

Algoritmy v digitální kartografii

Konvexní obálky

Zimní semestr 2018/2019

Tereza Kulovaná
Markéta Pecenová

Obsah

1	Zadání	2
2	Popis a rozbor problému	3
3	Algoritmy	3
3.1	Jarvis Scan	3
3.2	Quick Hull	4
3.3	Sweep Line	5
4	Problematické situace	6
5	Vstupní data	7
6	Výstupní data	7
6.1	Grafy a tabulky	7
7	!Aplikace	8
8	Dokumentace	10
8.1	Algorithms	10
8.1.1	CHJarvis	10
8.1.2	QHull	10
8.1.3	qh_loc	10
8.1.4	CHSweepLine	11
8.1.5	get2LinesAngle	11
8.1.6	getPointLineDistance	11
8.1.7	getPointLinePosition	12
8.2	Draw	12
8.2.1	paintEvent	12
8.2.2	setCh	12
8.2.3	generateSet	12
8.2.4	clearAll	13
8.3	SortByXAsc	13
8.4	SortByYAsc	13
8.5	Widget	14
8.5.1	on_ch_button_clicked	14
8.5.2	on_clear_button_clicked	14
8.5.3	on_set_button_clicked	14
9	!Závěr	15
10	Zdroje	16

1 Zadání

Zadání úlohy bylo staženo ze stránek předmětu 155ADKG.

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

Výstup: $\mathcal{H}(P)$.

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

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

Vstupní množiny bodů včetně vygenerovaných konvexních obálek vhodně vizualizujte. Pro množiny $n \in \langle 1000, 1000000 \rangle$ vytvořte grafy ilustrující doby běhu algoritmů pro zvolenou n . Měření proveďte pro různé typy vstupních množin (náhodná množina, rastr, body na kružnici) opakovaně (10x) a různou 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. Rozhodněte, která z těchto metod je s ohledem na časovou složitost a typ vstupní množiny P nejvhodnější.

Hodnocení:

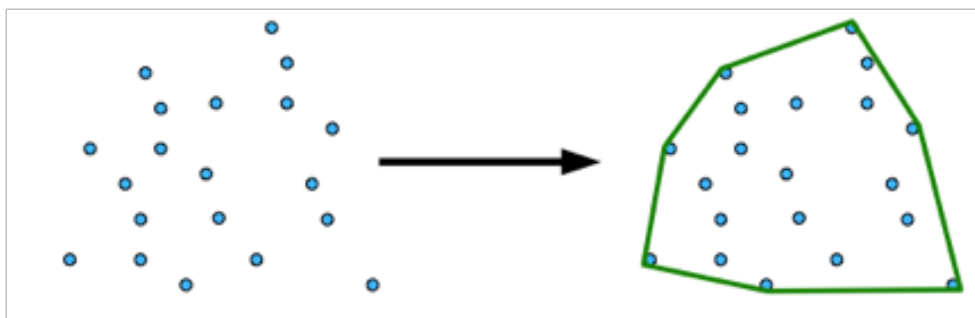
Krok	Hodnocení
Konstrukce konvexních obálek metodami Jarvis Scan, Quick Hull, Sweep Line.	15b
Konstrukce konvexní obálky metodou Graham Scan	+5b
Konstrukce striktně konvexních obálek pro všechny uvedené algoritmy.	+5b
Ošetření singulárního případu u Jarvis Scan: existence kolineárních bodů v datasetu.	+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. 2-3. Bonusová úloha č. 5 byla částečně implementována v rámci základního zadání plus navíc bylo přidáno vykreslení elipsy a čtverce.

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 .

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 sestavit v libovolném \mathbb{R}^n prostoru, avšak pro účely této úlohy byl uvažován pouze prostor \mathbb{R}^2 . V rámci úlohy se testuje výpočetní rychlost použitých algoritmů při konstrukci obálek nad danými vstupními množinami bodů.



Obrázek 1: Ukázka konvexní obálky (zdroj)

Vzniklá aplikace k tvorbě konvexní obálek využívá tří výpočetních algoritmů: *Jarvis Scan*, *Quick Hull* a *Sweep Line*.

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. Proved': $q \in H$
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} : $\angle p_{j-1}, p_j, p_{j+1} = \max$
 - II. Proved': $p_{j+1} \in H$
 - III. Přeindexuj: $p_{j-1} = p_j, p_j = p_{j+1}$

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 a přidává je do konvexní obálky dané poloroviny H_i .

Globální procedura:

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

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. Proved': $q_3 \in H$
6. Lokální procedura pro S_U
7. Proved': $q_1 \in H$
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 $p_{max} \exists$)
 - opakuji krok I. nad první nově vzniklou přímkou
 - $p_{max} \in H_i$
 - opakuji krok I. nad druhou nově vzniklou přímkou

3.3 Sweep Line

Algoritmus *Sweep Line* neboli *Metoda zametací přímky* je dalším z algoritmů, které byly pro vytváření konvexních obálek implementovány. Jeho princip je založen na imaginární přímce, která se postupně přesouvá zleva doprava nad všemi body množiny S . Body, které „přejede“, přidá do dočasné konvexní obálky \bar{H} , která je následně upravena, aby byla konvexní. Pro tento algoritmus je opět nutné předzpracování vstupních dat (seřadit body $\in S$ podle souřadnice X) s náročností $O(n \cdot \log(n))$. Další nevýhodou je citlivost algoritmu na singularity, konkrétně na duplicitní body. Ty je vhodné během předzpracování odstranit.

Algoritmus je postaven na znalosti pozice již vyhodnocených bodů vůči nově přidávanému bodu ukládáním jejich indexů do proměnných p (předchůdce) a n (následník). Zároveň je pro správné fungování algoritmu nutné dodržovat CCW orientaci (proti směru hodinových ručiček). Algoritmus má celkem tři fáze: iniciální a dvě iterativní.

V první fázi seřadíme body p_i z množiny S vzestupně podle souřadnice X . Následně z prvních dvou bodů vytvoříme přímku, jejíž koncové body umístíme do konvexní obálky H a indexy bodů umístíme do n a p . V první iterativní fázi vyhodnotíme, zda další přidávaný bod leží v horní či dolní polorovině v závislosti na jeho souřadnici Y vzhledem k předchozímu bodu, a opět obousměrně vyhodnotíme indexy n a p . Ve druhé iterativní fázi opravujeme dočasnou konvexní obálku \bar{H} na konvexní H vložím horních a dolních tečen a vynecháním nekonvexních vrcholů.

Zjednodušený zápis algoritmu:

1. Seřazení p_i podle souřadnice X

2. Pro body p_0, p_1 proved':

$$n[0] = 1, n[1] = 0$$

$$p[0] = 1, p[1] = 0$$

3. Pro všechna $p_i \in S, i > 1$ proved':

$$\text{Podmínka } (y_i > y_{i-1}) \rightarrow S_U: p[i] = i-1, n[i] = n[i-1]$$

$$\text{Jinak } \rightarrow S_L: n[i] = i-1, p[i] = p[i-1]$$

$$\text{Oprava indexů: } n[p[i]] = i, p[n[i]] = i$$

Dokud $n[n[i]]$ je vpravo od přímky $(i, n[i])$:

$$p[n[n[i]]] = i, n[i] = n[n[i]]$$

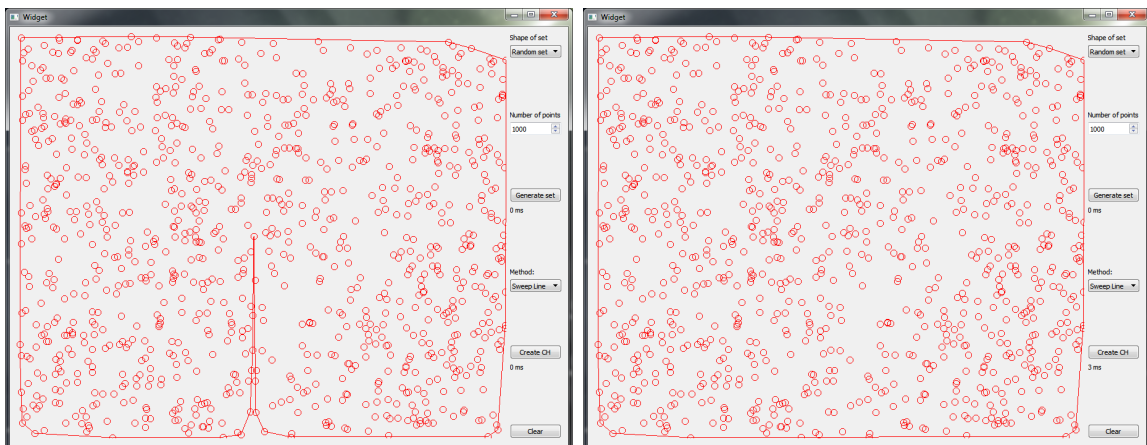
Dokud $p[p[i]]$ je vlevo od přímky $(i, p[i])$:

$$n[p[p[i]]] = i, p[i] = p[p[i]]$$

4 Problematické situace

V algoritmu *Sweep Line* bylo nutné ošetřit singularitu, která způsobovala generování nekonvexní obálky. Konkrétně bylo nutné odstranit duplicitní body. Zároveň však toto opatření rapidně navýšilo výpočetní dobu algoritmu, zejména u množiny s vyšším počtem bodů.

Podmínka $(p_i == p_j) \rightarrow$ bod p_j odstraněn



Obrázek 2: Neopravený (vlevo) vs. opravený (vpravo) Sweep Line algoritmus

Dalším problémem bylo, jak zajistit, aby byl generovaný rastr pravidelný. To bylo ošetřeno zaokrouhlením počtu vstupních bodů směrem nahoru tak, aby po odmocnění vznikl pravidelný rastr.

$$\text{počet bodů v řádce/sloupce} = \text{roundUp}(\sqrt{\text{num_of_points}})$$

Jarvis singularita!!!

5 Vstupní data

Aplikace si na základě ručního zadání vstupních parametrů uživatelem sama vygeneruje potřebná vstupní data.

Z rozbalovací nabídky **Shape of set** uživatel volí prostorové uspořádání generované množiny bodů. Na výběr jsou možnosti *Random set*, *Grid* a *Circle*. V kolonce **Number of points** uživatel volí, kolik bodů bude generováno. Lze tak učinit buď přímým zadáním počtu bodů, nebo zvýšením/snížením počtu bodů o 1000 bodů šipkami na boku. Aplikace omezuje maximální a minimální počet generovaných bodů na interval $<1,1000000>$. Množina bodů se vygeneruje stisknutím tlačítka *Generate set*.

Uživatel má dále možnost volit, jaký výpočetní algoritmus bude použit pro tvorbu konvexní obálky. Rozbalovací nabídka **Method** nabízí celkem tři výpočetní algoritmy: *Jarvis Scan*, *Quick Hull* a *Sweep Line*. Konvexní obálka je generována stisknutím tlačítka *Create CH*. Pokud před spuštěním procesu nebyla vygenerována žádná vstupní množina bodů, uživatel je upozorněn chybovou hláškou.

6 Výstupní data

Vygenerovaná množina bodů a její konvexní obálka je vykreslena v grafickém okně aplikace. Aplikace dále vypisuje časy [ms], jak dlouho trvalo generování množiny a jak dlouho nad danou množinou běžel výpočetní algoritmus.

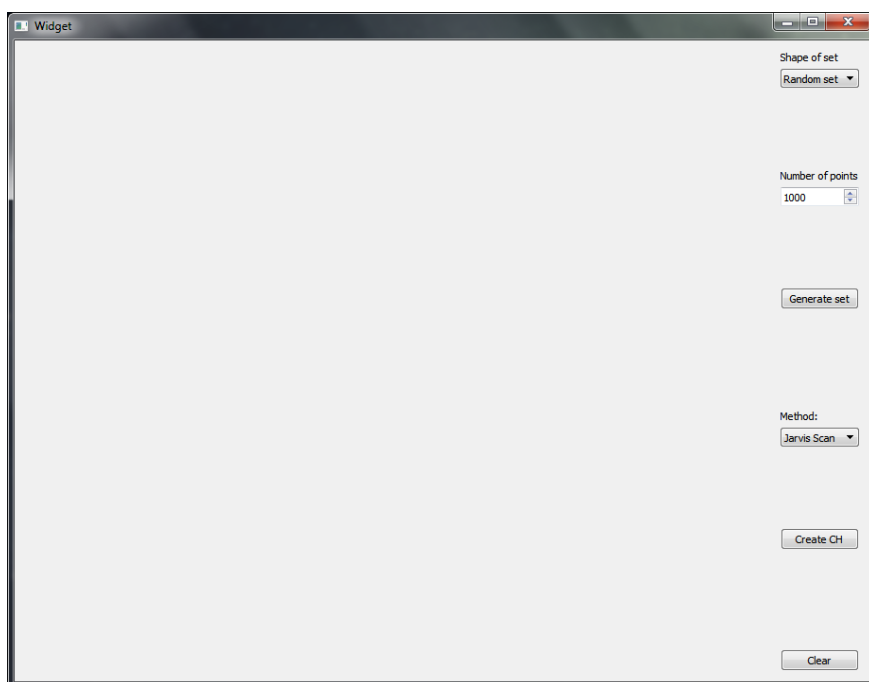
6.1 Grafy a tabulky

V rámci testování byly!!!! grafy
tabulky i s rozptyly

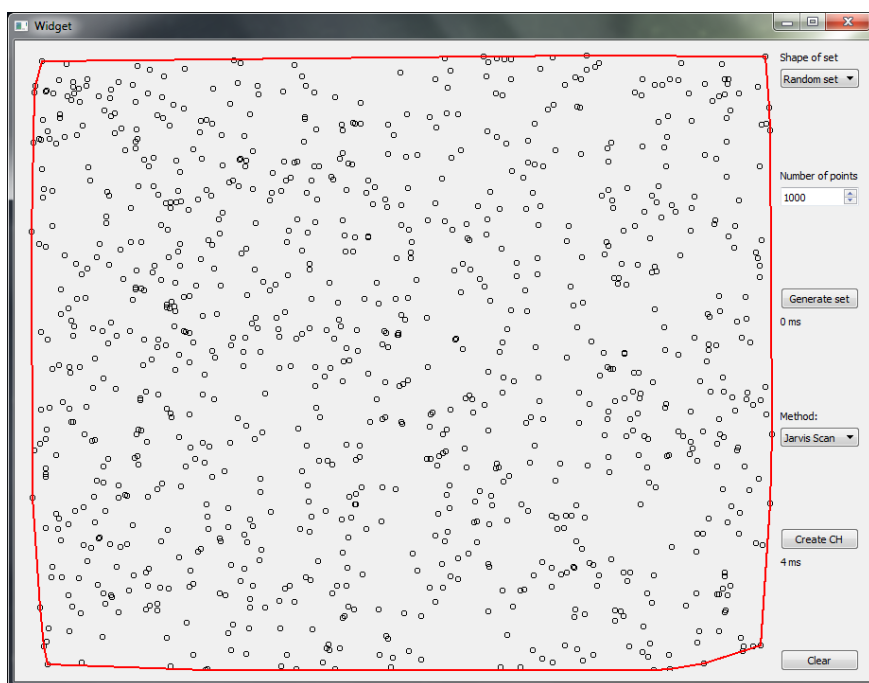
Jméno	Příjmení	Dílo
Linus	Torvalds	Napsal Linuxové jádro
Stallman	Richard	Zakladatel nadace GNU. Autor editoru Emacs
Donald	Knuth	Vytvořil sázecí systém T _E X

7 !Aplikace

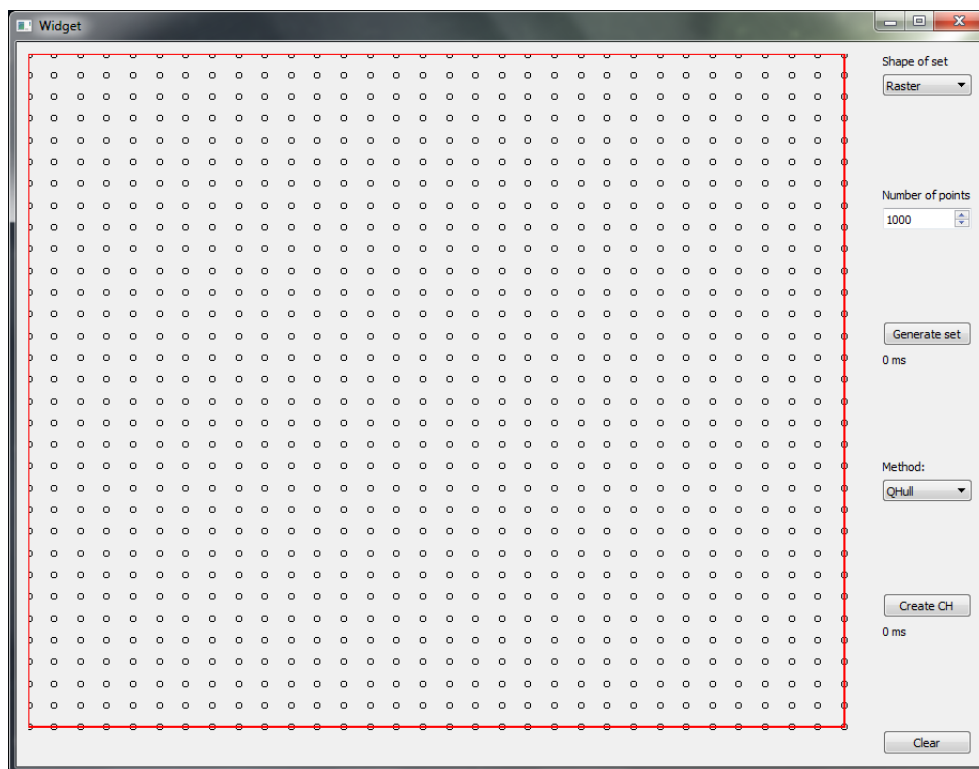
V následující kapitole je představen vizuální vzhled vytvořené aplikace tak, jak ji vidí prostý uživatel.



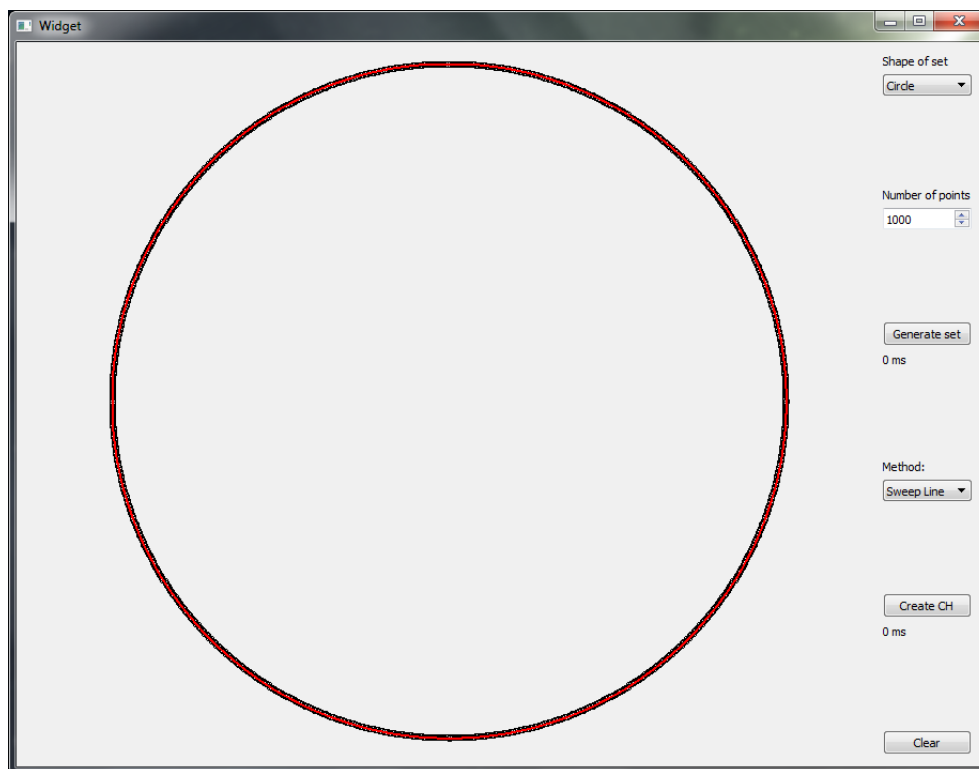
Obrázek 3: Výchozí vzhled aplikace po spuštění



Obrázek 4: Výstup při použití algoritmu *Jarvis Scan* nad množinou *Random* o 1000 bodech



Obrázek 5: Výstup při použití algoritmu *QHull* nad množinou *Grid* o 1000 bodech



Obrázek 6: Výstup při použití algoritmu *Sweep Line* nad množinou *Circle* o 1000 bodech

8 Dokumentace

Tato kapitola obsahuje dokumentaci k jednotlivým třídám.

8.1 Algorithms

Třída *Algorithms* obsahuje tři základní metody, které nad vstupní množinou bodů vytváří konvexní obálky. Dále obsahuje pomocné metody k výpočtu úhlu mezi dvěma přímkami a metody k určování vztahu bodu a přímky.

8.1.1 CHJarvis

Metoda **CHJarvis** vytváří konvexní obálku nad vstupní množinou bodů za použití algoritmu *Jarvis Scan*. Na vstupu je vektor bodů třídy **QPoint**. Návrátová hodnota je polygon třídy **QPolygon**, který obsahuje uspořádané body tvořící konvexní obálku.

Input:

- *vector <QPoint> points*

Output:

- **QPolygon**

8.1.2 QHull

Metoda **QHull** vytváří konvexní obálku nad vstupní množinou bodů za použití algoritmu *Quick Hull*. Na vstupu je vektor bodů třídy **QPoint**. Návrátová hodnota je polygon třídy **QPolygon**, který obsahuje uspořádané body tvořící konvexní obálku.

Input:

- *vector <QPoint> points*

Output:

- **QPolygon**

8.1.3 qh_loc

Metoda **qh_loc** je lokální procedura algoritmu **QHull**, která se volá rekurzivně a hledá nejvzdálenější bod od přímky v dané polorovině a přidává jej do konvexní obálky poloroviny H_i . Na vstupu jsou dvě proměnné typu **int**, které obsahují index počátečního (s) a koncového bodu (e) dané přímky, vektor bodů vstupní poloroviny třídy **QPoint** a polygon třídy **QPolygon**, ve kterém jsou uloženy body konvexní obálky. Návrátová hodnota je typu **void**.

Input:

- **int** s

- `int e`
- `vector <QPoint> ss`
- `QPolygon h`

8.1.4 CHSweepLine

Metoda **CHSweepLine** vytváří konvexní obálku nad vstupní množinou bodů za použití algoritmu *Sweep Line*. Na vstupu je vektor bodů třídy `QPoint`. Návrátová hodnota je polygon třídy `QPolygon`, který obsahuje uspořádané body tvořící konvexní obálku.

Input:

- `vector <QPoint> points`

Output:

- `QPolygon`

8.1.5 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 p1`
- `QPoint p2`
- `QPoint p3`
- `QPoint p4`

Output:

- `double`

8.1.6 getPointLineDistance

Metoda **getPointLineDistance** počítá nejkratší (kolmou) vzdálenost bodu q od přímky tvořené dvěma body. Na vstupu jsou 3 body typu `QPoint`, návratová hodnota typu `double` vrací vzdálenost bodu q od přímky.

Input:

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

Output:

- `double`

8.1.7 `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

8.2 Draw

Třída *Draw* obsahuje metody, které generují a vykreslují vstupní množinu bodů. Dále vykresluje konvexní obálku dané množiny.

8.2.1 `paintEvent`

Metoda **`paintEvent`** vykresluje vygenerovanou vstupní množinu bodů a její konvexní obálku. Návratová hodnota je typu *void*.

Input:

- `QPaintEvent` *e

8.2.2 `setCh`

Metoda **`setCh`** slouží k vymazání všech vykreslených dat. Návratová hodnota metody je typu *void*.

8.2.3 `generateSet`

Metoda **`generateSet`** slouží ke generování vstupní množiny bodů. Na vstupu má čtyři proměnné typu `int`, které definují prostorové uspořádání generovaných bodů, jejich počet a šířku a výšku kreslicí plochy. Metoda defaultně nastavuje počet generovaných bodů na hodnotu 1000 a omezuje maximální možnou hodnotu, kterou lze ručně nastavit, na 1000000 bodů. Rozměr kreslicí plochy definuje maximální možné souřadnice, kterých generované body mohou nabývat, a zároveň je z něj odvozován poloměr vykreslované kružnice a elipsy. Návratová hodnota metody vektor bodů třídy `QPoint`.

Input:

- `int shape_index` (0 → random, 1 → grid, 2 → circle)
- `int num_of_points` (rozsah: 1000 – 1000000)
- `int canvas_width`
- `int canvas_height`

Output:

- `vector <QPoint>`

8.2.4 clearAll

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

8.3 SortByXAsc

Třída **SortByXAsc** 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 p1`
- `QPoint p2`

Output:

- 0 → bod `p2` má nižší *x* souřadnici
- 1 → bod `p1` má nižší *x* souřadnici

8.4 SortByYAsc

Třída **SortByYAsc** 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 p1`
- `QPoint p2`

Output:

- 0 → bod `p2` má nižší souřadnici Y
- 1 → bod `p1` 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 **on_ch_button_clicked**

Metoda **on_ch_button_clicked** na základě uživatelem zvoleného algoritmu generuje konvexní obálku nad vstupní množinou bodů. Metoda zároveň počítá čas [ms], během kterého algoritmus vypočítá konvexní obálku, a vypíše ho do aplikace. Do výpočtu času není zahrnuto vykreslování konvexní obálky.

8.5.2 **on_clear_button_clicked**

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

8.5.3 **on_set_button_clicked**

Metoda **on_set_button_clicked** generuje množinu bodů na základě zadaných vstupních parametrů uživatelem. Metoda zároveň počítá čas [ms], jak dlouho množinu trvalo vygenerovat, a vypíše ho do aplikace.

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 hodnoty 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 potrénoval 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>
3. *SOPUCH, Pavel. LaTeX v kostce* [online][cit. 12. 11. 2018].
Dostupné z: <http://www.it.cas.cz>