

155ADKG: Geometrické vyhledávání bodu

Datum opravy: 8.12.2017

Petra Millarová, Oleksiy Maybrodskyy

Contents

1	Zadání	3
2	Popis a rozbor problému	4
3	Popisy algoritmů	5
3.1	Ray crossing algorithm	5
3.2	Winding Number Algorithm	6
4	Problematické situace a singularity	7
4.1	Bod ležící na hraně polygonu	7
4.2	Bod je identický s vrcholem jednoho až n polygonů	7
5	Vstupní data	8
6	Výstupní data	8
7	Ukázky aplikace	9
8	Dokumentace	13
8.1	Popis tříd	13
8.1.1	Algorithms	13
8.1.2	Draw	15
8.1.3	Widget	16
9	Závěr	17
9.1	Náměty na vylepšení	17
	References	17

1 Zadání

Následuje kopie oficiálního zadání úlohy. Autoři z nepovinných bodů zadání implementovali všechny kromě algoritmu pro automatické generování nekonvexních polygonů.

Úloha č. 1: Geometrické vyhledávání bodu

Vstup: Souvislá polygonová mapa n polygonů $\{P_1, \dots, P_n\}$, 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

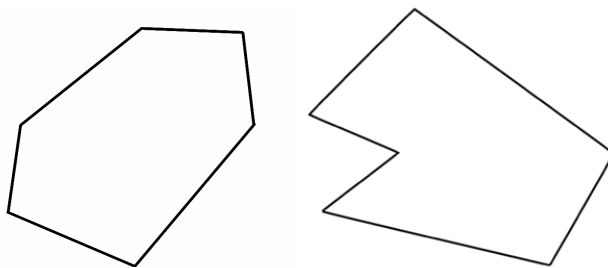
2 Popis a rozbor problému

Tato úloha se věnuje řešení praktického problému určování pozice uživatelem zadaného bodu q vůči polygonům načteným ze souboru. Jako implementaci si lze zjednodušeně představit zjišťování polohy konkrétního bodu kliknutím na digitální mapě.

Nechť ve dvojrozměrné kartézské soustavě existuje množina bodů tvořena n body. Uzavřením tohoto této množiny vznikne polygon. Polygon může nabývat jak konvexní, tak nekonvexní tvar.

Polygon je konvexní právě tehdy, když poloha všech bodů je vůči jakékoli přímce procházející vedle polygonu, vždy na stejné straně.

Rozdíl mezi konvexním a nekonvexním polygonem je možné názorně vidět na obrázcích níže.



obr 1.: konvexní polygon (vlevo) a nekonvexní polygon (vpravo)

Pokud následně polygon rozdělíme na menší polygonové útvary, pak polohu zvoleného bodu q můžeme popsat následovně:

1. Bod q se nachází uvnitř polygonu $q \in P_i$
2. Bod q se nachází vně všech polygonů $q \notin P_i$
3. Bod se nachází na hraně jednoho $q \notin P_i$ nebo dvou polygonů $q \in P_{i,i+1}$
4. Bod je totožný s vrcholem jednoho polygonu nebo více polygonů $q \in P_{i,i+1,\dots,i+n}$

Výpočet se bude provádět na základě metod **Ray Crossing algorithm** a **Winding Number algorithm**. Jejich výpočet je popsán v následujících kapitolách.

3 Popisy algoritmů

V dané úloze jsou použity následující algoritmy, avšak existují i další možnosti, jak polohu bodu určit (metoda pásů, Line Sweep algorithm aj.)

3.1 Ray crossing algorithm

Nechť existuje uzavřený polygon ve dvojrozměrné kartézské soustavě, tvořený n body. Nechť následně existuje bod q , kteréhož polohu se snažíme určit. Proložíme-li bodem q nekonečný počet paprsků směrem k polygonu, pak pro jednotlivý paprsek nastane jedna z následujících situací:

1. počet průsečíků paprsku k je roven sudému počtu, pak se bod q nachází vně polygonu $q \notin P_i$
2. počet průsečíků paprsku k je roven lichému počtu, pak se bod q nachází uvnitř polygonu $q \in P_i$

Zároveň mohou nastat singularity, respektive jisté situace, kdy algoritmus "nefunguje" a nedokáže přímo nalézt správný výsledek. V algoritmu ray crossing se konkrétně jedná o tyto případy:

1. Bod se nachází na hraně jednoho $q \in P_i$ nebo dvou polygonů $q \in P_{i,i+1}$
2. Bod je totožný s vrcholem jednoho polygonu nebo více polygonů $q \in P_{i,i+1,\dots,i+n}$

Řešením je posun, respektive redukce vrcholů polygonů směrem k poloze bodu q .

Hledaný algoritmus je možné popsat následovně:

1. Inicializace bodů polygonu p_i , počet průsečíku = 0;
2. Redukce souřadnic x bodů polygonu k bodu q , respektivě k paprskovému segmentu, $x'_i = x_i - x_q$.
3. Redukce souřadnic y bodů polygonu k bodu q , respektivě k paprskovému segmentu, $y'_i = y_i - y_q$.
4. Znovu pro ostatní body daného polygonu p_i /
5. if $(y'_i \leq 0) \& \& (y_{i+1}' > 0) \parallel (y'_i > 0) \& \& (y_{i+1}' \leq 0)$.
6. $x'_m = (x_i + 1' y'_i - x'_i y_i + 1') / (y'_{i+1} - y'_i)$.
7. Sčítání počtu redukovaných bodů, pro $x'_i > 0$
8. Pokud je počet průsečíků sudý, pak $q \in P$, pokud není, pak $q \notin P$

3.2 Winding Number Algorithm

Nechť existuje uzavřený polygon ve dvojrozměrné kartézské soustavě, tvořený pomocí n bodů. Nechť následně existuje bod q , polohu kteréhož se snažíme určit. Z pohledu bodu q provedeme orientaci směru, ze které se pak následně určí součet všech úhlů na jednotlivé body uvedeného polygonu.

Součtový úhel bude dále značen jako w . Výpočet je lepší provádět proti směru hodinových ručiček, jelikož v případě tohoto směru hodnota počítaných oběhů Winding Number Ω nabývá kladných hodnot. Je třeba také pamatovat, že hodnota Ω je uváděna v počtech oběhů a je záporná při oběhu po směru hodinových ručiček a kladná ve směru opačném. Do výpočtu také vstupuje tolerance ϵ , která zahrnuje chyby vzniklé zaokrouhlováním a strojovou přesností. Dle uvedených matematických podmínek mohou nastat následující případy:

1. $w = 2\pi$, pak $q \in P_i$
2. $w < 2\pi$, pak $q \notin P_i$

Níže je uveden algoritmus výpočtu:

1. Vstup $\omega = 0$, tolerance ϵ
2. Orientace z bodu q každého následujícího bodu p_{i+1} od orientaci na bod p_i
3. Určení úhlu $\omega_i = \angle p_i, q, p_{i+1}$
4. $\omega = \omega + \omega_i$, pro bod vprávo od orientace na bod p_i , pokud je bod vlevo od orientace na bod p_i , pak $\omega = \omega - \omega_i$
5. Pokud platí podmínka $(|\omega - 2\pi| < \epsilon)$, pak $q \in P$
6. Pokud neplatí podmínka $(|\omega - 2\pi| < \epsilon)$, pak $q \notin P$

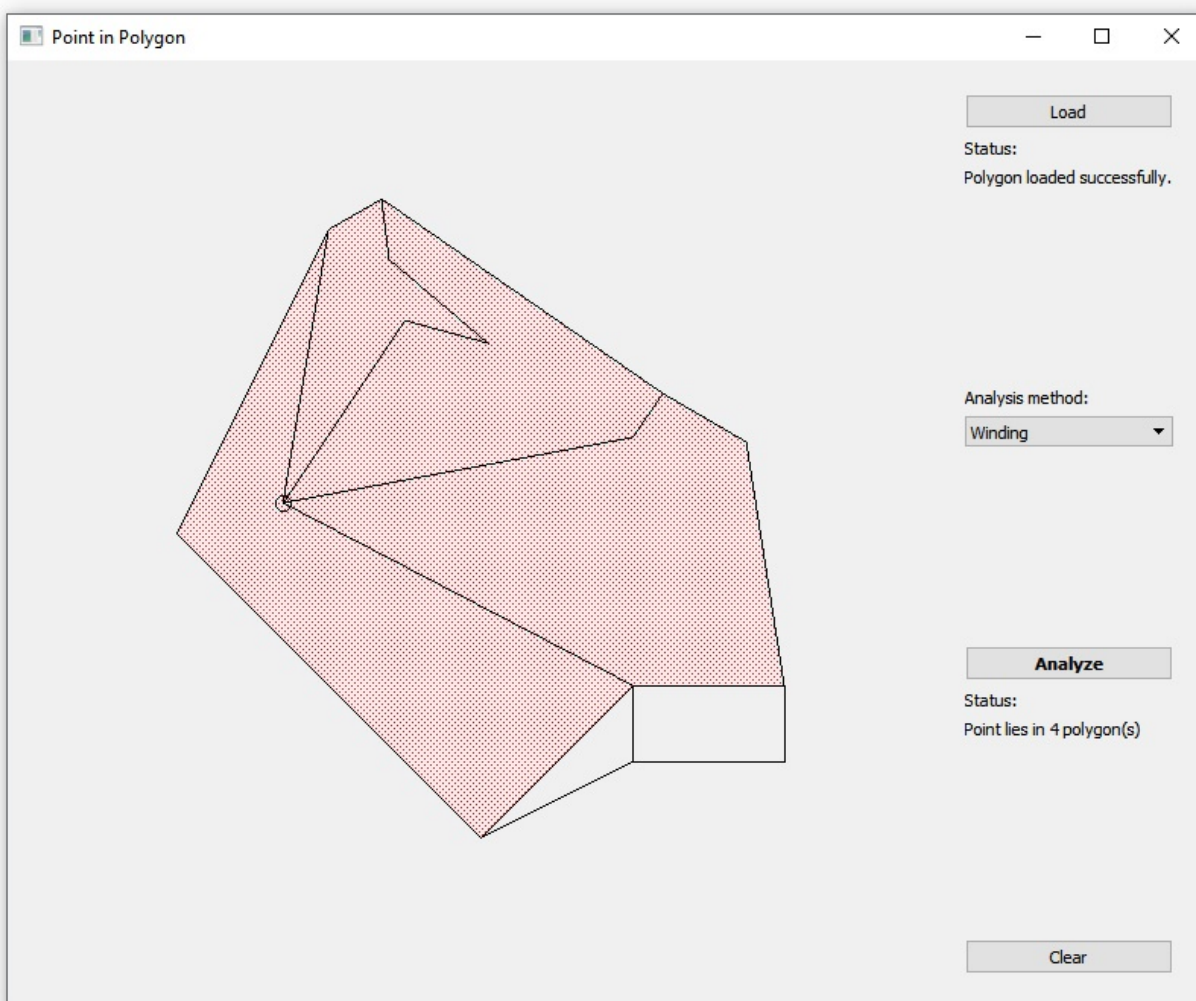
4 Problematické situace a singularity

4.1 Bod ležící na hraně polygonu

Pokud vyjdeme z podmínky, že bod se nachází na hraně polygonu, pak z pohledu geometrie je jasné, že svýrající úhel mezi krajními body dané hrany bude roven R neboli. Pokud ale ještě zavedeme chybu toleranci ϵ , pak $R + \epsilon$. Jelikož bod leží na hraně, pak zároveň patří do obou polygonů.

4.2 Bod je identický s vrcholem jednoho až n polygonů

Byl proveden prohledávací test, jestli souřadnice bodu q jsou identické se souřadnicemi kterého koliv polygonu. Pokud test byl úspěšný, pak se již dále neprováděl a rovnou se vybarvil konkrétní polygon. Pokud během testu se ukázalo, že souřadnice jsou identické s vrcholem, který je společný pro několik polygonů, pak se vybarvilo hned několik polygonů.



5 Vstupní data

Do programu vstupují dvě odlišné hodnoty:

1. analyzovaný bod q .
2. soubor polygonů.

Analyzovaný bod q , vstupuje na základě ručního vstupu přes GUI, tedy zmačknutím levého tlačítka myši v grafickém okně.

Vstupní (.txt) soubor obsahuje polygony zadané jednotlivými body.

Složený vstupních dat .txt: První řádek obsahuje počet vstupujících polygonů souřadnici X

Druhý řádek obsahuje počet bodů prvního polygonu

Třetí řádek obsahuje vypsání souřadnice X vstupního polygonu

Čtvrtý řádek obsahuje vypsání souřadnice Y vstupního polygonu

Další řádek obsahuje počet bodů druhého polygonu, následující vstup je identický se vstupem prvního polygonu

Při vstupu textového souboru program již rovnou rozezná počet vstupních polygonů a jejich rozměry, což značně ulehčuje následující práci. Vstupní souřadnice mohou být, jak celočíselné, tak s desetinou tečkou. Souřadnice X a Y se sekvenčně ukládají do proměnné **QPoint**, která se ukládá do proměnné vektoru typu **std::vector<QPoint>**, který znázorňuje polygon, který se následně ukládá do proměnné typu **std::vector<vector<QPoint>>**, která sloučuje všechny polygony.

6 Výstupní data

Výstup je vizualizací řešeného problému v grafickém okně. Také v grafickém okně je slovně popsáno, do kolika polygonu náleží hledaný bod.

Po načtení vstupních dat **Load**, v pravém horním rohu grafického okna, se polygony ihned vykreslí a po libovolným kliknutí levého tlačítka myši se vykreslí také vstupní bod.

Podle výsledků daného testu se červeně zvýrazní polygony, do kterých patří hledaný bod.

7 Ukázky aplikace

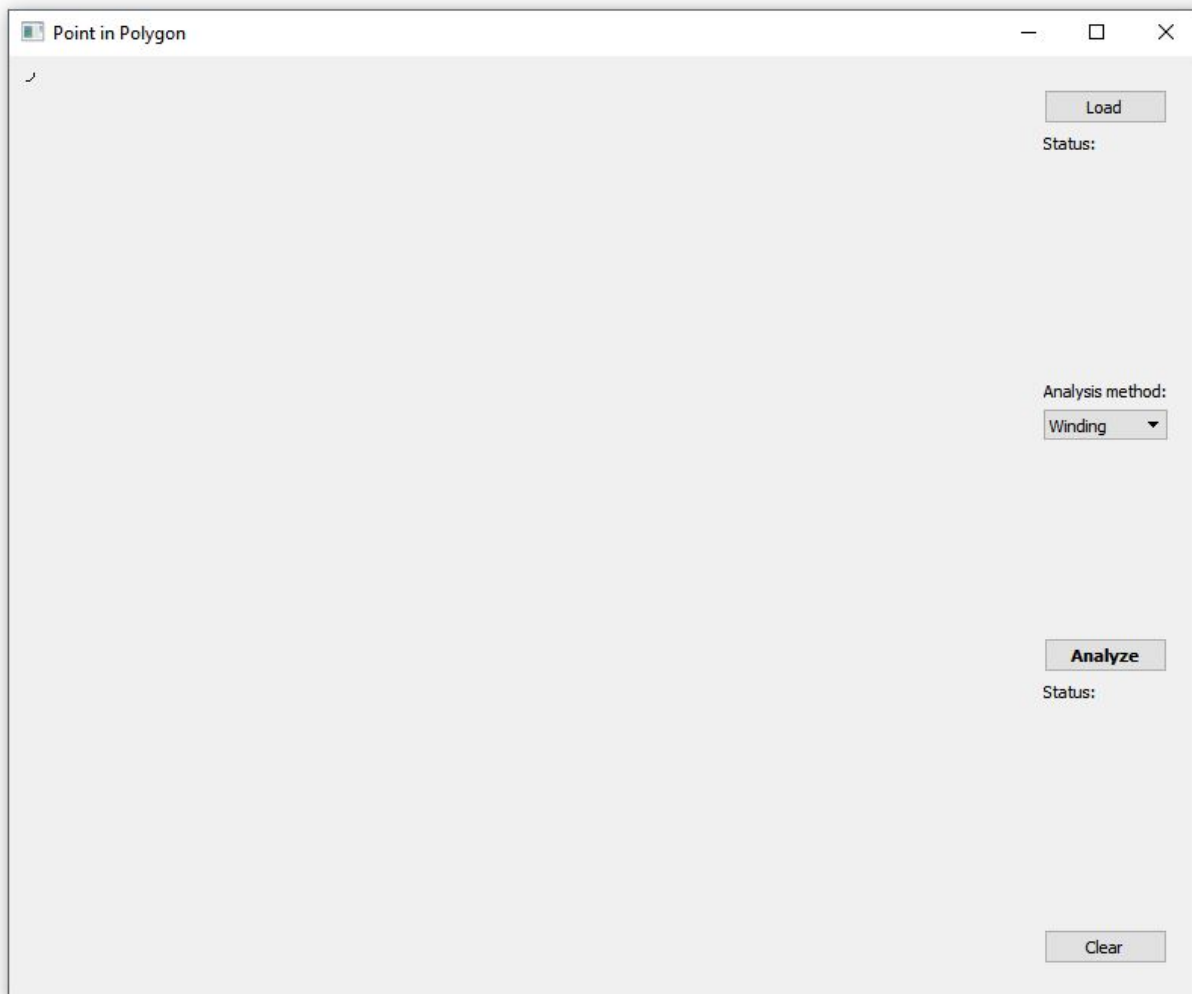


Figure 1: Aplikace po spuštění

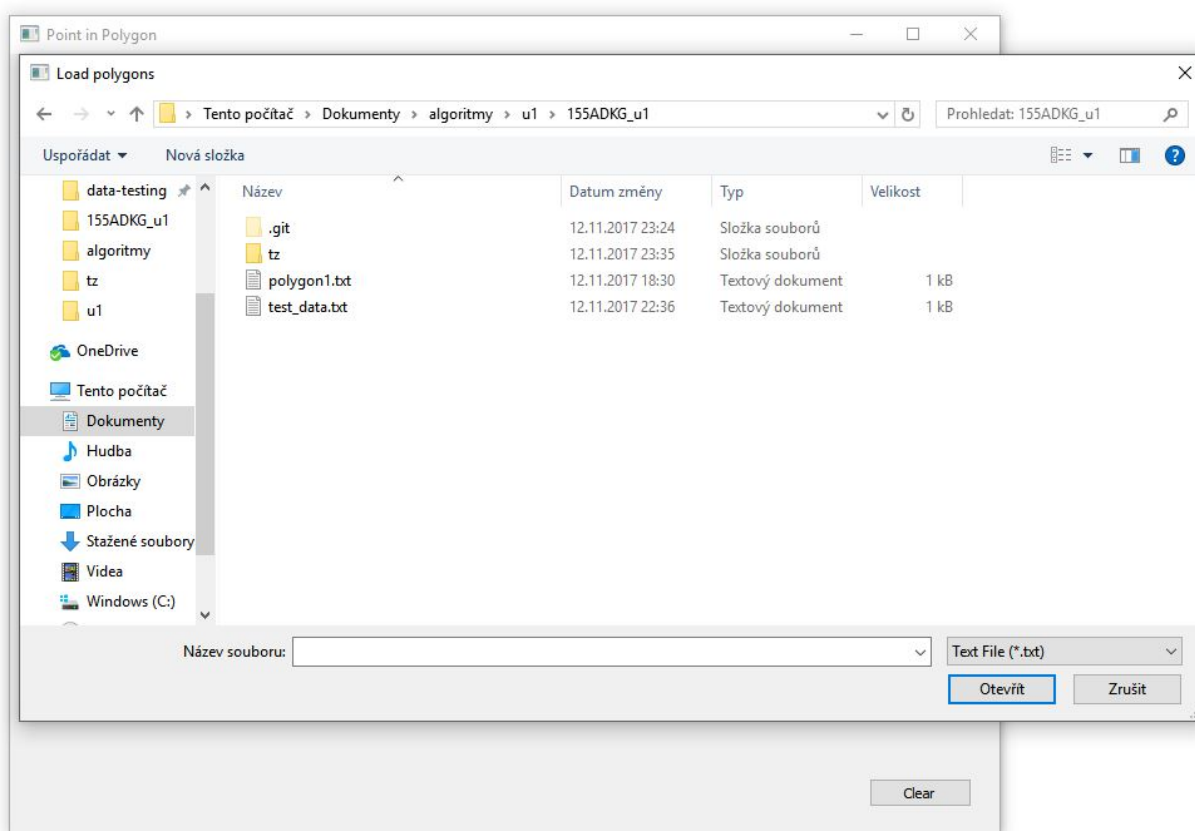


Figure 2: Načtení dat

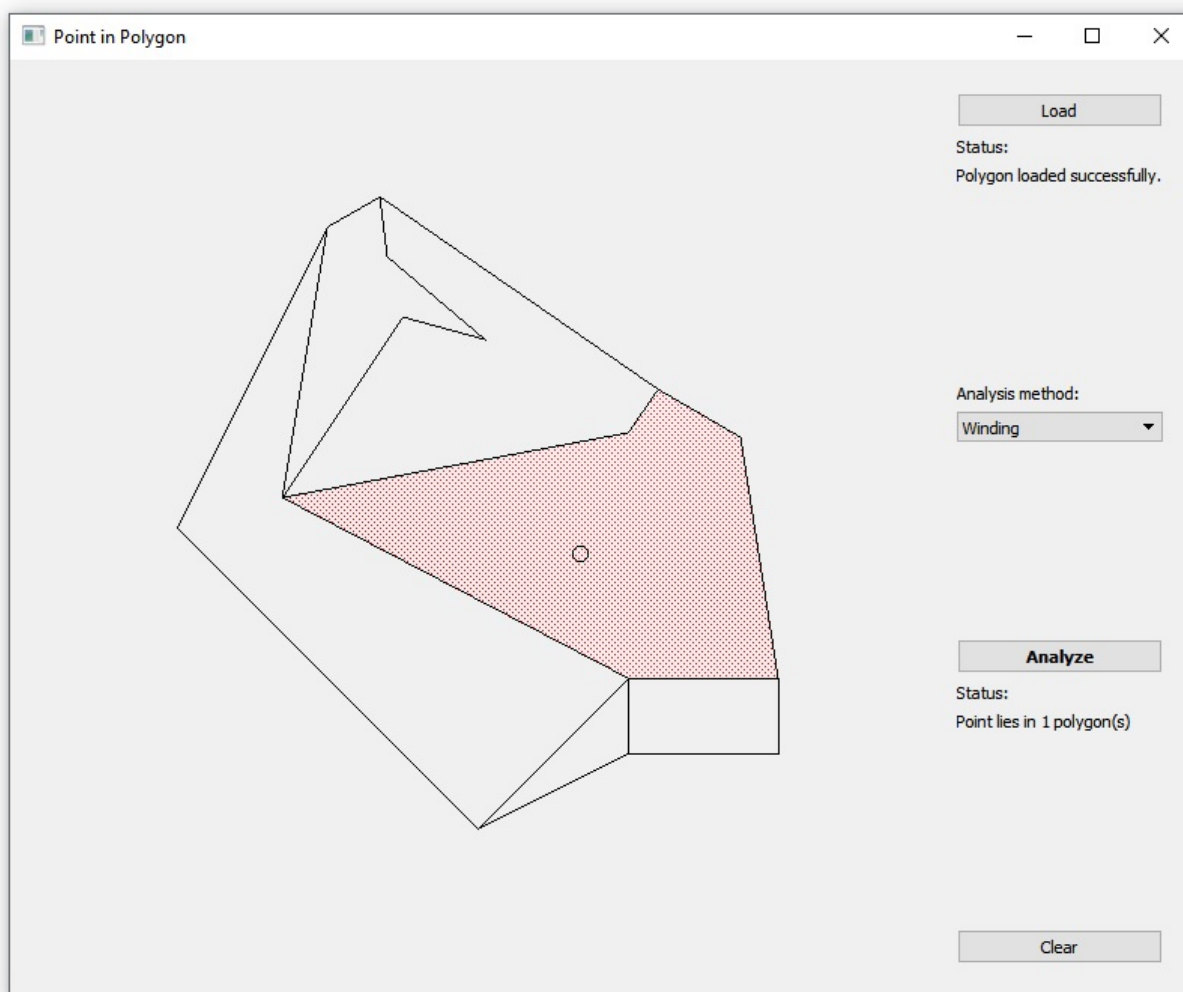


Figure 3: Po načtení a stisknutí tlačítka "Analyze"

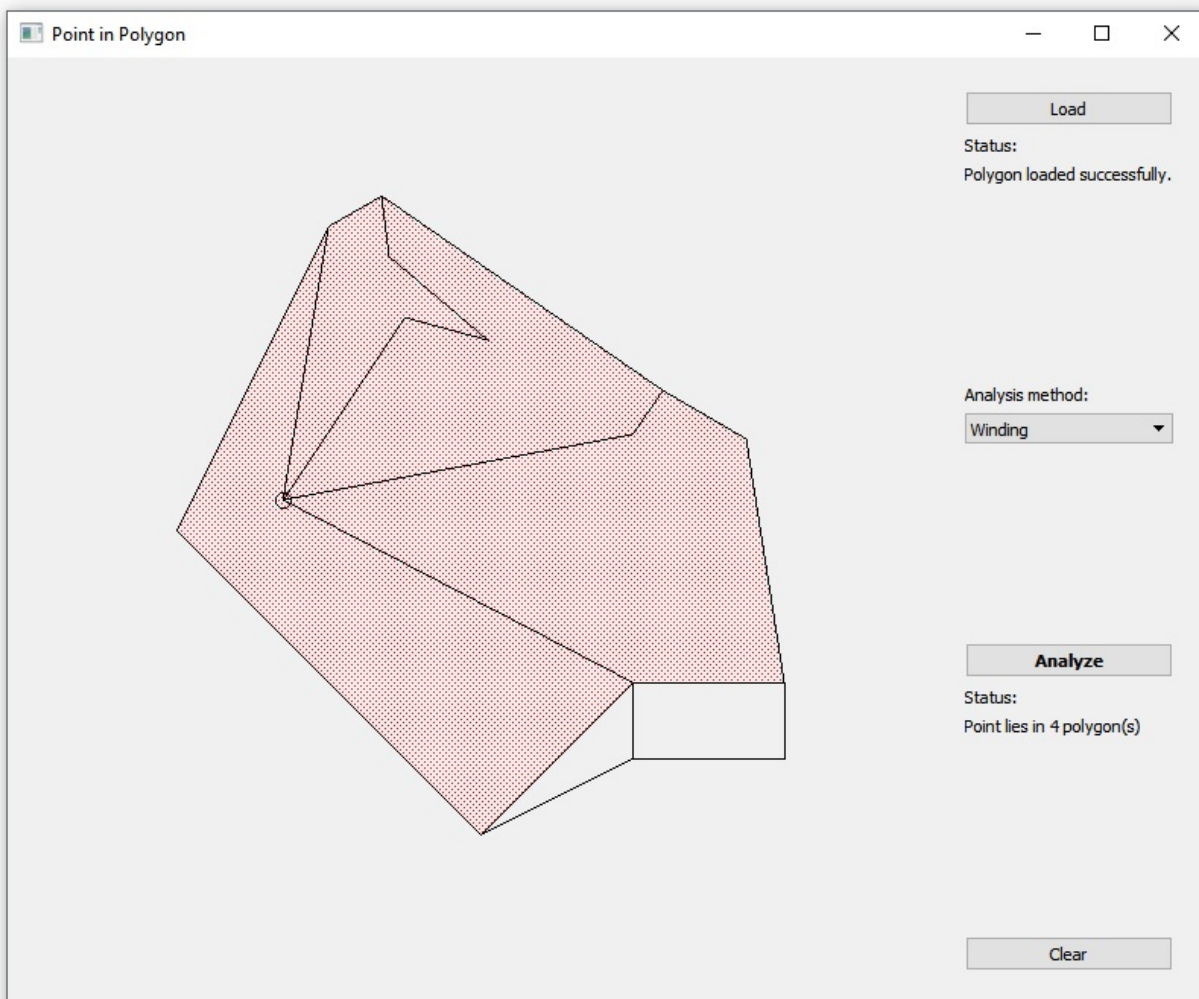


Figure 4: "Výstup s více polygony"

8 Dokumentace

8.1 Popis tříd

8.1.1 Algorithms

Třída byla vytvořena pro provedení výpočetních funkcí, je tvořená **Ray** a **Winding Number** algoritmy. Součástí těchto algoritmů jsou funkce pro iteraci algoritmu a výpočet úhlu mezi dvěma vektory a zjišťování pozice bodu vůči přímce.

getPosition je funkce, která popisuje polohu bodu vůči linii. Do funkce vstupují 3 hodnoty typu **QPoint**, které reprezentují hodnoty dvou vektorů. Návrátová hodnota této funkce je celé číslo **integer**.

Input:

1. *QPoint* &q
2. *QPoint* &a
3. *QPoint* &b

Output:

1. *int*

getAngle provádí výpočet úhlu mezi dvěma vektory. Do funkce vstupují 4 hodnoty typu **QPoint**, které reprezentují hodnoty dvou vektorů. Jelikož se jedná o výpočet úhlu, tak logicky návratovou hodnotou je funkce typu **double**.

Input:

1. *QPoint* &p1
2. *QPoint* &p2
3. *QPoint* &p3
4. *QPoint* &p4

Output:

1. *double*

getWindingPos určuje polohu bodu vůči vektoru polygonů. Návrátová hodnota je **bool** která obsahuje informaci jestli je bod uvnitř, respektivě na hraně, nebo jestli je mimo polygon.

Input:

1. *QPoint* &q

2. $std :: vector < QPoint > pol$

Output:

1. $bool$

getRayPos určuje polohu bodu vůči vektoru polygonů. Návrátová hodnota je **bool** která obsahuje informaci jestli je bod uvnitř, respektivě na hraně, nebo jestli je mimo polygon.

Input:

1. $QPoint \&q$
2. $std :: vector < QPoint > pol$

Output:

1. $bool$

iterateWindingPos vrací konkrétní indexy polygonů, ve kterých se bod q nachází. Uvnitř funkce dochází k volání funkce **getWindingPos** která právě uvádí polohu bodu v polygonu.

Input:

1. $QPoint \&q$
2. $std :: vector < std :: vector < QPoint >> polist$

Output:

1. $std :: vector < int >$

iterateRayPos vrací konkrétní indexy polygonů, ve kterých se bod q nachází. Uvnitř funkce dochází k volání funkce **getRayPos** která právě uvádí polohu bodu v polygonu.

Input:

1. $QPoint \&q$
2. $std :: vector < std :: vector < QPoint >> polist$

Output:

1. $std :: vector < int >$

8.1.2 Draw

Draw slouží k vykreslování testovaného bodu q a vykreslování polygonů včetně jejich výplně s ohledem na umístění bodu q .

mousePressEvent funkce slouží pro přímé načítání souřadnic bodu q z programu. `QPoint`. Funkce neobsahuje návratovou hodnotu.

Input:

1. `QMouseEvent *e`

Output:

1. none

paintEvent primární účel funkce je grafické znázornění načtených polygonů, pokud bod se nachází bezprostředně uvnitř polygonu, tak aj jejich barevná výplň.

Input:

1. `QPaintEvent *e`

Output:

1. none

clear, jak již název vypovídá funkce slouží k vyčištění dosavadních dat a umožňuje tak znovrozjetí aplikace.

Input:

1. none

Output:

1. none

loadData funkce naplňuje vektor polygonu. Součástí funkce jsou oznamení jednotlivých chyb v případě špatných vstupních hodnot, nebo jejich neodstačující počet, nebo že načtení proběhlo úspěšně.

Input:

1. `constchar*` path
2. `std::ifstream` &file
3. `QString` &status

Output:

1. none

8.1.3 Widget

Je to třída vytvořena pro práci s grafickým rozhraním celého programu. Přes ní se provádí načtení souborů a grafické znázornění výsledku aplikace.

on pushButton clear clicked

Provedení obnovy programu.

Input:

1. none

Output:

1. none

on pushButton analyze clicked provádí analýzu jednotlivých požadavků uživatele, jestli je požadován algoritmus Ray nebo Winding Number provádí předspracování dat. Zobrazí pomocí *label* výsledky na obrazovce.

Input:

1. none

Output:

1. none

on pushButton load clicked Primární účel funkce je načtení souboru dat z textového souboru .txt. Otevírá grafické rozhraní vyhledávání cesty k souboru.

Input:

1. none

Output:

1. none

9 Závěr

Autoři splnili většinu bodů zadání a vznikl program, který načítá soubor polygonů, následně uživatele nechá umístit bod a po stisknutí tlačítka určí, zda a ve kterých polygonech bod leží.

9.1 Náměty na vylepšení

Aplikace, ač funkční a splňující daný účel, má spoustu nedostatků, které by bylo dobré v budoucnu odstranit. Autoři zde uvádí pár těch nejzjevnějších.

Vykreslování dat: Aplikace bez problému vykreslí body, které se vejdou do jejího okna 665x605px. Problém nastává až tehdy, když jsou souřadnice větší než tato hodnota. Tato chyba jde odstranit vhodnou transformací okna (nebo souřadnic), která by ale probíhala na základě načtených dat.

Souřadnicové osy: Vykreslovací okno má v Qt, stejně jako ve většině podobných nástrojů, počátek souřadnic v levém horním rohu, kladnou osu x vpravo a kladnou osu y směrem dolů. Tento model se však neshoduje ani s geodetickými souřadnicemi používanými na našem území (kladná y doleva, kladná x dolů), ani s klasickým označením os (kladná x doprava, kladná y nahoru). Proto se body v současné verzi zobrazují jinak, než by možná uživatel očekával. Vhodným řešením by byla opět transformace.

Přesnější souřadnice: V současném stavu aplikace sice načítá data ve formátu `double`, avšak datová struktura `std::vector<std::vector<QPoint>>` body načítá bez desetinných míst jako typ `int`. V Qt knihovnách existuje i datový typ `QPointF`, který ukládá body jako typ `float`. Změně však bude potřeba přizpůsobit porovnávání čísel v algoritmech.