Algoritmy v digitální kartografii

Konvexní obálky

Zimní semestr 2018/2019

Tereza Kulovaná
Markéta Pecenová

Obsah

1 Zadání

Zadání úlohy bylo staženo ze stránek předmětu 155ADKG.

Vstup: množina $P = \{p_1, ..., p_n\}$, $p_i = [x, y_i]$.

Výstup: $\mathcal{H}(P)$.

Nad množinou P implementujete následující algoritmy pro konstrukci $\mathcal{H}(P)$:

- Jarvis Scan,
- Quick Hull,
- Swep Line.

Vstupní množiny bodů včetně vygenerovaných konvexních obálek vhodně vizualizujte. Pro množiny $n \in < 1000, 1000000 >$ vytvořte grafy ilustrující doby běhu algoritmů pro zvolená n. Měření proveďte pro různé typy vstupních množin (náhodná množina, rastr, body na kružnici) opakovaně (10x) a různá n (nejméně 10 množin) s uvedením rozptylu. Naměřené údaje uspořádejte do přehledných tabulek.

Zamyslete se nad problematikou možných singularit pro různé typy vstupních množin a možnými optimalizacemi. Zhodnoťte dosažené výsledky. Rozhodněte, která z těchto metod je s ohledem na časovou složitost a typ vstupní množiny P nejvhodnější.

Hodnocení:

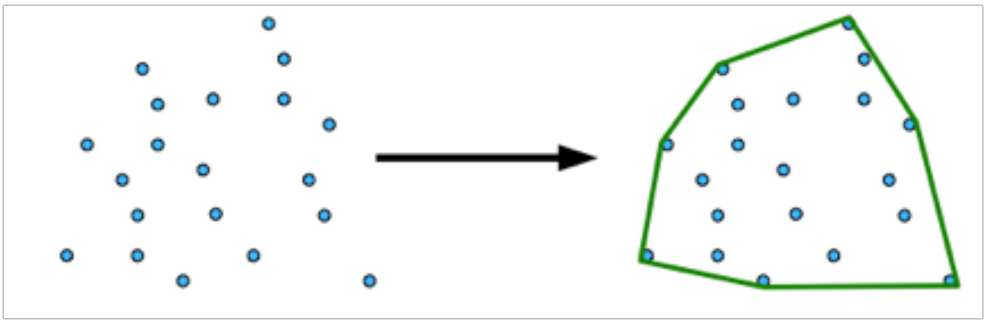
Krok	**Hodnocení**
Konstrukce konvexních obálek metodami Jarvis Scan, Quick Hull, Seep Line.	15b
Konstrukce konvexní obálky metodou Graham Scan	*+5b*
Konstrukce striktně konvexních obálek pro všechny uvedené algoritmy.	*+5b*
Ošetření singulárního případu u Jarvis Scan: existence kolineárních bodů v datasetu.	*+2b*
Konstrukce Minimum Area Enclosing box některou z metod (hlavní směry budov).	*+5b*
Algoritmus pro automatické generování konvexních/nekonvexních množin bodů různých tvarů (kruh, elipsa, čtverec, star-shaped, popř. další).	*+4b*
Max celkem:	**36b**

V rámci této úlohy byly implementovány bonusové úlohy č. xx.

2 Popis a rozbor problému

Úloha **Konvexní obálky** se zabývá vytvořením aplikace, která nad vybranou vstupní množinou S vytvoří tzv. konvexní obálku. Konvexní obálka je nejmenší konvexní mnohoúhelník C obsahující všechny body z množiny S. V rámci úlohy se testuje výpočetní rychlost použitých algoritmů při konstrukci obálek nad danými vstupními množinami bodů.

Své využití konvexní obálky nalézají v mnoha oborech. V kartografii se hojně využívají při detekci tvarů a natočení budov pro tvorbu minimálních ohraničujících obdélníků (???). Dále jsou vhodné pro analýzu tvarů či shluků. Konvexní obálky lze sestrojit v libovolném $\mathbb{R}^n$ prostoru, avšak pro účely této úlohy byl uvažován pouze prostor $\mathbb{R}^2$.



Obrázek 1: Ukázka konvexní obálky (*zdroj*)

Vzniklá aplikace k tvorbě konvexní obálek využívá čtyř výpočetních algoritmů: *Jarvis Scan*, *Quick Hull*, *Sweep line* a *Graham Scan*. Čtvrtý z uvedených algoritmů patří mezi bonusové úlohy, které bylo možno implementovat.

3 Algoritmy

Tato kapitola se zabývá popisem algoritmů, které byly v aplikaci implementovány.

3.1 Jarvis Scan

Prvním zvoleným algoritmem je *Jarvis Scan*. Způsob, jakým vytváří konvexní obálku, nápadně připomíná balení dárků (proto je též občas nazýván jako *Gift Wrapping Algorithm*). Mezi nevýhody tohoto algoritmu patří nutnost předzpracování dat a nalezení tzv. pivotu. Algoritmus dále není vhodný pro velké množiny bodů a ve vstupní množině S nesmí být žádné tři body kolineární. Časová náročnost algoritmu je až $O(n^2)$, jeho výhodou však je velmi snadná implementace.

Mějme množinu bodů S a pivota $q \in S$, jehož souřadnice Y je minimální ze všech bodů, a přidejme ho do konvexní obálky H. Následně do H přidejme takový bod, který s posledními dvěma body přidanými do konvexní obálky svírá maximální úhel. Na začátku výpočtu je nutno inicializovat pomocný bod s, jehož souřadnice X je minimální a Y shodná s pivotem q, který zajistí dostatečný počet bodů pro výpočet prvního úhlu. Algoritmus končí ve chvíli, kdy nově přidaným bodem do konvexní obálky H je opět pivot q.

Zjednodušený zápis algoritmu lze zapsat způsobem uvedeným níže:

1. Nalezení pivota q: $q = \min(y)$
2. Inicializace pomocného bodu s: $s = [\min(x), \min(y)]$
3. Proveď: $q \in H$
4. Inicializace: $p_{j-1} = s$, $p_j = q$
5. opakuj kroky I–III, dokud $p_{j+1} \neq q$:
   I. Najdi p_{j+1}: $\sphericalangle p_{j-1}, p_j, p_{j+1} = \max$
   II. Proveď: $p_{j+1} \in H$
   III. Přeindexuj: $p_{j-1} = p_j$, $p_j = p_{j+1}$

Singularity!!!!

3.2 Quick Hull

Druhý algoritmus použitý v aplikaci je *Quick Hull*, který k výpočtu konvexní obálky využívá strategii *Divide and Conquer*. Hlavní výhodou algoritmu je jeho rychlost, která není ovlivňována velkým počtem rekurzivních kroků, jak tomu bývá u jiných algoritmů. Časová náročnost výpočtu bývá v nejhorším případě $O(n^2)$ a nastává tehdy, pokud všechny body množiny S náleží konvexní obálce H.

Mějme body q_1, resp. q_3, jejichž souřadnice X je minimální, resp. maximální ze všech bodů z množiny S. Veďme těmito body pomyslnou přímku, která prostor rozdělí na horní (S_U) a dolní (S_L) polorovinu. Zbylé body množiny S roztřídíme do daných polorovin podle jejich pozice od přímky. V polorovině následně hledáme bod, který je od dané přímky nejvzdálenější, přidáme ho do konvexní obálky dané poloroviny a přímkami spojíme bod s krajními body přímky předchozí. Proces opakujeme, dokud od nově vzniklých přímek již neexistují vhodné body. Na závěr do konvexní obálky H přidáme bod q_3, body konvexní obálky H_U z poloroviny S_U, bod q_1 a nakonec body konvexní obálky H_L poloroviny S_L. Do konvexní obálky H je důležité přidávat body v tomto pořadí, jinak by došlo k nesprávnému vykreslení konvexní obálky H.

Algoritmus *Quick Hull* se skládá z globální a lokální procedury. Globální část zahrnuje rozdělení množiny na dvě poloroviny a spojení již nalezených bodů konvexních obálek polorovin do jediné H. V lokální části se rekurzivně volá metoda, která hledá nejvzdálenější body od přímky v dané polorovině.

Mezi singularity!!!

Globální procedura:

1. Inicializace: $H = \emptyset$, $S_U = \emptyset$, $S_L = \emptyset$
2. Nalezení $q_1 = \min(x)$, $q_3 = \max(x)$
3. Proveď: $q_1 \in S_U$, $q_3 \in S_U$, $q_1 \in S_L$, $q_3 \in S_L$

4. Potupně pro všechna $p_i \in S$:

   Podmínka (p_i je vlevo od q_1, q_3) $\rightarrow S_U$

   Jinak $p_i \rightarrow S_L$

5. Proveď: $q_3 \in H$
6. Lokální procedura pro S_U
7. Proveď: $q_1 \in H$
8. Lokální procedura pro S_U

Lokální procedura nad polorovinou S_i:

I. Pro všechny $p_i \in S_i$ kromě bodů přímky:

   Podmínka (p_i vpravo) $\rightarrow$ vzdálenost d_i

   Podmínka ($d_i > d_{max}$) $\rightarrow d_{max} = d_i$, $p_{max} = p_i$

II. Podmínka (bod p_{max} $\exists$)

   opakuj krok I. nad první nově vzniklou přímku

   $p_{max} \in H_i$

   opakuj krok I. nad druhou nově vzniklou přímku

3.3 !Sweep Line

3.4 !Graham Scan

4 !Problematické situace

V úloze bylo nutné ošetřit sigularity, zda bod q neleží v hraně některého z polygonů či v jejich vrcholech. Pro vyřešení tohoto problému byla použita metoda *getDistanceEdgeQ* třídy **Algorithms**, která porovnává vzdálenost dvou bodů p_1 a p_2 na přímce p se sumou vzdáleností těchto bodů k danému bod q. Je-li rozdíl vzdáleností menší než mezní hodnota ϵ, je bod q vyhodnocen, že leží na přímce.

$| d_{p_1,p_2} - \sum(d_{p_1,q} + d_{q,p_2}) | < \epsilon \rightarrow q$ náleží přímce p.

5 !Vstupní data

Aplikace požaduje dva druhy vstupních dat:

1. soubor daných polygonů
2. daný bod q

Seznam bodů jednotlivých polygonů je uložen v textovém souboru polygon.txt. Pro vykreslení jednotlivých polygonů v aplikaci je nutno tento soubor do aplikace nahrát pomocí tlačítka *Load*. K vygenerování souřadnic bodů byla použita online aplikace ze stránek mobilefish.com. Struktura souboru s polygony je následující:

První řádek: počet polygonů v souboru
Sloupec 1: číslo polygonu, jehož součástí daný bod je
Sloupec 2: souřadnice X daného bodu polygonu
Sloupec 3: souřadnice Y daného bodu polygonu

Bod q není součástí textového souboru, do aplikace vstupuje na základě ručního zadání uživatelem. Pro zadání bodu je nutné v aplikaci kliknout levým tlačítkem myši do okna s polygony.

6 !Výstupní data

Vytvořená aplikace dále vypisuje dobu, po kterou výpočet probíhal v závislosti na zvoleném výpočetním algoritmu, počtu vstupním bodů a na jejich prostorovém uspořádání. Výstupem úlohy je vypsání v grafickém okně aplikace, v jaké poloze se analyzovaný bod vůči polygonům nachází. Polygony, kterým bod náleží, jsou barevně zvýrazněny.

6.1 Grafy a tabulky

7 !Aplikace

V následují kapitole je představen vizuální vzhled vytvořené aplikace tak, jak ji vidí prostý uživatel.

8 !Dokumentace

Tato kapitola obsahuje dokumentaci k jednotlivým třídám.

8.1 !Algorithms

Třída *Algorithms* obsahuje metody, které určují polohu zvoleného bodu q vzhledem k polygonu. Dále obsahuje pomocné metody pro výpočet úhlu mezi třemi body a na zjištění polohy bodu vzhledem k přímce.

8.1.1 get2LinesAngle

Metoda **get2LinesAngle** počítá úhel mezi dvěma přímkami. Na vstupu jsou 4 body typu `QPoint`, návratová hodnota typu `double` vrací velikost úhlu v radiánech. Body p_1 a p_2 definují první přímku, zbylé dva body druhou přímku.

Input:

- `QPoint` p_1
- `QPoint` p_2
- `QPoint` p_3
- `QPoint` p_4

Output:

- `double`

8.1.2 getPointLineDistance

8.1.3 getPointLinePosition

Metoda **getPointLinePosition** určuje polohu bodu q vzhledem k přímce tvořené dvěma body. Na vstupu jsou 3 body typu `QPoint`, návratová hodnota je nově definovaný typ `TPosition`.

Input:

- `QPoint` q
- `QPoint` a
- `QPoint` b

Output:

- `LEFT` $\rightarrow$ bod se nachází vlevo od přímky
- `RIGHT` $\rightarrow$ bod se nachází vpravo od přímky
- `ON` $\rightarrow$ bod se nachází na přímce

8.1.4 CHJarvis

8.1.5 QHull

8.1.6 qhloc!!!!!podrtzitko

8.2 !Draw

Třída *Draw* obsahuje metody, které slouží k vykreslení polygonů, analyzovaného bodu q a výstupních dat.

8.2.1 !paintEvent

Metoda **paintEvent** vykresluje polygony a analyzovaný bod q. Návratová hodnota je typu *void*.

Input:

- QPaintEvent *e

8.2.2 !mousePressEvent

Metoda **mousePressEvent** slouží k načtení souřadnic bodu q. Návratová hodnota je typu *void*.

Input:

- QMouseEvent *e

8.2.3 !clearCanvas

Metoda **clearCanvas** slouží k vymazání všech vykreslených dat. Metoda neobsahuje žádné proměnné na vstupu a návratová hodnota je typu *void*.

8.2.4 !fillPolygon

Metoda **fillPolygon** barevně vyšrafuje polygon, ve kterém se nachází analyzovaný bod q. Návratová hodnota je typu *void*.

Input:

- $std :: vector<std :: vector<QPoint>>$ poly_fill

8.2.5 !loadPolygon

Metoda **loadPolygon** slouží k nahrání bodů jednotlivých polygonů do aplikace. Součástí metody je i kontrola, zda se soubor úspěšně nahrál, zda vůbec obsahuje nějaké polygony a zda jsou polygony tvořeny aspoň 3 body. Návratová hodnota je typu *QString* vrací hlášku, zda byly polygony úspěšně nahrány.

Input:

- const char* path

Output:

- QString

8.3 SortByXAsc

Třída obsahuje jedinou metodu(??konstruktor) **SortByXAsc**, která má na vstupu dva body typu `QPoint`, návratová hodnota je typu `bool`. Metoda vrací bod s nižší souřadnicí X. Mají-li oba body shodnou souřadnici X, vrací bod s nižší souřadnicí Y.

Input:

- `QPoint` p_1
- `QPoint` p_2

Output:

- 0 $\rightarrow$ bod p_2 má nižší x souřadnici
- 1 $\rightarrow$ bod p_1 má nižší x souřadnici

8.4 SortByYAsc

Třída obsahuje jedinou metodu(??konstruktor) **SortByYAsc**, která má na vstupu dva body typu `QPoint`, návratová hodnota je typu `bool`. Metoda vrací bod s nižší souřadnicí Y. Mají-li oba body shodnou souřadnici Y, vrací bod s nižší souřadnicí X.

Input:

- `QPoint` p_1
- `QPoint` p_2

Output:

- 0 $\rightarrow$ bod p_2 má nižší souřadnici Y
- 1 $\rightarrow$ bod p_1 má nižší souřadnici Y

8.5 !Widget

Metody třídy **Widget** slouží pro práci uživatele s aplikací. Až na jednu výjimku nemají metody na vstupu nic, návratové hodnoty všech metod jsou typu *void*.

8.5.1 !writeResult

Metoda **writeResult** vrací polohu bodu q vzhledem k polygonu na základě vstupní hodnoty typu *int*.

Input:

- int

8.5.2 !on_load_button_clicked

Metoda **on_load_button_clicked** načítá data z textového formátu. Uživatel sám vyhledává cestu k požadovanému souboru.

8.5.3 !on_analyze_button_clicked

Metoda **on_analyze_button_clicked** vypisuje na obrazovku polohu bodu q vzhledem k polygonům po zvolení požadovaného algoritmu a kliknutí na tlačítko *Analyze*.

8.5.4 !on_clear_button_clicked

Metoda **on_clear_button_clicked** vrací aplikaci do výchozí polohy smazáním všeho, co bylo vykresleno.

9 !Závěr

V rámci úlohy *Geometrické vyhledávání bodu* byla vytvořena aplikace, která určuje polohu analyzovaného bodu q vzhledem k polygonu. Z důvodu větší časové náročnosti úlohy než obě autorky původně očekávaly nebyly implementovány všechny bonusové úlohy a některé části kódu mohly být řešeny lépe. Tato opravená verze technické zprávy je aktualizována o několik vzorců a o text v sekci Zadání.

Jedná se například o použití tříd QPoint a QPolygon, které na vstupu mají hodnoty typu *int* a které mohly být nahrazeny třídami QPointF a QPolygonF, jež na vstupu mají hodnot typu *float*, což je pro práci se souřadnicemi bodů praktičtější. Dále jako ne zrovna nejšťastnější řešení hodnotíme množství vstupních hodnot, které mají na vstupu metody *getPointLinePosition*, *getTwoVectorsAngle* a *getDistanceEdgeQ* třídy **Algorithms**. Vhodné by bylo nahradit jednotlivé souřadnice třídou QPoint, resp. QPointF.

Dále mohlo být implementováno více kontrolních podmínek pro načítání bodů polygonu z textového souboru, například ošetření, zda soubor neobsahuje text, zda všechny body mají x a y souřadnici apod. Umisťování bodu q do hran či vrcholů polygonů vyžaduje notnou dávku trpělivosti, aby se zobrazil korektní výsledek. Experimentálně bylo zjištěno, že aplikace zvládá lépe (tzn. je potřeba méně pokusů na kliknutí) zobrazovat polohu bodu q ve vrcholech než v hranách, ač obě situace jsou v kódu ošetřené.

Úloha přinesla i pozitivní přínos v tom směru, že se autorky mohly potrénovat v psaní kódu v jazyce C++ a oprášit své znalosti psaní v prostředí LaTeX.

10 Zdroje

1. *BAYER, Tomáš. Geometrické vyhledávání bodů* [online][cit. 24. 10. 2018]. Dostupné z: https://web.natur.cuni.cz

2. *CS 312 - Convex Hull Project* [online][cit. 10. 11. 2018]. Dostupné z: http://mind.cs.byu.edu