ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE FAKULTA STAVEBNÍ KATEDRA GEOMATIKY					
Název předmětu: **Algoritmy digitální kartografie a GIS**					
Úloha: U1	**Název úlohy:** Geometrické vyhledávání bodu				
Akademický rok: 2024/2025	**Semestr:** letní	**Studijní skupina:** 101C	**Vypracoval:** Michal Kovář Filip Roučka	**Datum:** 15. 3. 2025	**Klasifikace:**

1 Zadání

Implementujte Ray Crossing Algorithm pro geometrické vyhledávání incidujícího polygonu obsahujícího zadaný bod q nad souvislou polygonovou mapou složenou z n polygonů $P_1, ..., P_n$.

- Implementujte algoritmus Ray Crossing pro určení polohy bodu q vzhledem k polygonu P_i.
- 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 nebo použít existující geografická data (např. mapa evropských států).
- Polygony budou načítány z textového souboru ve zvoleném formátu.
- Pro datovou reprezentaci jednotlivých polygonů použijte špagetový model.

2 Bonus

- *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 Ray Crossing 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.*
- *Rychlé vyhledání potenciálních polygonů (bod uvnitř min-max boxu).*
- *Řešení pro polygony s dírami.*
- *Načtení vstupních dat ze *.shp.*

3 Popis problému

Problém lokalizace bodu v polygonové mapě spočívá v určení toho, ve kterém polygonu se nachází daný bod. Tento úkol je klíčový v geografických informačních systémech (GIS) a je známý jako *Point Location Problem*[1].

3.1 Formulace problému

Dáno:

- Množina n bodů $\{p_i\}$, které tvoří vrcholy m polygonů $\{P_j\}$.
- Bod q, jehož polohu vůči polygonům chceme určit.

Určováno:

- Polygon P, který obsahuje bod q, nebo informace, že bod q neleží v žádném polygonu.

3.2 Techniky řešení problému

- **Half-plane test** - Spočívá v porovnání polohy bodu vůči každé hraně polygonu. Nelze jej využít pro nekonvexní polygony, kde mohou existovat hrany obklopující bod z různých stran. V praxi se tento problém řeší rozdělením nekonvexního polygonu na několik konvexních částí, na které lze test aplikovat jednotlivě.
- **Ray Crossing Algorithm** – Sleduje počet průsečíků polopřímky vycházející z bodu q s hranami polygonu. Pokud je počet průsečíků lichý, bod leží uvnitř polygonu; pokud je sudý, bod se nachází vně.
- **Winding Number Algorithm** – Vyhodnocuje součet úhlových změn mezi hranami polygonu a bodem. Pokud je výsledný součet nenulový, bod leží uvnitř polygonu.

4 Popis algoritmů

4.1 Ray Crossing Method

Bodem q vedena polopřímka r (tzv. ray):

$$r(q) : y = y_q. \tag{1}$$

Počet průsečíků paprsku r s oblastí P určuje příslušnost bodu q:

$$k \mod 2 = \begin{cases} 1, & q \in P, \\ 0, & q \notin P. \end{cases} \tag{2}$$

4.1.1 Vlastnosti

- Rychlejší než Winding Number Algorithm.
- Problém singularity: q na hranici polygonu.
- Špatné přiřazení bodu p_i do oblasti P, pokud je hrana příliš krátká.

Metoda `Ray_Crossing_Algorithm`

Vstup: bod q, polygon P
Výstup: číslo udávající polohu bodu vzhledem k P (0 - vně, 1 - uvnitř, 2 - na hraně, 3 - ve vrcholu)
$k \leftarrow 0$ {Počet průsečíků}
$n \leftarrow$ počet vrcholů polygonu P
for $i \leftarrow 0$ **to** $n - 1$ **do**
  $ii \leftarrow (i + 1) \mod n$
  $x'_i \leftarrow P[i].x - q.x$
  $y'_i \leftarrow P[i].y - q.y$
  $x'_{ii} \leftarrow P[ii].x - q.x$
  $y'_{ii} \leftarrow P[ii].y - q.y$
  {Kontrola zvláštních případů}
  if $(|x'_i| \leq \varepsilon \wedge |y'_i| \leq \varepsilon) \vee (|x'_{ii}| \leq \varepsilon \wedge |y'_{ii}| \leq \varepsilon)$ **then**
    return 3 {Bod leží ve vrcholu}
  end if
  $d_{i,ii} \leftarrow \sqrt{(P[i].x - P[ii].x)^2 + (P[i].y - P[ii].y)^2}$
  $d_{i,q} \leftarrow \sqrt{(P[i].x - q.x)^2 + (P[i].y - q.y)^2}$
  $d_{ii,q} \leftarrow \sqrt{(P[ii].x - q.x)^2 + (P[ii].y - q.y)^2}$
  if $|d_{i,q} + d_{ii,q} - d_{i,ii}| \leq \varepsilon$ **then**
    return 2 {Bod leží na hraně}
  end if
  {Ray Crossing Test}
  if $(y'_{ii} > 0 \wedge y'_i \leq 0) \vee (y'_i > 0 \wedge y'_{ii} \leq 0)$ **then**
    $x'_m \leftarrow \frac{x'_{ii} y'_i - x'_i y'_{ii}}{y'_{ii} - y'_i}$
    if $x'_m > 0$ **then**
      $k \leftarrow k + 1$
    end if
  end if
end for
if $k \mod 2 = 0$ **then**
  return 0 {Bod leží vně polygonu}
else
  return 1 {Bod leží uvnitř polygonu}
end if

4.2 Výpočet Winding Number Ω

Suma Ω všech rotací ω_i (měřená CCW) je dána vztahem:

$$\Omega(q, P) = \frac{1}{2\pi} \sum_{i=1}^{n} \omega(p_i, q, p_{i+1}), \tag{3}$$

kde $\omega(p_i, q, p_{i+1})$ je orientovaný úhel mezi sousedními vrcholy mnohoúhelníku vzhledem k bodu q.

4.2.1 Vlastnosti

- Lepší ošetření singulárních případů než u paprskového algoritmu.
- Pomalejší než paprskový algoritmus.
- Problematický případ $q \equiv p_i$.
- Nutnost předzpracování $O(N)$.

Metoda `Winding_Number_Algorithm`

Vstup: bod q, polygon P
Výstup: číslo udávající polohu bodu vzhledem k P (0 - vně, 1 - uvnitř, 2 - na hraně, 3 - ve vrcholu)
$\omega \leftarrow 0$ {Inicializace součtu úhlů (winding number)}
$n \leftarrow$ počet vrcholů polygonu P
for $i \leftarrow 0$ **to** $n - 1$ **do**
  $ii \leftarrow (i + 1) \mod n$
  $p_i \leftarrow P[i]$, $p_{ii} \leftarrow P[ii]$
  $x'_i \leftarrow p_i.x - q.x$, $y'_i \leftarrow p_i.y - q.y$
  $x'_{ii} \leftarrow p_{ii}.x - q.x$, $y'_{ii} \leftarrow p_{ii}.y - q.y$
  {Kontrola zvláštních případů}
  if $(|x'_i| \leq \varepsilon \wedge |y'_i| \leq \varepsilon) \vee (|x'_{ii}| \leq \varepsilon \wedge |y'_{ii}| \leq \varepsilon)$ **then**
    return 3 {Bod leží ve vrcholu}
  end if
  $d_{i,ii} \leftarrow \sqrt{(p_i.x - p_{ii}.x)^2 + (p_i.y - p_{ii}.y)^2}$
  $d_{i,q} \leftarrow \sqrt{(p_i.x - q.x)^2 + (p_i.y - q.y)^2}$
  $d_{ii,q} \leftarrow \sqrt{(p_{ii}.x - q.x)^2 + (p_{ii}.y - q.y)^2}$
  if $|d_{i,q} + d_{ii,q} - d_{i,ii}| \leq \varepsilon$ **then**
    return 2 {Bod leží na hraně}
  end if
  {Výpočet úhlu a jeho příspěvku k winding number}
  $cos\gamma \leftarrow \frac{d_{i,q}^2 + d_{ii,q}^2 - d_{i,ii}^2}{2 \cdot d_{i,q} \cdot d_{ii,q}}$
  $\theta \leftarrow \arccos(cos\gamma)$
  {Výpočet orientace pomocí determinantů}
  $\det \leftarrow (p_{ii}.x - p_i.x) \cdot (q.y - p_i.y) - (p_{ii}.y - p_i.y) \cdot (q.x - p_i.x)$
  if $\det > 0$ **then**
    $\omega \leftarrow \omega + \theta$ {Bod je v levé polorovině}
  else if $\det < 0$ **then**
    $\omega \leftarrow \omega - \theta$ {Bod je v pravé polorovině}
  end if
end for
{Rozhodnutí na základě součtu úhlů}
if $|\omega| \geq 2\pi - \varepsilon$ **then**
  return 1 {Bod leží uvnitř polygonu}
else
  return 0 {Bod leží vně polygonu}
end if

4.3 Rychlé vyhledávání potenciálních polygonů pomocí min-max boxu

Bod se nachází v oblasti, kterou tvoří extrémy vstupního polygonu. Kdy pro bod musí platit:

$$\begin{aligned} x_q \geq x_{P\,\min} \wedge x_q \leq x_{P\,\max} \\ y_q \geq y_{P\,\min} \wedge y_q \leq y_{P\,\max} \end{aligned} \tag{4}$$

Používá se společně s algoritmem Winding Number nebo metodou Ray Crossing. Pokud bod leží v min-max boxu daného polygonu, je následně ověřen pomocí algoritmu Winding Number nebo metody Ray Crossing.

Metoda `Min_Max_Box`

1: **Vstup:** bod q, polygon P
2: **Výstup:** `true`, pokud bod leží uvnitř min-max boxu, jinak `false`
{Inicializace extrémních hodnot}
3: $x_{\min} \leftarrow \min(x_{\min}, P.x)$
4: $x_{\max} \leftarrow \max(x_{\max}, P.x)$
5: $y_{\min} \leftarrow \min(y_{\min}, P.y)$
6: $y_{\max} \leftarrow \max(y_{\max}, P.y)$
{Získání souřadnic bodu q}
7: $x_q \leftarrow q.x$
8: $y_q \leftarrow q.y$
{Test, zda bod leží uvnitř min-max boxu}
9: **if** $x_q \geq x_{\min} \wedge x_q \leq x_{\max} \wedge y_q \geq y_{\min} \wedge y_q \leq y_{\max}$ **then**
10: **return** `true` {Bod leží uvnitř min-max boxu}
11: **else**
12: **return** `false` {Bod leží mimo min-max box}
13: **end if**

4.3.1 Vlastnosti

- Velmi rychlé vyhledávání.
- Nedokáže univerzálně odhalit zda je bod uvnitř polygonu.
- Doplňková metoda pro urychlení vyhledávání.

5 Problematické situace

5.1 Polygon s dírami

Pro práci s polygonem, který může obsahovat díry, je nutné uchovávat údaje o vnější straně polygonu, která je jediná, a o dírách, kterých může být více. Je tedy potřeba zajistit správné přiřazení děr ke každému polygonu. Při vykreslování je nezbytné odlišit vykreslení oblasti, která znázorňuje díru, od zbytku polygonu. Tento problém byl rozdělen na dvě části: jednu, která analyzuje polygon, a druhou, která se zabývá jeho vykreslováním.

5.1.1 Vytvoření polygonu s dírami

Body polygonu, nebo díry daného polygonu jsou ukládány do pomocných proměnných typu `QPolygonF`. Ty jsou následně ukládány do korespondujících proměnných.

- Pro analýzu polygonu s dírami byla vytvořena struktura `Polygon_`, která se skládá z `outer` a `holes`. `Outer` představuje jeden polygon typu `QPolygonF`, zatímco `holes` je vektor typu `std::vector<QPolygonF>`, který obsahuje všechny díry. Po vytvoření polygonu nebo jeho díry je objekt přidán do pomocné proměnné typu `Polygon_`. Po dokončení jednoho polygonu a zahájení nového, nebo při kreslení bodu, je aktuální polygon přidán do `std::vector<Polygon_>`, který uchovává všechny polygony.
- Pro vykreslování polygonu je polygon a díry ukládán do pomocné proměnné typu `QPainterPath`. Po dokončení jednoho polygonu a zahájení nového, nebo při kreslení bodu, je aktuální polygon přidán do `std::vector<QPainterPath>`, který uchovává všechny polygony pro vykreslování.

Pro správné manuální vkládání polygonu a jeho děr, je potřeba rozlišit vstup pro díry a pro vnější polygon. Pro tvorbu vnější strany polygonu bylo přiřazeno levé tlačítko myši a pro tvorbu díry bylo přiřazeno pravé tlačítko myši. Je potřeba proto stanovit podmínky pro správné vytvoření polygonu s dírami, které nesmí být porušeny.

- Nesmí být jako první vytvářena díra.
- Nesmí se vytvářet díra do nedokončeného polygonu.
- Po dokončení děr, musí být polygon uložen do vektoru všech polygonů.

5.1.2 Atributy a metody pro vytváření polygonu s dírami

Atributy:

- `QPolygonF currentPolygon` - aktuální polygon.
- `QPolygonF currentHole` - aktuální díra.
- `Polygon_ curentCPolygon` - aktuální polygon komplexní struktury.
- `QPainterPath curentPolygonWH` - aktuální polygon pro vykreslení.
- `std::vector<QPainterPath> polygonsWH` - vektor komplexní struktury polygonů.
- `std::vector<Polygon_> polygonComplex` - vektor polygonů pro vykreslení.
- `bool isPolygonReady = false` - je polygon připravený.

Metody:

- `void mousePressEvent(QMouseEvent *e)` - obsluha událostí myši pro kreslení polygonů.
- `void mousePressEventLeft(QMouseEvent *e)` - zpracování levého tlačítka myši.
- `void mousePressEventRight(QMouseEvent *e)` - zpracování pravého tlačítka myši.

Obsluha událostí myši pro kreslení polygonů

```
1:  Vstup: událost myši e
    {Zpracování levého tlačítka myši}
2:  if e.button = LeftButton then
3:    if currentPolygon je prázdný a isPolygonReady = true then
4:      polygonComplex.push_back(curentCPolygon)
5:      curentCPolygon.outer ← ∅
6:      curentCPolygon.holes ← ∅
        {Uložit a vyčistit aktuální kreslený polygon}
7:      polygonsWH.push_back(curentPolygonWH)
8:      curentPolygonWH ← ∅
9:    end if
      {Vymazat případné neukončené díry}
10:   currentHole ← ∅
      {Resetovat stav polygonu}
11:   isPolygonReady ← false
      {Zpracovat událost levého tlačítka}
12:   volání mousePressEventLeft(e)
13: end if
    {Zpracování pravého tlačítka myši}
14: if e.button = RightButton then
15:   if not isPolygonReady or currentPolygon není prázdný then
16:     zobrazit zprávu "Dokončete polygon před přidáním děr"
17:   else
18:     volání mousePressEventRight(e)
19:   end if
20: end if
    {Obnovit vykreslení}
21: repaint
```

Zpracování levého tlačítka myši

```
1: Vstup: událost myši e
   {Pokud se jedná o dvojklik, ukončit polygon}
2: if e.type = MouseButtonDblClick then
3:    if currentPolygon není prázdný then
4:       curentCPolygon.outer ← currentPolygon
         {Uzavřít polygon přidáním prvního bodu na konec}
5:       currentPolygon.push_back(currentPolygon.first)
6:       curentPolygonWH.addPolygon(currentPolygon)
         {Označit polygon jako dokončený}
7:       isPolygonReady ← true
8:       currentPolygon ← ∅
9:    end if
10:   repaint
11:   return
12: end if
   {Získat souřadnice kliknutí}
13: x ← e.pos.x
14: y ← e.pos.y
15: if add_point = true then
16:   q.x ← x
17:   q.y ← y
18: else
19:   p ← QPointF(x, y)
20:   currentPolygon.push_back(p)
21: end if
```

Zpracování pravého tlačítka myši

```
1: Vstup: událost myši e
   {Pokud se jedná o dvojklik, ukončit díru}
2: if e.type = MouseButtonDblClick then
3:    if currentHole není prázdný then
4:       curentCPolygon.holes.push_back(currentHole)
         {Uzavřít díru přidáním prvního bodu na konec}
5:       currentHole.push_back(currentHole.first)
6:       curentPolygonWH.addPolygon(currentHole)
         {Vymazat díru}
7:       currentHole ← ∅
8:    end if
9:    repaint
10:   return
11: end if
   {Přidat bod do aktuální díry}
12: x ← e.pos.x
13: y ← e.pos.y
14: p ← QPointF(x, y)
15: currentHole.push_back(p)
```

5.1.3 Analýza polygonu s dírami

K analýze, zda se bod nachází v polygonu, je nejprve nutné ověřit, zda neleží v některé z děr daného polygonu. Pokud ano, bod není součástí polygonu. Proto se nejprve provádí analýza jednotlivých děr a teprve poté kontrola vůči samotnému polygonu.

Analýza polygonu s dírami

```
1: inicializace
   {Pro každý polygon provést testování}
2: for all polygon_i v polygons do
3:    polygon ← polygon_i.outer
4:    holes ← polygon_i.holes
5:    resH ← 0
6:    res ← 0
      {Nejprve ověřit min-max box}
7:    if Algorithms.minMaxBox(q, polygon) then
8:       if holes není prázdné then
9:          for all hole v holes do
10:            resH ← Algoritmus pro analýzu vztahu polygonu a díry
11:         end for
12:      end if
         {Pokud bod není v díře, analyzovat hlavní polygon}
13:      if resH ≠ 1 then
14:         res ← Algoritmus pro analýzu vztahu polygonu a díry
15:         Další zpracování
16:      end if
17:      Další zpracování
18:   end if
19: end for
20: Další zpracování
```

5.1.4 Vykreslení polygonu s dírami

Pro správné vykreslení polygonu nebo děr je potřeba před předáním pomocného polygonu typu `QPolygonF` do proměnné typu `QPainterPath` přidat počáteční bod na konec pomocného polygonu.

5.2 Zvýraznění polygonů

Po určení, zda se bod nachází v polygonu, je nutné vybrané polygony znovu vykreslit jinou barvou. Jakmile jsou polygony identifikovány, musí být jejich vykreslení aktualizováno. Pokud se vykreslení výsledků zruší, je potřeba obnovit původní barvu polygonů. Když bod leží v daném polygonu je přidán do `std::vector<QPainterPath> selectedPolygonsWH`, když se výběr zruší tak jsou prvky z `std::vector<QPainterPath> selectedPolygonsWH` vymazány. Při překreslování, jsou všechny polygony znovu vykresleny

Vykreslení vybraných polygonů

```
1: Vykreslení všech polygonů
2: for all polygon v polygonsWH do
3:    painter.setPen(Qt :: GlobalColor :: red)
4:    painter.setBrush(Qt :: GlobalColor :: yellow)
5:    painter.drawPath(polygon)
6: end for
7: Vykreslení vybraných polygonů s opačnými barvami
8: for all selectedPolygon v selectedPolygonsWH do
9:    painter.setPen(Qt :: GlobalColor :: yellow)
10:   painter.setBrush(Qt :: GlobalColor :: red)
11:   painter.drawPath(selectedPolygon)
12: end for
```

5.3 Souřadnice shapefilu

Při nahrávání a vykreslování polygonů ze shapefile je nutné provést transformaci souřadnic z geografického souřadnicového systému do souřadnicového systému okna aplikace.

- **Načtení hranic shapefile**
  Nejprve se pomocí funkce `SHPGetInfo` zjistí minimální a maximální souřadnice polygonů obsažených ve shapefile (bounding box).
- **Zjištění rozměrů okna aplikace**
  Rozměry vykreslovací plochy se zjistí funkcemi `width()` a `height()`. Tyto hodnoty definují maximální prostor, který může být využit pro zobrazení polygonů.
- **Výpočet měřítka transformace**
  Aby se polygon přizpůsobil velikosti vykreslovací oblasti vypočte se měřítkový koeficient:

$$\text{scale} = \min\left(\frac{\text{widgetWidth}}{\text{maxX} - \text{minX}}, \frac{\text{widgetHeight}}{\text{maxY} - \text{minY}}\right) \tag{5}$$

- **Výpočet posunu polygonu**
  Po aplikaci měřítka je třeba polygon zarovnat do středu vykreslovací plochy. Posun v ose X a Y se vypočte následovně:

$$\text{offsetX} = \frac{\text{widgetWidth} - (\text{maxX} - \text{minX}) \cdot \text{scale}}{2} - \text{minX} \cdot \text{scale} \tag{6}$$

$$\text{offsetY} = \frac{\text{widgetHeight} - (\text{maxY} - \text{minY}) \cdot \text{scale}}{2} + \text{maxY} \cdot \text{scale} \tag{7}$$

- **Transformace jednotlivých bodů**
  Každý bod X a Y v polygonu se transformuje následujícím způsobem:

$$\text{transformedX} = \text{x} \cdot \text{scale} + \text{offsetX} \tag{8}$$

$$\text{transformedY} = -\text{y} \cdot \text{scale} + \text{offsetY} \tag{9}$$

- **Uložení transformovaných bodů**
  Transformované body jsou uloženy do datové struktury pro polygon.

6 Vstupní data

Data pro analýzu lze do aplikace získat manuálním zadáním, popřípadě načtením z textového souboru, nebo ze souboru shapefile.

6.1 Vstupní data od uživatele

Data jsou získány čtením souřadnic cursoru myši nad widgetem aplikace.

6.1.1 Polygon

- Vnější strana polygonu
  - Levé tlačítko vloží bod do polygonu.
  - Dvojklik levého tlačítka ukončí polygon.
- Díra v polygonu
  - Pravé tlačítko vloží bod do díry.
  - Dvojklik pravého tlačítka ukončí tvorbu díry.

6.1.2 Bod

Bod je ukládán po přepnutí typu vstupu z polygonu na bod. Bod je přidán zmáčknutím levého tlačítka myši.

6.2 Načítání dat z textového souboru

Při načítání dat z textového souboru je třeba, aby soubor měl specifický formát. Každý bod v polygonu je reprezentován souřadnicemi x a y, oddělenými čárkou. Každý polygon je oddělen prázdným řádkem. Bod je označen vložením do složených závorek a je vždy načten jen poslední bod v textovém souboru. Příklad formátu souboru:

```
100, 100
100, 200
200, 200
200, 100

100, 200
200, 200
200, 300
100, 300

200, 200
300, 200
300, 300
200, 300

{200, 200}
{200, 250}
```

V tomto formátu:

- Každá skupina souřadnic (například $100, 100$; $100, 200$; $200, 200$; $200, 100$) tvoří jeden polygon.
- Každý prázdný řádek mezi skupinami souřadnic označuje konec jednoho polygonu a začátek dalšího.
- Souřadnice obklopené složenými závorkami, jako například {200, 200}, označují bod, přičemž načten je vždy poslední bod.

Před načtením jsou vymazány veškeré údaje, které jsou načtené, nebo vytvořené v aplikaci. Polygony z textového souboru jsou načítány do proměnné pro analýzu polygonů, tak do proměnné pro vykreslování polygonů.

6.3 Načítání dat z shapefile souboru

Načítání dat z formátu shapefile probíhá pomocí knihovny `shapelib`, přičemž je použita její upravená verze pro C++. Funkce `loadPolygonFromShapefile` načítá soubor shapefile a každý objekt (tedy polygon) v souboru je přetvořen na seznam bodů. Následně jsou tyto body transformovány tak, aby se správně zobrazily v uživatelském rozhraní aplikace.

- Soubor je otevřen pro čtení pomocí funkce `SHPOpen`, která otevře shapefile soubor v režimu čtení binárního souboru `"rb"`. Pokud není soubor dostupný, aplikace zobrazuje chybovou hlášku.
- Po otevření shapefile souboru se načítají základní informace o počtu entit (polygonů), typu tvaru a souřadnicích hranic (bounding box) pomocí funkce `SHPGetInfo`. Pokud soubor neobsahuje žádné entity, aplikace zobrazí varovnou zprávu.
- Pro každý polygon se získají souřadnice jeho vrcholů pomocí funkce `SHPReadObject`, která načte polygon z shapefile souboru. Tyto souřadnice jsou následně transformovány, aby se správně zobrazily na widgetu aplikace. Změna měřítka škáluje polygony a posun je použit pro umístění polygonů na správné místo na obrazovce.

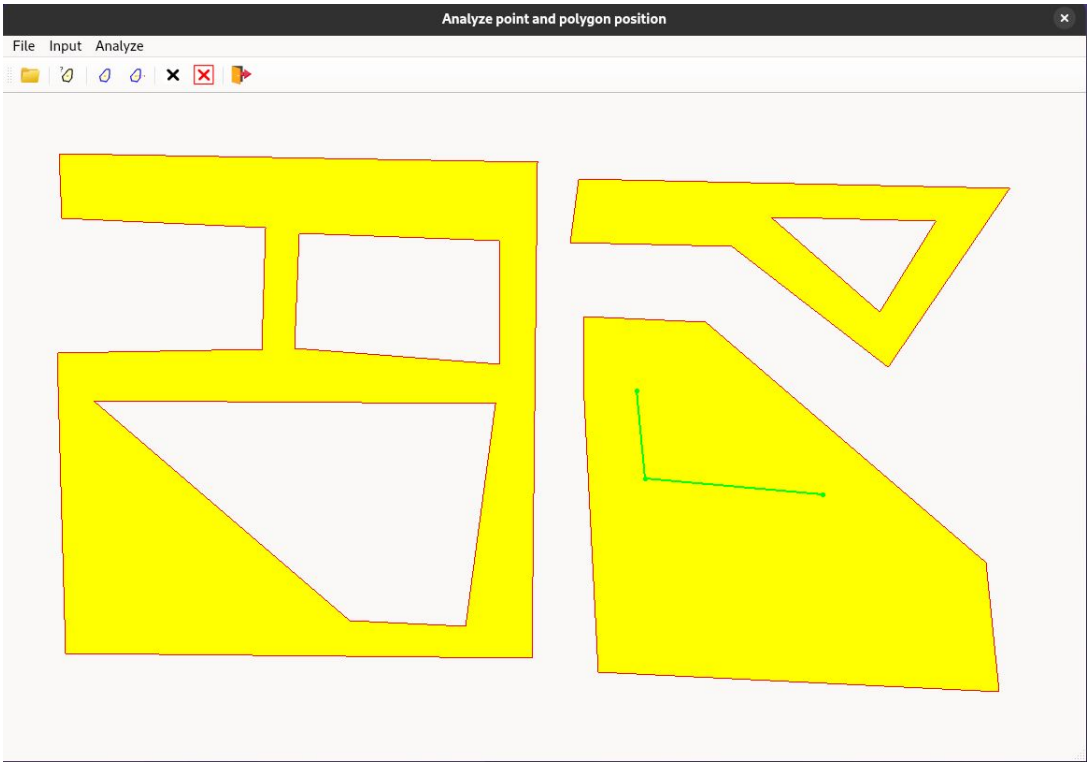
- Funkce `SHPDestroyObject` je volána po zpracování každého polygonu, aby se uvolnila paměť alokovaná pro daný objekt shapefile.
- Polygony jsou následně ukládány do struktury, která se skládá z vnějšího obvodu a případných děr.
- Po zpracování všech polygonů je soubor zavřen pomocí funkce `SHPClose`.

Pro ukázku byla použita data hranic krajů České republiky získaná z RÚIAN (Registr územní identifikace, adres a nemovitostí). Tato data obsahují hranice jednotlivých krajů ČR.
Před načtením jsou vymazány veškeré údaje, které jsou načtené, nebo vytvořené v aplikaci. Polygony ze souboru shapefile jsou načítány do proměnné pro analýzu polygonů, tak do proměnné pro vykreslování polygonů.

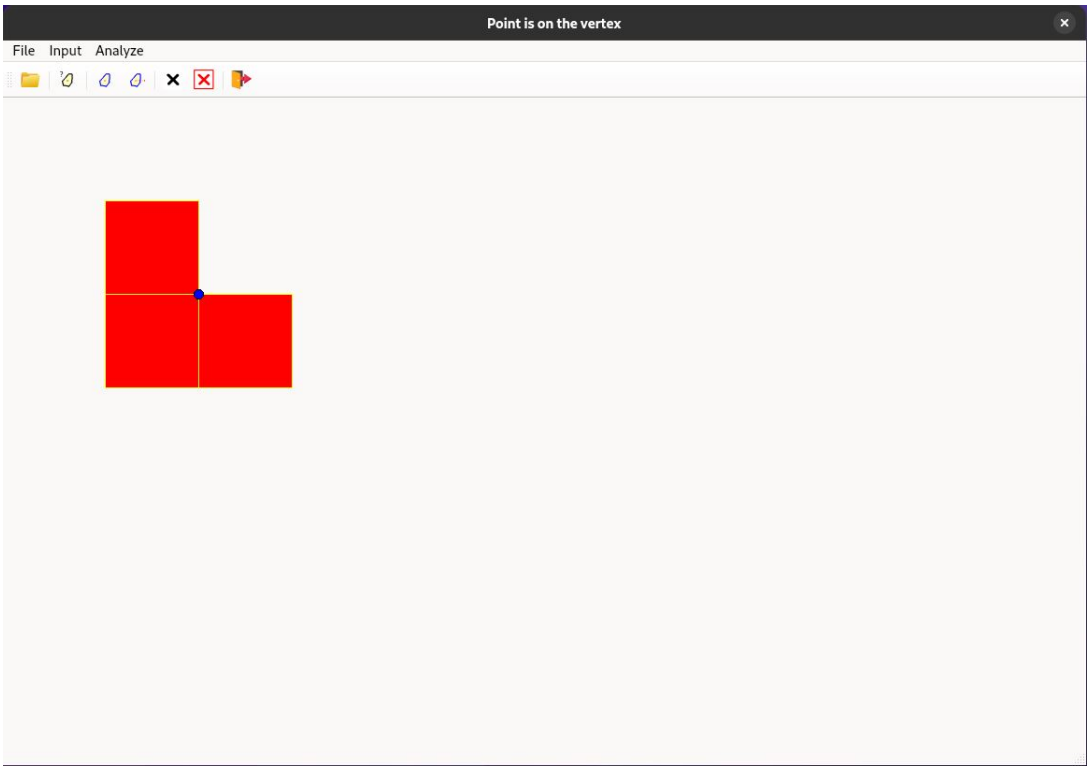
7 Výstupní data

Aplikace neprodukuje žádná výstupní data ve formě souborů. Výstupem je pouze vizualizace v aplikaci, kde jsou polygony zobrazeny v polygonové mapě a obarveny podle toho, zda obsahují zadaný bod. Dále aplikace vypisuje polohu bodu vůči jednotlivým polygonům (např. zda bod leží uvnitř polygonu, na hranici polygonu, ve vrcholu polygonu, nebo je mimo polygon).

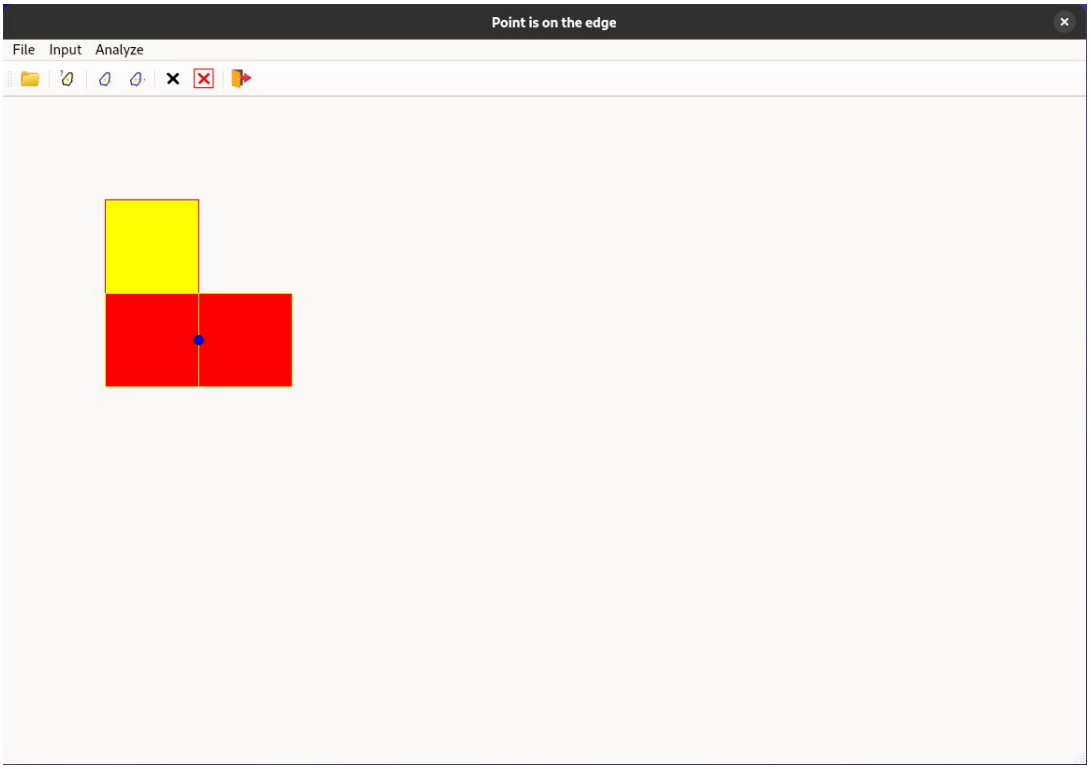
8 Výsledná aplikace



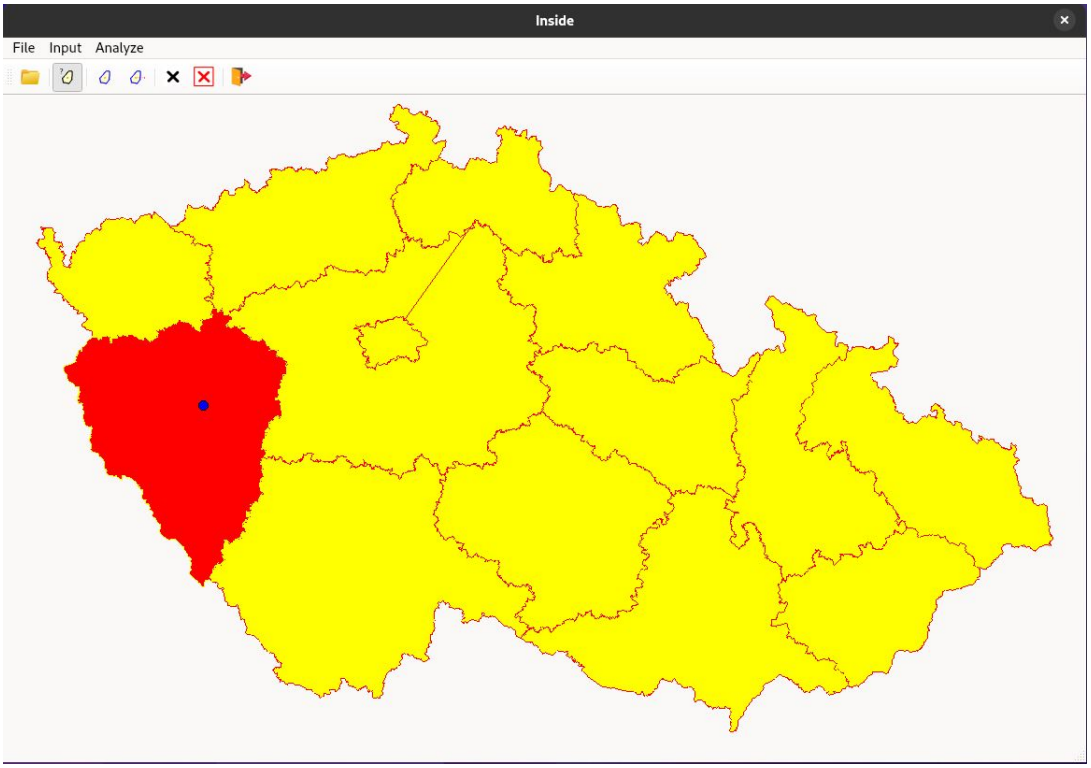
Obrázek 1: Ukázka vzhledu výsledné aplikace



Obrázek 2: načtená data z txt s bodem ve vrcholu tří polygonů



Obrázek 3: načtená data z txt s bodem na hraně dvou polygonů



Obrázek 4: načtená data z shp s bodem uvnitř polygonu

9 Dokumentace

9.1 Použité knihovny

Kód využívá následující knihovny:

- `Qt` – Knihovna pro správu grafických objektů, včetně tříd jako `QPointF`, `QPolygonF`, `QPainterPath`.
- `shapelib` – Knihovna pro práci se soubory formátu Shapefile [2]. Upravená verze této knihovny do C++ byla použita z projektu shapefile-viewer-qt od Zhihao Liu [3].

9.2 Třída Algorithms

Třída poskytuje metody pro geometrické analýzy bodů a polygonů.

Veřejné statické metody:

- `short analyzeRayCrossing(const QPointF &q, const QPolygonF &pol)` – Analyzuje vztah bodu a polygonu pomocí algoritmu Ray Crossing.
- `short analyzeWindingNumber(const QPointF &q, const QPolygonF &pol)` – Analyzuje vztah bodu a polygonu pomocí algoritmu Winding Number.
- `bool minMaxBox(const QPointF &q, const QPolygonF &pol)` – Kontroluje, zda bod leží uvnitř minmaxboxu polygonu.

Soukromé statické metody:

- `double calculateDistance(const QPointF &p1, const QPointF &p2)` – Vypočítá euklidovskou vzdálenost mezi dvěma body (`p1` a `p2`).
- `double calculateCosineValue(double l_qi, double l_qi1, double l_ii1)` – Vypočítá kosinusový úhel mezi vektory.
- `double calculateDeterminant(const QPointF &p1, const QPointF &p2, const QPointF &q)` – Vypočítá determinant matice tvořené třemi body (`p1`, `p2`, `q`).
- `short checkSingularities(const QPointF &q, const QPolygonF &pol, int i, int ii)` – Ověří výskyt singulárních bodů v polygonu.

9.3 Třída Draw

Třída pro vykreslování polygonů a interakci s uživatelem.

Veřejné metody:

- `void mousePressEvent(QMouseEvent *e)` – Zpracuje událost kliknutí myší.
- `void paintEvent(QPaintEvent *event)` – Vykreslí polygony a body.
- `void switch_source()` – Přepíná typ vstupních dat.
- `void loadPolygonFromFile(const QString &fileName)` – Načte polygon z `*.TXT`.
- `void loadPolygonFromShapefile(const QString &fileName)` – Načte polygon z `*.SHP`.
- `void clearPolygons()` – Vymaže všechny polygony.
- `void addSelectedPolygon(const QPainterPath& selection)` – Přidá vybraný polygon.
- `void clearSelectedPolygons()` – Odstraní všechny vybrané polygony.

Soukromé metody:

- `void mousePressEventLeft(QMouseEvent *e)` – Zpracovává kliknutí levého tlačítka myši.
- `void mousePressEventRight(QMouseEvent *e)` – Zpracovává kliknutí pravého tlačítka myši.
- `void addPointToPath(QPointF p)` – Přidává bod `p` do aktuální cesty, která tvoří polygon.

9.4 Třída MainForm

Hlavní uživatelské rozhraní aplikace.

Soukromé sloty:

- `void on_actionPoint_Polygon_triggered()` – Přepnutí vstupu bod/polygon
- `void on_actionRay_Crossing_triggered()` – Spustí analýzu metodou Ray Crossing.
- `void on_actionWinding_Number_triggered()` – Spustí analýzu metodou Winding Number.
- `void on_actionOpen_triggered()` – Otevře nabídku pro výběr souboru.
- `void on_actionClear_data_triggered()` – Vymaže aktuální data.
- `void on_actionClear_all_triggered()` – Vymaže všechny objekty.
- `void on_actionExit_triggered()` – Ukončí aplikaci.

10 Závěr

V rámci této úlohy byla vytvořena aplikace, která umožňuje analýzu polohy bodu vůči polygonové mapě pomocí dvou algoritmů: `Ray Crossing Algorithm` a `Winding Number Algorithm`. Aplikace umožňuje uživateli ruční tvorbu polygonové mapy nebo nahrání polygonů ze souboru ve formátech `*.TXT` a `*.SHP`. Uživatel může měnit polohu bodu na mapě a aplikace vizuálně zvýrazní výsledek analýzy. Pro vyhodnocení polohy bodu byly ošetřeny speciální případy, kdy bod leží na hraně polygonu nebo je totožný s některým z jeho vrcholů. Aplikace umožňuje zvýraznit odpovídající polygon vizuálně pomocí změny barvy na červenou. Pro zrychlení vyhledávání relevantních polygonů byla použita metoda min-max boxů. Implementace byla provedena v jazyce C++ s využitím frameworku Qt pro grafické rozhraní.

10.1 Další Možné Neřešené Problémy a Náměty na Vylepšení

- **Podpora dalších formátů:** V současnosti aplikace podporuje načítání polygonů ze souborů ve formátech TXT a SHP. Bylo by vhodné rozšířit podporu také o formáty jako GeoJSON nebo GeoPackage, které jsou běžně využívány v GIS aplikacích.
- **Generování náhodných polygonů:** Aplikace by mohla být rozšířena o možnost generování náhodných polygonů.
- **Ukládání vytvořených polygonů:** Aktuálně aplikace umožňuje načítání polygonů, ale ne jejich ukládání. Přidání možnosti exportu polygonů do souboru by zlepšilo použitelnost aplikace.
- **Generování textového souboru s výsledky analýzy:** Aplikace by mohla být rozšířena o možnost generování textového souboru obsahujícího výsledky analýzy polohy bodu vůči polygonům. To by usnadnilo uchování a sdílení výsledků analýz.
- **Dávkové zpracování:** Dalším vylepším by mohla být implementace dávkového zpracování, které by umožnilo uživateli analyzovat více bodů nebo polygonů najednou bez nutnosti interaktivního zadávání dat.
- **Možnosti interakce s polygonovou mapou:** Aktuální implementace umožňuje pouze základní práci s polygony. Bylo by možné rozšířit funkcionalitu o editaci polygonů, například odebírání jednotlivých vrcholů nebo přidávání nových.
- **Zoomování a posouvání mapy:** Aplikace v současnosti nezahrnuje podporu pro přibližování a posun mapy, což by mohlo usnadnit práci s polygony.
- **Odstranění jednotlivých prvků:** Dalším možným vylepšením by byla možnost selektivního mazání nakreslených prvků, například odstranění konkrétního polygonu nebo jeho editace, aniž by bylo nutné smazat veškeré polygony.
- **Implementace Half-Plane Testu:** Dalším rozšířením by mohla být implementace metody Half-Plane Test. Pro její správnou funkčnost by však bylo nutné nejprve ověřit, zda je polygon konvexní, a pokud ne, rozdělit jej na konvexní části.

- **Nastavení:** Přidat tlačítko umožňující nastavení parametrů, jako jsou barva, tloušťka a velikost.
- **Rušení aktuálně kresleného prvku:** Možnost zrušit aktuální kresbu pomocí tlačítka.
- **Čtení polygonů s dírami:** Implementovat načítání polygonů uložených v textovém souboru nebo shapefile tak, aby správně identifikovalo a uložilo díry.
- **Polygon s dírami:** Zamezit tvorbě děr mimo polygon nebo uvnitř již existující díry.
- **Aktualizace vektoru polygonů před analýzou:** Jakmile se vytvoří nový polygon přidáním bodu, neproběhne jeho analýza okamžitě, protože ještě není uložen ve vektoru polygonů. Analýza se provede až při vytvoření dalšího bodu nebo polygonu. Proto je vhodné před analýzou nejprve aktualizovat vektor polygonů.
- **Formát textového souboru:** Vytvoří načítání, aby bylo možné načíst textový soubor s různými formáty.
- **Vizuál aplikace:** Zlepšit vizuální stránku aplikace.

Odkazy

[1] Tomáš Bayer. *Algoritmy v digitální kartografii*. 2008. vyd. Praha: Karolinum, 2008. ISBN: 978-80-246-1499-1.

[2] Frank Warmerdam. *Shapelib*. `http://shapelib.maptools.org/`. Přístup k 2025-03-16. 1999-2002.

[3] Zhihao Liu. *Shapelib pro C++*. `https://github.com/zhihao-liu/shapefile-viewer-qt/tree/master/shapelib`. Přístup k 2025-03-16. 2025.