

# Úloha 1 - Geometrické vyhledávání bodu

## Algoritmy počítačové kartografie

Tomáš Hřebec, Kateřina Obrazová

Praha 2022

## 1 Zadání úlohy

### 1.1 Povinná část

*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 Winding Number Algorithm pro geometrické vyhledání incidujícího polygonu obsahující 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.

### 1.2 Volitelná část

Analýza polohy bodu (uvnitř/vně) metodou Ray Algorithm.

Ošetření singulárního případu u Ray Algorithm: bod leží na hraně polygonu.

Ošetření singulárního případu u obou algoritmů: bod je totožný s vrcholem jednoho či více polygonů.

Zvýraznění všech polygonů pro oba výše uvedené singulární případy.

## 2 Popis problému

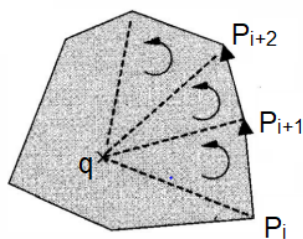
Je dán fixní polygon  $P$  s  $n$  počtem stran a bod  $q$ . Leží bod  $q$  v polygonu  $P$ ? Tento dotaz můžeme přirovnat ke kliknutí myši na obrazovku počítače. Avšak hardware stroje může umožnit řešení, která se vyhýbají geometrii. V našem případě uvažujeme problém z hlediska výpočetní geometrie. (O'Rourke 1998)

### 2.1 Poloha bodu vůči mnohoúhelníku (polygonu)

Používají se dva přístupy, kdy je možné otestovat, zda bod  $q$  je vnitřním bodem rovinného polygonu.

#### 2.1.1 Konvexní polygon

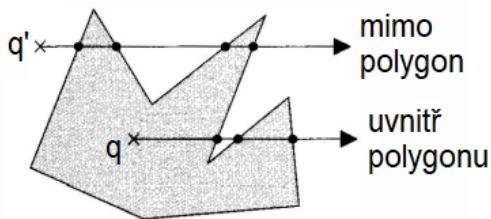
Konvexní polygon vzniká jako průnik polorovin definovaných shodně nalevo (napravo) od jeho orientovaných hran. „Zvolíme-li shodnou orientaci vlevo, pak vnitřní bod musí pro všechny hrany ležet v levých polorovinách vymezených hranami. (Obr. 1) Použijeme test velikosti orientovaného obsahu trojúhelníka  $P_i P_{i+1} q$ , který musí být pro všechny vrcholy  $P_i$  vždy  $\leq 0$ .“ (Žára a kol. 2004, str. 564)



Obr. 1 - Zjišťování polohy bodu vůči konvexnímu polygonu

#### 2.1.2 Nekonvexní polygon

Z testovaného bodu  $q$  libovolným směrem vyšleme polopřímku a počítáme, kolikrát protne hranici mnohoúhelníku. (Obr. 2) Je-li počet průtnutí lichý, bod  $q$  je uvnitř mnohoúhelníku (polygonu). Je-li počet průtnutí sudý, bod  $q$  leží vně mnohoúhelníku (polygonu). (Žára a kol. 2004)



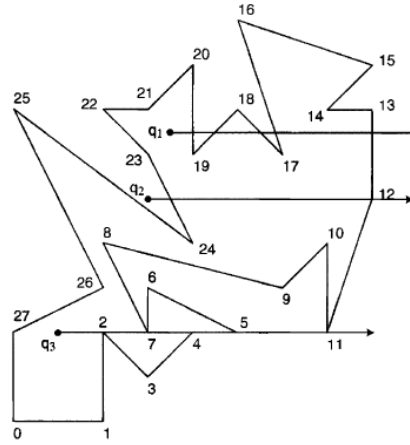
Obr. 2 - Zjišťování polohy bodu vůči nekonvexnímu polygonu

### 3 Popis algoritmu

Pro řešení konvexního polygonu se používá opakovaný Half-Plane Test a Ray Crossing algoritmus. Pro řešení nekonvexního polygonu se používají algoritmy Winding Number a Ray Crossing generalizovaný pro nekonvexní polygony.

#### 3.1 Ray Crossing Method

Algoritmus vede z bodu  $q$  polopřímku (paprsek, ray)  $R$  v libovolném směru. Dále je zjištěn počet průsečíků  $R$  s  $\partial P$ . Je-li počet prolnutí lichý, bod  $q \in P$ , pokud sudý, bod  $q \notin P$ . (Obr. 3)



Obr. 3 - Ray Crossing Algoritmus

Ke zjištění polohy bodu  $q_3$ , je potřeba upravit paprsek  $R$  tak, aby byl horizontálně doprava. „Jedním ze způsobů, jak odstranit většinu obtíží, je vyžadovat, aby se hrana mohla počítat jako křížení  $R$ , jeden z koncových bodů  $e$  musí být přísně nad  $R$  a druhý koncový bod na nebo pod. Neformálně se má za to, že  $e$  zahrnuje svůj spodní koncový bod, ale vylučuje svůj horní koncový bod. Při použití této konvence pro  $q_3$  (Obr. 3) se hrany  $(1, 2)$  a  $(2, 3)$  nekříží (žádná hrana nemá koncový bod striktně výše). Hrany  $(6, 7)$  a  $(7, 8)$  se počítají jako křížení ( $v_7$  je na nebo pod),  $(3, 4)$  a  $(4, 5)$  se nekříží a  $(5, 6)$ ,  $(10, 11)$ , a  $(11, 12)$  se všechny kříží. Celkový počet pěti křížení znamená, že  $q \in P$ . Všimněte si, že žádná hrana kolineární s paprskem se nepočítá jako křížení, protože nemá žádný bod přísně nahoře (O'Rourke 1998, s. 240).“

V rámci tohoto algoritmu mohou nastat situace, které je nutné v rámci algoritmu ošetřit:

- bod  $q$  je totožný s vrcholem,
- polopřímka  $R$  je kolineární s hranou tj. ležící na stejné přímce.

Ovšem tato varianta algoritmu neumí detekovat stav  $q \in \partial P$ . Je tedy nutné použít dva paprsky s opačnou orientací  $r_1$  (levostranný),  $r_2$  (pravostranný), jejichž počet průsečíků  $k_l$  a  $k_r$  je udržován. Pokud tedy  $q \in \partial P$ , tak počet levostranných a pravostranných průsečíků je různý.

Hlubší popis algoritmu je v následujícím pseudokódu:

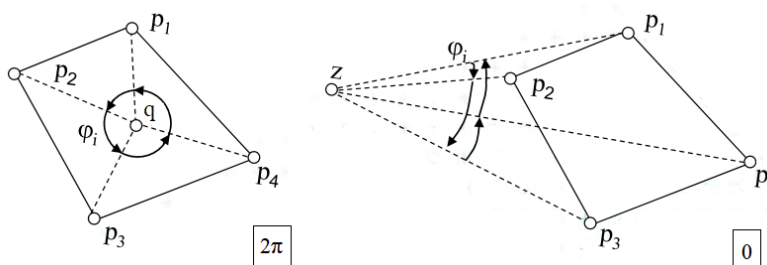
1. Inicializuj počet průsečíků v pravé polorovině na 0
2. Inicializuj počet průsečíků v levé polorovině na 0
3. Zjisti počet vrcholů polygonu  $n$
4. **Projdi** jednotlivé vrcholy  $p_i$  polygonu:
  - Souřadnice vrcholů  $p_i$  a  $p_{i-1}$  transformuj do lokálního systému s počátkem v  $q$
  - **Když** bod  $p_i$  leží v počátku:
    - Vrchol  $p_i$  je totožný s  $q$  a vrať, že  $q$  je uvnitř polygonu a ukonči výpočet
  - **Když** úsečka  $p_i$  a  $p_{i-1}$  protíná osu  $y$  z pohledu z horní poloroviny:
    - Vypočítej souřadnici  $x$  jejich průsečíku
    - **Je-li** tato souřadnice v pravé polorovině:
      - Přičti k počtu průsečíků v pravé polorovině 1
5. **Když** součty průsečíků vpravo a vlevo nejsou oba naráz sudé nebo oba liché:
  - Bod je na hraně polygonu
6. **Jinak když** je součet průsečíků vpravo lichý:
  - Bod je uvnitř polygonu
7. **Jinak:**
  - Bod je mimo polygon.

### 3.1.1 Singularity u algoritmu Ray Crossing

## 3.2 Winding Number algoritmus

V tomto algoritmu, jak již z názvu vypovídá, hraje roli winding number  $\omega$ . Definujeme ho jako počet, kolikrát se mnohoúhelník  $P$  otočí kolem bodu  $q$ . Je-li bod  $q$  uvnitř polygonu  $P$ , je  $\omega$  nenulové. To samé platí, pokud je bod  $q$  mimo polygon  $P$  a polygon se neotáčí kolem bodu. (Obr. 4, Kumar, Bangi 2018)

$$\omega = \begin{cases} 0; & \text{když } q \text{ je mimo } P, \\ n > 0; & \text{když se } P \text{ otáčí } n\text{-krát proti směru hod. ručiček,} \\ n < 0; & \text{když se } P \text{ otáčí } n\text{-krát ve směru hod. ručiček.} \end{cases} \quad (1)$$



Obr. 4 - Winding Number algoritmus

Hlubší popis metody je v následujícím pseudokódu:

1. Zjistí počet vrcholů polygonu  $n$
2. Inicializuj součet úhlů mezi definovaným bodem a hranou polygonu na 0
3. **Projdi** jednotlivé vrcholy  $p_i$  polygonu:
  - Zjistí polohu bodu  $q$  vůči přímce  $qp_i$
  - Vypočti úhel mezi přímkami  $qp_i$ ,  $qp_{i+1}$
  - **Když** je bod v levé polorovině,
    - Přičti úhel k celkovému součtu úhlů
  - **Když** je bod v pravé polorovině,
    - Odečti úhel od celkového součtu úhlů
  - **Jinak**,
    - Leží-li bod  $q$  mezi body  $p_i$  a  $p_{i+1}$ , bod se nachází na hraně a ukončí výpočet
12. **Když** je celkový součet úhlů roven  $2\pi$ :
  - Bod je uvnitř polygonu
14. **Jinak** je vně polygonu.

V rámci tohoto algoritmu může nastat situace, která je nutná v rámci algoritmu ošetřit:

- bod  $q$  je totožný s vrcholem.

Zda bod  $q$  je totožný s vrcholem zjistíme tak, že při výpočtu směrnice bude jedna ze složek nulová. Při porovnání Point-in-location a Winding Number zjišťujeme, že při jednoduchém mnohoúhelníku (tj. nemá žádné vlastní průsečíky) obě metody dávají stejný výsledek pro všechny body. Ovšem pro nejednoduché polygony dávají tyto dvě metody různé výsledky. „Například, když se mnohoúhelník překrývá sám se sebou, zjistí se, že body v oblasti překrytí jsou vně pomocí crossing number, kde pro Winding Number je znázorněno jako uvnitř“ (Kumar, Bangi 2018, s. 550). V porovnání s algoritmem Ray Crossing, je algoritmus Winding Number 20 krát pomalejší. (Felkel 2016)

## 4 Data

Vstupem jsou polygonové vrstvy ve formátu *.shp* a bod  $q$ . První polygonová oblast byla náhodně vybraná oblast polygonů obcí a pravidelná síť polygonů. Výstupem je grafické znázornění polygonů, které obsahují analyzovaný bod  $q$ . Bod  $q$  je bodem vstupu, který je vybrán na základě uživatele.

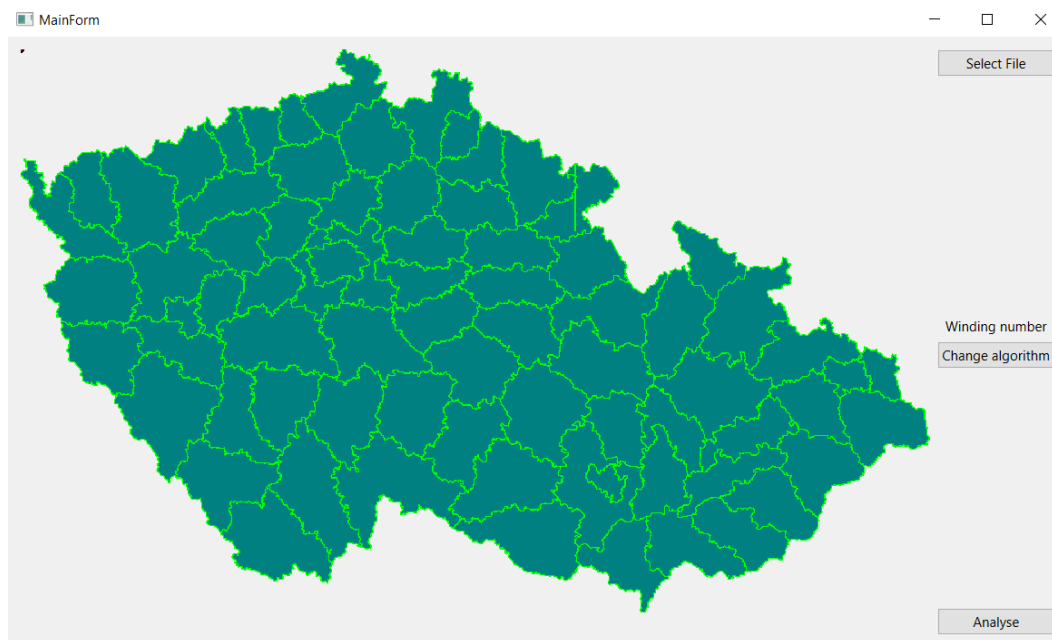
## 5 Dokumentace

Program je rozdělen do tří modulů. Modul *Mainform.py* je převážně vygenerován pomocí softwaru QT Creator a slouží k vytvoření uživatelského rozhraní. Modul *Algorithm.py* obsahuje algoritmy pro analýzu vzájemné polohy bodu a polygonů. Modul *Draw.py* slouží především pro propojení předešlých dvou modulů,

k zajištění vizualizace vstupu a výstupu a k volbě algoritmu.

## 5.1 Modul Mainform.py

Tento modul obsahuje třídu `Ui_Mainform`, která je vytvořena automaticky pomocí QT Creatoru (Obr. 5). Tato třída byla posléze doplněna o tři metody, které se spustí při interakci s uživatelským rozhraním. Metoda *SelectFile*, která při zmáčknutí stejnojmenného tlačítka, vyvolá dialogové okno, kde je možné zvolit formát *shapefile* se vstupními daty. Dále načte aktuální velikost okna, do kterého budou data vykresleny a zavolá funkci z modulu *Draw.py*. Po stisknutí tlačítka „Analyse“ se spustí stejnojmenná metoda, která rozhodne, který algoritmus pro analýzu polohy bodu a polygonu využije a následně jej spustí. Metoda *Change Algorithm* se zavolá při zmáčknutí tlačítka „Change Algorithm“ a slouží k výběru a k vizualizaci aktuálně zvoleného algoritmu.



Obr. 5 - Náhled uživatelského rozhraní

## 5.2 Modul Draw.py

Tento modul obsahuje třídu `Draw`, kterou dědí od třídy `QWidget` z knihovny PyQt6. Na začátku se inicializuje proměnná *q*, jako formát *QPoint*. To umožňuje do *q* ukládat souřadnice *x* a *y*. Do této proměnné se nahraje vstupní analyzovaný bod. Dále se inicializuje list *polygons*, který ponese jednotlivé vstupní polygony. Další proměnná *res* ve formátu *list* bude zaručovat uchování výsledků a proměnná *algorithm* nese informaci o aktuálně zvoleném výpočetním algoritmu.

První metoda v třídě `mousePressEvent` při stisknutí pravého tlačítka v uživatelském rozhraní v rozmezí vykreslovací plochy odečte aktuální souřadnice kurzoru *x* a *y* a uloží je pod *q*. Následně zavolá metodu pro vykreslení. Metoda vykreslení `paintEvent` definuje, jak se budou jednotlivé objekty třídy vykreslovat.

Vykreslí všechny polygony v proměnné *polygons*. U těchto polygonů je zvolena barva na základě výsledku, zda je hledaný bod uvnitř, či vně. Když výsledky ještě nejsou k dispozici (při prvním načtení polygonu), jsou všechny polygony stejné barvy. Dále jsou zde parametry pro vykreslení analyzovaného bodu.

V metodě *loadPolygons* je nejprve načtena cesta ke vstupním datům. Když není zvolena žádná cesta, metoda se ukončí. Dále je načten vstupní shapefile. Data z tohoto shapefile jsou převedeny do formátu *QPolygon*. Následně tyto data musejí být ještě musejí být zobrazena do lokálního souřadnicového systému, který je definován velikostí okna. Takto transformovaná data jsou poté jako jednotlivé polygony vloženy do listu *polygons*. Metoda *switchAlgorithm* zajišťuje přepínání mezi algoritmy. Aktuální algoritmus ukládá do proměnné *algorithm*. Poslední metody v této třídě jsou *getter*y, které vracejí list vstupních polygonů a bod *q*.

### 5.3 Modul Algorithm.py

Tento modul obsahuje dvě třídy. Každá z nich představuje jeden z výpočetních algoritmů - Winding Number nebo Ray Crossing. Třída *Winding\_num* tedy obsahuje metody k analýze vzájemné polohy bodu a polygonů pomocí metody Winding Number. První metoda této třídy - *getPointAndLinePosition* - zjišťuje, jestli se daný bod *q* nachází v pravé či levé polorovině od přímky definované dvěma body  $p_1$  a  $p_2$ . Když je v levé, metoda vrací 1, v pravé 0 a když je bod kolineární, vrací hodnotu -1. Metoda *get2LinesAngle* má na vstupu 4 body, které definují 2 přímky, u nichž je zjišťován úhel, který spolu svírají.

Hlavní metoda této třídy - *getPositionPointAndPolygon* - zjišťuje vzájemnou polohu daného bodu a polygonu. Metoda vrací 1 pro bod uvnitř polygonu, nebo pro bod na hraně a ve vrcholu. Pro bod vně polygonu vrací 0. Druhá třída - *Ray\_cross* - využívá k analýze algoritmus Ray Crossing. První metoda - *setLocalCoordinates* - převede souřadnice vrcholu polygonu tak, aby jejich počátek byl v analyzovaném bodu *q*. Vráť vrchol v lokálních souřadnicích s počátkem v bodu *q*.

Metody *getCrossingStatusU* a *getCrossingStatusL* jsou si velice podobné. Obě metody analyzují, zda úsečka definovaná body  $p_i$  a  $p_{i-1}$  protíná osu *y* z pohledu z horní poloroviny a dolní poloroviny. Metody vracejí hodnotu *True*, když dochází ke křížení a *False*, když nikoli.

Hlavní metoda této třídy se opět nazývá *getPositionPointAndPolygon* a zjišťuje vzájemnou polohu daného bodu a polygonu, nyní však využívá postupu algoritmu Ray Crossing.

## 6 Závěr

Program by měl umět pracovat se singularitami jako je bod na hraně nebo ve vrcholu polygonu. Program by bylo možné vylepšit a zrychlit tím, že při určení vztahu bod-polygon byl bod uvnitř, výpočet tím byl zastaven a následně výsledek vizualizován. Nyní vždy za každé situace program prohlíží všechny vstupní polygony. Dále by mohlo být rozšířeno uživatelské rozhraní o tlačítko *Clear*. Do programu by ještě mohla být přidána funkce, kdy se při zvětšení okna polygony automaticky překreslí na danou velikost. Nyní se polygony vykreslí do velikosti okna odečtené při jejich nahrávání.



## 7 Seznam literatury

FELKEL, P. (2016): Geometric Searching Part 1: Point Location. Computational Geometry. Praha, 45 s.

KUMAR, G. N., BANGI, M. (2018): An Extension to Winding Number and Point-in-Polygon Algorithm. IFAC, vol. 51, s. 548-553.

O'ROURKE, J. (1998): Computational Geometry in C. Second Edition. Cambridge University Press, 392 s.

ŽÁRA, J., BENEŠ, B., SOCHOR, J., FELKEL, P. (2004): Moderní počítačová grafika. Computer Press, Brno, 609 s.