České vysoké učení technické v Praze
Fakulta stavební

155ADKG: Množinové operace s polygony

Michael Kala
Anna Zemánková

1 Zadání

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

2 Údaje o bonusových úlohách

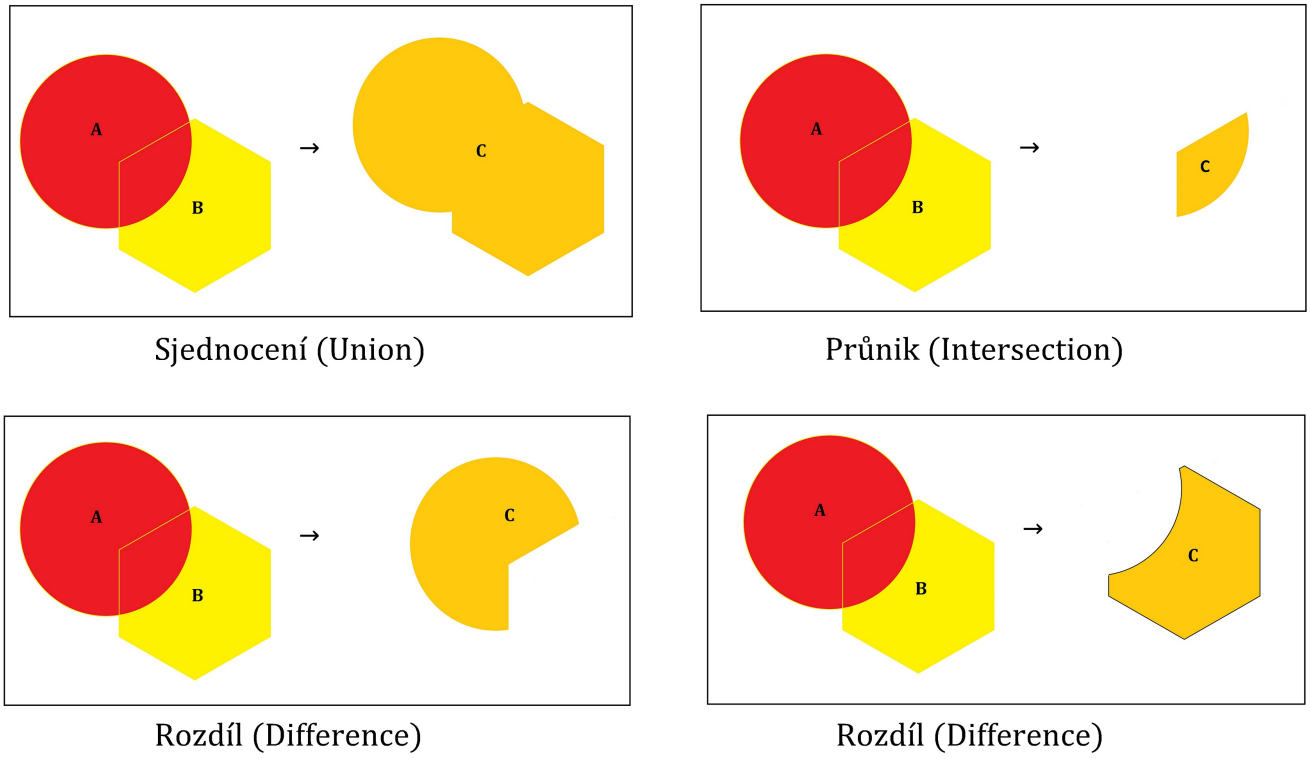
V rámci této úlohy nebyly zpracovány žádné bonusy.

3 Popis a rozbor problému

Mějme dva polygony A $\{p_i\}$ a B $\{p_j\}$.

Množinové operace:

- Sjednocení (Union): $C = A \cup B$
- Průnik (Intersection): $C = A \cap B$
- Rozdíl (Difference): $C = A \cap \bar{B}$ nebo $C = B \cap \bar{A}$



Obrázek 1: Příklady množinových operací

4 Popis algoritmů

4.1 Výpočet průsečíků

4.1.1 Implementace metody

1. $for(i = 0; i < n, i++)$
2. $\quad M = map(double, QPointFB)$ // Vytvoření mapy
3. $\quad for(j = 0; j < m, j++)$
4. $\quad\quad$ if $b_{ij} = (p_i, p_{(i+)1\%n} \cap q_j, q_{(j+)1\%m}) \neq 0$ // Existuje průsečík
5. $\quad\quad\quad M[\alpha_i] \longleftarrow b_{ij}$ //Přidej do M
6. $\quad\quad\quad$ ProcessIntersection (b_{ij}, β, B, j) //Zpracuj první průsečík pro e_j
7. $\quad$ if $(||M|| > 0)$ // Nějaké průsečíky jsme nalezli
8. $\quad\quad for \forall m \in M :$ // Procházej všechny průsečíky v M
9. $\quad\quad\quad b \longleftarrow m.second$ // Získej 2. hodnotu z páru
10. $\quad\quad\quad$ ProcessIntersection (b, α, A, i) //Zpracuj první průsečík pro e_i

Process Intersection

1. $if(|t| < \epsilon) :$
2. $\quad P[i] \longleftarrow inters$ //Startovní bod průsečíkem
3. $if(|t - 1| < \epsilon) :$
4. $\quad P[(i + 1)\%m] \longleftarrow inters$ // Koncový bod průsečíkem
5. $else :$
6. $\quad$ ProcessIntersection $i \longleftarrow i + 1$ //Inkrementuj pozici
7. $\quad$ if $P \longleftarrow (b, i)$ // Přidej průsečík na pozici i+1

4.2 Fragmenty

4.2.1 Vytvoření fragmentů - implementace metody

1. $i \longleftarrow 0$
2. $while(g(P[i]) \neq g \vee P[i] \neq inters)$ //Dokud P[i] není průsečík s orientací g
3. $\quad i \longleftarrow i + 1$
4. $if(i \equiv n)$ return; //Žádný bod s touto orientací neexistuje
5. $i_s \longleftarrow i$ //Zapamatuj startovní index prvního průsečíku
6. do
7. $\quad f = \emptyset$ //Prázdný fragment
8. $\quad if(createFragmentFromVertices(i_s, P, g, if))$ //Nalezen fragment
9. $\quad\quad if(s)f.reverse()$ //Swapuj prvky fragmentu, je-li třeba
10. $\quad\quad F[f[0]] \longrightarrow f$ //Přidej fragment do mapy, klíč poč. bod
11. $\quad i \longleftarrow (i + 1)\%m$
12. $while(i \neq i_s)$ //Dokud nedojdeme k počátečnímu průsečíku

createFragmentFromVertices

1. $if(g(P[i]) \neq g \vee P[i] \neq inters)$ //Bod není průsečíkem s orientací g
2. $\quad return\ false$
3. for (;;)
4. $\quad f \longleftarrow P[i]$ // Přidej bod do fragmentu
5. $\quad i \longleftarrow (i + 1)\%n$
6. $\quad if(i \equiv i_s)$ //Obešli jsme celý polygon
7. $\quad\quad return\ false$
8. $\quad if(g(P[i]) \neq g)$ // První bod s rozdílou orientací
9. $\quad\quad f \longleftarrow P[i]$ //Přidej ho do seznamu
10. $\quad\quad return\ true$

4.2.2 Sestavení oblastí z fragmentů - implementace metody

1. $for \forall f \in F$
2. $P \longleftarrow \emptyset$ //Vytvoř prázdný polygon
3. $s \longleftarrow f.first$ //Najdi startovní bod fragmentu
4. $if(!f.second.first)$ //Pokud fragment již nebyl zpracován
5. $if(createPolygonFROMFragments(s, F, P))$
6. $C \longleftarrow P$ //Přidej polygon do seznamu

4.2.3 Vytvoření polygonu z fragmentů - implementace metody

1. $QPointn \longleftarrow s$ //Inicializuj následující bod
2. for (;;) //Projdi všechyn fragmenty tvořící polygon
3. $f \longleftarrow F.find(n)$ //Najdi navazující fragment
4. $if(f \equiv F.end)$ // Fragment s takovým poč. bodem neexistuje
5. $return\ false$
6. $f.second.first \longleftarrow true$ //Fragment označen za zpracovaný
7. $n \longleftarrow f.second.second.back()$ // Najdi následující bod
8. $P \longleftarrow f.second.second - \{f.second.second[0]\}()$ // Přidej bez poč. bodu
9. $if(n \equiv s)$ // Obešli jsme polygon, jsme na startu
10. $return\ true$

4.3 Množinové operace

4.3.1 Implementace metody

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

5 Vstupní data

Dva polygony mohou být "nakresleny"klikáním do kanvasu, pomocí tlačítka *Polygon A/B* lze přepínat mezi polygony.

Dále mohou být polygony nahrány pomocí tlačítka *Load polygon*, v tom případě jsou data nahrána ze vstupního souboru *.txt, kde jsou souřadnice lomových bodů polygonů v tomto pořadí:

1. počet bodů v prvním polygonu
2. souřadnice X a Y (s desetinnou tečkou, pokud se nejedná o celé číslo) bodů prvního polygonu
3. počet bodů v druhém polygonu
4. souřadnice X a Y bodů druhého polygonu atd.

Př.: Vstupní soubor se 2 polygony, kde první polygon obsahuje 3 body a druhý polygon 4 body bude vypadat následovně:

3 12.2 4.1 8.3 11.4 6.8 7.9
4 1.1 2.2 3.3 4.4 5.5 6.6 7 8

Vše může být zapsáno do jednoho řádku, odděleno mezerami nebo jako v příkladu výše může každý útvar začínat na novém řádku.

6 Výstupní data

Výsledek aplikace je zobrazen graficky na kanvasu (červenou barvou).

7 Ukázka vytvořené aplikace

7.1 Algorithms

V třídě Algorithms jsou staticky implementovány algoritmy počítající kovexní obálku a minimální ohraničující obdélník (včetně vodící linie).

- Výčtový typ **TPosition**
  - Typ využitý jako návratová hodnota členské metody **getPointLinePosition**.
  - **LEFT = 0**
  - **RIGHT = 1**
  - **ON = 2**
- Metoda **getPointLinePosition**
  - Tato metoda slouží k určení polohy bodu vůči přímce. Návratovou hodnotou je výčtový typ **TPosition**.
  - Vstup
    - **QPointF &q** - určovaný bod
    - **QPointF &a, &b** - body přímky
  - Výstup
    - **LEFT** - bod vlevo od přímky
    - **RIGHT** - bod vpravo od přímky
    - **ON** - bod na přímce
- Metoda **getTwoVectorsAngle**
  - Tato metoda slouží k určení úhlu mezi 2 přímkami. Její návratovou hodnotu je **double**.
  - Vstup
    - **QPointF &p1, &p2** - body první přímky
    - **QPointF &p3, &p4** - body druhé přímky
  - Výstup
    - Úhel mezi 2 přímkami
- Metoda **getPointLineDistance**
  - Tato metoda slouží k výpočtu vzdálenosti bodu od přímky. Její návratovou hodnotou je **double**
  - Vstup
    - **QPointF &q** - určovaný bod
    - **QPointF &a, &b** - body přímky
  - Výstup
    - Vzdálenost bodu od přímky

- Přetížená metoda **rotateByAngle**
  - Tato metoda slouží k rotaci dané množiny o úhel. Jejím návratovým typem je **void**.
  - Vstup
    - Přetížení 1
      - **std::vector<QPointF> &points** - vektor bodů, jež má být orotován
      - **double angle** - úhel, o který má rotace být provedena
    - Přetížení 2
      - **QPolygonF &points** - polygon, jež má být orotován
      - **double angle** - úhel, o který má rotace být provedena
    - Přetížení 3
      - **QLineF &points** - úsečka, jež má být orotována
      - **double angle** - úhel, o který má rotace být provedena
- Metoda **getDistance**
  - Tato metoda slouží k výpočtu vzdálenosti dvou bodů. Jejím výstupním typem je **double**.
  - Vstup
    - **QPointF &a, &b** - body, mezi kterými je vzdálenost počítána
  - Výstup
    - Vypočtená vzdálenost
- Metoda **jarvisScanCH**
  - Tato metoda slouží k výpočtu konvexní obálky pomocí algoritmu Jarvis Scan. Během výpočtu je ošetřována singularita existence kolineárních bodů v datasetu. Jejím výstupním typem je **QPolygonF**.
  - Vstup
    - **std::vector <QPointF> points** - vektor bodů, kolem nichž má být vytvořená konvexní obálka.
  - Výstup
    - Polygon obsahující kovexní obálku.
- Metoda **grahamScanCH**
  - Tato metoda slouží k výpočtu konvexní obálky pomocí algoritmu Graham Scan. Jejím výstupním typem je **QPolygonF**.
  - Vstup
    - **std::vector <QPointF> points** - vektor bodů, kolem nichž má být vytvořená konvexní obálka.
  - Výstup

        * Polygon obsahující kovexní obálku.

- Metoda **quickHullCH**
    - Tato metoda slouží k výpočtu konvexní obálky pomocí algoritmu Quick Hull. Jejím výstupním typem je **QPolygonF**.
    - Vstup
        * **std::vector <QPointF> points** - vektor bodů, kolem nichž má být vytvořená konvexní obálka.
    - Výstup
        * Polygon obsahující kovexní obálku.

- Metoda **quickHullLocal**
    - Pomocná metoda k výpočtu konvexní obálky metodou Quick Hull. Jejím výstupním typem je **void**.
    - Vstup
        * **int s, e** - index počátečního a koncového bodu dělící přímky
        * **std::vector <QPointF> &points** - vektor bodů, kolem nichž má být vytvořená konvexní obálka.
        * **QPolygonF &poly_ch** - polygon obsahující body konvexní obálky
    - Výstup
        * Polygon obsahující kovexní obálku.

- Metoda **sweepLineCH**
    - Tato metoda slouží k výpočtu konvexní obálky pomocí algoritmu Sweep Line. Jejím výstupním typem je **QPolygonF**.
    - Vstup
        * **std::vector <QPointF> points** - vektor bodů, kolem nichž má být vytvořená konvexní obálka.
    - Výstup
        * Polygon obsahující kovexní obálku.

- Metoda **generatePoints**
    - Metoda pro generování zadaného počtu a tvaru bodů. Jejím výstupním typem je **std::vector <QPointF> points**.
    - Vstup
        * **QSizeF &canvas_size** - rozměry kreslícího plátna, ze kterých se determinuje rozsah generovaných bodů
        * **int point_count** - počet bodů, který se má generovat
        * **std::string shape** - tvar vytvářené množiny bodů (random, grid, na kružnici, na elipse, na čtverci)

  - Výstup
    * Vektor nagenerovaných bodů.

- Metoda **minimalRectangle**
  - Metoda pro výpočet minimálního ohraničujícího obdélníku a hlavní linie. Jejím výstupním typem je **void**.
  - Vstup
    * **QPolygonF &poly_ch** - polygon obsahující konvexní obálku
    * **QPolygonF &minimal_rectangle** - polygon, do kterého jsou počítány body minimálního ohraničujícího obdélníku
    * **QLineF &direction** - hlavní linie minimálního ohraničujícího obdelníku (resp. do této proměnné je počítaná)
    * **bool compute_dir_line** - ukazatel určující zda-li má být počítána hlavní linie minimálního ohraničujícího obdélníku

7.2 Draw

Třída draw slouží k vykreslení vygenerovaných (nebo naklikaných) bodů, vypočteného minimálního ohraničujícího obdélníku a hlavní linie minimálního ohraničujícího obdélníku. V této třídě jsou zároveň nagenerované body zbavené duplicit a vypočtené konvexní obálky se zde omezují na striktní konvexní obálky (vše v metodě **setCH**. Třída dědí od třídy **QWidget**.

- Členské proměnné
  - **std::vector <QPointF> points** - vektor obsahující nagenerované nebo naklikané body
  - **QPOlygonF ch** - polygon obsahující body konvexní obálky
  - **QPolygonF rect** - polygon obsahující body minimálního ohraničujícího obdélníku
  - **QLineF direction** - hlavní linie minimálního ohraničujícího obdélníka
- Metoda **paintEvent**
  - Tato metoda slouží k vykreslení nagenerovaných (nebo naklikaných) bodů, konvexní obýlky, minimálního ohraničujícího obdélníka a hlavní linie minimálního ohraničujícího obdélníka. Metoda se volá pomocí metody **repaint()**. Návratovým typem je **void**.
  - Vstup
    - **QPaintEvent *e**
- Metoda **mousePressEvent**
  - Metoda sloužící k uložení bodu do členské proměnné **points** určeného kliknutím myší nad kreslícím plátnem. Jejím návratovým typem je **void**.
  - Vstup
    - **QMouseEvent *e**
- Metoda **setCH**
  - Tato metoda slouží pro kontrolu duplicity generovaných bodů, pro kontrolu alespoň 3 bodů, k zavolání příslušného algoritmu pro vypočtení konvexní obálky a k omezení konvexní obálky na striktně konvexní obálku. Metoda počítá dobu trvání výpočetních algoritmů. Jejím návratovým typem je **double**.
  - Vstup
    - **std::string &selected_algorithm** - uživatelsky vybraný algoritmus pro počítání konvexní obálky
  - Výstup
    - Čas trvání výpočtu.
- Metoda **setRect**

  - Tato metoda slouží pro zavolání algoritmu pro výpočte minimálního ohraničujícího obdélníku a jeho hlavní linie. Jejím návratovým typem je **void**.
  - Vstup
    * **bool draw_dir_line** - uživatelsky nastavený indikátor, zda-li se má vypočítat hlavní linie minimálního ohraničujícího obdélníku

- Metoda **setPoints**
  - Metoda volající algoritmus pro generování bodů daného počtu a tvaru. Jejím návratovým typem je **void**.
  - Vstup
    * **QSizeF &canvas_size** - rozměr kreslícího plátna pro pozdější určení rozsahu generování bodů
    * **int count** - počet bodů, jež se má generovat
    * **std::string &shape** - tvar, do kterého se body mají generovat

- Metoda **clearCanvas**
  - Metoda, která maže obsah kreslícího okna. Jejím návratovým typem je **void**. Do metody nevstupují žádné parametry.

7.3 SortByXAsc, SortByYAsc, SortByAngleAsc

Třídy sloužící jako sortovací kritérium - podle rostoucí souřadnice x resp. souřadnice y (při stejných souřadnicích x resp. y je druhým kritériem druhá souřadnice) a podle rostoucího úhlu mezi body (při stejném úhlu je druhým kritériem vzdálenosti mezi body).

7.4 Widget

Tato třída slouží ke komunikaci s GUI. Třída dědí od třídy QWidget. Všechny její metody slouží jako sloty k signálům z GUI, nemají žádné vstupní hodnoty a jejich návratovým typem je void.

- Metoda **on_createCHButton_clicked** - reaguje na zmáčknutí tlačítka pro vypočtení konvexní obálky, volá metodu **setCH** z třídy **Draw**, zapisuje čas výpočtu do GUI.
- Metoda **on_generateButton_clicked** - reaguje na zmáčknutí tlačítka pro generování bodů, volá metodu **setPoints** z třídy **Draw**.
- Metoda **on_clearButton_clicked** - reaguje na zmáčknutí tlačítka pro vymazání obsahu kreslícího plátna, volá metodu **clearCanvas** z třídy **Draw**.
- Metoda **on_createRectButton_clicked** - reaguje na zmáčknutí tlačítka pro vypočtení minimálního ohraničujícího obdélníka, volá metodu **setRect** z třídy **Draw**.
- Metoda **on_helpButton_clicked** - reaguje na zmáčknutí tlačítka pro volání nápovědy, volá okno s nápovědou **help_dialog** z třídy **HelpDialog**.

7.5 HelpDialog

Třída sloužící pro vykreslení okna s nápovědou.

8 Přílohy

- Príloha č.1: Testování výpočetních dob algoritmů - "Testovani.pdf"

9 Závěr

9.1 Návrhy na vylepšení

- bla bla

10 Zdroje

1. BAYER, Tomáš. Konvexní obálky [online][cit. 3.1.2019].
   Dostupné z: https://web.natur.cuni.cz/ bayertom/images/courses/Adk/adk9.pdf

2. BAYER, Tomáš. Konvexní obálky [online][cit. 3.1.2019].
   Dostupné z: https://web.natur.cuni.cz/ bayertom/images/courses/Adk/adkcv4.pdf