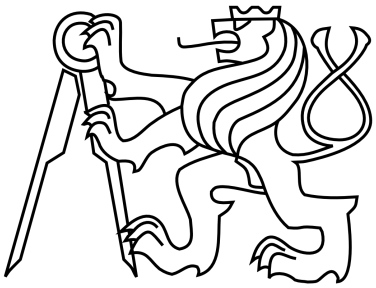
České vysoké učení technické v Praze
Fakulta stavební



155ADKG Algoritmy v digitální kartografii

Množinové operace s polygony

Bc. Lukáš Kettner Bc. Martin Hulín
17.12.2019

Obsah

1 Zadanie

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

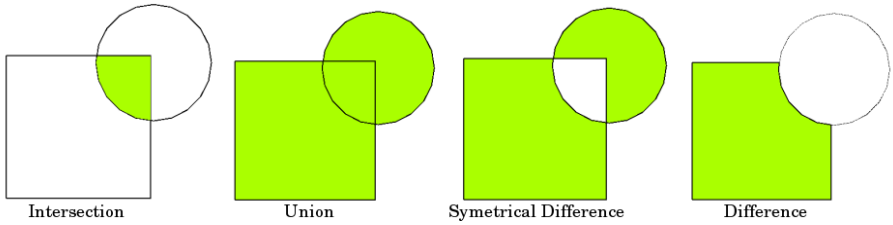
1.1 Bonusové úlohy

V rámci úlohy sú vypracované tieto bonusové úlohy

- Riešenie pre polygóny obsahujúce otvory

2 Popis a rozbor problému

Podstatou úlohy je tvorba aplikácie, v ktorej grafickom rozhraní bude možné prevádzať základné množinové operácie. V rámci úlohy sa zoberáme operáciami prienik, zjednotenie a rozdiel polygónov A a B.



Obr. 1: Typy množinových operácií s polygónmi

3 Popis použitých algoritmov

3.1 Výpočet priesečníkom, zotriedenie a aktualizácia

Využili sme funkciu get2LinesPosition. Táto funkcia kontroluje hrany z polygonu A a polygonu B na existenciu priesečníku. V úlohe bol použitý datový typ QPointFB, ktorý uchováva hodnoty parametrov alfa a beta. Tento typ je odvodený od typu QPointF. Pokiaľ priesečník existoval spočítali sme jeho súradnice.

Pri výpočte priesečníku môžu nastať nasledujúce možnosti :

1. úsečky sú kolineárne
2. úsečky sú rovnobežné
3. úsečky sú rôznobežné
4. úsečky sú mimobežné

Tieto hodnoty sa ukladajúdo datového typu map - kľúč je parameter alfa / beta, hodnota priesečník. Priesečníky boli ďalej vložené do správneho polygónu na správnu pozíciu pomocou funkcie processIntersection.

3.1.1 Implementácia metódy processIntersection

1. Nastavenie tolerancie epsilon
2. $if((t >= epsilon)\&\&(t <= 1 - epsilon)$
3. $i+ = 1$
4. $polygon.insert(polygon.begin()+i, pi$ // priradť priesečník polygónu na pozíciu

3.1.2 Implementácia funkcie computePolygonIntersection

1. Cyklus $for(inti = 0; i < pa.size(); i++)$ - prechádame celý polygón A
2. Vytvotenie map $std :: map < double, QPointFB > intersections$
3. Cyklus $for(intj = 0; j < pb.size(); j++)$ - prechádame celý polygón B

4. $if(get2LinesPosition(...) == INTERSECTED$ podmienka ak existuje prisečník

5. Získaj hodnoty alpha, beta, ulož priesečník do mapy na základe alpha $intersections[alpha] = p_i$

6. $processIntersection(pi, beta, pb, j)$

7. Ak bol nájdený aspoň jeden priesečník

8. prejdi mapu $for(std :: pair < double, QPointFB > item : intersections)$

9. získaj druhú hodnotu z páru $QPointFBpi = item.second$

10. $processIntersection(pi, alfa, pa, i)$

3.2 Ohodnotenie vrcholov

Tento algoritmus uplatňuje ako ohodnocovacie pravidlo polohu vrcholu v polygóne voči druhému vrcholu. Rozsah hodnotenia v závislosti na polohe môže byť Inner, Outer, On. Tieto hodnoty boli uložené do nového datového typu TPointPolygonPosition. K určeniu polohy sme využili Winding Number algoritmus.

3.2.1 Implementácia metódy setPosition

1. Cyklus $for(inti = 0; i < n; i + +)$ - prechádame celý polygón A

2. Výpočet stredového bodu hrany

3. $doublemx = (pa[i].x() + pa[(i + 1)\%n].x())/2;$

4. $doublemy = (pa[i].y() + pa[(i + 1)\%n].y())/2;$

5. Uloženie bodu $QPointFBm(mx, my)$;

6. Určenie polohy metodou Winding Number $TPointPolygonPositionposition = positionPointPolygonWinding(m, pb)$;

7. Uloženie pozície počiatočného vrholu hrany

3.3 Ohodnotenie hran

Výber hran pre množinové operácie znázorňuje nasledujúca tabuľka.

Operácia	PolygonA	PolygonB
Union	outer	outer
Intersect	inner	inner
DifferenceAB	outer	inner
DifferenceBA	inner	outer

3.4 Vytvorenie hran

Vrcholy, ktorým náleží príslušne ohodnotenie sme následne spojili do hrán a uložili do vektoru, ktorý je vykreslovaný.

3.4.1 Implementácia metódy selectEdges

1. Cyklus $for(inti = 0; i < n; i++)$ prechádzame celý polygon
2. Nájdenie vhodnej hrany
3. $Edgee(pol[i], pol[(i+1)\%pol.size()])$; vytvorenie hrany
4. $edges.push_back(e)$; pridanie hrany do vektoru hran

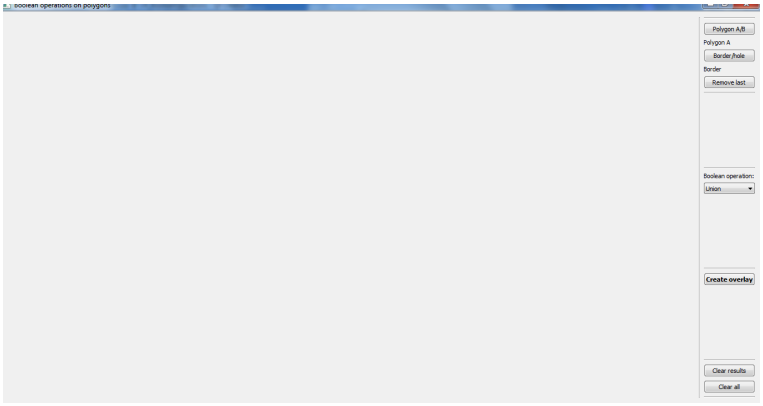
4 Vstupné dáta

Vstupnými dátami sú dva polygóny, naklikané ručne v grafickom rozhraní aplikácie. Pomocou tlačítka PolagonA/B je možné prepínať medzi kresbou jednotlivých polygónov.

5 Výstupné - generovanie množinových operácií

Výstupnými dátami je graficky reprezentované zobrazenie množinových operácií.

6 Ukážka vytvorenej aplikácie



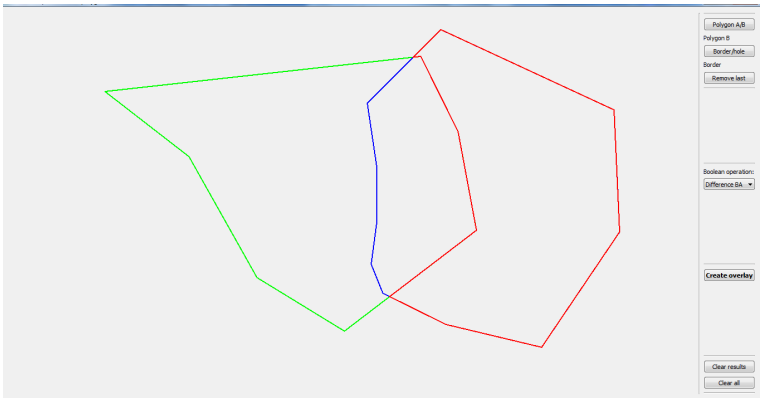
Obr. 2: Ukážka grafického rozhrania aplikácie

Obr. 3: Ukážka grafického rozhrania aplikácie - 2 polygóny

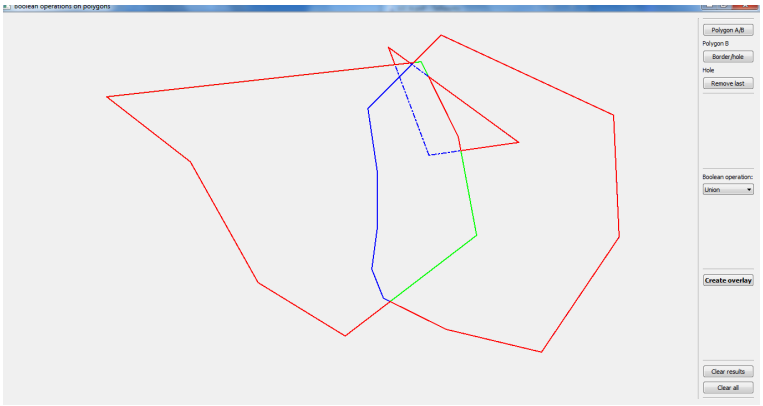
Obr. 4: Ukážka grafického rozhrania aplikácie - Union

Obr. 5: Ukážka grafického rozhrania aplikácie - Intersect

Obr. 6: Ukážka grafického rozhrania aplikácie - Difference AB



Obr. 7: Ukážka grafického rozhrania aplikácie - Difference BA



Obr. 8: Ukážka grafického rozhrania aplikácie - Holes pri operácii Union

7 Dokumentácia

7.1 Trieda Algorithms

Triedu Algorithms sme použili pre deklarovanie funkcií pre výpočtové algoritmy tvorby konvexnej obálky a pre deklaráciu ich pomocných metód.

7.1.1 Členské premenné

static QPoint pivot

- bod obsahujúci súradnice pivota

QPoint kolinear X point

- bod obsahujúci súradnice pomocného bodu, kolineárneho s osou X

7.1.2 Metódy

getAngle2Vectors

- Slúži k určeniu uhlu medzi dvoma priamkami. Jej návratovou hodnotou je double
- na vstupe má : súradnice bodov p_1, p_2, p_3, p_4 určujúcich prvú a druhú priamku
- výstupom je hodnota uhlu medzi priamkami

getPointLinePosition

- Slúži na určenie polohy bodu voči priamke. Jej návratovou hodnotou je integer.
- na vstupe má : súradnice určovaného bodu q , súradnice bodov priamky p_1 p_2
- na výstupe hodnoty :

  -1 – bod sa nachádza na priamke

  0 – bod sa nachádza vpravo od priamky

  1 – bod sa nachádza vľavo od priamky

getPointLineDistance

- je pomocnou metódou pre metódu positionPointPolygonWinding. Slúži na určenie polohy bodu voči priamke. Jej návratovou hodnotou je double.
- na vstupe má : súradnice určovaného bodu q , súradnice bodov priamky p_1 p_2
- na výstupe hodnotu vzdialenosti určovaného bodu od priamky.

Distance

- Slúži na porovnanie vzdialenosti dvoch bodov.

- na vstupe má : súradnice bodov p_1 a p_2
- na výstupe hodnotu vzdialenosti medzi dvoma bodmi.

Angle

- Slúži na porovnanie veľkosti dvoch uhlov.
- na vstupe má : súradnice bodov p_1 a p_2
- na výstupe väčší uhol.

jarvisScan, qHull, grahamScan, sweepLine

- Metódy slúžia k výpočtu konvexnej obálky podľa zvoleného výpočetného algoritmu. Vstupným typom je QPolygon
- na vstupe je vektror bodov std::vector<QPoint>points
- na výstupe je polygón obsahujúci množinu bodov a konvexnú obálku.

qh

- Metódy slúžia k výpočtu konvexnej obálky podľa algoritmu Quick Hull. Je to pomocná metóda. Vstupným typom je void.
- na vstupe je index počiatočného a koncového bodu deliacej priamky start, end
- std::vector<QPoint>points vektor bodov okolo ktorých vytvárame konvexnú obálku
- QPolygon convexHull ktorý obsahuje body konvexnej obálky
- na výstupe je polygón obsahujúci konvexnú obálku.

7.2 Trieda Draw

Trieda Draw slúži ku grafickému vykresleniu množiny modov a konvexnej obálky nad touto množinou.

7.2.1 Členské premenné

std::vector<QPoint>points

- vektor bodov okolo ktorých vytvárame konvexnú obálku

QPolygon ch

- polygón obsahujúci body konvexnej obálky

7.2.2 Metódy

paintEvent

- slúži k vykresleniu naklikaných a vygenerovaných bodov, vykresleniu konvexnej obálky. Návratovým typom je void.

void mousePressEvent

- slúži k vykresleniu bodu stlačením tlačidla myši, v okamihu stlačenia tlačidla na myši sa uložia súradnice bodu do vektoru points. Návratovým typom je void.

void clearCH

- slúži k vymazaniu konvexnej obálky. Návratovým typom je void.

void clearPoints

- slúži k vymazaniu množiny bodov. Návratovým typom je void.

std::vector<QPoint>getPoints

- vektor, ktorý slúži k vráteniu množiny bodov points.

setCH

- slúži na prevedenie konvexnej obálky do vykresľovacieho okna.

generatePoints

- slúži ku generovaniu množiny bodov. Na vstupe je zadaná metóda, počet bodov, šírka a výška.

- na výstupe je vygenerovaná množina bodov podľa užívateľského zadania.

std::vector<QPoint>generatePointsU2

- slúži ku generovaniu množiny bodov. Na vstupe je zadaná metóda, počet bodov, šírka a výška.
- na výstupe je vygenerovaná množina bodov podľa užívateľského zadania. Generovanie konvexnej obálky prebehne automaticky 10x pre všetky typy tvaru generovanej množiny bodov a počty generovaných bodov v intervale od 1 000 do 1 000 000.

7.3 Tridy SortByX, SortByY

Sú to triedy, ktoré obsahujú zoraďovacie kritériá. Pomocou týchto funkcií zoradíme súbor bodov podľa X alebo podľa Y súradnice.

7.4 Trieda Widget

Tieda Widget obashuje metódy ktoré sú odkazom na sloty umožňujúce vykonávať príkazy z grafického rozhrania aplikácie. Nemajú žiadne vstupné hodnoty, návratovým typom je void.

7.4.1 Metódy

on_pushButton_createCH_clicked

- tlačidlo **Create convex hull** po kliknutí naň sa vygeneruje konvexná obálka množiny bodov

on_pushButton_clearPoints_clicked

- tlačidlo **Clear points** po kliknutí naň sa vymaže množina bodov

on_pushButton_clearCH_clicked

- tlačidlo **Clear convex hull** po kliknutí naň sa vymaže konvexná obálka

on_pushButton_generatePoints_clicked

- tlačidlo **Generate points** po kliknutí naň sa vygenerujú body v zvolenom tvare a počte

on_pushButton_solveU2_clicked

- tlačidlo **Solve U2** po kliknutí naň sa vykoná automatické generovanie konvexnej obálky 10x pre každý typ rozmiestnenia bodov (raster, kruh, náhodné rozmiestnenie bodov) a počet bodov n $\in$ $\langle$ 1000, 5000, 10 000, 25 000, 50 000, 75 000, 100 000, 250 000, 500 000, 750 000, 1000 000 $\rangle$. Pre každé generovanie konvexnej obálky je počítaná doba behu, ktorá sa spolu s počtom a typom generovaných bodov, typom algoritmu a poradím opakovania je ukladaná do textového súboru

8 Záver

Výsledkom úlohy je funkčná aplikácia a grafická prezentácia doby trvania jednotlivých algoritmov pre rôzne množiny bodov. Po vypracovaní úlohy sme došli k nasledujúcemu záveru. Najvhodnejší algoritmus pre generovanie konvexných obálok je Sweep Line. Táto metóda dosiahla najlepšie výsledky vo všetkých testovacích prípadoch. Obzvlásť výrazný časový rozdie oproti ostatným algoritmoml bol pri zväčšujúcej sa množine bodov usporiadanej v kruhovom tvare. Algoritmy Graham Scan a Jarvis Scan sú o poznanie pomalšie, hlavne pri rastri a náhodne generovanej množine bodov. Algoritmus Quick Hull dosahuje pre rastrovú a náhodnú množinu bodov takmer totožnú rýchlosť ako algoritmus Sweep Line, pri kruhovom rozložení bodov je pomalší, obdobne rýchly ako Graham Scan.

V úlohe sme sa neimplementovali generovanie množiny bodov v tvare elipsy a star - shaped rozloženia. Bolo by zaujímavé porovnať doby behov algoritmov aj pre takéto rozloženia množín bodov.

Zoznam obrázkov