České vysoké učení v Praze

Fakulta stavební

155ADKG - Algoritmy digitální kartografie

Úloha 4: Množinové operace s polygony

Radek Novotný

Zadání:

Úloha č. 4: Množinové operace s polygony

Vstup: množina n polygonů $P=\{P_1, ..., P_n\}$.

Výstup: množina m polygonů $P' = \{P'_1, ..., P'_m\}$.

S využitím algoritmu pro množinové operace s polygony implementujte pro libovolné dva polygony $P_i, P_j \in P$ následující operace:

- Průnik polygonů $P_i \cap P_j$,
- Sjednocení polygonů $P_i \cup P_j$,
- Rozdíl polygonů: $P_i \cap \overline{P}_j$, resp. $P_j \cap \overline{P}_i$.

Jako vstupní data použijte existující kartografická data (např. konvertované shape fily) či syntetická data, která budou načítána z textového souboru ve Vámi zvoleném formátu.

Grafické rozhraní realizujte s využitím frameworku QT.

Při zpracování se snažte postihnout nejčastější singulární případy: společný vrchol, společná část segmentu, společný celý segment či více společných segmentů. Ošetřete situace, kdy výsledkem není 2D entita, ale 0D či 1D entita.

Pro výše uvedené účely je nutné mít řádně odladěny algoritmy z úlohy 1. Postup ošetření těchto případů diskutujte v technické zprávě, zamyslete se nad dalšími singularitami, které mohou nastat.

Hodnocení:

Krok	**Hodnocení**
Množinové operace: průnik, sjednocení, rozdíl	20b
Konstrukce offsetu (bufferu)	*+10b*
Výpočet průsečíků segmentů algoritmem Bentley & Ottman	*+8b*
Řešení pro polygony obsahující holes (otvory)	*+6b*
Max celkem:	**44b**

Čas zpracování: 2 týdny

Bonusové úlohy:

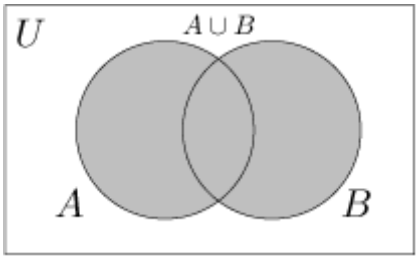
V rámci bonusových úloh byla řešena konstrukce buffer.

Popis problému:

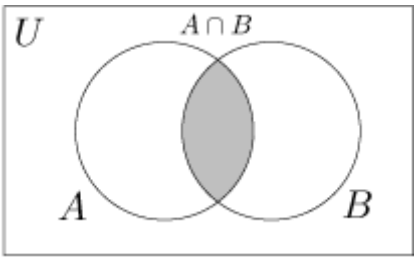
Množinové operace

Provést pro danou dvojici uzavřených a ohraničených polygonů množinové operace:

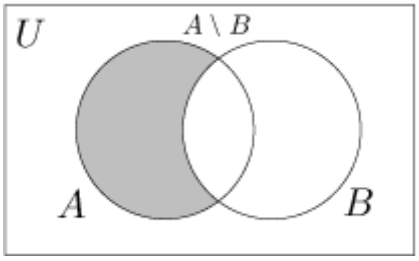
- Sjednocení (Union): $C = A \cup B$



- Průnik (Intersection): $C = A \cap B$



- Rozdíl (Difference): $C = A \cap \bar{B}$



Buffer

Pro polylinii vytvoříme oblast, jejíž hranice je ekvidistantou od polylinie.

Popis algoritmů:

Základními fázemi algoritmu jsou výpočet průsečíků mezi oběma polygony a jejich setřídění v závislosti na použité orientaci, použita CCW orientace. Tyto průsečíky jsou dále vedeny jako vrcholy polygonů. U všech vrcholů je ohodnocena jejich pozice vůči druhému polygonu.

Pomocí ohodnocení jsou vytvořeny fragmenty, ze kterých je v poslední fázi vytvořen výsledný polygon.

```
BooleanOperations(A, B, oper)

1: if (o(A)! = CCW)
2:     A.switchOr ()       //Změň orientaci na CCW
3: if (o(B)! = CCW)
4:     B.switchOr ()       //Změň orientaci na CCW
5: ComputeIntersections(A, B) //Urči průsečíky A, B
6: SetPositions(A, B)      //Urči polohu vrcholů vůči oblastem
7: map < QPointFI, pair < bool, vector < QPointFB >>> F
8: pos1 = (oper ≡ Intersection ∨ oper ≡ DifBA?Inner : Outer )
9: pos2 = (oper ≡ Intersection ∨ oper ≡ DifAB?Inner : Outer )
10: swap1 = (oper ≡ DifAB : true : false
11: swap2 = (oper ≡ DifBA) : true : false
12: CreateFragments(A, pos1, swap1, F)
13: CreateFragments(B, pos2, swap2, F)
14: MergeFragments(A, B, C)
```

Algoritmus výpočtu průsečíků:

ComputeIntersection(A,B)

```
1: for (i = 0; i < n; i++):
2:     M = map <double, QPointFB>                          //Vytvoření mapy
3:     for (j = 0; j < m; j++):
4:           if b_ij = (p_i , p_(i+1)%m) ∩ (q_j , q_(j+1)%m) ≠ ∅      //Existuje průsečík
5:                 M[α_i ] ← b_ij                          //Přidej do M
6:                 ProcessIntersection(b_ij,β,B,j)         //Zpracuj první průsečík pro e'_j
7:     if (|M| > 0)                                        //Nějaké průsečíky jsme nalezli
8:           for ∀m ∈ M:                                   //Procházej všechny průsečíky v M
9:                 b ← m.second                            //Získej 2. hodnotu z páru
10:                ProcessIntersection(b,α,A,i)            //Zpracuj aktuální průsečík pro e_i
```

Metoda `ProcessIntersection`:

```
1: if (|t| < ε):
2:     P[i] ← inters              //Startovní bod průsečíkem
3: if (||t| − 1| < ε):
4:     P[(i + 1)%m] ← inters      //Koncový bod průsečíkem
5: else:
6:     i ← i + 1                  //Inkrementuj pozici
7:     P ← (b, i)                 //Přidej průsečík na pozici i + 1.
```

Algoritmus vytvoření fragmentů:

CreateFragments(P, g, s, F)

```
1: i ← 0
2: while (g(P[i]) ≠ g ∨ P[i] ≠ inters)         //Dokud P[i] není průsečík s orientací g
3:     i ← i + 1
4: if (i ≡ n) return;                          //Žádný bod s touto orientací neexistuje
5: is ← i                                      //Zapamatuj startovní index prvního průsečíku
6: do
7:     f = ∅                                   //Prázdný fragment
8:     if (createFragmentFromVertices(is, P, g, i, f)) //Nalezen fragment
9:           if (s) f.reverse()                //Swapuj prvky fragmentu, je -li třeba
10:          F[f[0]] → f                       //Přidej fragment do mapy, klíč poč. bod
11:    i ← (i + 1)%m
12: while (i ≠ is)                             //Dokud nedojdeme k počátečnímu průsečíku
```

Metoda `createFragmentFromVertices(is, P, g, i, f)`

```
1: if g(P[i]) ≠ g ∨ P[i] ≠ inters //Bod není průsečíkem s orientací g
2:     return false
```

```
3: for (;;)
4:     f ← P[i]                    //Přidej bod do fragmentu
5:     i ← (i + 1)%n
6:     if (i ≡ is)                 //Obešli jsme celý polygon
7:           return false
8:     if g(P[i]) ≠ g              //První bod s rozdílnou orientací
9:           f ← P[i]              //Přidej ho do seznamu
10:          return true           //Úspěšně skončeno
```

Algoritmus pro spojení fragmentů

MergeFragments(F, C)

```
1: for ∀f ∈ F
2:     P ← ∅                       //Vytvoř prázdný polygon
3:     s ← f.first                 //Najdi startovní bod fragmentu
4:     if (!f.second.first)        //Pokud fragment již nebyl zpracován
5:           if(createPolygonFromFragments(s, F, P))
6:                 C ← P           //Přidej polygon do seznamu
```

createPolygonFromFragments(s, F, P)

```
1: QPoint n ← s                    //Inicializuj následující bod
2: for (;;)                        //Projdi všechny fragmenty tvořící polygon
3:     f ← F.find(n)               //Najdi navazující fragment
4:     if (f ≡ F.end)              //Fragment s takovým poč. bodem neexistuje
5:           return false
6:     f.second.first ← true       //Fragment označen za zpracovaný
7:     n ← f.second.second.back() //Najdi následující bod
8:     P ← f.second.second − {f.second.second[0]} //Přidej bez poč. Bodu
9:     if (n ≡ s)                  //Obešli jsme polygon, jsme na startu
10:          return true
```

offsetPolyline(P, d, Δϕ)

```
1: Cd (L) ← ∅                                    //Prázdný offset polylinie
2: Cd (e) ← createOffset(P[0], P[1], d, Δϕ)     //Offset prvního segmentu
3: Cd (L) ← Cd (e)                               //Přidání prvního offsetu
4: for ∀i = 1, ..., n-1
5:     Cd (e) ← createOffset(P[k], P[(k + 1)%n], d, Δϕ) //Offset k- tého segmentu
6:     C'd (L) ← ∅                               //Vytvoření pomocné struktury
7:     for ∀j = 0, ..., m-1                      //Pro všechny IN/OUT segmenty stávajícího offsetu
8:           C'd (e) = Cd (e) ∩ Cd (L)[j]        //Sjednocení se stávajícím offsetem
9:           if C0 d (e) ≡ ∅                     //Hole z předchozích iterací
10:                C'd (L) ← Cd (L)[j]           //Přidej stávající Hole do Cd (Lk )
11:          else                                //Nový Solid, Holes
12:                C'd (L) ← C'd (e)             //Přidej nový Solid a Holes do Cd (Lk )
13:    Cd(L) ← C'd (L)                           //Aktualizuj offset polyline: Cd (Lk ) = Cd (Lk-1)
```

createOffset(p1, p2, d, Δϕ)

```
1: c ← ∅
2: ϕ ← atan2(y2 − y1, x2 − x1)
3: ϕs,1 ← ϕ + π/2
4: ϕe,1 ← ϕs,1 + π
5: sampleArc(p1, d, ϕs,1, ϕe,1, Δϕ, c)
6: ϕs,2 ← ϕ − π/2
7: ϕe,2 ← ϕs,2 + π
8: sampleArc(p2, d, ϕs,2, ϕe,2, Δϕ, c)
9: return c
```

Vstupní data

Vstupem jsou dva polygony. Ty mohou být zadány buď manuálním zadáváním jednotlivých bodů do kanvasu pomocí levého tlačítka myši. Mezi polygony se přepíná pomocí tlačítka Polygon A/B.

Druhou možností je nahrání textového souboru se souřadnicemi. Tento soubor musí být ve tvaru:

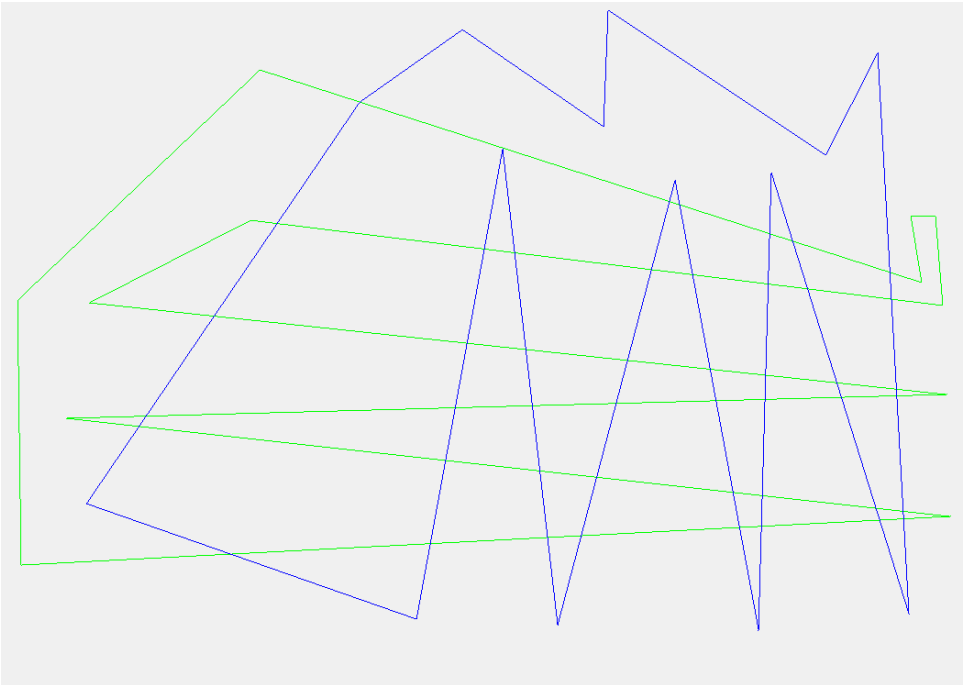
X Y
X Y
X Y

Každý polygon v jednom souboru, každý bod na jednom řádku, jednotlivé souřadnice odděleny mezerou.

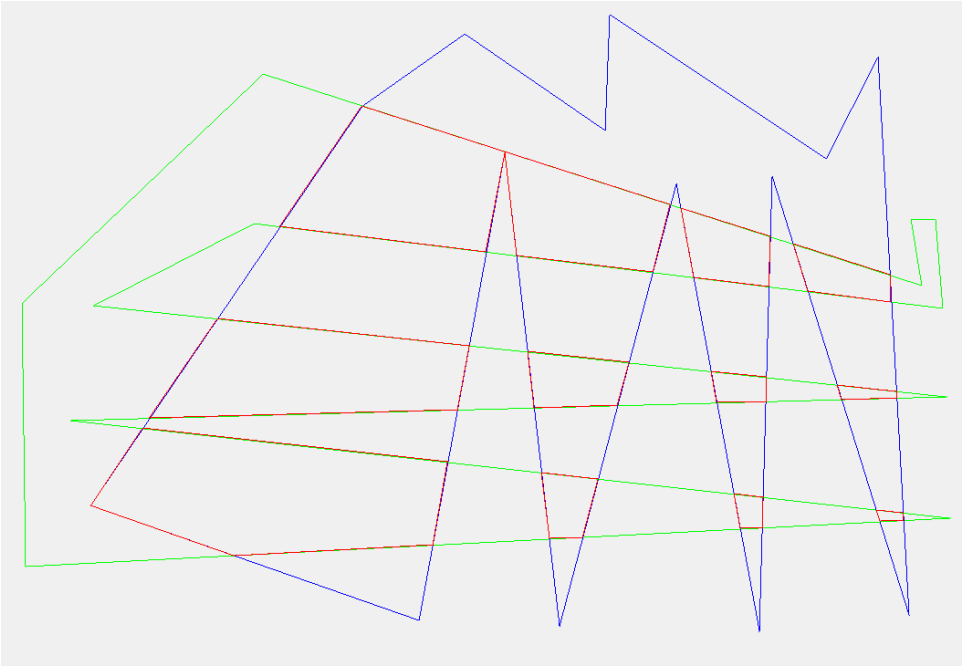
Výstupní data:

Výsledkem je grafické zobrazení dané množinové operace.

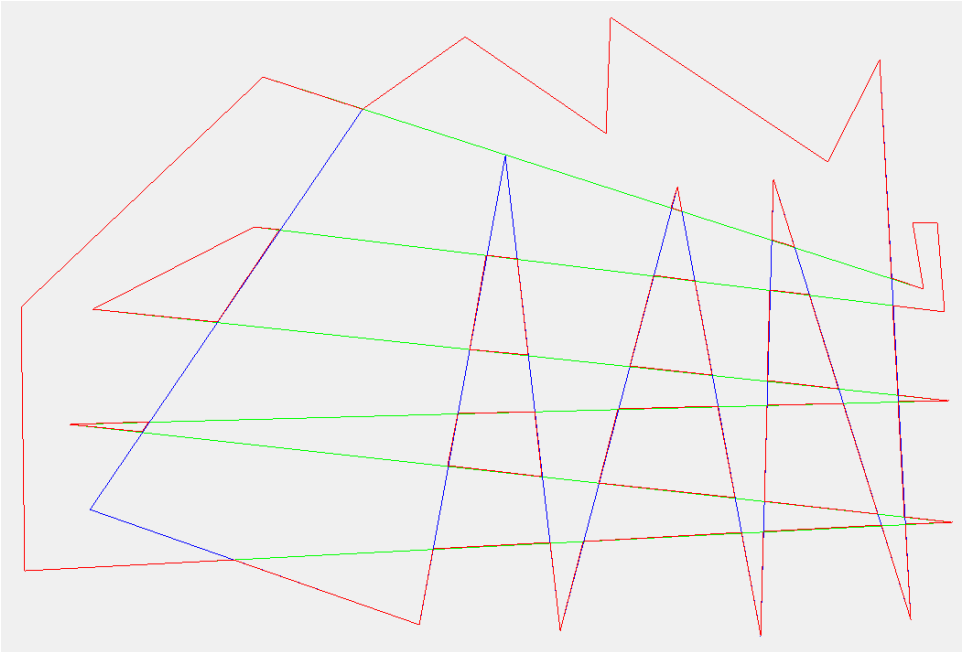
Printscreen:



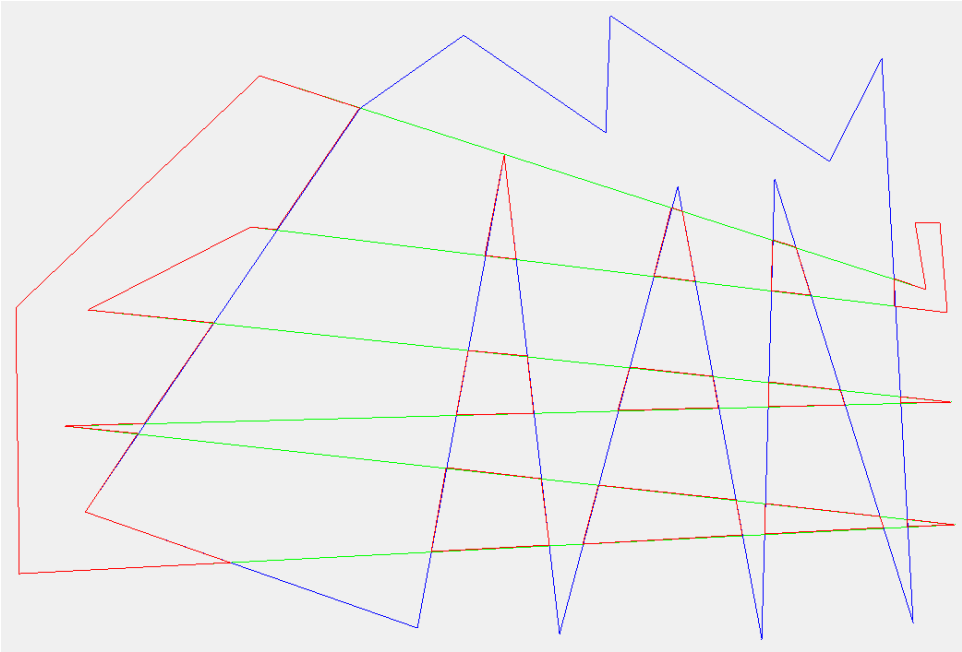
Obr.1 Vstupní polygony



Obr.2 Intersection



Obr.3 Union



Obr.4 Difference

Zhodnocení:

Byl vytvořen program umožňující nahrát nebo ručně vložit polygony a počítat na nich množinové operace - sjednocení, průsečík a rozdíl.

Problematické situace:

Program nedokáže počítat s otvory v polygonech, ty do něj ani nelze vložit. Pokud ručně zadáme polygon, jehož strany se protínají, program množinové operace nepočítá. Špatně se mohou zobrazovat polygony, jež mají jen jeden společný bod.