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:

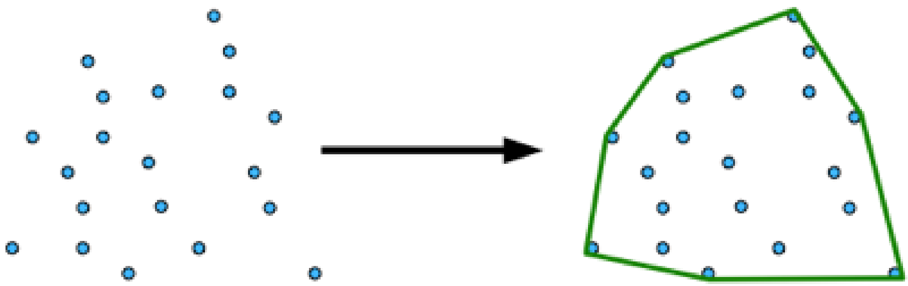
Krok	**Hodnocení**
Konstrukce konvexních obálek metodami Jarvis Scan, Quick Hull, Seep Line.	15b
Konstrukce konvexní obálky metodou Graham Scan	*+5b*
Konstrukce striktně konvexních obálek pro všechny uvedené algoritmy.	*+5b*
Ošetření singulárního případu u Jarvis Scan: existence kolineárních bodů v datasetu.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

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 sestrojit 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. 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: $p_{j-1} = p_j$, $p_j = p_{j+1}$

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. Proveď: $q_1 \in S_U$, $q_3 \in S_U$, $q_1 \in S_L$, $q_3 \in S_L$

4. Postupně 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_L

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

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 $\bar{\text{H}}$, 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 $O(n.\log(n))$. 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 $\bar{\text{H}}$ na konvexní H vložením 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 proveď:

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

   $p[0] = 1$, $p[1] = 0$
3. Pro všechna $p_i \in S$, $i > 1$ proveď:

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

   Jinak $\rightarrow S_L$: n[i] = i-1 p[i] = p[i-1]

   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]]

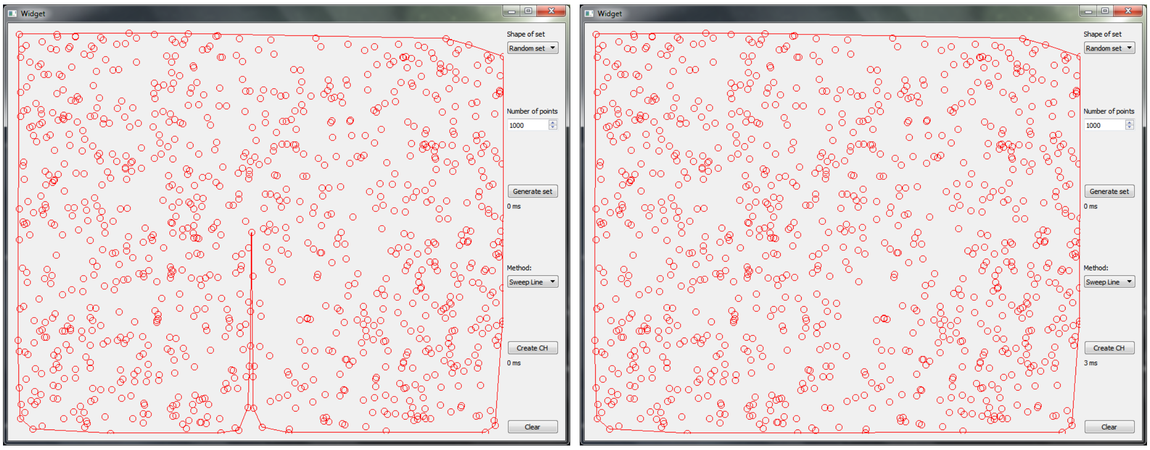
   Dokud $p[p$[i]] je vlevo od přímky (i, p[i]):

   $n[p[p$[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 (viz. Obrázek 2)



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

Algoritmus *Jarvis Scan* generuje špatné výsledky, není-li ošetřeno, že žádné tři body nejsou spolu kolineární. Tento problém se zejména projevoval při generaci setu *Grid*, který jich obsahuje spoustu. Singularita byla ošetřena vypočtením úhlu $\sphericalangle p_{jj}, p_j, p_i$, a pokud byl menší, než stanovená mez ϵ, byl pro další výpočty vybrán bližší z kolineárních bodů.

- Podmínka $(\sphericalangle p_{jj}, p_j, p_i) < \epsilon$

  Podmínka $(d_{p_j,p_i} < d_{min}) \rightarrow$ vyber bod p_i, $d_{min} = d_{p_j,p_i}$

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.

počet bodů v řádce/sloupci $= roundUp(\sqrt{num\_of\_points})$

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 šipkami na boku. Aplikace omezuje minimální a maximální 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.

V rámci testování byly nad množinami bodů *Random set*, *Grid* a *Circle* postupně spuštěny všechny tři algoritmy. Pro každou množinu a algoritmus byla pro daný počet bodů $n = \{1000, 5000, 10000, 25000, 50000, 75000, 100000, 200000, 500000, 1000000\}$ aplikace spuštěna 10x, aby bylo získáno dostatečné množství testovacích dat. Z každé testované množiny bylo tedy získáno celkem 300 testovacích dat. Data byla ukládána do textového souboru a následně zpracována v *Excelu*.

6.1 Tabulky

Níže jsou uvedeny průměrné časy běhu algoritmů pro daný počet bodů n z 10 měření pro jednotlivé množiny. Tabulky se všemi naměřenými hodnotami a vypočtenými rozptyly jsou uvedeny v souboru *testing.xlsx*.

Průměrný čas výpočtu – Random set			
n / algoritmus	Jarvis Scan [ms]	Quick Hull [ms]	Sweep Line [ms]
1000	4.5	0.2	1.1
5000	36.3	1.7	22.5
10000	94.8	3.2	92.6
25000	475.9	6.3	539.3
50000	2078.2	13.1	2091.7
75000	4424.1	19.0	4450.4
100000	8890.2	27.9	7818.6
200000	32830.6	52.0	31694.4
500000	126622.3	130.8	179290.2
1000000	378372.1	245.7	635048.4

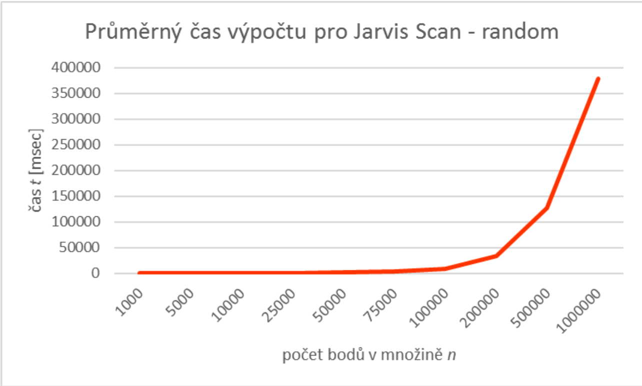
Průměrný čas výpočtu – Grid			
n / algoritmus	Jarvis Scan [ms]	Quick Hull [ms]	Sweep Line [ms]
1000	25.7	0.1	1.0
5000	284.5	0.8	22.6
10000	794.2	1.6	84.4
25000	3207.6	4.1	462.9
50000	9096.5	9.1	1719.1
75000	16612.3	14.6	3723.1
100000	25434.8	19.4	6763.6
200000	71724.0	33.4	27294.5
500000	272534.8	89.5	179383.6
1000000	567696.9	187.1	745138.1

Průměrný čas výpočtu – Circle			
n / algoritmus	Jarvis Scan [ms]	Quick Hull [ms]	Sweep Line [ms]
1000	44.7	1.5	1.0
5000	441.5	7.2	17.5
10000	875.0	14.0	71.1
25000	2382.7	34.0	389.2
50000	4847.1	67.3	1502.8
75000	7265.5	100.7	3468.3
100000	9729.6	133.4	6254.1
200000	19879.8	265.2	25542.0
500000	48700.1	673.7	185213.4
1000000	73295.8	1373.5	799618.4

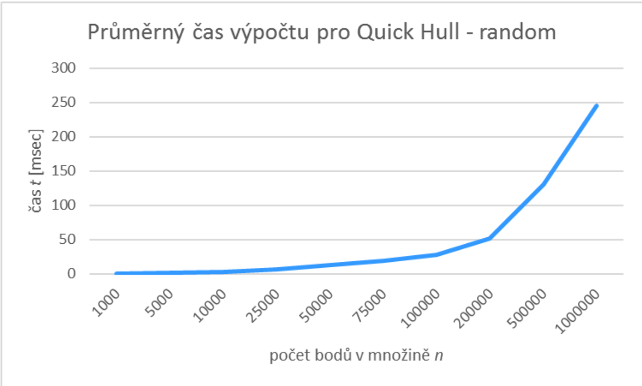
6.2 Grafy

Níže přiložené grafy zobrazují závislost mezi počtem bodů množiny a průměrným časem výpočtu pro jednotlivé algoritmy.

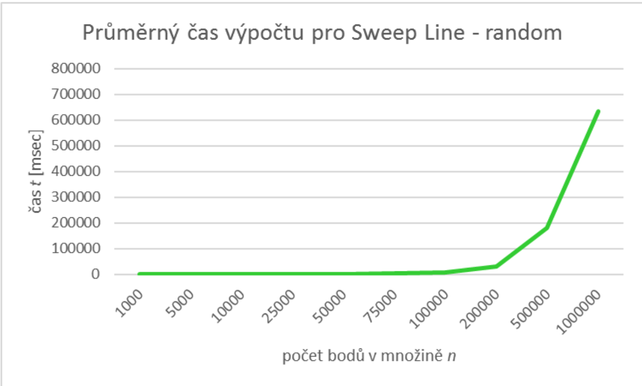
6.2.1 Random set



Obrázek 3: Graf pro Jarvis Scan - random

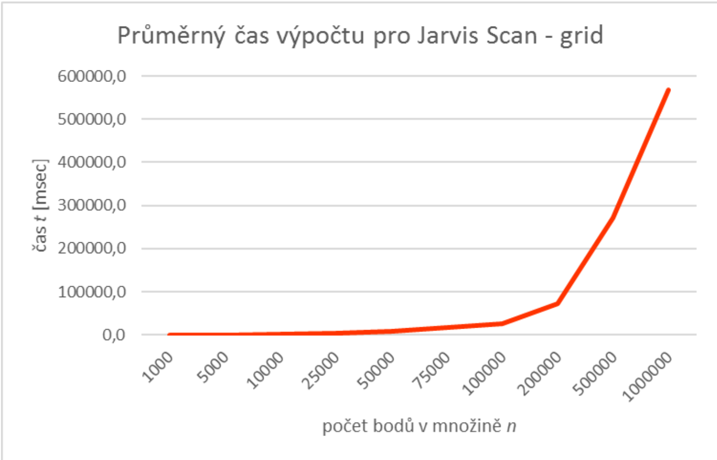


Obrázek 4: Graf pro Quick Hull - random

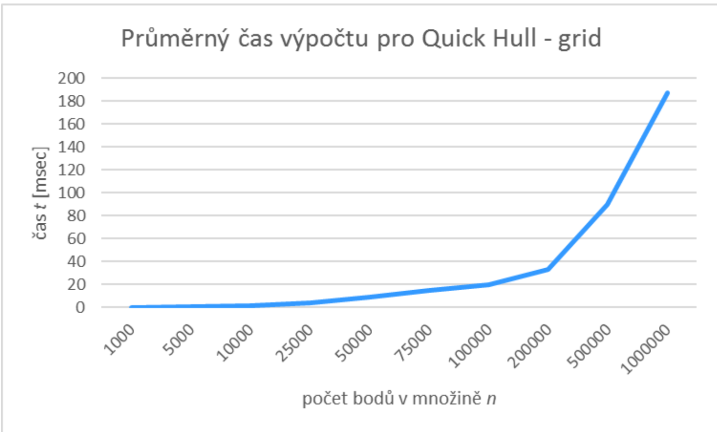


Obrázek 5: Graf pro Sweep Line - random

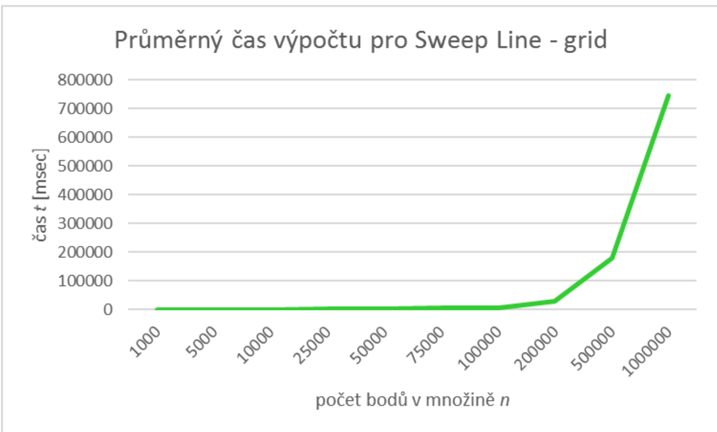
6.2.2 Grid



Obrázek 6: Graf pro Jarvis Scan - grid

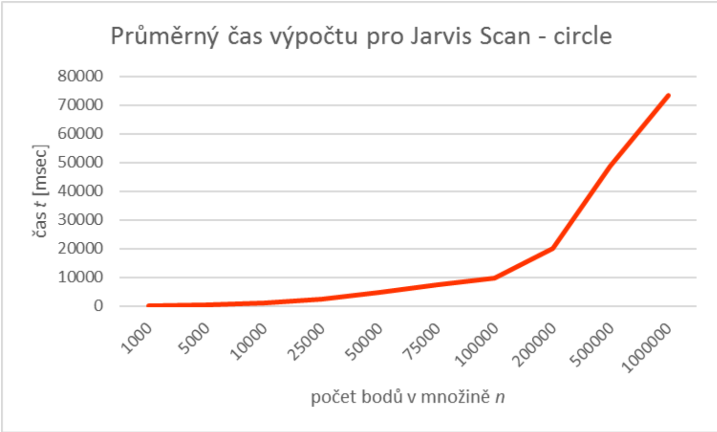


Obrázek 7: Graf pro Quick Hull - grid

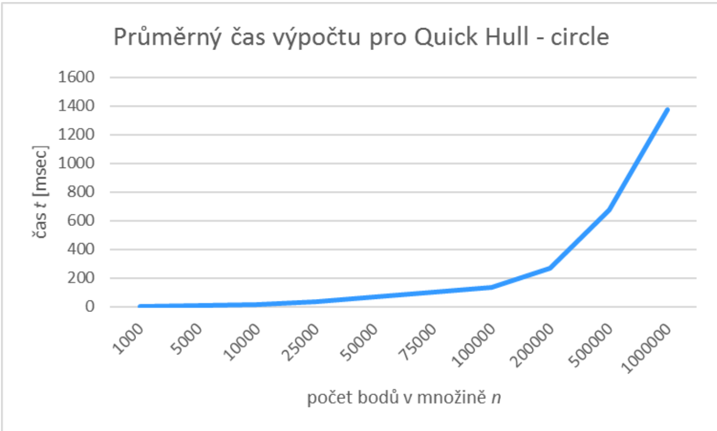


Obrázek 8: Graf pro Sweep Line - grid

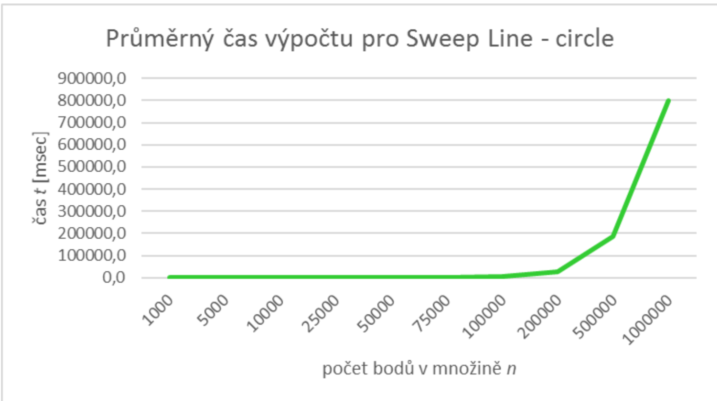
6.2.3 Circle



Obrázek 9: Graf pro Jarvis Scan - circle

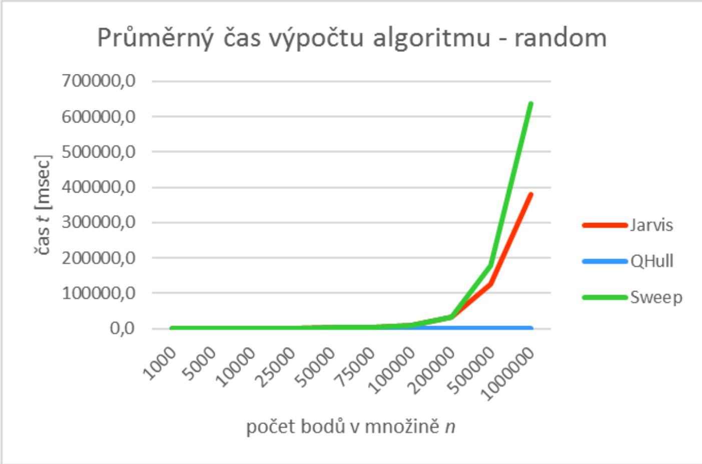


Obrázek 10: Graf pro Quick Hull - circle

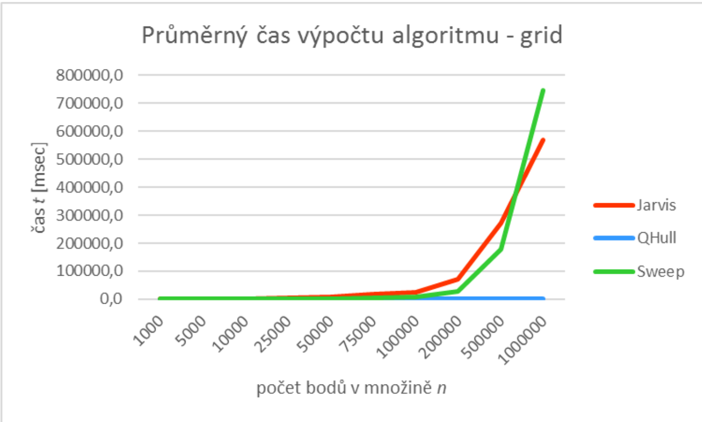


Obrázek 11: Graf pro Sweep Line - circle

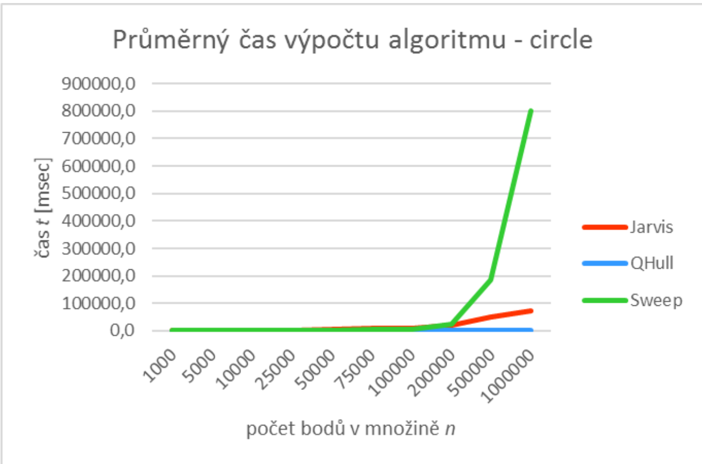
6.2.4 Porovnání



Obrázek 12: Graf pro množinu *Random*



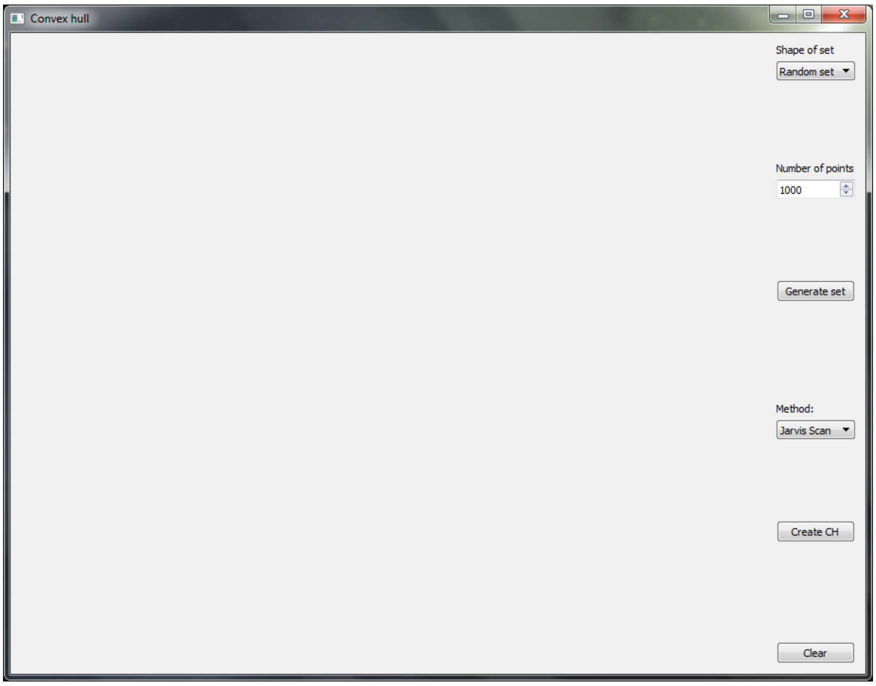
Obrázek 13: Graf pro množinu *Grid*



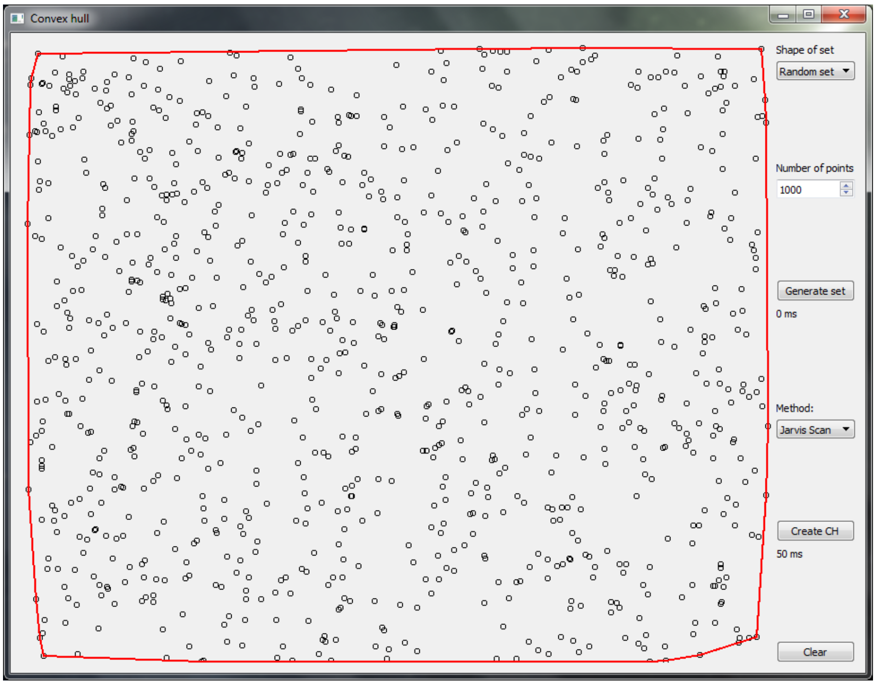
Obrázek 14: Graf pro množinu *Circle*

7 Aplikace

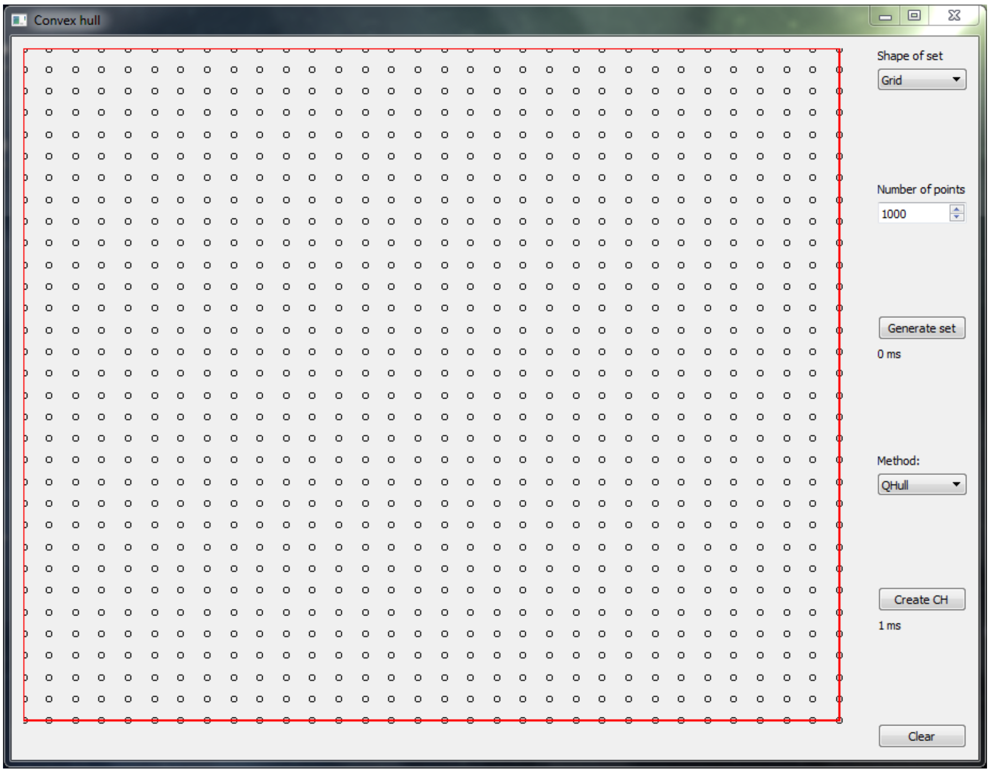
V následují kapitole je představen vizuální vzhled vytvořené aplikace tak, jak ji vidí prostý uživatel.



Obrázek 15: Výchozí vzhled aplikace po spuštění



Obrázek 16: Výstup při použití algoritmu *Jarvis Scan* nad množinou *Random* o 1000 bodech



Obrázek 17: Výstup při použití algoritmu *Quick Hull* nad množinou *Grid* o 1000 bodech

Obrázek 18: 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, metody k určování vztahu bodu a přímky a metodu pro vytvoření striktně konvexní obálky.

CHJarvis

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

Input:

- *vector* <`QPoint`> *points*

Output:

- `QPolygon`

QHull

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

Input:

- *vector* <`QPoint`> *points*

Output:

- `QPolygon`

qh_loc

Metoda **qh_loc** je lokální procedura algoritmu *Quick Hull*, 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. Návratová hodnota je typu `void`.

Input:

- `int` s

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

CHSweepLine

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

Input:

- *vector* <`QPoint`> *points*

Output:

- `QPolygon`

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`

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`

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

strictCH

Metoda **strictCH** ze vstupního polygonu vytváří striktně konvexní obálku. Přebytečné body konvexní obálky, které leží na přímce, jsou z polygonu odstraněny. Na vstupu má polygon typu `QPolygon`, návratová hodnota je typu `void`.

Input:

- `QPolygon` ch

getDistance

Metoda **getDistance** počítá vzdálenost mezi dvěma body. Metoda byla implementována v rámci ošetření případu kolineárních bodů pro algoritmus *Jarvis Scan*. Na vstupu jsou 2 body typu `QPoint`, návratová hodnota typu `double` vrací vzdálenost mezi dvěma body.

Input:

- `QPoint` a
- `QPoint` b

Output:

- `double`

8.2 Draw

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

paintEvent

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

Input:

- QPaintEvent *e

setCh

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

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 kreslící plochy. Metoda defaultně nastavuje počet generovaných bodů na hodnotu 1000 a omezuje maximální možnou hodnotu, kterou lze nastavit, na 1000000 bodů. Rozměr kreslící plochy definuje maximální možné souřadnice, kterých generované body mohou nabývat, a zároveň je z něj odvozován poloměr vykreslované kružnice a elipsy. Návratová hodnota metody je vektor bodů třídy `QPoint`.

Input:

- `int` *shape_index* (0 → random set, 1 → grid, 2 → circle, 3 → ellipse)
- `int` *num_of_points* (rozsah: 1000 – 1000000)
- `int` *canvas_width*
- `int` *canvas_height*

Output:

- *vector* <`QPoint`>

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` 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 **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` 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. Metody na vstupu nemají žádné parametry a návratové hodnoty jsou typu `void`.

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.

on_clear_button_clicked

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

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 *Konvexní obálky* byla vytvořena aplikace, která nad vstupní množinou bodů vytváří striktně konvexní obálky. V rámci testování, která trvalo dlouho do noci a použitým počítačům dala pořádně zabrat, byla shromážděna data průměrné doby výpočtu striktně konvexní obálky pro jednotlivé algoritmy. Z časových důvodů byly implementovány jen některé bonusové úlohy.

Po provedených testech považujeme za nejvhodnější algoritmus pro výpočet konvexní obálky pro všechny typy množin algoritmus *Quick Hull*, a to i přesto, že by měl na kružnici dávat horší výsledky. Ze všech použitých algoritmů je suverénně nejrychlejším, což se výrazně projevilo hlavně na velkých množinách. Výpočet konvexní obálky *Jarvis Scanem* na velkých množinách trval dlouho, avšak překvapila nás rychlost, s jakou se vypořádal s kružnicí. Nicméně výpočet obálky mu i na malých množinách trval o poznání déle než ostatním algoritmům. Jako nevyhovujícím pro všechny typy množin (zejména pokud obsahují velké množství bodů) byl shledán *Sweep Line* algoritmus. Vytvoření konvexní obálky nad množinou o 1 milionu bodů mu trvalo 12-15 minut(pro srovnání, *Quick Hull* to zvládal pod 1.5 s). Vliv na výpočetní dobu jistě má odstranění duplicitních bodů ze vstupní množiny, avšak je to nezbytný krok ke správnému fungování algoritmu.

Závěrem by bylo vhodné podotknout, že data z testování nejsou 100% spolehlivá. Již v průběhu testování bylo zaznamenáno, že doba výpočtu algoritmu velmi závisí na výkonu použitého počítače (rozdíl v rychlostech byl až dvojnásobný) a také na tom, zda jsou v době testování na počítači spuštěny jiné aplikace (např. prohlížeč) nebo se provádí jiné úkony (např. psaní technické zprávy). To může být jednou z příčin vzniku odchylek, které se v datech občas vyskytují. Pro zachování přibližně konzistentních podmínek při testování byly použity dva notebooky s podobným výkonem.

Do budoucna by jistě šla rozšířit nabídka generovaných množin bodů a naprogramovat celková automatizace testování. Aktuální verze kódu pro testování obsahovala pouze cyklus na 10 opakování téhož výpočtu. Dále by mohl být naprogramován další výpočetní algoritmus, *Graham Scan*, na který již autorky neměly čas. Mezi pozitivní přínosy úlohy zajisté patří objevení způsobu hromadného exportu grafů z *Excelu* do formátu *.png.

10 Zdroje

1. *BAYER, Tomáš. Geometrické vyhledávání bodů* [online][cit. 10. 11. 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

4. *Create LaTeX tables online* [online][cit. 15. 11. 2018].
   Dostupné z: https://www.tablesgenerator.com