Úloha 1 - Geometrické vyhledávání bodu

Algoritmy počítačové kartografie

Sabina Fukalová, Klára Linková

Praha 2023

1 Zadání úlohy

1.1 Povinná část

Vstup: Souvislá polygonová mapa n polygonů $\{P_1, ..., P_n\}$*, analyzovaný bod q.*

Výstup: P_i*,* $q \in P_i$*.*

Nad polygonovou mapou implementujete Ray 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 Winding Number Algorithm.

Ošetření singulárního případu u Winding Number 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 a rozbor problému

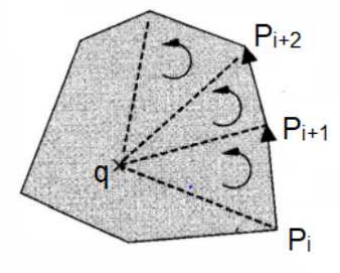
Zjišťování vzájemné polohy bodu a uzavřené konvexní či nekonvexní oblasti je často řešenou úlohou v oblasti digitální kartografie. Po kliknutí na bod v digitální mapě je potřeba vědět, uvnitř které oblasti se daný bod nachází, a s touto oblastí je poté možné provádět další operace jako je např. zvýraznění, výpočet plochy, obvodu nebo získání informací o polygonu (Bayer 2008).

Existují tři možnosti vzájemné polohy bodu q a uzavřené oblasti P s počtem hran n – bod q leží uvnitř uzavřené oblasti P, na hranici uzavřené oblasti P nebo vně uzavřené oblasti P. Pro zjištění, zda je daný bod vnitřním bodem rovinného polygonu se používají dva přístupy dle toho, jestli se jedná o konvexní nebo nekonvexní polygon.

2.1 Konvexní polygon

Pro konvexní polygon platí, že vznikne jako průnik polorovin definovaných shodně nalevo (v případě CCW orientace, tj. proti směru hodinových ručiček) od jeho orientovaných hran. Vnitřní bod tedy musí ležet v levých polorovinách vzhledem ke všem hranám (obrázek 1). Žára a kol. (2004, str. 564) uvádí *„Použijeme test velikosti orientovaného obsahu trojúhelníka P_iP_{i+1}, který musí být pro všechny vrcholy P_i vždy <= 0.“*

Obrázek 1 - Zjišťování polohy bodu vůči konvexnímu polygonu

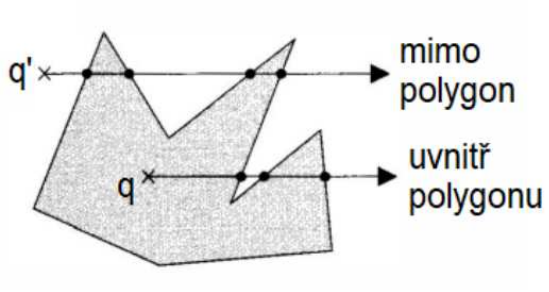


Zdroj: Žára a kol. (2004)

2.2 Nekonvexní polygon

Pro nekonvexní polygony se používá postup, při němž je z testovaného bodu vyslána polopřímka libovolným směrem a zjišťuje se počet průsečíků tohoto paprsku s hranicí polygonu. Pokud je počet průsečíků liché číslo, bod leží uvnitř polygonu (obrázek 2). V případě, že polopřímka protne hranici v sudém počtu případů, bod leží vně polygonu.

Obrázek 2 - Zjišťování polohy bodu vůči nekonvexnímu polygonu



Zdroj: Žára a kol. (2004)

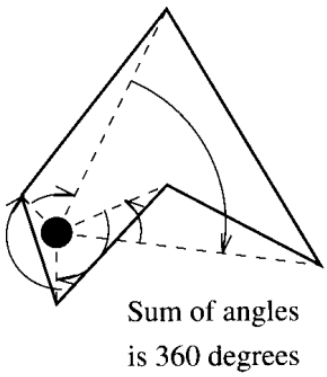
3 Popis algoritmu

Pro řešení konvexního polygonu se používá opakovaný Half Plane Test a Ray Crossing algoritmus. K řešení nekonvexních polygonů se používá algoritmus Winding Number a Ray Crossing upravený pro nekonvexní polygony.

3.1 Winding Number algoritmus

Algoritmus Winding Number vychází z myšlenky, že pozorovatel stojí v bodě q, u něhož testujeme polohu vůči polygonu. Pokud by se chtěl pozorovatel postupně podívat do všech vrcholů a bod q by ležel uvnitř mnohoúhelníku, musel by se otočit o úhel 2π. V případě, že by bod ležel vně mnohoúhelníku, otočil by se pozorovatel o úhel menší než 2π. Spojnice q-P_i (průvodič) opíše v prvním případě úhel 2π (obrázek 3) a ve druhém případě úhel menší než 2π. Velikost výsledného úhlu je rovna součtu jednotlivých rotací q-P_i. Otáčí-li se průvodič ve směru hodinových ručiček, je hodnota rotace záporná. Pokud se průvodič otáčí ve směru hodinových ručiček, hodnota rotace ω je kladná (Bayer 2008). Hodnota Ω udává, kolikrát se mnohoúhelník P otočí kolem bodu q (jedná se o násobky 2π).

Obrázek 3 - Winding Number algoritmus v případě bodu uvnitř polygonu



Zdroj: Haines (1994)

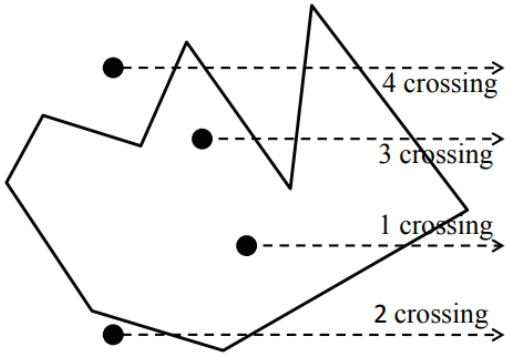
Bližší popis algoritmu

1. Zjisti počet vrcholů n polygonu
2. Inicializuj $\Omega = 0$, tolerance ϵ
3. **Projdi** jednotlivé vrcholy P_i polygonu:
    - Urči polohu bodu q vzhledem k přímce P_iP_{i+1}
    - Urči velikost úhlu ω_i mezi přímkami qP_i a qP_{i+1}
    - **Když** q leží v levé polorovině:
        - Přičti úhel k celkovému součtu úhlů
    - **Když** q leží 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, ukonči výpočet
4. **Když** je odchylka od 2π menší než tolerance ϵ:
    - Bod q je uvnitř polygonu
5. **Jinak** je bod q vně polygonu.

3.2 Ray Crossing algoritmus

Z bodu q je vedena polopřímka r (paprsek, ray) v libovolném směru. Tento algoritmus analyzuje polohu budu q vůči polygonu P na základě počtu průsečíků polopřímky s hranami polygonu. Je-li počet průsečíků lichý, bod q se nachází uvnitř polygonu P. Pokud je počet průsečíků sudý, popř. roven 0, bod q leží vně polygonu P (obrázek 4). Pokud je právě jeden průsečík totožný s testovacím bodem q, leží bod q na hraně polygonu P. Pro jednoduchost je vhodné použít polopřímku, která je vodorovná (Bayer 2008).

Obrázek 4 - Zjišťování polohy bodů vzhledem k polygonu za použití Ray Crossing algoritmu



Zdroj: Yan, Zhao, Ng (2012)

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

- bod q je totožný s vrcholem P_i – polopřímka r protíná dva segmenty (v koncovém bodě jedné hrany a počátečním bodě následující hrany), čímž hrozí detekce dvou průsečíků
- hrana P_iP_{i+1} je podmnožinou polopřímky r - není možné najít průsečíky polopřímky s hranou mnohoúhelníku (Bayer 2023)

Druhá situace se vyskytuje u pravoúhlých polygonů, jejichž hrany jsou rovnoběžné s osami x a y. Z tohoto důvodu se místo testovací polopřímky používá testovací úsečka rovnoběžná s osou x a koncovým bodem za hranicemi min-max boxu (Bayer 2008).

Pro ošetření nejednoznačného určení počtu průsečíků je možné provést modifikaci tohoto algoritmu tak, že z testovaného bodu q jsou na obě strany vedeny polopřímky rovnoběžné s osou x, které rozdělí oblast na dvě podoblasti O_1 a O_2. V jedné leží body P_i, které mají souřadnice y v intervalu $y_i < y_q$, a ve druhé leží body, které mají souřadnice y v intervalu $y_i \geq \mathrm{y}_q$ (Bayer 2008). Průsečík M hrany a polopřímky započítáváme, pokud se hrana nachází v obou polorovinách nebo jen v horní, nebo se hrana nachází v obou polorovinách nebo jen v dolní, tzn.

$$(y_{i-1} \leq \mathrm{y}_q \wedge y_i > y_q) \vee \ (y_{i-1} > y_q \wedge \mathrm{y}_i \leq \mathrm{y}_q)$$

Pro detekci bodu ležícího na hraně polygonu P je nutné použít dva paprsky s opačnou orientací r_1 (levostranný) a r_2 (pravostranný). Pokud bod q leží na hraně polygonu P, zbytek po dělení dvěma pro levostranné a pravostranné průsečíky je různý (Bayer 2023).

Bližší popis algoritmu

1. Inicializuj počet průsečíků v pravé polorovině na 0
2. Inicializuj počet průsečíků v levé polorovině na 0
3. Zjisti počet vrcholů n polygonu P
4. **Projdi** jednotlivé vrcholy P_i polygonu:
   - Transformuj souřadnici x vrcholů polygonu do nového souřadnicového systému s počátkem v bodě q podle vzorce $x'_i = x_i - x_q$
   - Transformuj souřadnici y vrcholů polygonu do nového souřadnicového systému s počátkem v bodě q podle vzorce $y'_i = y_i - y_q$
   - **Když** bod P_i leží v počátku:
     - Vrchol P_i je totožný s q; vrať, že q je uvnitř polygonu P a ukonči výpočet
   - **Když** počáteční a koncový bod segmentu leží v opačných polorovinách, nebo jeden v horní polorovině a druhý na přímce:
     - Segment je vhodný, spočítej souřadnici x průsečíku
     - **Když** souřadnice x průsečíku je kladná
       - Jedná se o vhodný průsečík, přičti k počtu průsečíků v pravé polorovině hodnotu 1
5. **Když** součty průsečíků v pravé a levé polorovině nejsou oba sudé nebo liché:
   - Bod je na hraně polygonu
6. **Jinak když** je součet průsečíků vpravo lichý:
   - Bod je uvnitř polygonu
7. **Jinak**
   - Bod leží mimo polygon

4 Data

Vstupem jsou polygonové vrstvy ve formátu *.shp* a bod q. Polygonové vrstvy znázorňují městské obvody a městské části Prahy a Brna. Výstupem je grafické znázornění polygonů, které obsahují analyzovaný bod q. Bod q je zadaný uživatelem.

5 Dokumentace

Program je rozdělen do tří do sebe propojených Python kódů – Algorithm, Draw a Mainform.

5.1 Algorithm.py

Modul *Algorithm* obsahuje funkcionalitu samotných zadaných a použitých algoritmů – *Ray Crossing* a *Winding Number*. Kód tedy obsahuje dvě třídy, kdy každá z nich reprezentuje jednotlivou metodu.

Algoritmus Ray Crossing

Prvním krokem bylo definování první metody *setLocalCoordinates* (viz obrázek 5). Tato metoda převede souřadnice vybraného vrcholu polygonu tak, aby jejich hodnota odpovídala souřadnicím počátečního bodu q. To znamená, že jako výstup metoda vrací tzv. lokální souřadnice daného vrcholu na místě bodu q.

Obrázek 5 - Metoda *setLocalCoordinates*

```
class Ray_crossing:
    def __init__(self):
        pass

    def setLocalCoordinates(self, p: QPoint, q: QPoint):

        #Výpočet souřadnic bodu
        x = p.x() - q.x()
        y = p.y() - q.y()

        #Nové souřadnice bodu
        p_shifted = QPoint(x, y)

        return p_shifted
```

Následujícími definovanými metodami algoritmu jsou *getCrossingStatusU* a *getCrossingStatusL* (obrázek 6). Princip jejich funkcionality je stejný. Analyzují, zda přímka, jež je definovaná body P_i a P_{i-1}, protíná osu y z pohledu horní a dolní poloroviny. Pokud metody identifikují křížení přímky a osy y, vrací hodnotu True, pokud však žádné křížení neproběhne, metody vrací hodnotu False.

Obrázek 6 - Metoda *getCrossingStatusU* a *getCrossingStatusL*

```python
#Analýza křížení linií v horní poloroviné
def getCrossingStatusU(self, p1: QPoint, p2: QPoint):
    if (p1.y() > 0) != (p2.y() > 0):
        return True
    return False

#Analýza křížení linií ve spodní poloroviné
def getCrossingStatusL(self, p1: QPoint, p2: QPoint):
    if (p1.y() < 0) != (p2.y() < 0):
        return True
    return False
```

Metoda *getPositionPointAndPolygon* počítá vzájemnou polohu daného bodu q a polygonu (obrázek 7).

Obrázek 7 - Metoda *getPositionPointAndPolygon*

```python
#Počet křížení mezi Q-ray a hranami polygonu
def getPositionPointAndPolygon(self, q: QPoint, pol: QPolygon):
    k_r = 0
    k_l = 0
    n = len(pol)

    #Změna souřadnic bodů na lokální souřadnice
    for i in range(n):
        p1 = self.setLocalCoordinates(pol[(i + 1) % n], q)  # p[i]
        p2 = self.setLocalCoordinates(pol[i], q)  # p[i-1]

        #Pokud je bod na vrcholu
        epsilon = 1.0e-10
        if abs(0 - p1.x()) < epsilon and abs(0 - p1.y()) < epsilon:
            return 1

        #Křížení v pravé poloroviné
        if self.getCrossingStatusU(p1, p2):
            x_m = (p1.x() * p2.y() - p2.x() * p1.y()) / (p1.y() - p2.y())

            #Součet křížení
            if x_m > 0:
                k_r += 1

        #Křížení v levé poloroviné
        if self.getCrossingStatusL(p1, p2):
            x_m = (p1.x() * p2.y() - p2.x() * p1.y()) / (p1.y() - p2.y())

            #Součet křížení
            if x_m < 0:
                k_l += 1


    #Pokud je bod na hraně polygonu
    if k_l % 2 != k_r % 2:
        return 1

    #Pokud je bod uvnitř polygonu
    elif k_r % 2 == 1:
        return 1

    #Pokud je bod mimo polygon
    else:
        return 0
```

Winding Number algoritmus

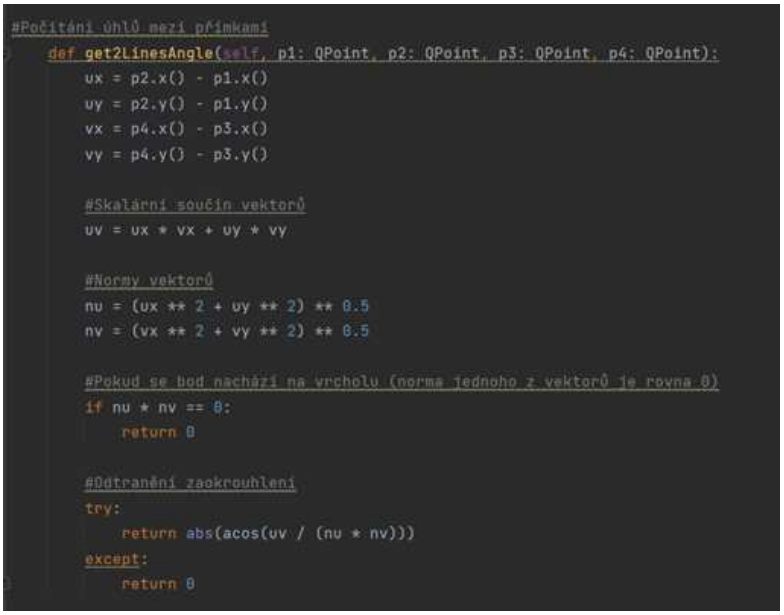
Jako první krok byla definovaná metoda *getPointAndLinePosition* (obrázek 8). Ta zjišťuje, jestli se zadaný bod Q nachází v levé či pravé polorovině od specifikované přímky, která je zadaná jako vzdálenost mezi body p1 a p2. Pokud je bod definován právě v levé polorovině, metoda vrací hodnotu 1, v pravé zase vrací hodnotu 0, a pokud je bod vyhodnocen na kolineární, tedy že leží právě na téže přímce, metoda vrací hodnotu -1.

Obrázek 8 - Metoda *getPointAndLinePosition*

```
class Winding_number:
    def __init__(self):
        pass

    def getPointAndLinePosition(self, a: QPoint, p1: QPoint, p2: QPoint):
        #Analýza bodu a přímky
        eps = 1.0e-10

        #Souřadnice
        ux = p2.x() - p1.x()
        uy = p2.y() - p1.y()
        vx = a.x() - p1.x()
        vy = a.y() - p1.y()

        #Determinant
        t = ux * vy - vx * uy

        #Pokud se bod nachází v levé polorovině
        if t > eps:
            return 1

        #Pokud se bod nachází v pravé polorovině
        if t < -eps:
            return 0

        #Kolineární bod
        return -1
```

Další definovanou metodou je *get2LinesAngle* (obrázek 9), jež zjišťuje úhel mezi dvěma danými přímkami, které jsou definované celkem čtyřmi body. Metoda je počítána jako vektorový součin přímek.

Obrázek 9 - Metoda *get2LinesAngle*



```
#Počítání úhlů mezi přímkami
    def get2LinesAngle(self, p1: QPoint, p2: QPoint, p3: QPoint, p4: QPoint):
        ux = p2.x() - p1.x()
        uy = p2.y() - p1.y()
        vx = p4.x() - p3.x()
        vy = p4.y() - p3.y()

        #Skalární součin vektorů
        uv = ux * vx + uy * vy

        #Normy vektorů
        nu = (ux ** 2 + uy ** 2) ** 0.5
        nv = (vx ** 2 + vy ** 2) ** 0.5

        #Pokud se bod nachází na vrcholu (norma jednoho z vektorů je rovna 0)
        if nu * nv == 0:
            return 0

        #Odtranění zaokrouhlení
        try:
            return abs(acos(uv / (nu * nv)))
        except:
            return 0
```

Poslední definovanou metodou tohoto algoritmu je *getPositionPointAndPolygon*. Ta zjišťuje vzájemnou polohu bodu a polygonu. Metoda vrátí hodnotu 1, pokud se bod nachází uvnitř tohoto polygonu, či na jeho hraně nebo jeho vrcholu. Pokud se bod nachází mimo specifikovaný polygon, metoda vrací hodnotu 0 (obrázek 10).

Obrázek 10 - Metoda *getPositionPointAndPolygon*

```
#Analyza pozice bodu vůči polygonu
def getPositionPointAndPolygon(self, q: QPoint, pol: QPolygon) -> int:

    n = len(pol)
    omega_sum = 0

    #Smyčka v uzlech
    for i in range(n):

        #Pozice q, pi, pi+1
        pos = self.getPointAndLinePosition(q, pol[i], pol[(i + 1) % n])

        #Uhel q, pi, pi+1
        omega = self.get2LinesAngle(q, pol[i], q, pol[(i + 1) % n])


        #Binding number
        if pos == 1:

            #Pro bod v levé polorovině
            omega_sum += omega

        elif pos == 0:
            #Pro bod v pravé polorovině
            omega_sum -= omega

        else:
            #Bod je kolineární
            x_check = (q.x() - pol[i].x()) * (q.x() - pol[(i + 1) % n].x())
            y_check = (q.y() - pol[i].y()) * (q.y() - pol[(i + 1) % n].y())

            if x_check <= 0 and y_check <= 0:
                return 1


    #Počáteční bod uvnitř polygonu
    epsilon = 1.0e-10

    if abs(abs(omega_sum) - 2 * pi) < epsilon:
        return 1

    #Počáteční bod mimo polygon
    return 0
```

5.2 Draw.py

Tento kód obsahuje hlavní třídu Draw, kterou dědí z třídy OWidget. Na začátku je inicializovaná proměnná q jako formát QPoint, který umožňuje do definované proměnné ukládat kartézské souřadnice. Do této proměnné jsou ukládány všechny počáteční analyzované body. Je vytvořen také seznam *polygons*, do kterého se ukládají všechny vstupní polygony. Do dále vytvořeného seznamu res se ukládají všechny mezivýsledky. Definovaná třída *mousePressEvent* v uživatelském rozhraní v rámci kreslící plochy uloží po kliknutí aktuální souřadnice kurzoru jako souřadnice x a y do bodu q. Ten se vykreslovací metodou *pointEvent* vykreslí. Je-li těchto bodů více vykreslí se polygon. Polygony jsou vybarveny podle toho, zdali se hledaný bod q nachází uvnitř polygonu či nikoliv.

Pomocí třídy *loadFile* je načten do grafického rozhraní vybraný shapefile. Jeho data pro komptabilitu jsou převedena do formátu QPolygon a do lokálního souřadnicového systému. Převedené polygony jsou uloženy do seznamu *polygons*.

5.3 Mainform.py

Modul Mainform je převážně vytvořen jako hlavní kód v Qt Creator. Kromě hlavní třídy Mainform byl kód doplněn o další definované třídy, které zaštiťují interakci s grafickým rozhraním. Třída selectfile vyvolá okno, ve kterém je možné vybrat daný shapefile, který obsahuje data k analýze. Třída *analyze* pracuje s výběrem jednotlivých definovaných algoritmů k analýze vzájemné polohy bodu a vybraných polygonů.

6 Závěr

Program by měl zvládnout pracovat se singularitami jako je bod na hraně nebo ve vrcholu polygonu. Je však limitován určeným uživatelským rozhraním z důvodu chybějící zkušenosti s tvorbou uživatelského rozhraní aplikací. Uživatelské rozhraní by mohlo být doplněno dalšími nástroji jako např. *Clear*.

7 Použité zdroje

BAYER, T. (2008): Algoritmy v digitální kartografii. Univerzita Karlova v Praze, Nakladatelství Karolinum, Praha, 251 s.

BAYER, T. (2023): Point Location Problem. Výukový materiál. Dostupné z: http://web.natur.cuni.cz/ bayertom/images/courses/Adk/adk3.pdf (cit. 11.3.2023).

HAINES, E. (1994): Point in Polygon Strategies. In: HECKBERT, P. S.: Graphics Gems IV. Academic Press, Computer Science Department, Carnegie Mellon University, Pittsburgh, Pennsylvania. 24-46.

YAN, D., ZHAO, Z., NG, W. K. (2012): Monochromatic and Bichromatic Reverse Nearest Neighbor Queries on Land Surfaces. Conference paper. The Hong Kong University of Science and Technology Clear Water Bay, Kowloon, Hong Kong, 11 s.

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