonu P,
- bod q je identický s některým z vrcholů polygonu P.

Pro testování vzájemné polohy bodu vůči polygonu existuje několik metod. V této úloze je naprogramována metoda Ray Crossing Algorithm (varianta s posunem těžiště polygonu) a metoda Winding Number Algorithm.

1.1 Údaje o bonusových úlohách

Kromě implementace výše zmíněných algoritmů pro geometrické vyhledávání bodu nad polygonovou mapou, byl ošetřeno několik singulárních případů. U Winding Number Algorithm byl ošetřen případ, kdy bod leží na hraně polygonu. U obou algoritmů došlo k podchycení případu, kdy bod je totožný s vrcholem jednoho či více polygonů. Pro uvedené singularity bylo provedeno zvýraznění polygonů zelenou barvou. Zároveň byl naprogramován algoritmus pro automatické genererování nekonvexních polygonů.

2 Popis použitých algoritmů

Cílem této úlohy je vytvořit aplikaci, která po zadání příslušného bodu q určí polygon P, do kterého zadaný bod přísluší. Při volbě metody je vždy nutné se zamyslet nad typem vstupních a výstupních dat a nad časovou náročností.

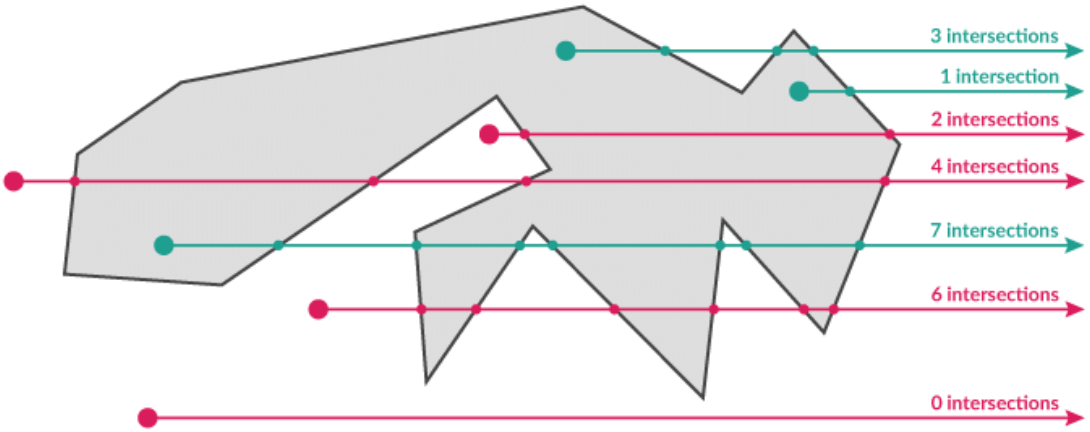
Řešení pro konvexní polygony je nejčastěji realizováno opakovaným testem polohy bodu q vůči každé hraně mnohoúhelníku P (half-plane test) nebo pomocí Ray Crossing Algorithm (paprskovitého algoritmu). Test polohy bodu q vůči hranám mnohoúhelníku P je princip, který ale nelze aplikovat na nekonvexní polygony.

K řešení této úlohy pro nekonvexní algoritmy jsou používány dva algoritmy: paprsový (Ray algorithm) a metoda oviíjení (Winding algorithm). Oba tyto přístupy převádí problém na opakované určení bodu vzhledem k mnohoúhelníku. Rozdílnou technikou řešení planární dělení roviny, což je jiná skupina algoritmů daleko mocnější, ale náročnější na implementaci. Do této skupiny patří například Metoda pásů, Line Sweep Algorithm či metoda trapezoidálních map.

2.1 Ray Crossing Algorithm

Ray Crossing Algorithm primárně slouží pro určení vztahu bod versus konvexní mnohoúhelník. Je však možné ho naprogramovat i pro nekonvexní polygony, což bude náš případ. Podstata algoritmu spočívá v tom, že ze zájmového bodu vedeme

polopřímku a počítáme její průsečíky s hranami polygonu. Tento algoritmus je o řád rychlejší než Winding algoritmus, problémem jsou však singularity. Princip paprskového algoritmu je zřejmý z obrázku 2.



Obrázek 2: Princip paprskového algoritmu [1]

Počet průsečíků paprsku r s hranou polygonu označme M. Platí, že:

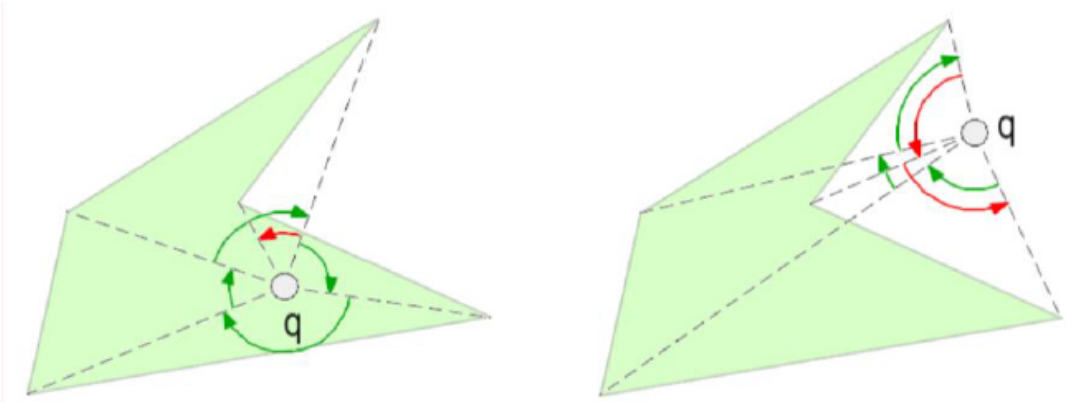
- pokud je M liché: bod náleží polygonu P ($q \in P$) ,
- pokud je M sudé: bod nenáleží polygonu P ($q \notin P$) .

2.1.1 Implementace algoritmu (varianta s redukcí)

1. Nastavení počtu průsečíků rovno nule: $intersects = 0$
2. Redukce souřadnic všech bodů polygonu vůči souřadnicím bodu q (lokální souřadnicový systém (q, x', y'))
3. Test podmínky: $if(y'_i > 0)\&\&(y'_{i-1} <= 0)\|(y'_{i-1} > 0)\&\&(y'_i <= 0)$
4. Při splnění podmínky provedení: $x'_m = (x'_i y'_{i-1} - x'_{i-1} y'_i)/(y'_i - y'_{i-1})$
5. Pokud $x'_m > 0$, zvýšení počtu průsečíků o jeden: $intersects = intersects + 1$
6. Určení, zda je počet průsečíků sudý či lichý: $if(intersect\%2) = 0$, pak: $q \in P$
7. V opačném případě: $q \notin P$

2.2 Winding Number Algorithm

Tento algoritmus je také nazýván Metoda ovíjení. Půjdeme postupně po vrcholech polygonu P a budeme sčítat nebo odečítat úhly mezi daným bodem q a vrcholy P (po směru hodinových ručiček přičítáme, proti směru odečítáme). Pokud je výsledný úhel roven 2π, byl dokončen celý kruh, lze tedy prohlásit, že bod q náleží polygonu P. Pokud je výsledný úhel roven 0, bod q polygonu P nenáleží. Princip Winding algoritmu je zřejmý z obrázku 3.



Obrázek 3: Princip Winding algoritmu [3]

Při této metodě je zapotřebí počítat Winding Number Ω. Pro toto číslo platí, že je rovno sumě všech rotací ω proti směru hodinových ručiček, které průvodič opíše nad všemi body:

$$\Omega = \frac{1}{2\pi} \sum_{i=1}^{n} \omega_i$$

.

Platí, že pokud je úhel $\sphericalangle p_i, q, p_{i+1}$ orientován ve směru hodinových ručiček, pak $\omega_i > 0$. Naopak pokud je úhel $\sphericalangle p_i, q, p_{i+1}$ orientován proti směru hodinových ručiček, pak $\omega_i < 0$. V závislosti na výsledné hodnotě Ω lze rozhodnout o poloze bodu q vůči polygonu P:

- pokud je $\Omega = 1$, platí $q \in P$,
- pokud je $\Omega = 0$, pak platí $q \notin P$.

2.2.1 Implementace metody

1. Nastavení výchozího úhlu $\omega = 0$, volba tolerance $\epsilon : \omega = 0, \epsilon = 1e - 6$
2. Určení úhlu: $\omega_i = \sphericalangle p_i, q, p_{i+1}$
3. Určení orientace o_i bodu q ke straně p_i, p_{i+1}
4. Volba podmínky - pokud bod vlevo: $\omega = \omega + \omega_i$, v opačném případě: $\omega = \omega - \omega_i$
5. Volba podmínky - pokud rozdíl: $|(|\omega| - 2\pi)| < \epsilon$, pak platí: $q \in P$, v opačném případě: $q \notin P$

2.3 Generování nekonvexního polygonu

Uživatel si nejprve zadá počet vrcholů polygonu, jež bude poté automaticky vygenerován. Pokud zvolí počet vrcholů 1,2 nebo 3 je zobrazena chybová hláška. Počet vrcholů polygonu musí být nejméně 4, aby mohl být výsledný obrazec nekonvexní. Souřadnice náhodných bodů jsou redukovány pomocí funkce modulo, aby bylo zajištěno zobrazení bodů ve vykreslovacím widgetu.

Uvnitř kódu je náhodně generováno (n - 1) bodů, jelikož jako n-tý bod je vygenerován střed všech vygenerovaných bodů. Od tohoto cetrálního bodu jsou poté vypočteny směrníky na všechny další body, díky čemuž mohly být body v polygonu seřazeny ve směru hodinových ručiček. Díky výpočtu směrníku je výsledný polygon topologicky korektní, ale je nutné brát ohled na kvadranty arcus tangens.

2.3.1 Implementace metody

1. Náhodné vygenerování (n-1) bodů: $x_{pi} = rand\%1000; y_{pi} = rand\%600$.
2. Vygenerování centrálního bodu jakožto průměru z již vygenerovaných souřadnic bodů.
3. Výpočet směrníků z centrálního bodu na jednotlivé body v závislosti na osách daného zobrazovacího okna.
4. Určení kvadrantu a správný přepočet úhlu.
5. Seřazení bodů v polygonu ve směru hodinových ručiček, tedy od nejmenšího směrníku.

3 Problematické situace a jejich rozbor

3.1 Ray Crossing Algorithm

Při programování tohoto algoritmu bylo řešeno několik problematických situací, tzv. singularit. K singularitě zde může dojít tehdy, pokud bod leží na hraně polygonu či pokud je bod identický s některým z vrcholů polygonu. Z tohoto důvodu se využívá upravená varianta Ray Crossing Algorithm, kdy je nejprve provedena redukce souřadnic bodů polygonu k zájmovému bodu a dále se testuje souřadnice y vybraného bodu polygonu a bodu předcházejícího. Počítají se pouze průsečíky M ležící v 1. kvadrantu upraveného souřadnicového systému, tedy souřadnice x bodu M musí být kladná. Problematické singulární situace můžeme vyjádřit následovně:

- Bod q je totožný s vrcholem polygonu P: $q[x_q, y_q] = p_i[x_i, y_i]$
- Bod q leží na některé z hran AB polygonu P: $q \in \overrightarrow{AB}$

První případ byl ošetřen porovnáním souřadnic bodu q a souřadnic bodu polygonu (tolerance totožnosti byla zvolena 1e-6). Druhý případ byl ošetřen pomocí funkce getPointLinePosition. Pokud bod náležel hraně polygonu či se nalézal ve vrcholu polygonu, byla návratová hodnota rovna -1. V takovém případě byl bod q vyhodnocen, že se v polygonu P nachází.

3.2 Winding Number Algorithm

K singulárním případům dochází pouze v případě, že $q \approx p_i$, tedy v případě že zájmový bod je téměř identický s některým z vrcholů polygonu. Pokud tomu nastane, vyhodnotí se funkce tak, že bod leží v obou polygonech. Problematické singulární situaci můžeme vyjádřit následovně:

- Bod q je totožný s vrcholem polygonu P: $q[x_q, y_q] \approx p_i[x_i, y_i]$

Tento případ byl ošetřen porovnáním souřadnic bodu q a souřadnic bodu polygonu (tolerance totožnosti byla zvolena 1e-6).

4 Vstupní data

Vstupní data tvoří analyzovaný bod q a textový soubor s body jednotlivých polygonů. Vstupem může také být automaticky vygenerovaný nekonvexní polygon.

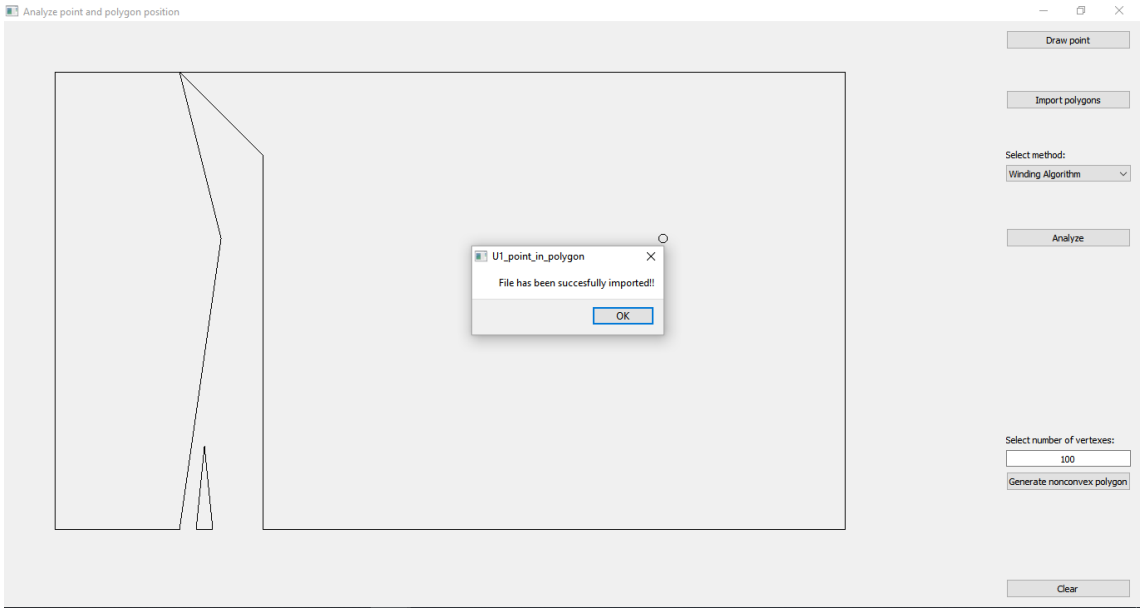
- Bod q je vložen uživatelem po ztisknutí tlačítka v grafickém okně aplikace.
- Souvislá mapa polygonů je ve formátu txt a obsahuje body jednotlivých polygonů. Má strukturu [číslo bodu, souřadnice X, souřadnice Y]. Soubor je seřazen od bodu č. 1 po n-tý bod polygonu. Jednotlivé body se nejprve uloží do proměnné typu QPointF a následně do proměnné polygonu QPolygonF. Jednotlivé polygony se poté uloží do proměnné typu ¡std::vector¿QPolygonF.

5 Výstupní data

Hlavním výstupem této úlohy je grafická aplikace, která umožňuje nahrání textového souboru s body polygonů a následné zobrazení těchto bodů a mnohoúhelníků. Zároveň je možné v aplikaci určit polohu bodu v rámci polygonu pomocí dvou různých metod - Winding algorithm a Ray Crossing algorithm. Dálším výstupem je zelené vyznačení toho polygonu, kterému bod q náleží. V rámci bonusové úlohy lze mezi výstupy zařadit i automatické vygenerování polygonu, který by měl splňovat podmínky nekonvexního mnohoúhelníku.

6 Ukázka aplikace

Do této kapitolky je zahrnuto několik ukázek vytvořené aplikace.



Obrázek 4: Úspěšné načtení souboru se souřadnicemi polygonů

Obrázek 5: Analýza Ray Crossing pro bod uvnitř polygonu

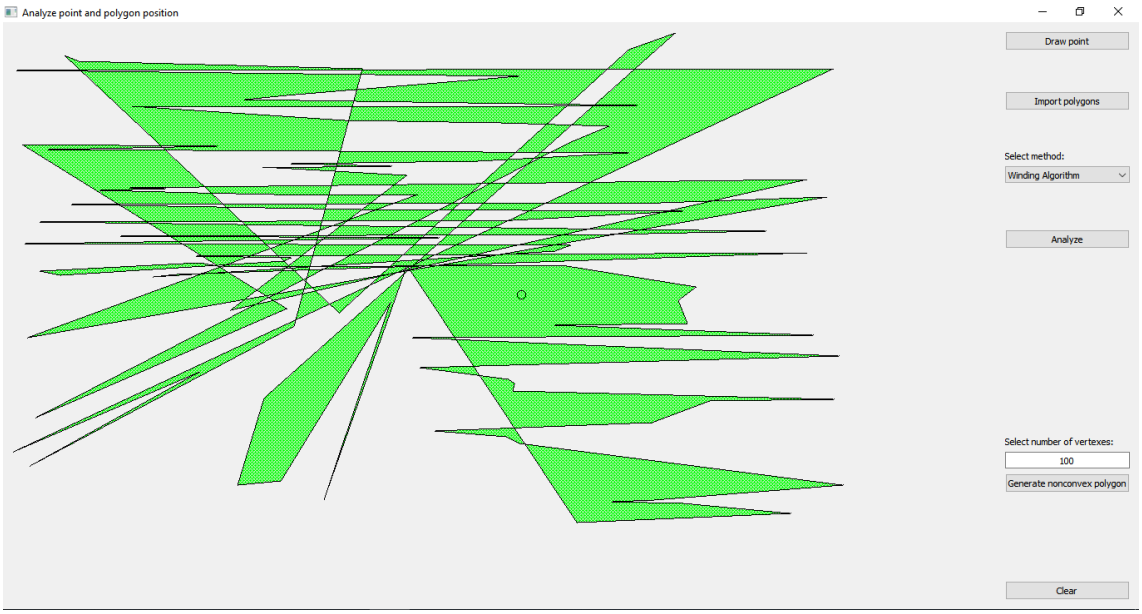
Obrázek 6: Analýza Ray Crossing pro bod na hraně polygonu

Obrázek 7: Analýza Ray Crossing pro bod ve vrcholu polygonu

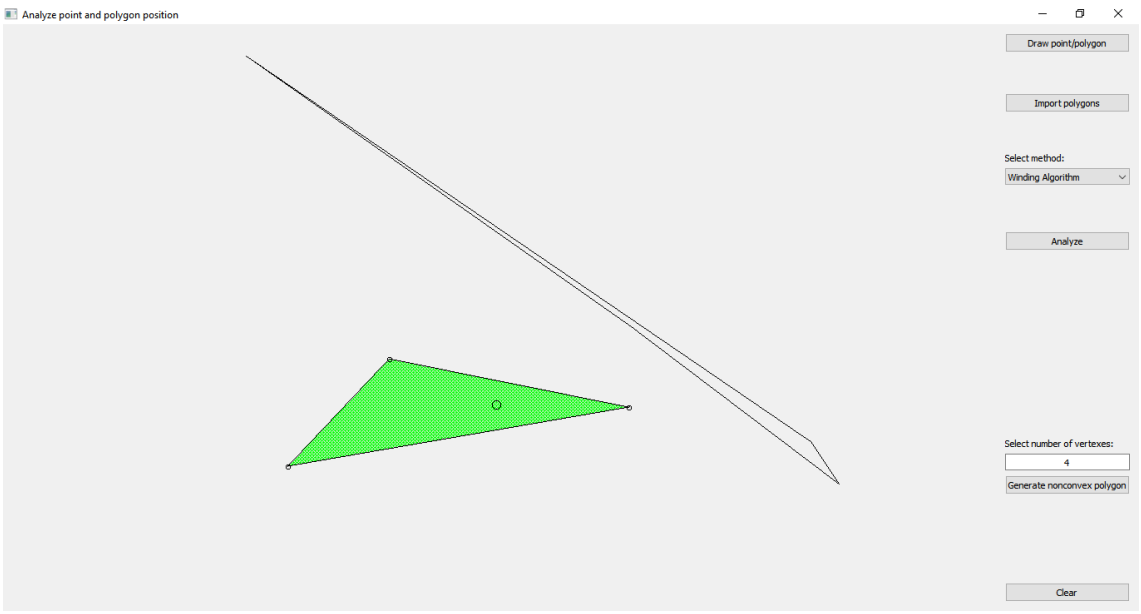
Obrázek 8: Analýza Winding pro bod uvnitř polygonu

Obrázek 9: Analýza Winding pro bod ve vrcholu polygonu

Obrázek 10: Analýza Winding nad vygenerovaným polygonem



Obrázek 11: Automaticky generovaný nekonvexní polygon se 100 vrcholy



Obrázek 12: Analýza manuálně a automaticky nakreslených polygonů

7 Technická dokumentace

7.1 Třídy

V aplikaci se nachází celkem tři třídy Algorithms, Draw a Widget. Každá z těchto tříd má svůj hlavičkový soubor, kde je deklarována spolu se svými metodami, a svůj zdrojový soubor, ve kterém jsou její metody konkrétně definovány.

7.1.1 Algorithms

Třída Algorithms obsahuje konstruktor a další čtyři metody, které jsou určeny pro výpočet algoritmů používaných v digitálním GIS. Datové typy u bodu a polygonu byly zvoleny s plovoucí desetinnou čárkou (floating point).

int positionPointPolygonRayCrossing(QPointF q, QPolygonF pol)
Jedná se o funkci, která vypočítá polohu bodu vůči polygonu využitím Ray Crossing Algorithm (paprskového algoritmu). Do funkce vstupuje určovaný bod q a polygon P, vůči kterému je poloha zjišťována. V případě, že výsledná hodnota funkce je rovna *1*, znamená to, že bod leží v polygonu, případně na hraně či vrcholu polygonu – singulární případy jsou ošetřeny pomocí funkce getPointLinePosition. V ostatních

případech leží bod mimo polygon.

int positionPointPolygonWinding(QPointF q, QPolygonF pol)
Tato funkce určí polohu bodu vůči polygonu metodou Winding Number Algorithm. Vstupem je určovaný bod *q* a polygon *P*, vůči kterému je poloha určována. Hned na počátku je také zvolena tolerance (minimální hodnota) *eps = 1e-6*. V případě, že výstupem je číslo *0*, bod leží mimo polygon, v případě že *1*, bod leží uvnitř polygonu. Pokud nenastane žádná z těchto uvedených možností, výstupem je hodnota *-1*. Ve funkci je mimo jiné ošetřena singularita, která může nastat v případě, že bod leží na hraně či ve vrcholu. Pokud je výstupem číslo rovno *1*, znamená to, že bod *q* buď leží v některém z vrcholů polygonu *P*, nebo leží na hraně polygonu, což je kontrolováno funkcí getPointLinePosition.

int getPointLinePosition(QPointF q, QPointF p1, QPointF p2)
Tato funkce má za úkol určit polohu bodu *q* vůči přímce zadané dvěma body *p1* a *p2*. Z vypočtených vektorů určíme determinant, jehož velikost je stěžejší pro správné určení polohy bodu *q*. Aby bylo možné s menším rozptylem definovat situaci, kdy bod leží přímo na přímce, byla inicializována tolerance (minimální hodnota) *eps = 1e-6*. Pokud je determinant větší než tato tolerance, bod se nechází v levé polorovině a funkce vrací hodnotu *1*. Pokud je menší, bod se nachází v pravé polorovině a funkce vrací hodnotu *0*. Pokud nenastane ani jeden z výše uvedených případů, znamená to, že bod leží blízko hrany a výstupem je hodnota *-1*.

double getAngle2Vectors(QPointF p1,QPointF p2,QPointF p3, QPointF p4)
V této funkci je pomocí norem a skalárního součinu počítán úhel mezi dvěma hranami zadanými čtyřmi body typu QPoint. Úhel je vypočten jako arcus cosinus poměru skalárního součinu a součinu obou velikostí. Defaultně se v prostředí počítá v radiánech, což bylo ponecháno.

7.1.2 Draw

Třída Draw dědí od třídy QWidget. Je v ní obsaženo několik metod a také kontruktor, který nastavuje počáteční kursor mimo kreslící okno.

void setDrawMode
Po spuštění aplikace je nutné stisknout tlačítko Draw Point pro vykreslení bodu. V případě, že se stiskne tlačítko znovu, je přivolána funkce k tvorbě polygonu. Další možností je importovat polygony ze souboru nebo automaticky generovat náhodné polygonu o předem vybraném počtu vrcholů.

void mousePressEvent
V této funkci je v závislosti na hodnotě textitdraw_mode vykreslen buď nový bod q, nebo nový bod polygonu P.

void paintEvent
Tato metoda slouží k vykreslení bodu q. Zároveň jsou díky této metodě vykresleny naimportované polygony nebo nakreslený polygon. V metodě je také naprogramováno zvýraznění polygonu, pokud bod q leží uvnitř, případně na hraně či ve vrcholu polygonu P.

void setResult
Metoda, která výsledek analýzy vytvořený po kliknutí na tlačítko Analyze ve třídě Widget, udělá viditelný i pro třídu Draw.
void clearCanvas
Metoda slouží k vymazání všech polygonů i bodu q a k překreslení. Volá se před automatickým generováním polygonu.

bool importPolygons(std::string &path)
Tato funkce slouží k importu textového souboru, ve kterém se nachází body polygonů. Struktura textového souboru je blíže popsána v kapitole Vstupní data. Při importu souřadnic se uživateli zobrazí dialogové okno, které informuje o úspěšném načtení, případně o chybě.

void generatePolygon
Pomocí této metody může uživatel vytvořit nekonvexní mnohoúhelník, který pak dále může analyzovat. V případě, že není dodržen limit vstupních hodnot, zobrazí se Message Window, které upozorňuje na chybné vložení dat.

QPointF getPoint
Metoda, která vrací privátní člen q třídy Draw.

QPointF getPolygon
Metoda, která vrací privátní člen P třídy Draw.

QPointF getPolygons
Metoda, která vrací privátní vektor polygonů P třídy Draw.

7.1.3 Widget

void on_clear_clicked
Díky této funkci se vyčistí okno aplikace. V aplikaci je funkce implementována tlačítkem Clear.

void on_drawMode_clicked
Touto funkcí se volá metoda setDrawMode, která je implementována ve třídě Draw. V aplikace se funkce nalézá pod tlačítkem Draw Points.

void on_analyze_clicked
Tato metoda zavolá vybraný algoritmus a zobrazí, zda se bod nachází v polygonu, případně na jeho hraně či v jeho vrcholu.

void on_importPolygons_clicked
Funkce se zavolá po kliknutí na tlačítko Import polygons. V těle funkce je definován způsob načtení dat.

void on_generatePolygon_clicked
Při stisknutí tlačítka Generate nonconvex polygon je touto funkcí přivolána metoda generatePolygon, která podle zadaného počtu vrcholů vykreslí nekonvexní polygon.

7.2 Popis bonusových úloh

Ošetření singulárního případu u Winding Number Algorithm, kdy bod leží na hraně polygonu

Ve funkci positionPointPolygonWinding je určována pozice bodu vůči linii funkcí getPointLinePosition. V této funkci je poté vypočtena vzdálenost určovaného bodu q od obou bodů linie. V případě, že rozdíl vzdálenosti p1p2 a součtu vzdálenosti Qp1 a Qp2 je téměř nulový, můžeme říci, že bod leží na přímce. Pro tento případ byla tolerance rozdílu obou vzdáleností volena vyšší než ostatní tolerance (e = 2).

Ošetření singulárního případu u obou metod, kdy je bod totožný s vrcholem dvou a více polygonů

Ve funkci positionPointPolygonWinding je na začátku kódu ošetřeno, zda vyhledávaný bod q není totožný s některým z vrcholů polygonu. Pokud je rozdíl menší než zvolená mezní hodnota, je funkce vyhodnocena tak, že bod leží uvnitř testovaného polygonu. Podobně je tento případ řešen u Ray Crossing algoritmu.

Zvýraznění všech polygonů pro oba výše uvedené singulární případy

Pokud uživatel zadá více polygonů, jsou analyzovány pomocí for cyklu. Ty polygony, u kterých je splněna podmínka, že leží uvnitř polygonu, jsou zvýrazněny zelenou barvou. Pokud tedy bod leží přesně na hraně dvou polygonů, zvýrazněny jsou oba polygony.

Tvorba nekonvexního polygonu

Pro tvorbu polygonu je nutné zvolit počet vrcholů. Defaultně je hodnota nastavena na číslo 4. Ve funkci pro generování polygonů je na začátku uvedena podmínka, že vkládané číslo nesmí být menší než 4. Pokud uživatel zadá číslo měnší než 4 nebo text, objeví se chybová hláška upozorňující na neplatný vstup.

8 Závěr

Výsledná aplikace má několik funkcionalit. Umí pomocí tlačítka Draw point/polygon nakreslit polygon a bod a jejich vzájemnou polohu analyzovat. Stejně tak dokáže stiknutím tlačítka Import polygons importovat soubor se souřadnicemi polygonů a tyto polygony pak vůči zadanému bodu analyzovat. Další variantou získání polygonu je automatické vygenerování polygonu, který může být dále také analyzován vůči vybranému bodu. Byla snaha o to splnit povinné i bonusové úlohy, nicméně aplikace má i tak mnoho nedokonalostí, které je třeba zmínit.

8.1 Náměty na vylepšení

- Při importu souřadnic polygonů ze souboru není nijak ošetřena chyba, která nastane pokud struktura vstupního textového souboru není správná. Námětem na vylepšení by bylo kontrolovat, jak vybraný soubor vypadá.
- Při manuálním generování polygonu se při přidávání bodů nesmaže odebíraná hrana polygonu.

Literatura

[1] Example of ray casting of a polygon. [online], Naposledy navštíveno 6. 10. 2019, dostupné z: https://www.researchgate.net/figure/Example-of-ray-casting-of-a-polygon-Even-crossings-for-a-point-outside-and-odd-crossings_fig12_317957001.

[2] Konvexní mnohoúhelník. [online], Naposledy navštíveno 6. 10. 2019, dostupné z:https://cs.wikipedia.org/wiki/Konvexní_mnohoúhelník.

[3] Bayer, T.: Geometrické vyhledávání bodů. [online], Naposledy navštíveno 6. 10. 2019, dostupné z: https://web.natur.cuni.cz/ bayertom/images/courses/Adk/adk3.pdf.