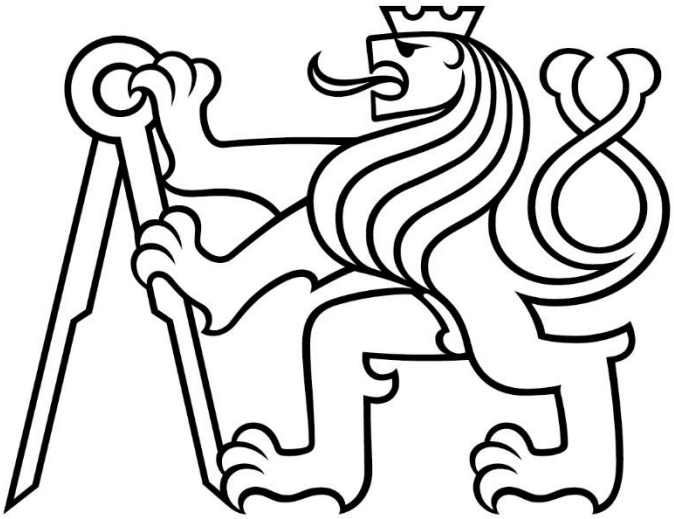


ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

FAKULTA STAVEBNÍ
KATEDRA GEOMATIKY

název předmětu

ALGORITMY V DIGITÁLNÍ KARTOGRAFII

úloha

U1: GEOMETRICKÉ VYHLEDÁVÁNÍ BODU

datum	vypracovali
17.10.2020	Bc. Josef Műnzberger, Bc. Martin Hudeček

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

V rovině je dána množina bodů P_1, P_2, …, P_n, které tvoří vrcholy mnohoúhelníku, a také bod q. Cílem úlohy je identifikovat takový mnohoúhelník, ve kterém se nachází bod q. Řešení tohoto problému se nabízí vícero a výběr metody závisí na vlastnostech testovaných mnohoúhelníků, které rozlišujeme konvexní a nekonvexní[1].

Test vzájemné polohy bodu a polygonu (mnohoúhelníku) může nabývat několika možných výsledků:

- bod q leží vně polygonu
- bod q leží uvnitř polygonu
- bod q leží na hraně polygonu
- bod q je totožný s jedním z vrcholů polygonu

K řešení úlohy byly aplikovány dvě metody: **Winding Number Algorithm** a **Ray Crossing Algorithm** neboli paprsková metoda.

1.1. Údaje o bonusových úlohách

Mimo hlavní část řešené úlohy (implementace dvou zmíněných algoritmů) byly také ošetřeny některé singulární případy: v rámci metody *Winding Number Algorithm* byl ošetřen případ, kdy bod leží na hraně polygonu (resp. leží velmi blízko některé z hran), v obou metodách byl pak řešen případ, kdy bod q je totožný s jedním z vrcholů polygonu (resp. je mu velmi blízký).
V případě, kdy program detekuje bod uvnitř, na hraně či přímo ve vrcholu polygonu, zvýrazní polygon zelenou barvou. Nakonec byl přidán algoritmus pro generování náhodných polygonů se zadaným počtem vrcholů[2].

2. Popis použitých algoritmů

Jak již bylo zmíněno výše, výběr použitých metod závisí na vstupních datech ve smyslu konvexnosti či nekonvexnosti polygonů. Zatímco pro konvexní mnohoúhelníky lze problém řešit poměrně snadno metodou *Half-plane test*, pro nekonvexní polygony bylo potřeba zvolit sofistikovanější řešení, kterými lze řešit úlohu rovněž pro polygony konvexní.

2.1. Winding Number Algorithm

V této metodě postupně procházíme vrcholy Polygonu P a zjišťujeme velikost úhlu sevřeného body P_i, q, P_{i+1}. V závislosti na výsledku implementované metody *half-plane test* tento úhel buď přičteme do nebo odečteme od proměnné Ω. Jakmile projdeme všechny body polygonu můžeme prohlásit, že pokud Ω = 0, pak bod q leží vně polygonu, v případě Ω = 1, bod q náleží testovanému polygonu. Byla vhodně zavedena malá tolerance, aby výsledný úhel Ω nemusel nabývat přesně hodnot 0 či 1. Definice chápe číslo *Winding Number* Ω jako sumu všech rotací ω_i měřená CCW, které musí průvodič opsat nad všemi body polygonu.

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

Tento vztah byl v algoritmu zjednodušen; obejdeme se bez konstanty 2π , aby pak Ω nabývala hodnot 0 či právě 2π.

---

[1] Žádné dvě strany konvexních mnohoúhelníků (na rozdíl od nekonvexních) nesvírají větší vnitřní úhel než 180°.
[2] Počet vrcholů musí být větší než 3, jsou generovány náhodné polygony, tj. konvexní, nekonvexní, simple i non-simple.

2.1.1. Slovní zápis algoritmu

- Nastavení Ω = 0, volba tolerance ξ = 1e-6
- Určení orientace$o_i$bodu q vzhledem ke straně P_i, P_{i+1}
- Určení úhlu ω_i sevřeného body P_i, q, P_{i+1}
- Volba podmínky: v závislosti na vyhodnocení testu *orientace* přičteme či odečteme ω_i
  - Bod q leží v levé polorovině: $\Omega = \Omega + \omega_i$
  - Bod q leží v pravé polorovině: $\Omega = \Omega - \omega_i$
- Volba podmínky: testujeme výraz $\left||\Omega| - 2\pi\right|$
  - Bod q náleží polygonu: $\left||\Omega| - 2\pi\right| < \xi$
  - Bod q nenáleží polygonu: $\left||\Omega| - 2\pi\right| \geq \xi$

2.2. Ray Crossing Algorithm

Jádro této metody tkví ve tvorbě polopřímky vedoucí z daného bodu q a následném součtu všech jejích průsečíků s polygonem. Polopřímka může být obecně libovolná, ale výhodné je zavést polopřímku rovnoběžnou s některou z os souřadnicového systému. Tato metoda je o řád rychlejší než *Winding Number Algorithm*, nicméně je potřeba ošetřit singulární případy (viz kapitola 3). V závislosti na počtu průsečíků polopřímky s polygonem můžeme určit polohu bodu q; sudý počet průsečíků znamená bod vně polygonu, lichý uvnitř.

2.2.1. Slovní zápis algoritmu

- Nastavení počtu průsečíků $k = 0$
- Redukce souřadnic všech bodů polygonu (zavedení lokálního souřadnicového systému neboli posunutí počátku souřadnicového systému do bodu q)
- Testování podmínky: $\left(y'_i < 0 \cap y'_{i-1} \geq 0\right) \cup \left(y'_i \leq 0 \cap y'_{i-1} > 0\right)$
- Pokud je splněna, provedeme výpočet x-ové souřadnice:

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

- Pokud je splněna další podmínka, že $x'_M > 0$, pak zvětšíme počet průsečíků k o jedna.
- Nakonec určíme, zda je počet průsečíků k sudý nebo lichý a tím rozhodneme, zda bod q náleží polygonu.

3. Problematické situace a jejich rozbor

3.1. Winding Number Algorithm

Problematický případ, který bylo u této metody nutné ošetřit, nastává, pokud bod q leží na hraně polygonu (nebo je totožný s jedním z vrcholů). Nejprve byla zvolena tolerance, jak blízko u hrany se může bod nacházet, aby byl považovaný jako ležící na hraně (obdobně platí pro toleranci totožnosti bodu s některým z vrcholů polygonu). Testování podmínky tolerance bylo implementováno již do metody getPointLinePosition, která je volána metodou getPositionWinding. V případě, že byla podmínka splněna, vrací se hodnota 2, resp. 3 (pokud $q \approx P_i$), pomocí které posléze program vytiskne polohu bodu (*outside, inside, border, vertex*).

3.2. Ray Crossing Algorithm

Singulárními případy této metody byly částečně ošetřeny analogicky k předchozí podkapitole (případy, kdy bod q leží na hraně či dokonce splývá s některým z vrcholů polygonu), nicméně jiná singularita nastává, pokud polopřímka prochází hranou polygonu. Toto lze vyřešit zavedením lokálního souřadnicového systému (posunutí počátku souřadnicového systému do

bodu *q*) a počet průsečíků zjišťujeme pouze pro I. kvadrant lokálního souřadnicového systému (testujeme, zda *x*-ová souřadnice průsečíku je kladná).

4. Vstupní data

Do programu vstupují dva elementy:

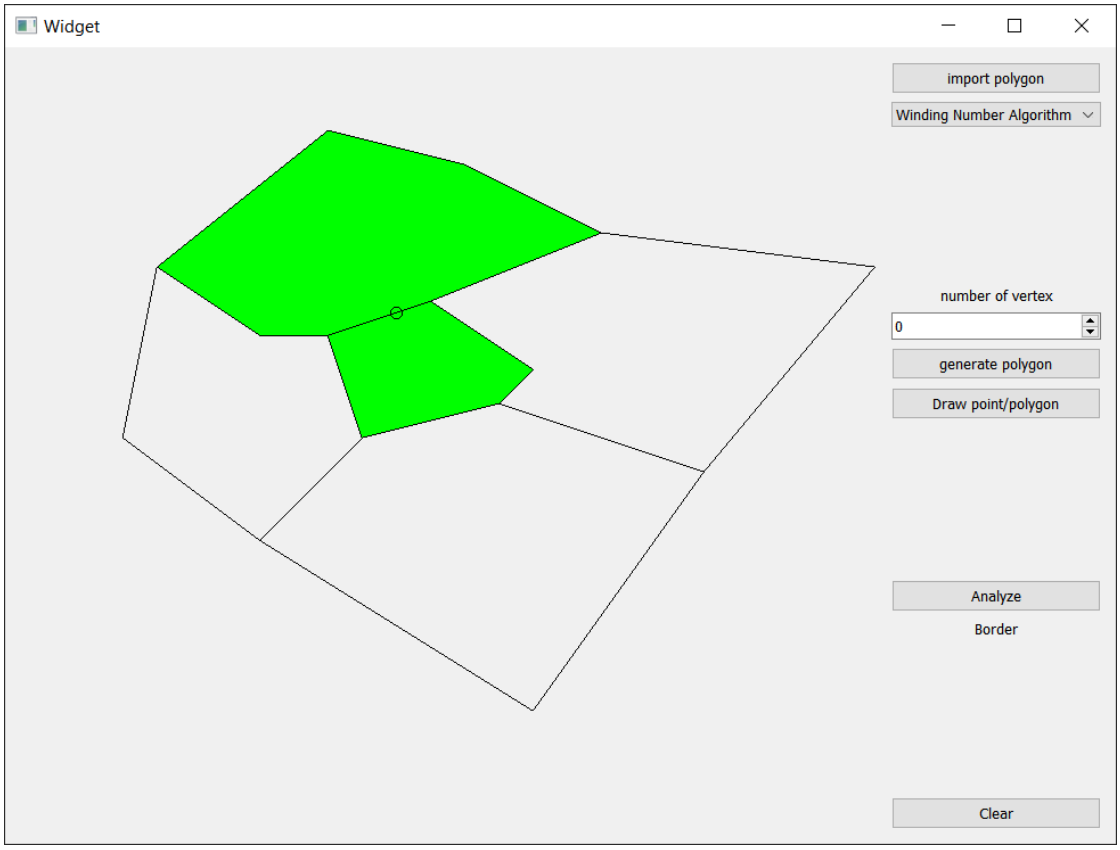
- Bod *q*, který je uživatelem zadáván manuálně kliknutím do grafického okna.
- Polygon *P*, který je možné
    - nakreslit ručně (naklikat vrcholy v grafickém okně)
    - importovat v souboru *.txt* obsahujícím souřadnice. Co se týče struktury textového souboru, řádky postupně obsahují ID *n-tého* bodu a jeho souřadnici X a souřadnici Y. Body řazeny dle ID.
    - náhodně vygenerovat po zadání požadovaného počtu vrcholů.

5. Výstupní data

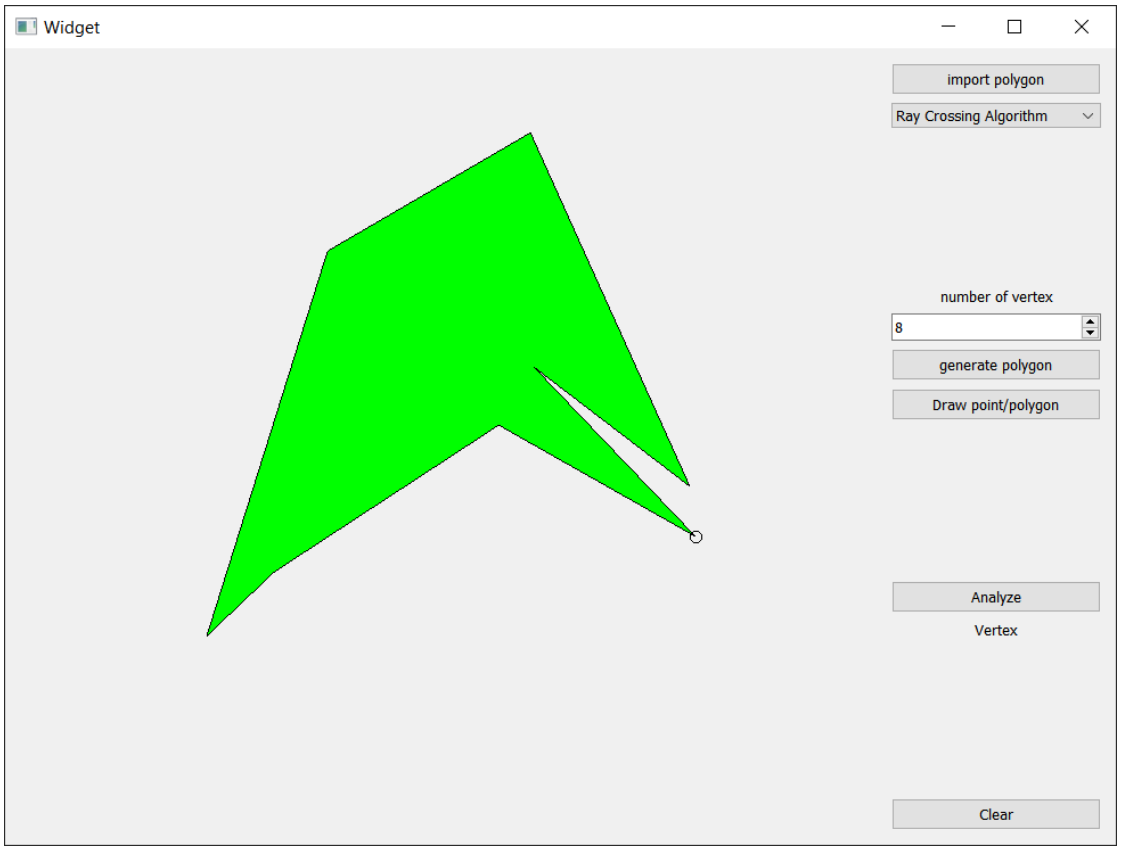
Výstupem programu je grafická aplikace, která polohově zobrazuje nakreslené, vygenerované či importované polygony a zadaný bod. Pomocí výběru ze dvou algoritmů (*Winding Number Algorithm* a *Ray Crossing Algorithm*) vyhodnotí polohu bodu vzhledem k polygonové mapě a vytiskne výslednou hlášku (*outside, inside, border, vertex*). V závislosti na poloze bodu aplikace barevně zvýrazní polygon, uvnitř kterého se bod nalézá; v případě umístění bodu na hraně či vrchol zabarví všechny zainteresované polygony, které danou hranu či vrchol sdílí.

6. Ukázka aplikace

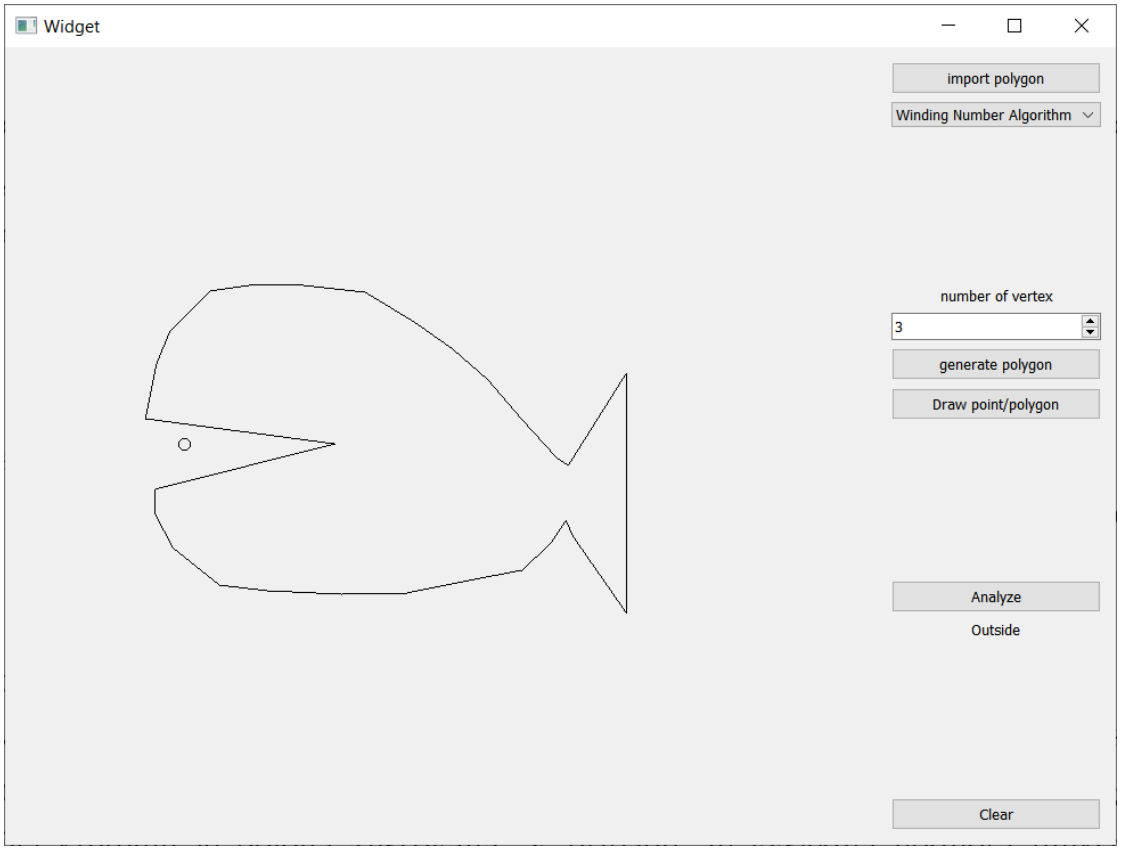
Tato kapitola nabízí přehled několika komentovaných screenshotů chodu aplikace.



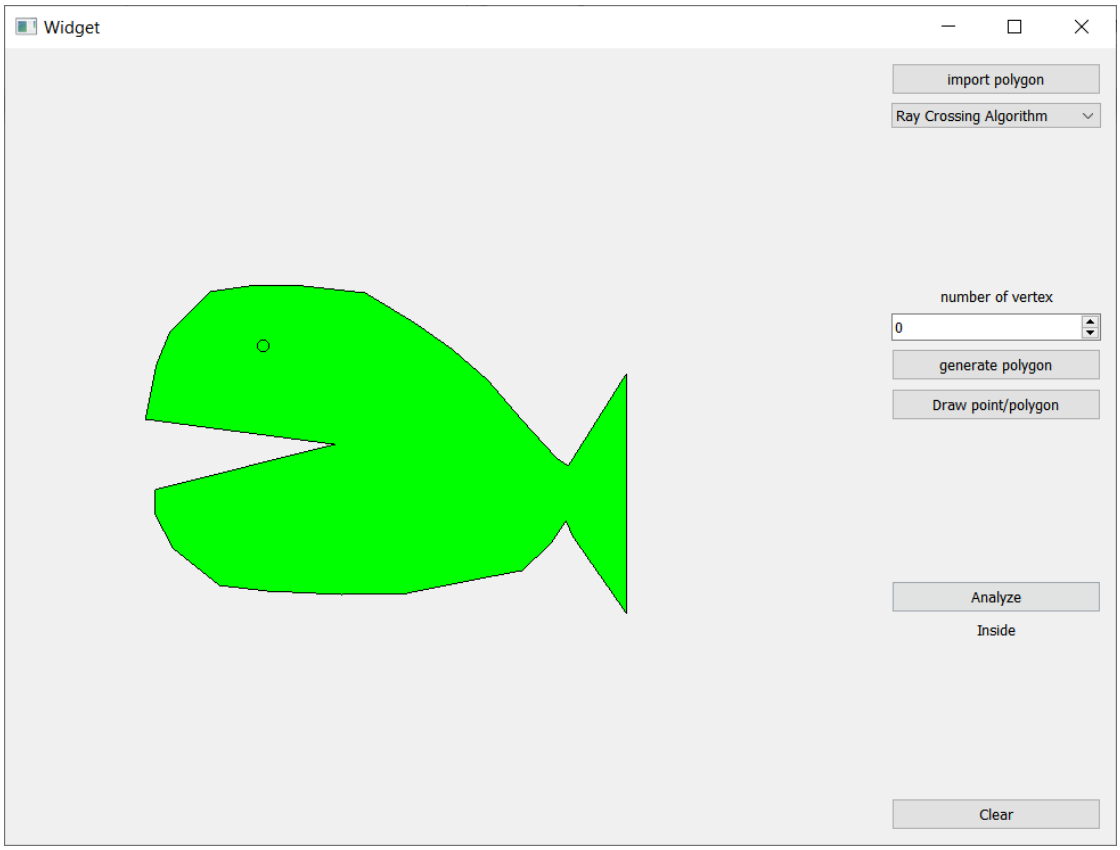
Metoda *Winding Number Algorithm* odhalila bod na hraně importované mapy polygonů, zabarveny byly polygony sdílející dotyčnou hranu.



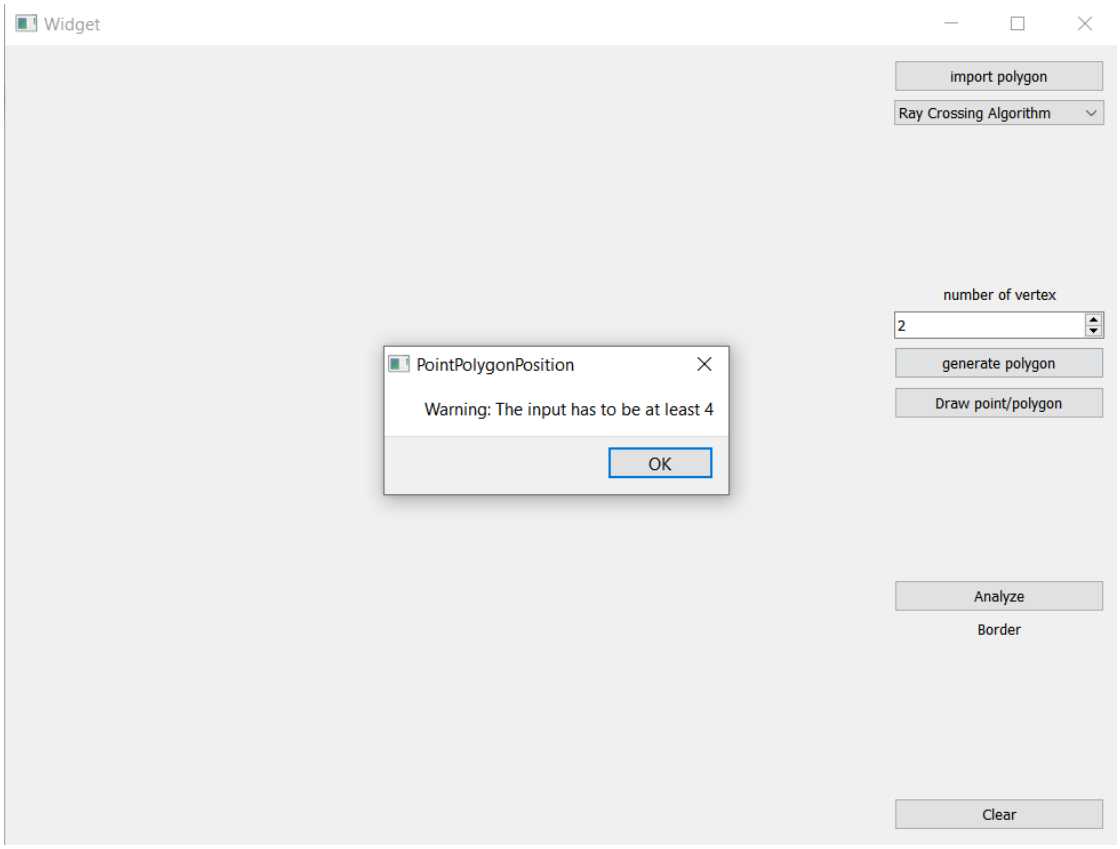
Metoda *Ray Crossing Algorithm* detekovala zadaný bod přímo v jednom z vrcholů náhodně vygenerovaného polygonu s osmi vrcholy.



Pomocí metody *Winding Number Algorithm* byl bod nalezen vně nakresleného polygonu.



Metoda *Ray Crossing Algorithm* odhalila zadaný bod uvnitř nakresleného polygonu.



Chybová hláška po zadání nedostatečného počtu vrcholů generovaného polygonu.

7. Technická dokumentace

Aplikace obsahuje tři třídy: Algorithms, Draw a Widget, níže následuje jejich detailnější rozbor.

7.1. Třída Algorithms

Mimo konstruktor zahrnuje třída Algorithms čtyři metody.

int getPointLinePosition(QPoint q, QPoint p1, QPoint p2)

Na začátku si metoda spočítá vzdálenosti *p1q, p2q a p1p2*. Poté zkoumá, zda bod *q* neleží na hraně (nebo v její těsné blízkosti) testováním podmínky, zda absolutní hodnota rozdílu *p1p2 – p1q+p2q* je menší nebo rovna zvolené toleranci. Dalším krokem je zjištění, zda bod *q* není totožný s jedním z vrcholů polygonu, pročež jsou zkoumány absolutní hodnoty rozdílů souřadnic X a Y bodů *q* a *p1*, resp. *p2*. Pokud splňují toleranci, prohlásíme bod *q* totožný s bodem p_i. Dále tato metoda počítá vektory *qp1, p1p2*, z kterých je pak přes determinant určeno, zda bod *q* leží v pravé či levé polorovině.

Metoda tedy vrací 4 možné výsledky: 0 (bod leží v pravé polorovině), 1 (bod leží v levé polorovině), 2 (bod leží na hraně), 3 (bod splývá s vrcholem).

double getAngle(QPoint p1, QPoint p2, QPoint p3, QPoint p4)

Tato funkce počítá úhel mezi dvěma hranami určenými čtyřmi body na vstupu dle známého vzorce:

$$\cos\omega = \frac{\vec{u}\cdot\vec{v}}{|\vec{u}|\cdot|\vec{v}|}$$

int getPositionWinding(QPoint q, std::vector<QPoint> pol)

Funkce určující polohu bodu vůči polygonu metodou *Winding Number Algorithm*, na vstupu je bod *q* a polygon. Na začátku je nastaven tolerance 1e-6, algoritmus funguje na principu popsaném v podkapitole 2.1., prochází tedy postupně úhly, které přičítá či odečítá do tzv. *Winding Number* a podle výsledku vrací buď hodnotu 0 (bod vně polygonu) či 1 (bod uvnitř polygonu). Funkce také ošetřuje singulární případy aneb využívá výstup z funkce *getPointLinePosition* (v takových případech vrací hodnotu 2 či 3).

int getPositionRay(QPoint q, std::vector<QPoint> pol)

Funkce určující polohu bodu vůči polygonu metodou *Ray Crossing Algorithm*, na vstupu je bod *q* a polygon. Na začátku jsou vypočteny redukované souřadnice posunutého lokálního souřadnicového systému a opět na principech popsaných v podkapitole 2.2. je vypočten počet průsečíků paprsku (polopřímky z bodu *q*) s polygonem, na jehož základě funkce vrátí 0 či 1. Opět byly pomocí funkce *getPointLinePosition* ošetřeny singulární případy jako v předchozí funkci (s návratem hodnot 2 či 3).

7.2. Třída Draw

void mousePressEvent

V závislosti na *draw_mode* je touto metodou vykreslen buď bod *q* nebo polygon.

void paintEvent

Touto metodou je vykreslen bod *q* a polygon (importovaný, zadaný ručně či generovaný).

void changeMode

Obsluhuje tlačítko Draw point/polygon. Po stisknutí přepíná mezi kresbou bodu či polygonu (mění hodnotu *draw_mode*).

void setResult, void setRes

Metody ukládající výsledek řešení analýzy ze třídy Widget do nové proměnné.

QPoint getPolygon

Tato metoda vrací polygon třídy Draw.

QPolygon getPol

Tato metoda vrací vektor polygonů třídy Draw.

QPoint getPoint

Tato metoda vrací bod *q* třídy Draw.

loadFile

Načítá mapu polygonů z textového souboru.

7.3. Widget

void on_pushButton_clicked

Slouží ke změně kreslícího módu (bod/polygon).

void on_pushButton_2_clicked

Metoda zavolá vybraný algoritmus a vytiskne řešení.

void on_pushButton_3_clicked

Metoda vyčistí okno aplikace.

void on_pushButton_4_clicked

Otevře možnost pro import textového souboru obsahující polygonovou mapu.

void on_pushButton_5_clicked

Tato metoda vykreslí náhodné polygony, které generuje v závislosti na zadaném počtu vrcholů.

7.4. Přehled bonusových úloh

Singulární případ u Winding Number Algorithm: bod leží na hraně

Tento problém popsaný v kapitole 3 byl vyřešen, jak již bylo také nastíněno, pomocí testování absolutní hodnoty rozdílu vzdáleností *p1p2* a *(qp1, qp2)*, pokud byla hodnota menší než zvolená tolerance, program vyhodnotí bod *q* jako ležící na hraně polygonu. Implementováno do metody *getPointLinePosition*.

Singulární případ u Winding Number Algorithm: bod leží ve vrcholu

Opět problém popsaný v kapitole 3, řešeno pomocí testování absolutní hodnoty rozdílů souřadnic X a Y bodů *q* a *p1*, resp. *p2*. Pokud splňují toleranci, program vyhodnotí bod *q* jako totožný s bodem p_i.

Zvýraznění všech polygonů pro oba singulární případy

Poloha bodu *q* je určena pomocí cyklu vzhledem ke všem importovaným polygonům a pro každý z nich je vyhodnocena podmínka, zda bod leží na hraně nebo vrcholu, po jejímž splnění je příslušný polygon vybarven zeleně.

Generování polygonu

Aplikace požaduje nejprve zadat počet vrcholů pro generování polygonu, pokud je počet menší než 4, program vrátí chybovou hlášku. Pokud je zadán korektní počet vrcholů, program generuje náhodný polygon, který může být konvexní či nekonvexní, zároveň simple nebo non-simple. Pro generované souřadnice je zavedena náhodná proměnná srand(time(NULL)), do které vstupuje čas interních hodin počítače, takže polygony jsou opravdu vždy náhodné a jiné. Pokud by nebylo užito takového postupu, řada náhodných generovaných polygonů se stejným počtem vrcholů by byla při každém spuštění aplikace stejná.

8. Závěr, náměty na vylepšení

Byla vytvořena solidní aplikace, která nabízí možnost analyzovat polohu bodu vůči polygonu. Bod je zadáván ručně uživatelem, polygon je možné taktéž nakreslit, vygenerovat či případně importovat. Vylepšení programu se nabízí v sekci generování náhodného polygonu. Jelikož by bylo vhodné ošetřit, aby polygony byly podle zadání bonusové úlohy nekonvexní. Každopádně snaha byla a výsledkem je určitý mezistupeň, který je funkční.