

České vysoké učení technické v Praze

Fakulta stavební



Algoritmy v digitální kartografii

Úloha 2: Konvexní obálky a jejich konstrukce

Skupina

Lucie Děkanová

Josef Pudil

Zimní semestr 2020/2021

Obsah

1. Zadání	3
2. Údaje o bonusových úlohách	3
3. Popis a rozbor problémů	4
4. Popis algoritmů	4
4.1 Jarvis Scan	4
4.2 Quick Hull	5
4.3 Sweep Line	7
5. Problematické situace a popsání bonusových úloh	8
5.1 Ošetření singulárního případu u Jarvis Scan: existence kolineárních bodů datasetu	8
5.2 Algoritmus pro automatické generování konvexních/nekonvexních množin bodů různých tvarů	8
5.2.1 Generátor náhodných bodů	8
5.2.2 Generátor mřížky	9
5.2.3 Generátor kružnice	9
5.2.4 Generátor elipsy	9
5.2.5 Generátor čtverce	9
5.2.6 Generátor star-shaped	9
6. Vstupní data	9
7. Výstupní data	10
8. Dokumentace	13
8.1 Třída Algorithms	13
8.2 Třída Draw	14
9. Testování	16
9.1 Jarvis Scan	16
9.2 Quick Hull	18
9.3 Sweep Line	19
10. Zhodnocení metod	20
10.1 Jarvis Scan	20
10.2 Quick Hull	20
10.3 Sweep Line	21
Závěr	21
Zdroje	22
Seznam obrázků	22

1. Zadání

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 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, 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

Čas zpracování: 3 týdny.

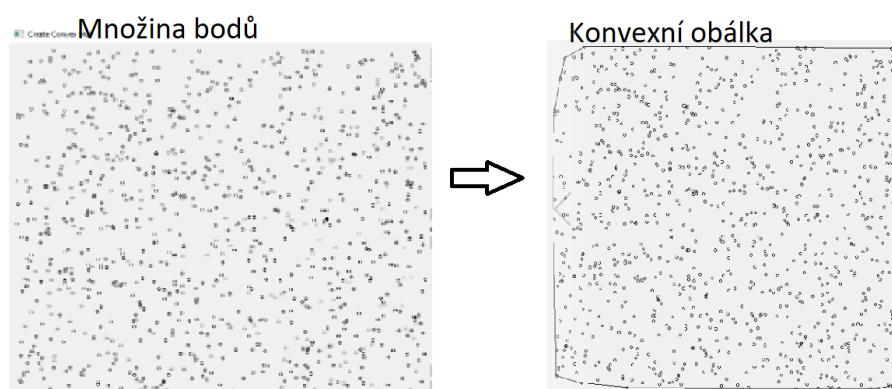
2. Údaje o bonusových úlohách

V této úloze byla vyřešena bonusová úloha *ošetření singulárního případu u Jarvis Scan: existence kolineárních bodů a algoritmus pro automatické generování konvexních/nekonvexních množin bodů různých tvarů*.

3. Popis a rozbor problémů

Definice konvexní obálky: Konvexní obálka množiny CH konečné množiny S v E^2 představuje nejmenší konvexní mnohoúhelník P obsahující množinu S .

Konvexní obálka je hranice, která spojuje konkrétní body množiny tak, aby se každý bod množiny nacházel buď uvnitř nebo na hranici obálky.



Obrázek 1: Příklad konvexní obálky nad množinou bodů

Pokud spojíme dva libovolné body množiny úsečkou, daná úsečka leží v konvexní obálce nebo je s ní totožná.

Hlavním cílem úlohy bylo vytvořit aplikaci, která využívá několika algoritmů pro tvorbu konvexních obálek. Jednou z povinných částí zadání bylo i měření času nutného pro výpočet daného algoritmu nad různými množinami bodů. Tato problematika byla řešena pomocí algoritmů *Jarvis Scan*, *Quick Hull* a *Sweep Line*.

Aplikace obsahuje funkci pro generování bodů, grafický výstup a jejich zpracování ve formě konvexní obálky.

Úloha byla vytvořena v programu *Qt Creator*.

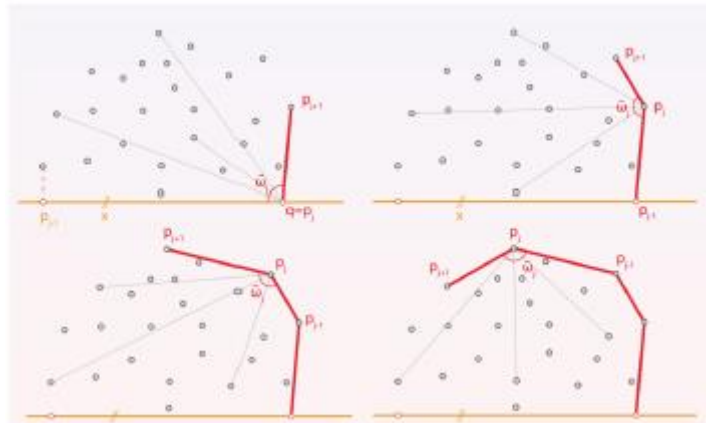
4. Popis algoritmů

Bylo využito algoritmů *Jarvis Scan*, *Quick Hull* a *Sweep Line*. V této kapitole je zapsán stručný popis všech algoritmů.

4.1 Jarvis Scan

Pomocí této metody lze vyhledávat body konvexní obálky na základě hledání maximálního úhlu. U algoritmu musí být ošetřeno, že 3 body neleží na jedné přímce. Tento algoritmus je časově náročnější. Je to algoritmus, který je nevhodný pro velké množiny bodů.

Jako první se u algoritmu nalézá pivot q , což je takový bod z množiny, který má nejmenší souřadnici y . Nalezený bod q bude prvním bodem konvexní obálky. Poté se hledá bod r takový, který tvoří s bodem q rovnoběžku s osou y . Po nalezení těchto dvou bodů jsou prováděny cykly hledání bodu, který má vzhledem k dané přímce, tvořené posledními dvěma body konvexní obálky maximální úhel. Pokud je takový bod nalezen, je přidán do konvexní obálky a cyklus se opakuje. Konec cyklu nastává ve chvíli, kdy je přidáván bod q .



Obrázek 2: Jarvis Scan

Algoritmus:

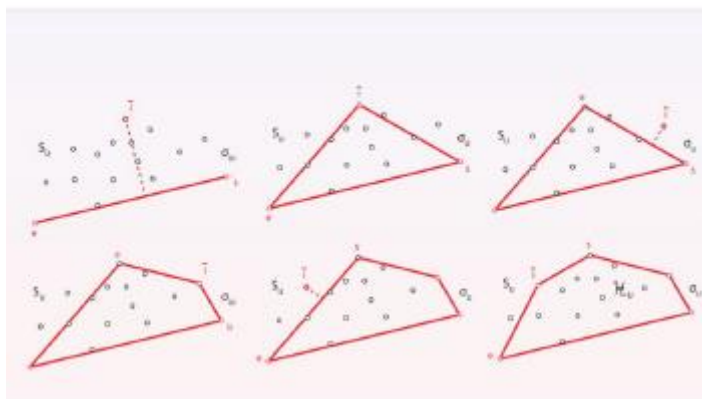
1. Nalezení pivota q , $q = \min(y_i)$
2. Přidání q do konvexní obálky H
3. Inicializace $p_{j-1} \in X, p_j = q, p_{j+1} = p_{j-1}$
4. Spuštění cyklu, dokud $p_{j+1} \neq q$
 - Nalezení bodu s maximálním úhlem, $p_{j+1} = \operatorname{argmax} \angle(p_{j-1}, p_j, p_i)$
 - Přidání bodu p_{j+1} do konvexní obálky
 - Změna indexu, $p_{j-1} = p_j; p_j = p_{j+1}$

4.2 Quick Hull

Tato metoda vyhledává body konvexní obálky pomocí hledání nejvzdálenějšího bodu od přímky, která je určena 2 body množiny bodů.

Ve většině případů funguje s časovou náročností $(n \log n)$, ale při špatném rozložení bodů dosahuje časové náročnosti (n^2) , kde n je počet bodů vstupní množiny.

Při tvorbě je nejdříve roztržiděn dataset podle velikosti x-ové souřadnice. Ze setříděných bodů je vybrán bod q_1 s minimální souřadnicí x a druhý q_3 s maximální souřadnicí x . Spojnice těchto bodů rozdělí množinu na dvě poloroviny, kde je každá z nich zpracovávána samostatně. V polorovině je rekurzivně vyvolána lokální procedura, která urči bod konvexní obálky takový, který je od přímky dané počátečním a koncovým bodem nejdéle.



Obrázek 3: Quick Hull

Algoritmus:

Lokální procedura (qh()):

1. Nalezení bodu \bar{p} napravo od spojnice ps, pe s maximální vzdáleností.
2. Pokud $\bar{p} \neq \emptyset$
 - Rekurzivní volání; $qh(s, \bar{t}, S, H)$ // horní polovina
 - Přidání bodu \bar{p} do konvexní obálky
 - Rekurzivní volání; $qh(\bar{t}, e, S, H)$ // dolní polovina

kde

S je množina bodů,

s je index počátečního bodu spojnice;

e je index koncového bodu spojnice,

\bar{t} je index nejvzdálenějšího bodu od spojnice.

Globální procedura:

1. Inicializace: $H = \emptyset, SU = \emptyset, SL = \emptyset$
2. Setřídění bodů podle X , $q_1 = \min(x_i), q_3 = \max(x_i)$
3. Přidání bodů q_1 a q_3 do SU // do horní poloroviny

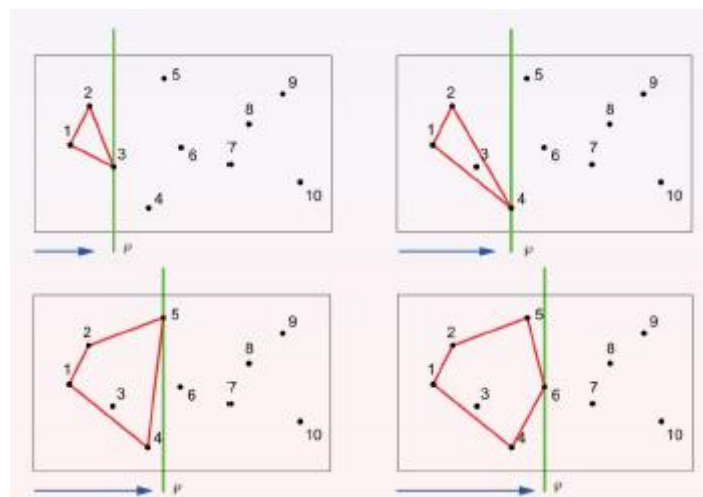
4. Přidání bodů $q1$ a $q3$ do SL // do dolní poloviny
5. For cyklus přes všechny body množiny
 - Pokud $pi \in \sigma l(q1, q3)$, přidání pi do SU // leží nalevo od spojnice
 - jinak přidání pi do SL
8. Přidání bodu $q3$ do konvexní obálky
9. Rekurze; $qh(1, 0, SU, H)$
10. Přidání bodu $q1$ do konvexní obálky
11. Rekurze; $qh(0, 1, SL, H)$

4.3 Sweep Line

Tvorba metodou *Sweep Line* je založena na přidávání bodů do konvexní obálky po jednom. Tvar obálky je postupně upravován.

V této metodě je rovina rozdělena na zpracovanou a nezpracovanou část. Složitost algoritmu je $(n \log n)$ a lze jej převést do vyšší dimenze. Nevýhodou je citlivost vůči duplicitním bodům.

V prvním kroku algoritmu je třeba setřídít celou množinu bodů podle velikosti souřadnice x . Dále algoritmus určí jako iniciální řešení dvojúhelník (trojúhelník), který je tvořený prvními dvěma (třemi) body setříděného datasetu. Po přidání dalšího bodu je konvexní obálka updatována a dále se rozšiřuje v kladném směru osy x .



Obrázek 4: Sweep Line

Algoritmus:

1. Setřídění množiny bodů podle X

- Inicializace vektorů bodů předchůdců $p(m)$ a následníků $n(m)$, kde m je počet všech bodů
- 3. Prvotní aproximace; $n[0] = 1, n[1] = 0, p[0] = 1, p[1] = 0$

4. For cyklus přes všechny body; $i = 2$

- Pokud $y_i > y_{i-1}$
 - $p[i] = i - 1; n[i] = n[i - 1]$
- Else $p[i] = p[i-1]; n[i] = i-1$
 - $n[p[i]] = i; p[n[i]] = i$
 - Dokud $n[n[i]] \in \sigma R(i, n[i])$
 - Změna indexů $p[n[n[i]]] = i; n[i] = n[n[i]]$
 - Dokud $p[p[i]] \in \sigma L(i, p[i])$
 - Změna indexů $n[p[p[i]]] = i; p[i] = p[p[i]]$

5. Problematické situace a popsání bonusových úloh

5.1 Ošetření singulárního případu u Jarvis Scan: existence kolineárních bodů datasetu

U této metody je třeba ošetřit, aby nenastala situace, kdy bod a má stejný úhel jako jiný bod b vzhledem ke stejné přímce.

Dochází kde k určování maximálního úhlu ve vrcholu konvexní obálky.

Tato singularita byla způsobena kolineárními body. Odstranění singularity bylo provedeno jako bonusová úloha. Pokud byl vypočtený úhel totožný s největším nalezeným úhlem, tak byla vypočtena vzdálenost od bodu konvexní obálky k bodu s počítaným úhlem. Pokud byla vzdálenost počítaná k bodu větší, tak je tento bod označen jako bod s maximálním úhlem. Bod je přeskočen a není zahrnut do konvexní obálky.

5.2 Algoritmus pro automatické generování konvexních/nekonvexních množin bodů různých tvarů

5.2.1 Generátor náhodných bodů

Náhodné body byly vygenerovány pomocí funkce *randompoints*. Pro správné zobrazení byly nastaveny rozměry okna. Po zadání počtu bodů v aplikaci je možné je vygenerovat.

5.2.2 Generátor mřížky

Nejdříve byl vytvořen náhodný levý horní roh a dále se generovaly vzdálenosti bodů po x a y souřadnici. Dále byla vypočtena velikost mřížky z odmocniny počtu bodů. Dále byly vytvořeny body po řádcích se stejným rozestupem.

5.2.3 Generátor kružnice

Nejdříve byl vytvořen náhodný střed, poté vytvořen poloměr a byl vypočten středový úhel $\phi = 2\pi/n$. V závislosti na počtu zadaných bodů byl tento úhel určen pro dva po sobě jdoucí body. Poté se postupně vytvořili jednotlivé body kružnice s konstantním rozestupem, kde úhel tvořený spojnicemi dvou sousedních bodů se středem kružnice byl roven ϕ . Body na kružnici byly vygenerovány pomocí rovnic:

$$X = X0 + r * \cos(\phi)$$

$$Y = Y0 + r * \sin(\phi)$$

5.2.4 Generátor elipsy

Elipsa byla vytvořena podobně jako kružnice, ale v tomto případě byly náhodně generovány hodnoty hlavní poloosy a a vedlejší poloosy b . Body byly generovány podle vzorců:

$$X = X0 + a * \cos(\phi)$$

$$Y = Y0 + b * \sin(\phi)$$

5.2.5 Generátor čtverce

Nejdříve byla vygenerována strana a (velikost strany), dále byl vytvořen levý horní roh a dopočítány zbylé 3 rohy. Poté byla rozdělena strana a na $k = n/4$ (kdy n je počet bodů). Poté se ke každému rohu přičítala/odečítala $k * i$.

5.2.6 Generátor star-shaped

Tento generátor je založen na generování dvou kružnic. Jedna s větším poloměrem a druhá s menším. Body jsou generovány tak, aby žádná dvojice bodů, kde jeden leží na menší kružnici a druhý na větší, neležela v jedné přímce se středem kružnic.

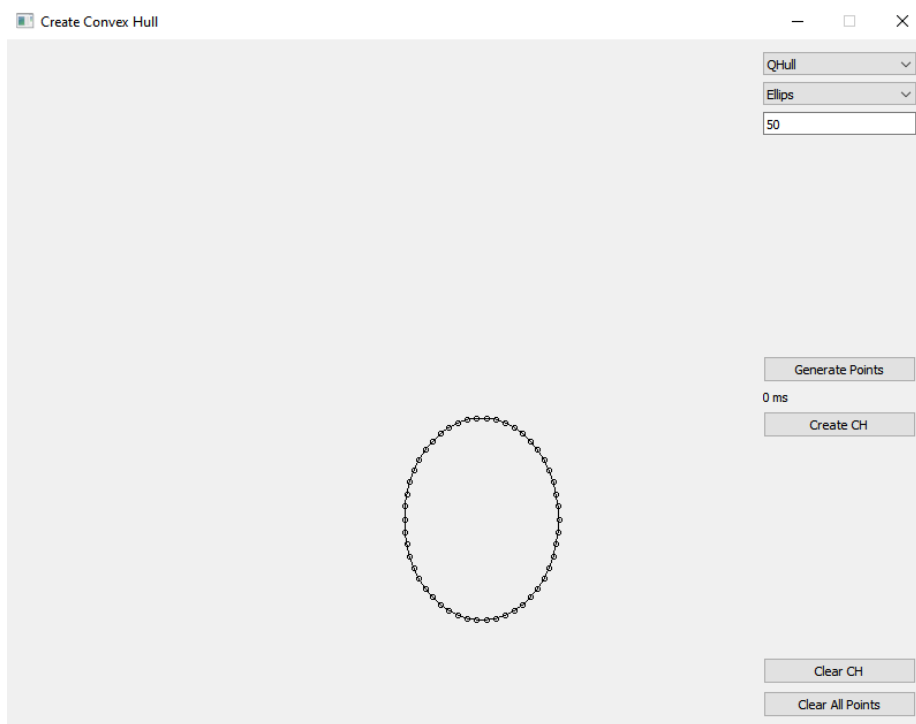
6. Vstupní data

Vstupní data jsou buď generována pomocí tlačítka nebo lze body kreslit ručně.

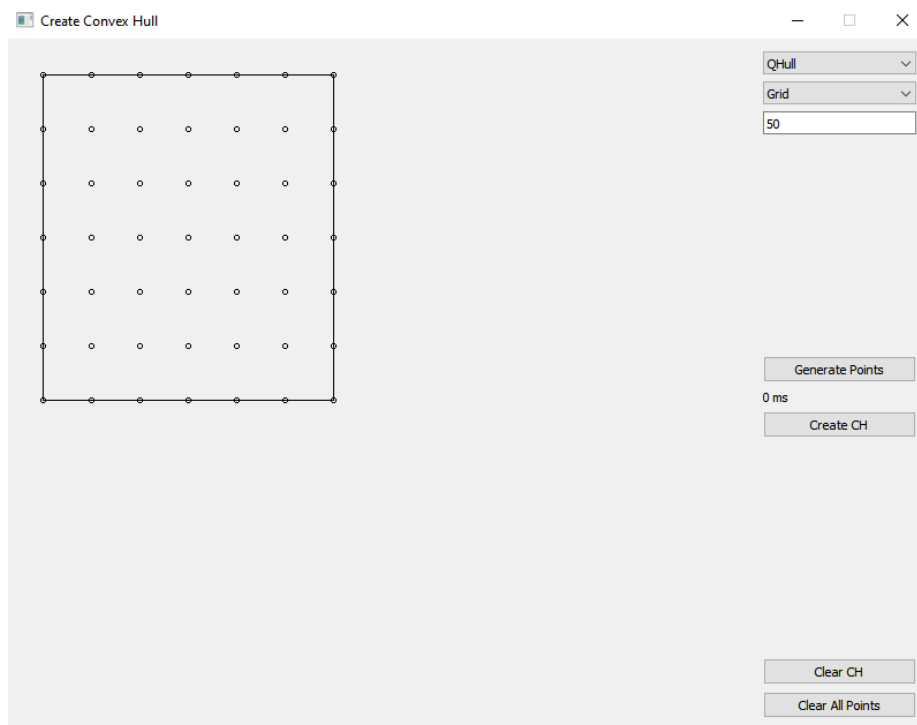
7. Výstupní data

Výstupem je grafická aplikace, která umožňuje generovat množinu bodů a tvořit nad ní konvexní obálky. V aplikaci je možné využít několika algoritmů – *Jarvis Scan*, *Quick Hull* nebo *Sweep Line*.

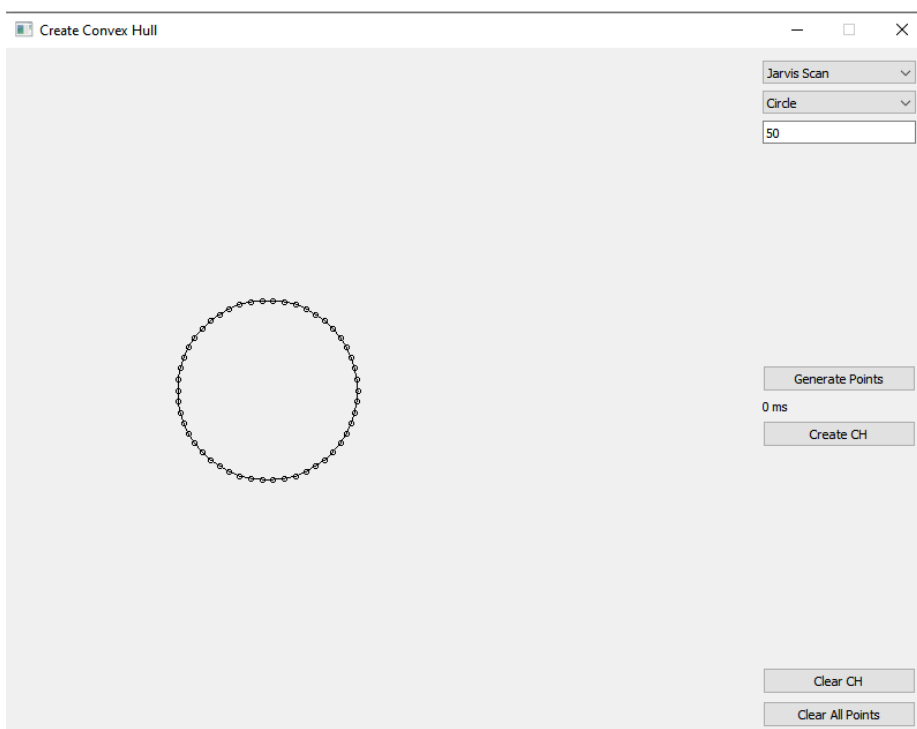
Ukázky aplikace:



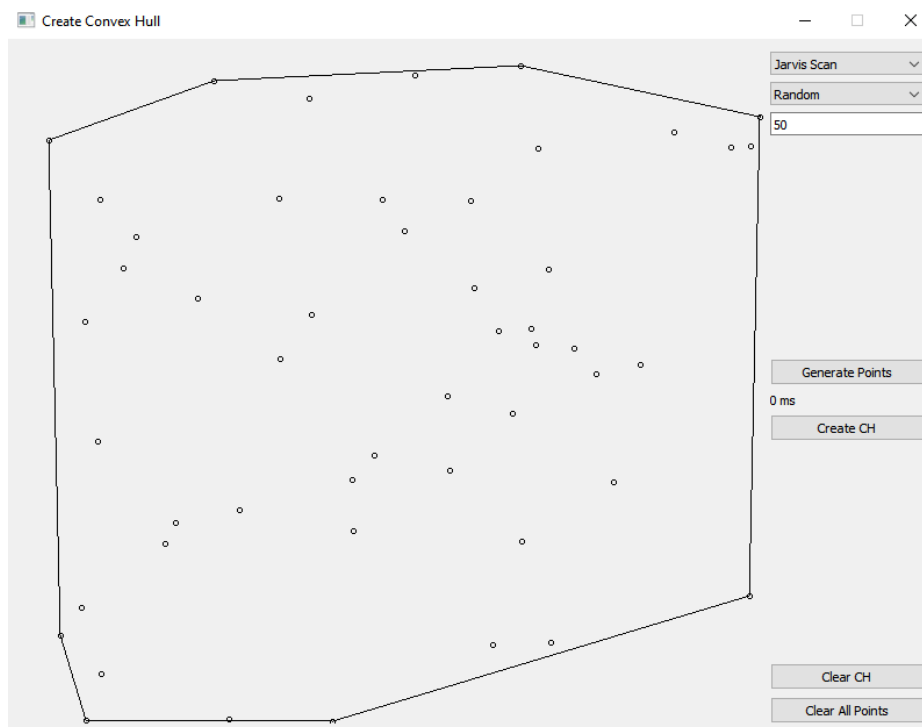
Obrázek 5: Vygenerovaná elipsa pomocí Quick Hull



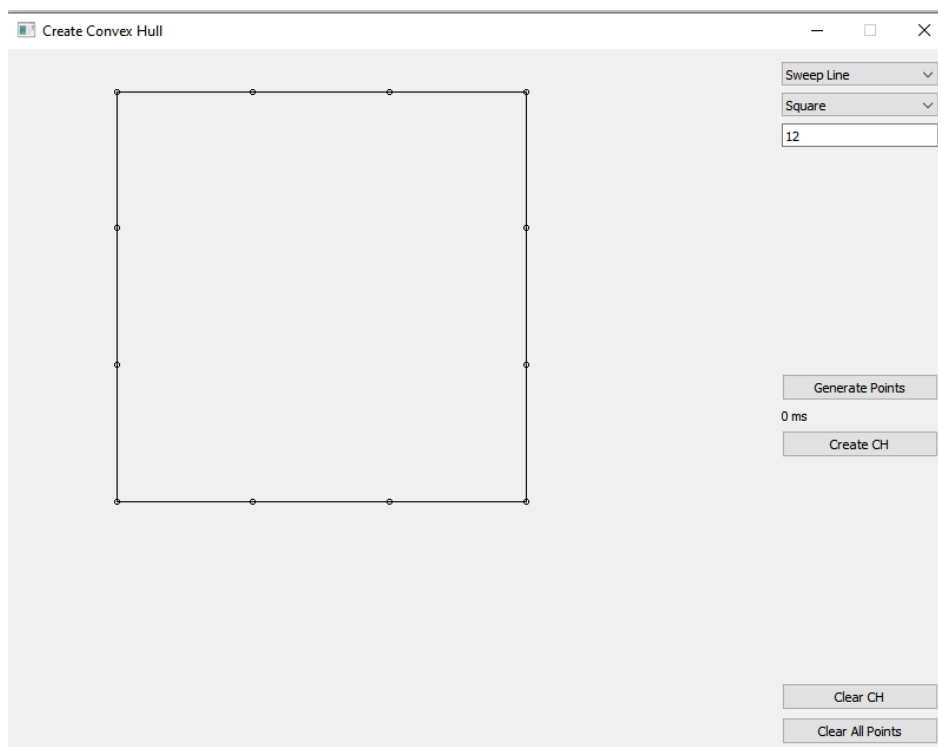
Obrázek 6: Vygenerovaný grid



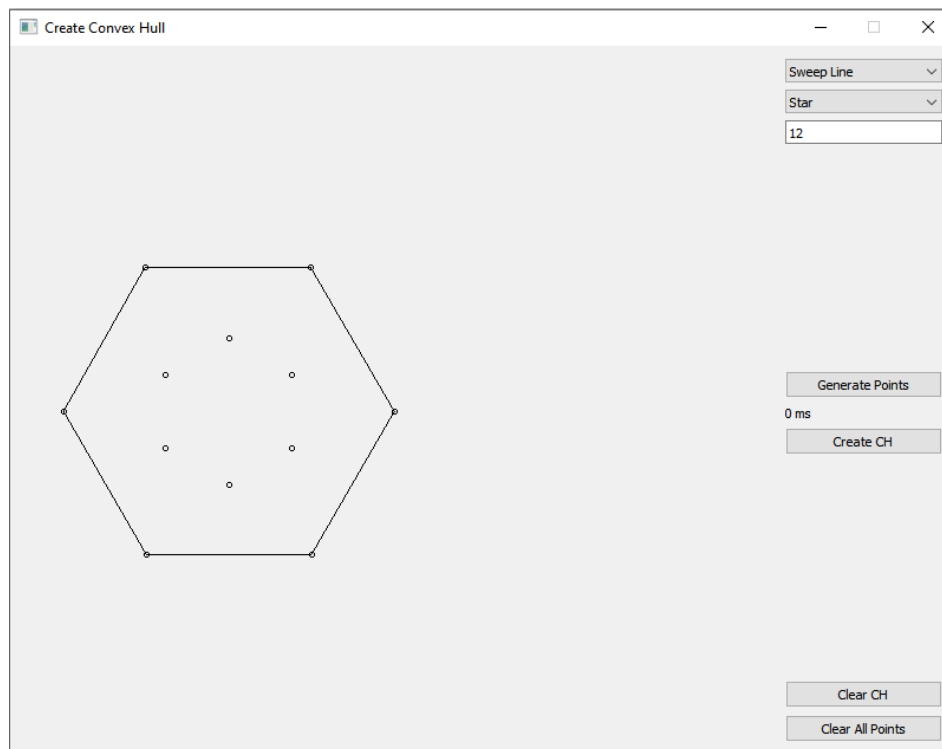
Obrázek 7: Vygenerovaná kružnice



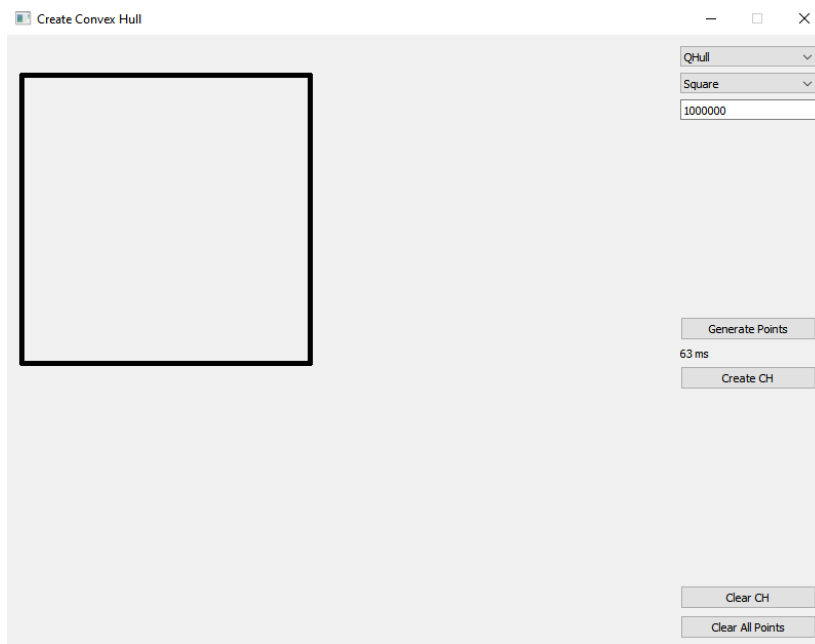
Obrázek 8: Random points s konvexní obálkou



Obrázek 9: Čtverec pomocí metody Sweep Line



Obrázek 10: Star-shaped pomocí Sweep Line



Obrázek 11: Čtverec pomocí Quick Hull

8. Dokumentace

8.1 Třída Algorithms

funkce:

```
double getAngle(QPoint &p1, QPoint &p2, QPoint &p3, QPoint &p4);
```

-funkce počítá úhel mezi vektory, definovány jsou body $p1$, $p2$, $p3$, $p4$

```
int getPointLinePosition(QPoint &q, QPoint &p1, QPoint &p2);
```

-funkce počítá orientaci bodu q vůči přímce tvořené z bodů $p1$ a $p2$, zda se nachází bod v levé, nebo právě polorovině, nebo na přímce. Definované prvky jsou bod q a linie tvořená z bodů $p1$ a $p2$

```
double getPointLineDist(QPoint &a, QPoint &p1, QPoint &p2);
```

-funkce počítá vzdálenost mezi bodem a přímkou, definován je bod a a přímka body $p1$ a $p2$

```
QPolygon jarvis(QPolygon &points);
```

-funkce vytváří konvexní obálku algoritmem *Jarvis Scan*, kolem definované množiny *Points*

```
QPolygon qhull(QPolygon &points);
```

-funkce vytváří konvexní obálku algoritmem quick hull, kolem definované množiny *points*.

```
void qh(int s, int e, QPolygon &points, QPolygon &ch);
```

-funkce pro rekursivní proceduru pro algoritmus *Quick Hull*

```
QPolygon sweepLine(QPolygon &points);
```

-funkce vytváří konvexní obálku algoritmem *Sweep Line*, kolem definované množiny *points*

```
double getLength(QPoint &p1, QPoint &p2);
```

-funkce počítá vzdálenost mezi body, definovány jsou body $p1$ a $p2$

8.2 Třída Draw

```
private:
```

```
QPolygon points;
```

-Proměnná, do které se budou ukládat body a pracovat s nimi

```
QPolygon ch;
```

-Proměnná, do které se ukládá konvexní obálka

```
QPolygon polygon;
```

-Proměnná pro vyhledání bodů

```
public:
```

```
void mousePressEvent(QMouseEvent *e);
```

-funkce pro snímání kurzoru myši a kliknutí

```
void paintEvent(QPaintEvent *e);
```

-funkce pro vykreslování

```
QPolygon & getPoints(){return points;}
```

-funkce pro získání bodů do proměnné *points*

```
QPolygon & getCH() {return ch;}
```

-funkce pro získání konvexní obálky do proměnné *ch*.

```
QPolygon & getPolygon (){return polygon;}
```

-funkce pro získání bodů do proměnné *polygon*

```
void setCH(QPolygon &ch_){ch = ch_;}
```

-funkce pro nastavení konvexní obálky

```
void setPoints(QPolygon Points){points = Points;}
```

-funkce pro nastavení bodů

```
QPolygon randompoints(int width, int height, int n);
```

-funkce generující náhodně body v kreslicím okně o daném počtu bodů, definovány jsou šířka okna *width*, výška okna *height* a počet bodů *n*

```
QPolygon circle(int width, int height, int n);
```

-funkce generující kruh o daném počtu bodů, definovány jsou šířka okna *width*, výška okna *height* a počet bodů *n*

`QPolygon grid(int width, int height, int n);`

-funkce generující mřížku v kreslicím okně o daném počtu bodů, definovány jsou šířka okna *width*, výška okna *height* a počet bodů *n*

`QPolygon ellips(int width, int height, int n);`

-funkce generující elipsu v kreslicím okně o daném počtu bodů, definovány jsou šířka okna *width*, výška okna *height* a počet bodů *n*

`QPolygon square(int width, int height, int n);`

-funkce generující čtverec v kreslicím okně o daném počtu bodů, definovány jsou šířka okna *width*, výška okna *height* a počet bodů *n*

`QPolygon star(int width, int height, int n);`

-funkce generující tvar hvězdy v okně o daném počtu bodů, definovány jsou šířka okna *width*, výška okna *height* a počet bodů *n*

9. Testování

V této kapitole jsou k nahlédnutí výsledky testování časové náročnosti u všech 3 metod.

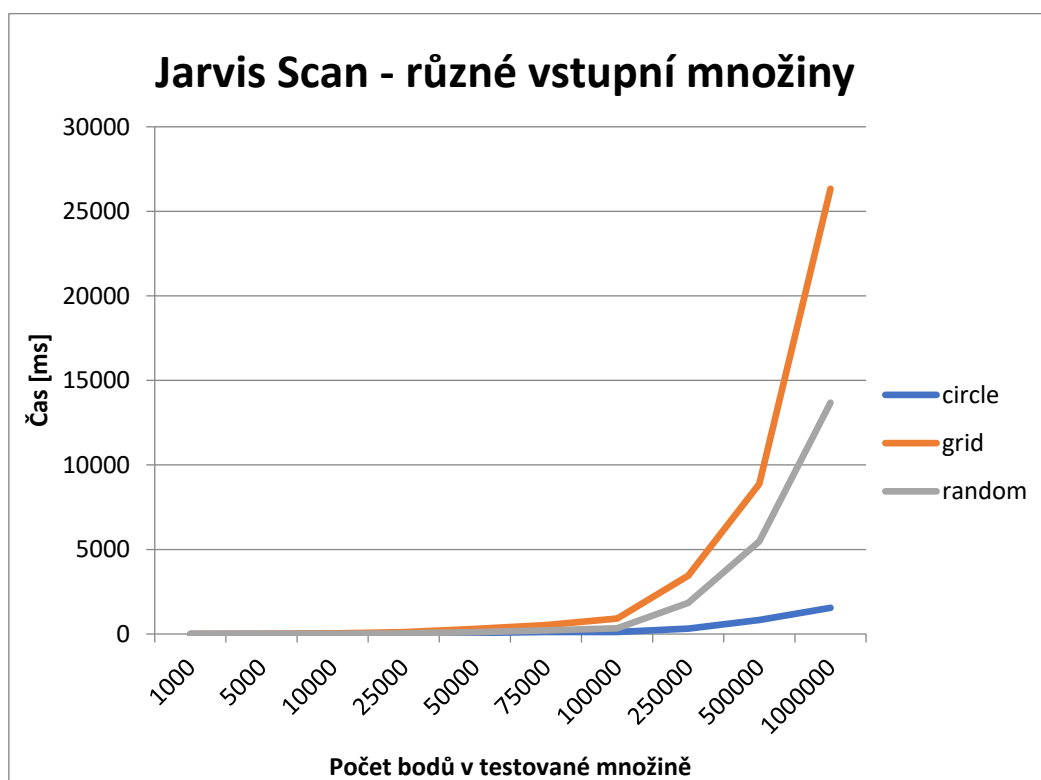
Červeně jsou vyznačeny měřené časy v ms. Pod tabulkami je vidět porovnání.

9.1 Jarvis Scan

Circle	1000	5000	10000	25000	50000	75000	100000	250000	500000	1000000
	2	5	24	38	80	150	67	324	898	1413
	2	4	14	29	57	96	114	258	596	2161
	1	4	9	21	103	98	96	266	433	1990
	0	2	11	36	84	160	101	295	1015	1726
	1	0	6	25	91	98	124	310	1084	871
	1	1	12	8	51	99	84	167	808	1972
	0	8	14	23	56	121	115	528	1146	1207
	1	5	21	28	42	156	100	560	333	1986
	1	2	21	34	82	106	221	174	1185	839
	1	1	13	39	33	113	158	312	765	1332
průměr	1	3,2	14,5	28,1	67,9	119,7	118	319,4	826,3	1549,7
rozptyl	0,4	5,36	29,85	80,49	480,49	602,61	1712,4	15315,04	79319,21	211658

Grid	1000	5000	10000	25000	50000	75000	100000	250000	500000	1000000
	1	11	32	110	290	533	931	3431	9240	27179
	0	11	33	117	285	517	918	3645	9145	26917
	0	11	32	111	305	528	922	3490	9141	26963
	0	11	30	105	308	518	865	3403	7857	26455
	1	11	30	119	291	513	897	3355	9327	27264
	0	10	33	117	293	542	823	3449	9235	22471
	0	11	32	60	297	546	977	3111	8939	25783
	1	12	30	116	289	530	975	3656	8221	26048
	0	11	31	114	297	541	900	3307	8774	26809
	0	11	27	115	310	526	951	3659	8894	27457
průměr	0,3	11	31	108,4	296,5	529,4	915,9	3450,6	8877,3	26334,6
rozptyl	0,21	0,2	3	275,64	66,05	114,84	2055,89	27270,44	208943	1912401

Random	1000	5000	10000	25000	50000	75000	100000	250000	500000	1000000
	0	3	8	27	109	195	404	1857	5526	14297
	0	3	7	35	100	241	344	1793	5429	13563
	0	3	10	33	113	231	230	1751	5492	13527
	0	3	7	26	108	154	326	1693	5542	13825
	0	2	10	24	127	210	354	1828	5575	13560
	0	3	8	39	119	230	345	1915	5403	13568
	0	3	8	41	110	192	370	1797	5397	13554
	0	2	11	32	119	257	386	2015	5407	13743
	0	3	5	37	116	220	346	1909	5165	13444
	0	5	6	44	110	241	308	1857	5874	13760
průměr	0	3	8	33,8	113,1	217,1	341,3	1841,5	5481	13684,1
variance	0	0,6	3,2	40,16	50,49	825,29	2068,81	7529,85	29032,8	54752,89

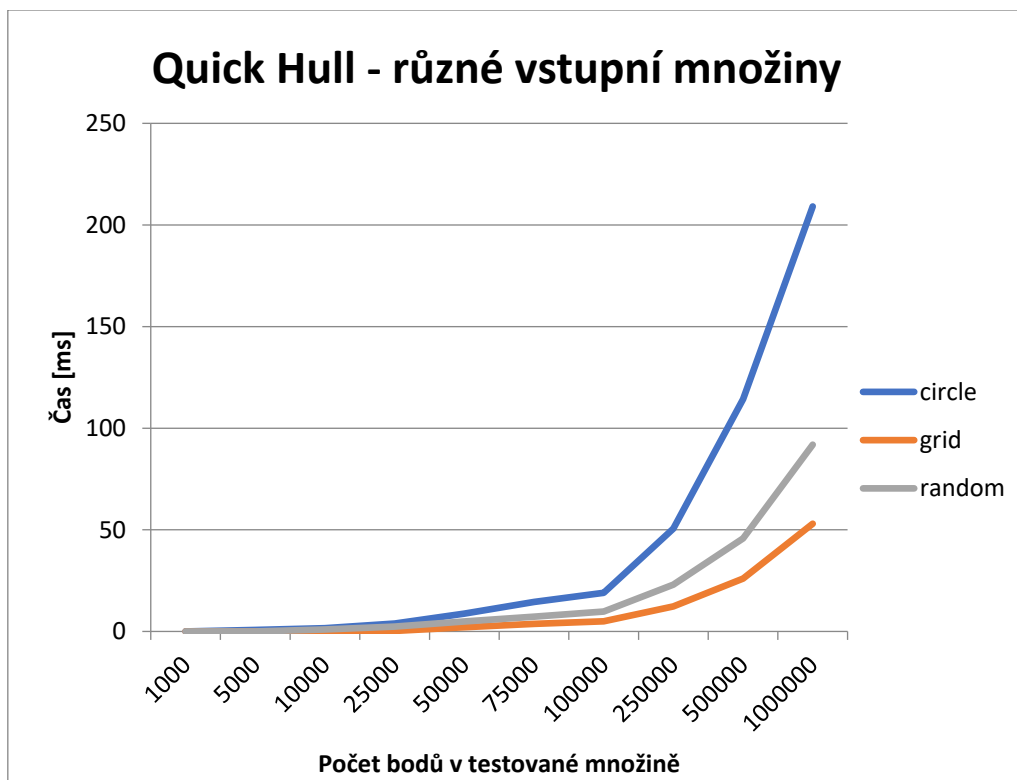


9.2 Quick Hull

Circle	1000	5000	10000	25000	50000	75000	100000	250000	500000	1000000
	0	1	2	4	6	11	13	67	118	259
	0	0	1	3	6	20	13	43	77	288
	0	0	0	4	13	16	12	39	146	129
	0	0	2	6	4	11	29	69	131	250
	0	1	2	3	6	19	21	18	105	274
	0	1	2	3	11	9	26	69	111	160
	0	1	2	2	7	17	27	66	121	134
	0	1	2	3	14	18	8	60	91	208
	0	1	1	4	9	9	15	50	133	164
	0	1	2	7	12	15	26	25	110	225
průměr	0	0,7	1,6	3,9	8,8	14,5	19	50,6	114,3	209,1
rozptyl	0	0,21	0,44	2,09	10,96	15,65	52,4	318,24	374,21	3135,49

Grid	1000	5000	10000	25000	50000	75000	100000	250000	500000	1000000
	0	0	0	1	2	4	5	12	26	53
	0	0	0	0	2	4	5	12	26	53
	0	0	0	0	2	4	5	13	26	53
	0	0	0	0	2	4	5	12	26	52
	0	0	0	0	2	4	5	12	26	53
	0	0	0	0	2	4	5	13	25	52
	0	0	0	1	2	3	5	13	26	52
	0	0	0	0	3	3	5	12	26	57
	0	0	0	0	2	4	5	12	27	53
	0	0	0	1	2	3	5	12	26	52
průměr	0	0	0	0,3	2,1	3,7	5	12,3	26	53
rozptyl	0	0	0	0,21	0,09	0,21	0	0,21	0,2	2

Random	1000	5000	10000	25000	50000	75000	100000	250000	500000	1000000
	0	0	1	2	4	7	9	25	52	86
	0	0	1	3	5	8	10	23	46	91
	0	0	1	2	6	7	9	22	45	85
	0	0	1	2	4	7	8	20	44	95
	0	0	1	3	5	8	11	22	43	93
	0	0	1	2	5	7	10	21	47	93
	0	0	1	3	5	7	11	25	42	97
	0	0	1	3	5	8	9	24	48	92
	0	0	1	2	5	7	10	22	46	90
	0	0	1	2	5	7	11	26	45	97
průměr	0	0	1	2,4	4,9	7,3	9,8	23	45,8	91,9
rozptyl	0	0	0	0,24	0,29	0,21	0,96	3,4	7,16	15,09

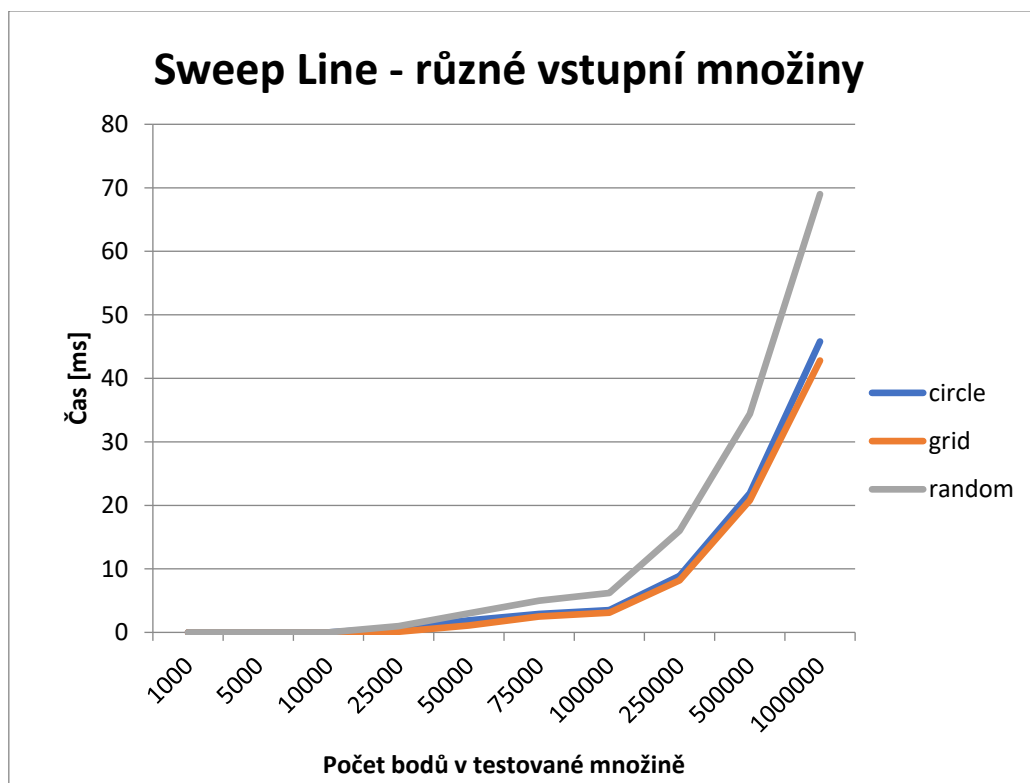


9.3 Sweep Line

Circle	1000	5000	10000	25000	50000	75000	100000	250000	500000	1000000
	0	0	0	1	2	2	4	9	22	45
	0	0	0	1	2	3	4	8	21	45
	0	0	0	0	2	4	4	8	23	46
	0	0	0	1	2	3	3	9	21	45
	0	0	0	1	2	3	4	9	22	44
	0	0	0	0	2	3	3	10	22	47
	0	0	0	1	2	3	3	9	22	47
	0	0	0	1	2	3	4	9	21	49
	0	0	0	0	1	3	3	9	23	43
	0	0	0	1	2	2	3	9	22	47
průměr	0	0	0	0,7	1,9	2,9	3,5	8,9	21,9	45,8
rozptyl	0	0	0	0,21	0,09	0,29	0,25	0,29	0,49	2,76

Grid	1000	5000	10000	25000	50000	75000	100000	250000	500000	1000000
	0	0	0	1	1	2	3	8	21	42
	0	0	0	0	1	3	3	8	20	42
	0	0	0	0	1	2	3	8	21	42
	0	0	0	0	2	3	3	8	20	43
	0	0	0	0	1	3	3	10	21	42
	0	0	0	0	1	3	3	8	21	43
	0	0	0	0	1	2	4	8	20	42
	0	0	0	0	1	3	3	8	21	42
	0	0	0	0	1	2	3	8	22	45
	0	0	0	0	1	2	3	8	21	45
průměr	0	0	0	0,1	1,1	2,5	3,1	8,2	20,8	42,8
rozptyl	0	0	0	0,09	0,09	0,25	0,09	0,36	0,36	1,36

Random	1000	5000	10000	25000	50000	75000	100000	250000	500000	1000000
	0	0	0	1	3	5	6	16	34	67
	0	0	0	1	3	5	6	16	37	68
	0	0	0	1	3	5	6	16	36	70
	0	0	0	1	3	5	6	16	34	68
	0	0	0	1	3	5	6	16	34	68
	0	0	0	1	3	4	7	16	33	70
	0	0	0	1	3	6	6	16	34	68
	0	0	0	1	3	5	6	16	34	69
	0	0	0	1	3	5	6	16	33	72
	0	0	0	1	3	5	7	16	35	70
průměr	0	0	0	1	3	5	6,2	16	34,4	69
rozptyl	0	0	0	0	0	0,2	0,16	0	1,44	2



10. Zhodnocení metod

10.1 Jarvis Scan

Tento algoritmus je nejnáročnější na výpočet s porovnáním s ostatními dvěma metodami. Tato metoda není příliš vhodná pro velké množiny bodů. Z grafu je patrné, že nejdéle trval výpočet množin v gridu. Křivka výpočetního času se v tomto případě láme v cca v 200 000, 300 000 a 700 000 bodech. Pokud srovnáme výpočet na gridu, random a kruhu, je výpočet gridu srovnatelně delší než zbylé dvě možnosti.

10.2 Quick Hull

U této metody v srovnání s kruhem a random vyšel nejnižší čas výpočtu u gridu. Nejhuře dopadl výpočet kruhu. Tento algoritmus je v porovnání s metodou Jarvis Scan rychlejší.

10.3 Sweep Line

Metoda Sweep Line vykazuje nejhorší výsledky u random bodů s nejdelší dobou výpočtu. Nejrychlejší byl v tomto případě výpočet množiny gridu, který je srovnatelný s výpočtem na kruhu. Při srovnání této metody s metodou Jarvis Scan, u Sweep Line dopadla lépe. Při porovnání s metodou Quick Hull také dosáhla lepších výsledků.

Závěr

Byla vytvořena aplikace pro výpočet a kresbu konvexní obálky nad množinou bodů vytvořenou klikáním, případně vygenerováním funkcemi pro různé tvary.

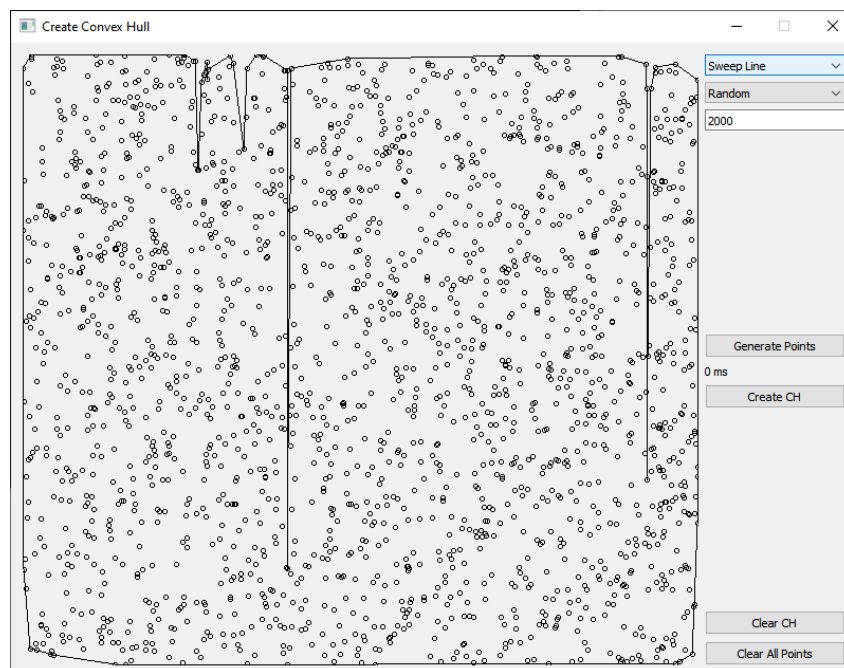
V úloze byl zkoumán čas, jaká funkce je nejúčinnější a nejrychlejší, pravděpodobně nejrychlejší je *Sweep Line* a pak *Quick Hull*, *Jarvis Scan* byl zajisté nejpomalejší.

Algoritmus *Sweep Line* obsahuje chybu při vytváření konvexní obálky kolem více než 10 000 bodů v horní části, což je pravděpodobně způsobeno duplicitou bodů.

Z bonusových úloh bylo splněno Ošetření existence kolineárních bodů v datasetu a algoritmus pro generování různých tvarů. A vytvoření generátoru pro tvorbu čtverce, hvězdice a elipsy.

Vylepšení:

Vylepšením by mohlo být vytvoření více bonusových úloh a opravení chyby u *Sweep Line*.



Zdroje

1. BAYER, Tomáš. [online] [cit. 12.11.2020].

Dostupné na: <https://web.natur.cuni.cz/~bayertom/images/courses/Adk/adk4.pdf>

Seznam obrázků

Obrázek 1: Příklad konvexní obálky nad množinou bodů	4
Obrázek 2: Jarvis Scan	5
Obrázek 3: Quick Hull	6
Obrázek 4: Sweep Line	7
Obrázek 5: Vygenerovaná elipsa pomocí Quick Hull.....	10
Obrázek 6: Vygenerovaný grid	11
Obrázek 7: Vygenerovaná kružnice	11
Obrázek 8: Random points s konvexní obálkou.....	12
Obrázek 9: Čtverec pomocí metody Sweep Line	12
Obrázek 10: Star-shaped pomocí Sweep Line	13
Obrázek 11: Čtverec pomocí Quick Hull.....	13