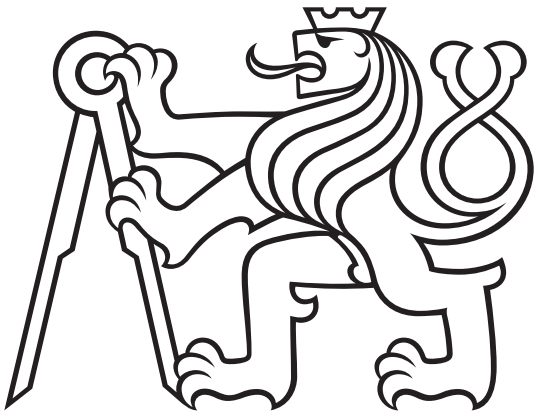
České vysoké učení technické v Praze
Fakulta stavební
program geodézie a kartografie
obor geomatika



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

1 Popis a rozbor problému **5**
1.1 Údaje o bonusových úlohách 6

2 Popis použitých algoritmů **6**
2.1 Ray Crossing Algorithm 6
2.1.1 Slovní zápis algoritmu (varianta s redukcí) 7
2.2 Winding Number Algorithm 8
2.2.1 Slovní zápis algoritmu 9
2.3 Generování nekonvexního polygonu 9
2.3.1 Slovní zápis algoritmu 9

3 Problematické situace a jejich rozbor **11**
3.1 Ray Crossing Algorithm 11
3.2 Winding Number Algorithm 11

4 Vstupní data **12**

5 Výstupní data **12**

6 Ukázka aplikace **13**

7 Technická dokumentace **17**
7.1 Třídy 17
7.1.1 Algorithms 17
7.1.2 Draw 19
7.1.3 Widget 20
7.1.4 sortByX 21
7.1.5 sortByY 21
7.1.6 sortByAngle 21
7.1.7 Angle 21
7.2 Popis bonusových úloh 22

8 Závěr **23**
8.1 Náměty na vylepšení 23

Literatura **24**

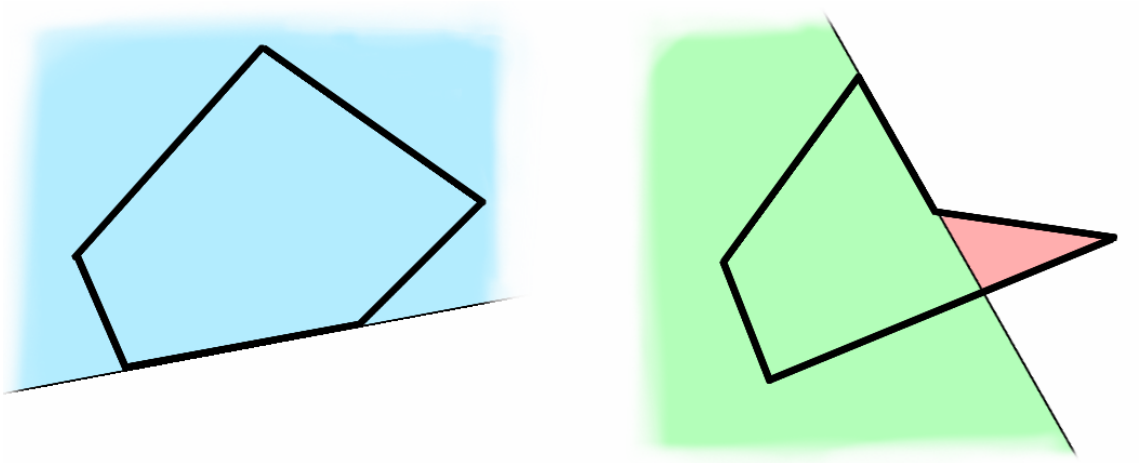
Seznam obrázků

1 Srovnání konvexního a nekonvexního (konkávního) mnohoúhelníku 5
2 Princip paprskového algoritmu 7
3 Princip Winding algoritmu 8
4 Úspěšné načtení souboru se souřadnicemi polygonů 13
5 Analýza Ray Crossing pro bod uvnitř polygonu 13
6 Analýza Ray Crossing pro bod na hraně polygonu 14
7 Analýza Ray Crossing pro bod ve vrcholu polygonu 14
8 Analýza Winding pro bod uvnitř polygonu 15
9 Analýza Winding pro bod ve vrcholu polygonu 15
10 Analýza Winding nad vygenerovaným polygonem 16
11 Automaticky generovaný nekonvexní polygon s 50 vrcholy 16
12 Analýza manuálně a automaticky nakreslených polygonů 17

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 [**?**]

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 (paprskový algoritmus) a metoda Winding Number Algorithm (metoda oviíjení).

1.1 Údaje o bonusových úlohách

Kromě implementace výše zmíněných algoritmů pro geometrické vyhledávání bodu nad polygonovou mapou, která je importována z textového souboru, bylo 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 obou (nebo více) společných polygonů zelenou barvou. Zároveň byl naprogramován algoritmus pro automatické genererování nekonvexních polygonů při zadaném počtu vrcholů.

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.

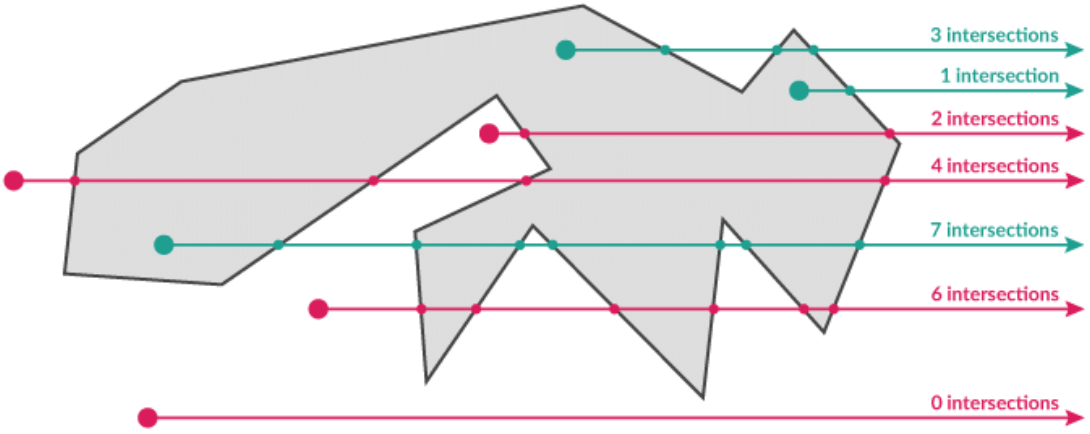
Řešení pro konvexní polygony je nejčastěji realizováno opakovaným testem polohy bodu q vůči každé hraně mnohoúhelníku P (half-plane test) nebo pomocí Ray Crossing Algorithm (paprskovitého algoritmu). Test polohy bodu q vůči hranám mnohoúhelníku P je princip, který ale nelze aplikovat na nekonvexní polygony.

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 je algoritmus planárního 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 [**?**]

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

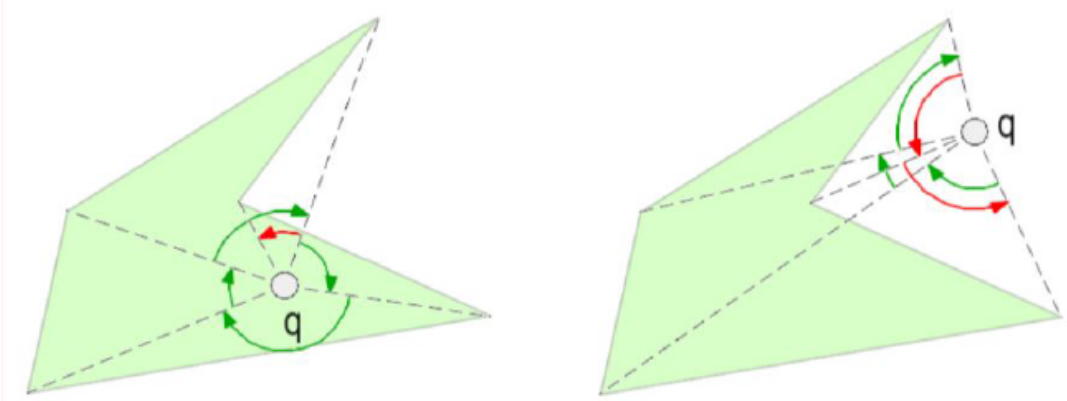
- pokud je M liché: bod náleží polygonu P ($q \in P$) ,
- pokud je M sudé: bod nenáleží polygonu P ($q \notin P$) .

2.1.1 Slovní zápis 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. Při této mětodě byla uvažována malá tolerance, jelikož je pravděpodobné, že výsledný úhel není roven přesně 2π nebo 0.



Obrázek 3: Princip Winding algoritmu [**?**]

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 Slovní zápis algoritmu

1. Nastavení výchozího úhlu $\omega = 0$, volba tolerance $\epsilon = 1e - 6$.
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 a pomocí velikosti zobrazovacího okna, aby bylo zajištěno zobrazení bodů ve vykreslovacím widgetu. Pro generování nekonvexního topologicky korektního polygonu (žádné hrany se nekříží) byla použita velká část Graham Scan algoritmu. Tento algoritmus funguje na principu zjišťování CCW (levotočivé) orientace trojúhelníku. Algoritmus Graham Scan je velmi rychlý a využívá se zejména ke kontrukci konvexních obálek.

Na počátku algoritmu setřídíme body podle Y souřadnice a bod s nejmenší Y souřadnicí označíme jako pivot (p). Poté se vypočteme směrnici vzhledem k ose x ke všem bodům množiny. Tyto body pak následně seřadíme podle velikosti směrnice. Takto seřazené body následně vykreslíme.

2.3.1 Slovní zápis algoritmu

1. Náhodné vygenerování (n-1) bodů: $x_{pi} = rand\%width; y_{pi} = rand\%height$.
2. Setřídění vstupní množiny bodů podle souřadnice y.
3. Nalezení pivota q: $q = min_{\forall p_i \in S}(y_i), q \in CH$ (pokud jsou souřadnice y u bodů stejné, vybere toho s nejvyšším indexem).
4. Nalezení bodu s ve směru osy x (levé rameno úhlu): $s = (q.x() - 1, q.y())$.

5. Opakuj pro všechny body vstupní množiny:

   Spočti $omega_i$: $omega_i = \angle(s, q, p_i)$

   Spočti $dist_i$: $dist_i = |qp_i|$

6. Setřídění bodů dle úhlu s osou x: $\forall p_i \in S$ sort by $\omega_i = \angle(s, q, p_i)$

7. Při nalezení stejného úhlu: $\omega_k = \omega_l \rightarrow$ neuvažovat bližší bod

8. Přidej body do polygonu, který pak bude vykreslován

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.

První případ singularity, kdy bod je identický s vrcholem polygonu, byl ošetřenve funkci getPointLinePosition porovnáním souřadnic bodu q a souřadnic bodu polygonu (tolerance totožnosti byla zvolena 1e-3). Druhý případ, kdy bod leží na hraně polygonu, byl ošetřen také pomocí funkce getPointLinePosition. Pokud bod náležel hraně polygonu či se nalézal ve vrcholu polygonu, byla návratová hodnota rovna funkce getPointLinePosition -2. V takovém případě byl bod q vyhodnocen, že se v polygonu P nachází. Tedy návratová hodnota funkce Ray Crossing je v takovém případě rovna 1.

Dále může nastat problém, že paprsek r prochází hranou polygonu, tedy není možné v této části jednoznačně určit průsečík. Z tohoto důvodu se využívá upravená varianta Ray Crossing Algorithm, kdy je nejprve provedena redukce souřadnic bodů polygonu k zájmovému bodu (lokální souřadnicový systém) 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.

3.2 Winding Number Algorithm

Nevýhodou této metody je její vyšší výpočetní náročnost než u Ray Crossing algoritmu. Nicméně výhodou je, že k singulárním případům dochází pouze v případěch, kdy $q \approx p_i$, tedy pokud zájmový bod splývá s některým z vrcholů polygonu, nebo pokud zájmový bod leží na hraně polygonu. Oba případy byly ošetřeny obdobně jako u výše popsaného Ray Crossing algoritmu rovnou při testu u funkce getPointLinePosition. Pokud dojde k některému z výše uvedených případů, oba dva sousední polygony budou vybarveny, jelikož bod q leží na hraně těchto polygonů. Může být vybarveno i více polygonů v případě, že bod q leží ve vrcholu, který náleží vícero polygonům.

4 Vstupní data

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

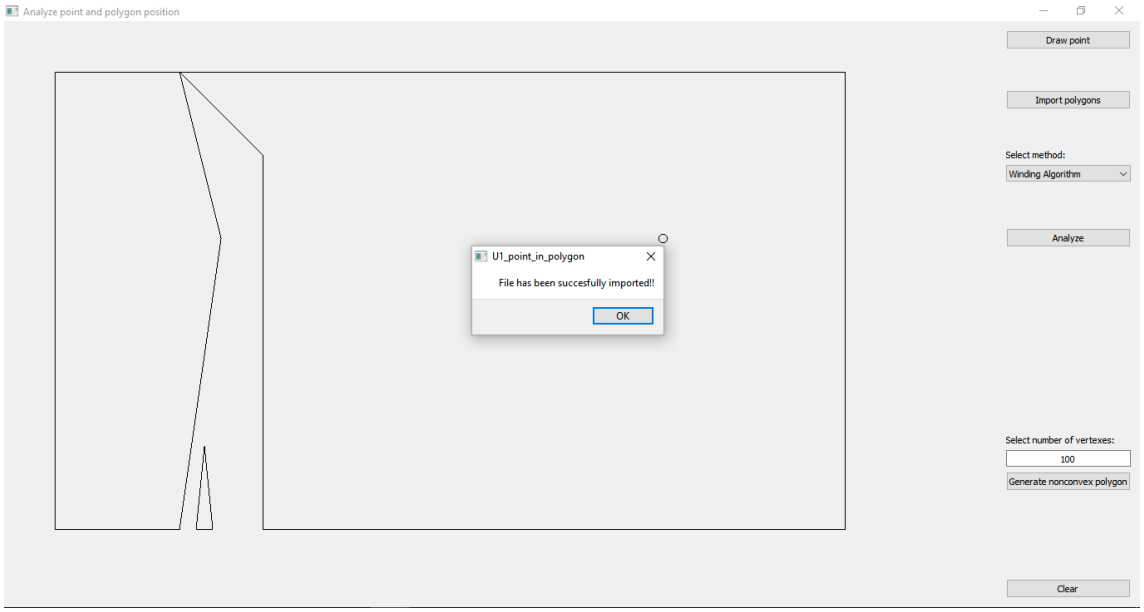
- Bod q je vložen uživatelem po ztisknutí tlačítka v grafickém okně aplikace.
- Souvislá mapa polygonů je ve formátu txt a obsahuje body jednotlivých polygonů. Má strukturu [číslo bodu, souřadnice X, souřadnice Y]. Soubor je seřazen od bodu č. 1 po n-tý bod polygonu. Jednotlivé body se nejprve uloží 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 a ošetření singularit, kdy bod náleží hraně nebo je identický s bodem polygonu. Zmíněné analýzy lze uplatnit také na automaticky vygenerováný nekonvexní polygon.

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

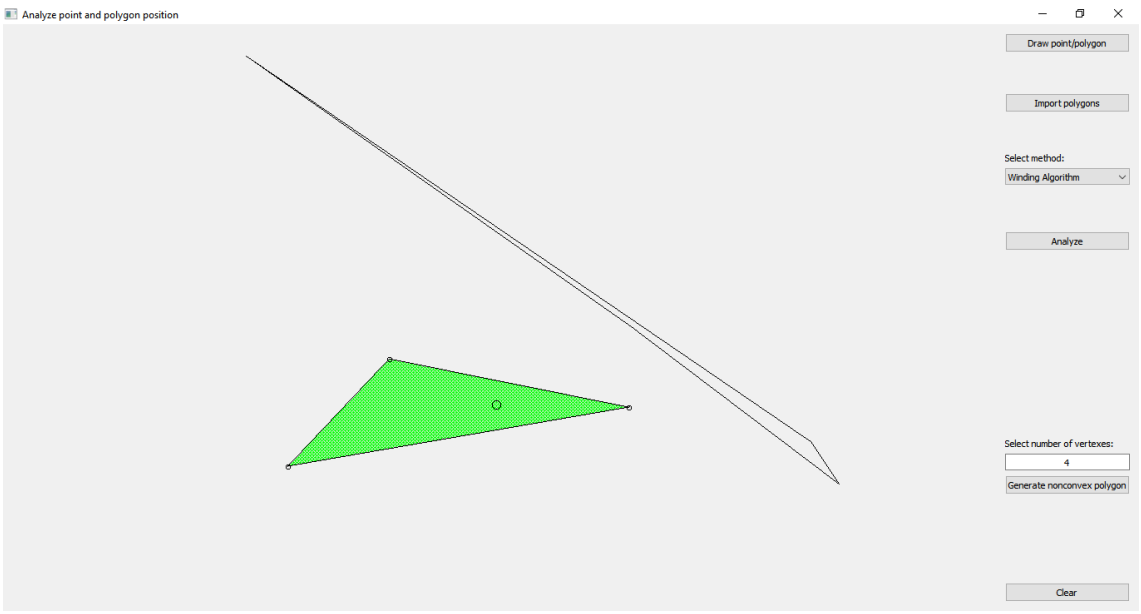
Obrázek 7: Analýza Ray Crossing pro bod ve vrcholu polygonu

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 s 50 vrcholy



Obrázek 12: Analýza manuálně a automaticky nakreslených polygonů

7 Technická dokumentace

7.1 Třídy

V aplikaci se nachází celkem šesti tříd - třídy Algorithms, Draw a Widget, sortByY, sortByX, sortByAngle. Každá z těchto tříd má svůj hlavičkový soubor, kde je deklarována spolu se svými metodami a svůj zdrojový soubor.

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-3*. 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*. Nejprve byly ve funkci ošetřeny singulární případy. Pomocí funkce Length2Points byly vypočteny dílčí vzdálenosti *p1q*, *p2q* a *p1p2*. Pokud rozdíl v absolutní hodnotě vzdálenosti *p1p2* a součtu vzdáleností *p1q*, *p2q* je menší než stanovená tolerance (zvolena 1e-1), bod se nachází blízko hrany a funkce vrací hodnotu -2. Dále se kontroluje, zda bod *q* nesplývá s bodem *p1* nebo *p2*, pokud ano, je rovněž vrácena hodnota -2. Teprve poté se z vektorů vypočítá determinant. 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, výstupem je hodnota *-1*.

double length2Points(QPointF p, QPointF q)
Vrací vzdálenost dvou bodů vypočtenou z Pythagorovy věty.

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 QPointF. 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 polygony o předem vybraném počtu vrcholů.

void mousePressEvent
V této funkci je v závislosti na hodnotě *draw_mode* vykreslen buď nový bod q, nebo je možné nakreslit ručně body 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 sloužící k vymazání všech polygonů i bodu q a k překreslení. Volá se před automatickým generováním polygonu P.

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 bod q třídy Draw.

QPointF getPolygon
Metoda, která vrací privátní polygon 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
Polygons.

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.

V rámci této metody je použit Graham Scan algoritmus.

7.1.4 sortByX

Třída, která definuje operátor přetížení *()*, který třídí body podle X souřadnice. Pokud je X souřadnice stejná, rozhoduje se podle souřadnice Y. Třídící funkce jsou využity při automatické tvorbě nekovexního polygonu.

7.1.5 sortByY

Třída, která definuje operátor přetížení *()*, který třídí body podle Y souřadnice. Pokud je Y souřadnice stejná, rozhoduje se podle souřadnice X.

7.1.6 sortByAngle

Třída, která definuje operátor přetížení *()*, který třídí úhly typu Angle (samostatně definovaná struktura) podle velikosti. Pokud je úhel stejný, rozhoduje vzdálenost mezi datovými prvky typu Angle.

7.1.7 Angle

Jedná se o strukturu, která definuje datový typ Angle, který je využíván při generování nekonvexního polygonu. Skládá se z bodu q typu QPointF, z úhlu od rovnoběžky s osou x k tomuto bodu (double) a ze vzdálenosti mezi počátečním bodem a bodem q.

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, ve které je 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 = 1e-1).

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

Ve funkci positionPointPolygonWinding je na začátku 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. Jelikož je pro tvorbu použit Graham Scan algoritmus, je výsledný polygon topologicky korektní (žádné dvě hrany se neprotínají).

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.