České vysoké učení technické v Praze
Fakulta stavební

Algoritmy v digitální kartografii

Konvexní obálky

Bc. Robin Pflug
Bc. Tomáš Klemsa

Obsah

1 Zadání úlohy

Vstup: *množina* $P = (p_1, ..., p_n), p_i = [x, y_i]$.
Výstup: *CH(P)*

Nad množinou P implementujete následující algoritmy pro konstrukci *CH(P)*:

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

Vstupní množiny bodů včetně vygenerovaných konvexních obálek vhodně vizualizujte. Pro množiny n $\epsilon < 1000, 1000000 >$ 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.

1.1 Bonusové úlohy

Ze zadání bonusových úloh byla řešena konstrukce konvexní obálky metodou *Graham Scan*, kontrukce striktně konvexních obálek , ošetření metody *Jarvis Scan*: existence kolineárních bodů v datasetu a byl vytvořen algoritmus pro automatické generování množin bodů různých tvarů.

Hodnocení:

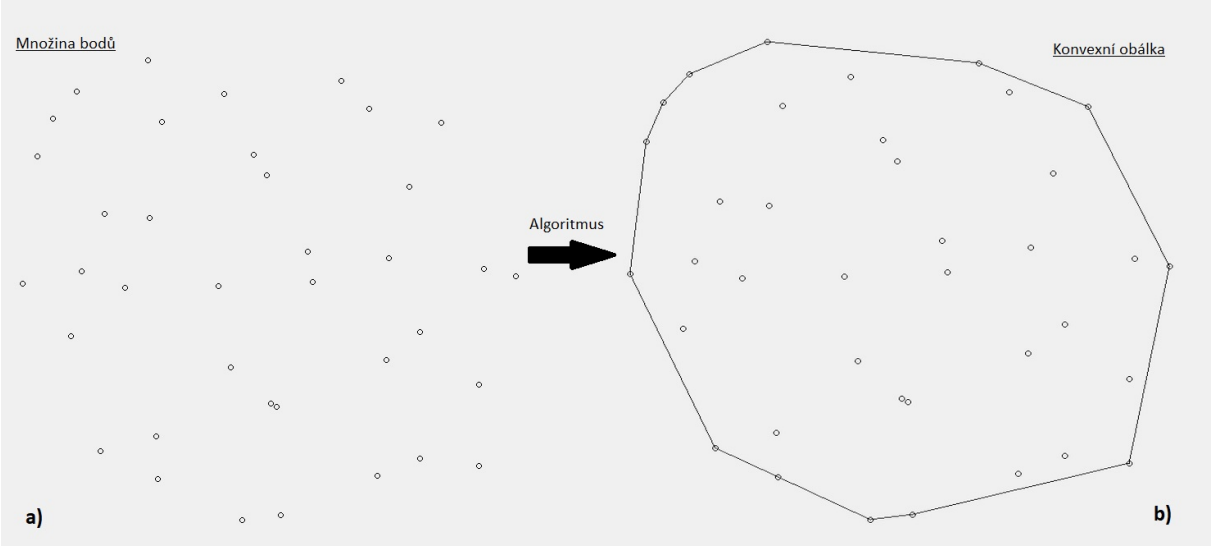
Krok	**Hodnocení**
Konstrukce konvexních obálek metodami Jarvis Scan, Quick Hull, Seep Line.	15b
Konstrukce konvexní obálky metodou Graham Scan	*+5b*
Konstrukce striktně konvexních obálek pro všechny uvedené algoritmy.	*+5b*
Ošetření singulárního případu u Jarvis Scan: existence kolineárních bodů v datasetu.	*+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**

Obrázek 1: Bodové hodnocení úlohy [zdroj: 1]

2 Obecná formulace a řešení problému

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

Konvexní obálku si lze představit jako co nejmenší množinu bodů poskládaných tak, že tvoří vrcholy uzavřeného konvexního polygonu. Všechny body zadané množiny se pak nacházejí uvnitř nebo na hraně tohoto polygonu.



Obrázek 2: Ukázka množiny bodů, kolem nichž je algoritmem vytvořena konvexní obálka.

Zadáním této úlohy bylo vytvořit aplikaci využívající různé algoritmy pro tvorbu konvexních obálek nad zadanou množinou bodů. Dalším cílem úlohy bylo zhodnotit časovou náročnost výpočtů těchto algoritmů nad ruznými množinami bodů. Problematika byla řešena aplikací celkem čtyř algoritmů: Jarvis Scan, Quick Hull, Sweep Line a Graham Scan. Widgedts aplikace byla vytvořena v jazyce C++ na platformě Qt Creator. Aplikace obsahuje funkci pro generování bodů, zpracování bodů a grafický výstup ve formě konvexní obálky i implementaci časové náročnosti jednotlivých algortimů.

3 Aplikované algoritmy

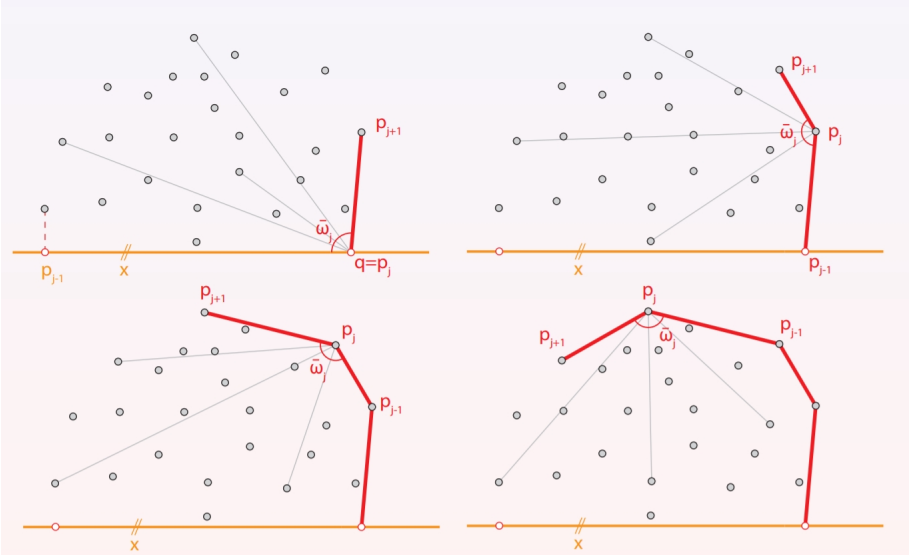
Algoritmy využívané pro tvorbu konvexní obálky: Jarvis Scan, Quick Hull, Sweep Line, Graham Scan.

3.1 Jarvis Scan

Jednoduchý algoritmus fungující na principu přidávaného bodu, který maximalizuje úhel vzhledem k posledním dvoum bodům obálky. $p_j + 1 = arg\,max_{\forall p_i \in P} \angle(p_{j-1}, p_j, p_i)$ [zdroj: 1]

Algoritmus lze rozšířit do R^3. Dataset musí být ošetřen tak, aby neobsahoval tři kolineárních body. Nevýhodou Jarvis Scan algoritmu je složitost $O(n^2)$. Z tohoto faktu vyplývá, že není vhodný pro velké množiny bodů.

Prvním krokem při tvorbě konvexní obálky nad množinou bodů metodou Jarvis Scan je nalezení pivota q. Jako pivot se určí takový bod ze zadané množiny, který má nejmenší souřadnici y. Nalezený bod, určený jako pivot q, bude prvním bodem konvexní obálky. Dalším krokem je inicializace takového bodu r, který tvoří s bodem q rovnoběžku s osou y. Po nalezení pivotu a bodu r budu provádět cyklicky hledání bodu, který má vzhledem k zadané přímce, tvořené posledními dvěmi body konvexní obálky, maximální úhel (v prvním kroku tohoto cyklu se jako poslední dva body obálky uvažují r a q. Pokud je takový bod nalezen, přidám tento bod do konvexní obálky a opakuji tento cyklus. Cyklus končí v momentě, kdy je jako přidávaný bod vyhodnocen bod q.



Obrázek 3: Jarvis Scan algoritmus [zdroj: 1]

3.1.1 Algoritmus Jarvis Scan

1. Nalezení pivota q,$q = min(y_i)$
2. Přidej $q \rightarrow CH$
3. Inicializuj: $p_j = q; p_{j+1} = p_{j-1}$
4. Opakuj, dokud $p_{j+1} \neq q$

   Nalezni p_{j+1}: $p_{j+1} = argmax_{\forall p_i \in P} \angle(p_{j-1}, p_j, p_i)$

   Přidej p_{j+1}: $p_{j+1} \rightarrow H$

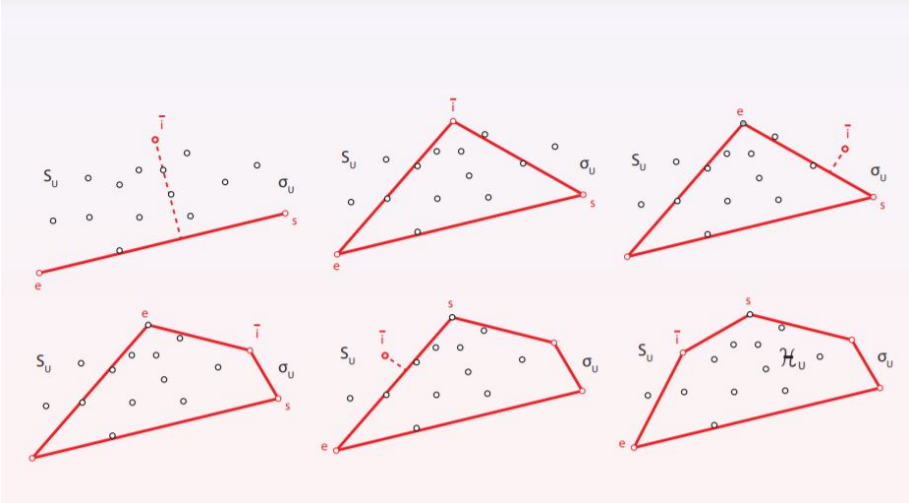
   $p_{j-1} = p_j; p_j = p_{j+1}$

[zdroj: 1]

3.2 Quick Hull

Algoritmus pro tvorbu konvexní obálky metodou Quick Hull využívá strategie Divide and Conquer. Ve většině případů funguje rychle s časovou naročností (n log n), ovšem při špatném rozložení bodů v datasetu dosahuje časové náročnosti (n^2). Rychlé provedení algoritmu je podmíněno malým počtem rekurzivních kroků. Při tvorbě expanduje konvexní obálka všemi směry.

Prvník krokem pro tvorbu kovexní obálky touto metodou je setřídění bodů datasetu podle velikosti souřadnice x_i. Z takto setříděných bodů vyberu jeden q_1 s minimální souřadnicí x a druhý q_3 s maximální souřadnicí x. Tyto dva body q_1 a q_3 rozdělí rovinu s množinou bodů na horní a dolní polorovinu. Takto vzniklé poloroviny s množinou bodů jsou zpracovávány samostatně. V polorovině je volána rekurzivně lokální procedura. Ta určí jako bod konvexní obálky takový, který je od zadané přímky dané počátečním a koncovým vrcholem nejdále.



Obrázek 4: Princip Quick Hull algoritmu [zdroj: 1]

3.2.1 Algoritmus Quick Hull globální metoda

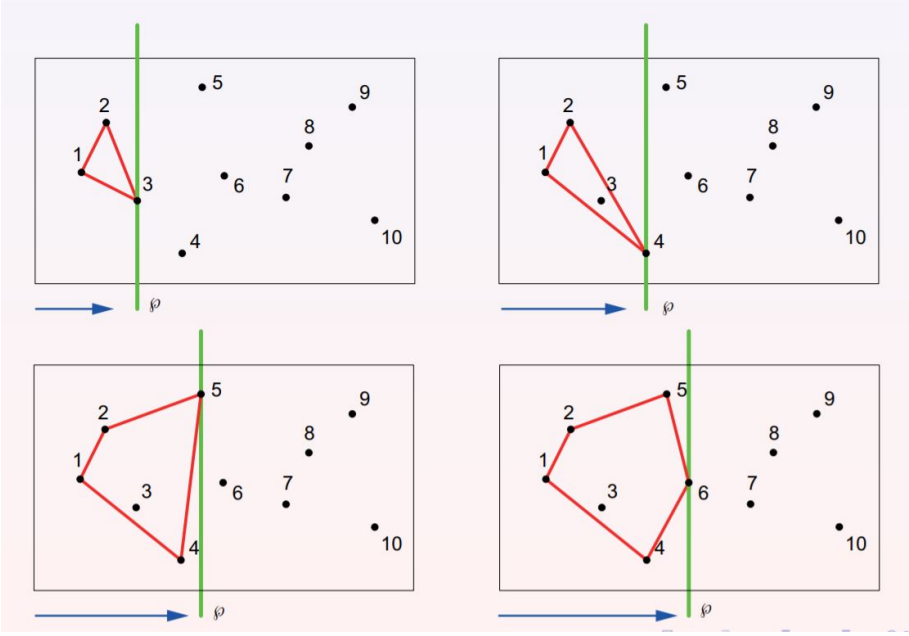
1. $CH = 0; S_U = 0; S_L = 0$
2. $q_1 = min_{\forall p_i \in S}(x_i); q_3 = max_{\forall p_i \in S}(x_i)$
3. $S_U \leftarrow q_1; S_U \leftarrow q_3; S_L \leftarrow q_1; S_L \leftarrow q_3$
4. for $\forall p_i \in S$
   if($p_i \in \sigma_l(q_1, q_3)) S_U \leftarrow p_i$
   else $S_L \leftarrow p_i$
5. $CH \leftarrow q_3$
6. Nalezení nejvzdálenějšího bodu c v horní části od přímky, přidání do množiny konvexní obálky a opakování vůči nově vzniklé přímce.
7. $CH \leftarrow q_1$
8. Opakování hledání nejvzdálenějšího bodu v dolní části.

[zdroj: 1]

3.3 Sweep line

Tvorba konvexní obálky metodou Sweep line využívá inkrementální konstrukci.
Princip inkrementální konstrukce je založen na přidávání bodů do konvexní obálky po jednom. Tvar konvexní obálky je tudíž postupně modifikován.
V metodě Sweep line je rovina rozdělena na zpracovanou a nezpracovanou část. Složitost algoritmu je (n log n) a lze jej převést do vyšší dimezne. Nevýhodou je citlivost vůči duplicitním bodům.

Prvním krokem algoritmu Sweep line je setřídění celé množiny bodu podle velikosti souřadnice x. Dále algoritmus určí jako iniciální řešení dvojúhelník (trojúhelník) tvořený prvními dvěma (třema) body setříděného datasetu. Po přidání dalšího bodu je konvexní obálka updatována a dále expanduje v kladném směru osy x.



Obrázek 5: Sweep line [zdroj: 1]

3.3.1 Algoritmus Sweep line

1. $SortP_s = sort(P)\ by\ x$

   Změna indexů následovníků: $n[0] = 1; n[1] = 0$

   Změna indexů předchůdců: $p[0] = 1; p[1] = 0$

2. $for\ p_i \in P_S, i{>}1$

   $if(y_i{>}y_{i-1})$

   $p[i] = i - 1; n[i] = n[i - 1]$

3. else $p[i] = p[i - 1]; n[i] = i - 1$

   $n[p[i]] = i; p[n[i]] = i$

   $while(n[n[i]]) \in \sigma_R(i, n[i])$

   Změna indexů: $p[n[n[i]]] = i; n[i] = n[n[i]]$

   $while(p[p[i]]) \in \sigma_L(i, p[i])$

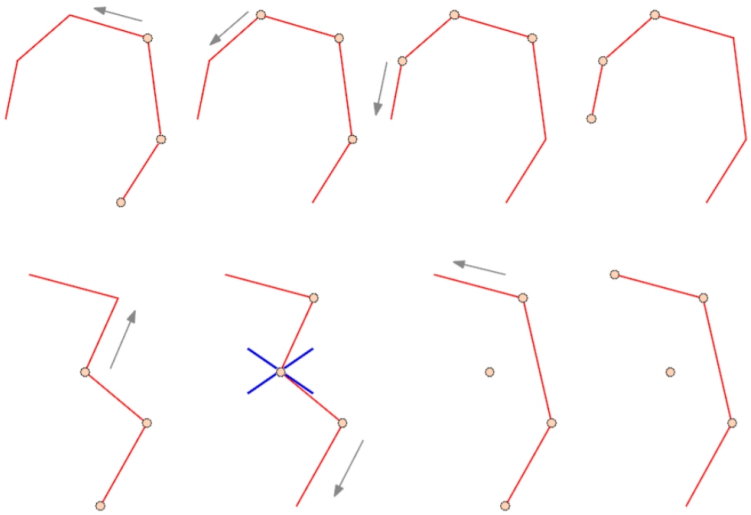
   Změna indexů: $n[p[p[i]]] = i; p[i] = p[p[i]]$

[zdroj: 1]

3.4 Graham Scan

Algortimus je založen na převodu star-shaped polygonu na konvexní obálku. Převod je realizován za pomoci kritéria levotočivosti. Metodu lze provádět i v R^3 a časová složitost je (n log n). Lze tedy říci, že algoritmus je vhodný pro rozsáhlé datasety.

Pro konstrukci konvexní obálky metodou Graham Scan je nutné v prvním kroku určit pivota q s takovou vlastností, že má nejmenší souřadnici y z celé zadané možiny bodů. Další krok je setřídění ostatních bodů datasetu podle velikostu úhlu daného rovnoběžkou s osou x procházející bodem q a přímkou danou hodnoceným bodem a bodem q. Poté jsou z datasetu odstraněny body o stejném úhlu. Odstraněn je bod ležící blíže bodu q. Bod q a bod o největším úhlu jsou zařazeny do konvexní obálky. Poté cyklicky procházím polohu bodu ze seřazené množiny a hodnotím jeho polohu vůči lini tvořené posledníma dovouma bodama v CH. Poku se nachází přidávaný bod vlevo od linie, je do konvexní obálky přidán. Dále se opakuje cyklus pro další bod ze seřazené množiny.
Pokud bod leží vpravo od linie, je poslední bod CH odebrán a cyklus se opakuje.



Obrázek 6: Princip vyhodnocení polohy bodu v metodě Graham Scan [zdroj: 6]

3.4.1 Algoritmus Graham Scan

1. $q = min_{\forall p_i \in S}(y_i), q \in H$
2. $\forall p_i \in S$ sort by $\omega_i = \angle(p_i, q, x)$
3. $\omega_k = \omega_l \rightarrow$ odstraň bod blíže q
4. $H \leftarrow q; H \leftarrow p_1$
5. $for j < n$
6. if p_j vpravo od předešlých bodů $\rightarrow popS$
7. else $p_j \rightarrow H$

[zdroj: 1]

4 Bonusové úlohy

4.1 Konstrukce konvexní obálky metodou Graham Scan

Řešení této úlohy je popsáno v kapitole Aplikované algoritmy - Graham Scan.

4.2 Konstrukce striktně konvexních obálek

Striktně konvexní obálky jsou takové, že po sobě následující 3 body neleží na stejné přímce. Striktně konvexní obálky byly tvořeny ve funkci *strictlyConvex*. Tato funkce příjmá na vstupu klasickou konvexní obálku ve formě polygonu. Z polygonu jsou nejprve odstraňovány duplicitní body. Poté je kontrolováno, zda tři po sobě jdoucí body neleží na stejné přímce. Pokud ano, prostřední bod je odstraněn z konvexní obálky.

1. $for(inti = 0; i < (ch.size() - 2); i + +)\{$
2. $if(getPointLinePosition(ch[i + 2], ch[i], ch[i + 1]) == -1)$
3. $ch.remove(i + 1);$
4. $i - -;\}$

4.3 Jarvis Scan: existence kolineárních bodů v datasetu

Pokud vstupní množina bodů obsahuje kolineární body, mohla by nastat situace kdy bod k má stejný úhel jako jiný bod l vzhledem ke stejné přímce $\angle(p_j - 1, p_j, p_k) = (p_j - 1, p_j, p_l)$. Pro ošetření tohoto případu byla přidána podmínka pro stejné úhly, která do konvexní obálky přidá z takovéto dvojice bodů ten, který je od bodu p_j vzdálenější.

4.4 Automatické generování množin bodů

Uživetal ve spodní části ovládacího panelu má možnost generovat náhodně body. V rozbalovacím seznamu může vybrat z položek Circle, Square, Random a Raster a ve spinboxu může zvolit počet generovaných bodů od 10 do 1000000. Tlačítkem generate points se body přidají do Okna.

Circle

1. $pi = ((double)rand()/RAND_MAX) * 2 * M_PI;$
2. $point.setX(center.x() + sin(pi) * r);$
3. $point.setY(center.y() + cos(pi) * r);$
4. $points.push_back(point);$

Square

1. $x = (((double)rand()/RAND_MAX) * a) - a/2;$
2. $y = (((double)rand()/RAND_MAX) * a) - a/2;$

3. $point.setX(center.x() + x)$;
4. $point.setY(center.y() + y)$;
5. if (rand() % 2)
6. $x = rand()\%2?center.x() - a/2 : center.x() + a/2$;
7. $point.setX(x)$;
8. $else$
9. $y = rand()\%2?center.y() - a/2 : center.y() + a/2$;
10. $point.setY(y)$;
11. $points.push_back(point)$;

Raster

1. $x = (((double)rand()/RAND_MAX) * (a + n)) - a/2$;
2. $y = (((double)rand()/RAND_MAX) * (a + n)) - a/2$;
3. $x = (((int)x + n/2)/n) * n$;
4. $y = (((int)y + n/2)/n) * n$;
5. $point.setX(x + center.x() - n)$;
6. $point.setY(y + center.y() - n)$;
7. $points.push_back(point)$;

5 Vstup dat do aplikace

Vstup dat v aplikaci lze realizovat dvěma způsoby. Uživatel sám vkládá body do grafického okna za pomoci kurzoru myši nebo vygeneruje náhodné body funkcemi z kapitoly *Bonusové úlohy - Automatické generování množin bodů*.

6 Výstup aplikace

Výstupem grafického rozhraní aplikace je uzavřený polygon konvexní obálky vytvořený zadanou metodou nad vstupní množinou bodů. Další částí výstupu je časová náročnost výpočtu daného algoritmu pro aktuální množinu bodů v datasetu.

7 Dokumentace

7.1 Třídy

7.1.1 Algorithms

Třída Algorithms obsahuje metody zajišťující výpočty daných algoritmů.

double getPointLineDistance(QPoint q, QPoint p1, QPoint p2)
Návratová hodnota: *double*;
Metoda vrátí vzdálenost bodu *q* od přímky dané body *p1* a *p2*.

int getPointLinePosition(QPoint q,QPoint p1,QPoint p2)
Návratová hodnota: *integer*;
Metoda vrátí polohu bodu *q* vůči přímce dané body *p1* a *p2*. Hodnota 1 je pro polohu vlevo od přímky, 0 pro polohu vpravo od přímky a -2 pro polohu bodu na přímce.

double getAngle2Vectors(QPoint p1,QPoint p2,QPoint p3,QPoint p4)
Návratová hodnota: *double*;
Metoda vrátí úhel mezi přímkou zadané body p1, p2 a přímkou zadané body p3 a p4.

QPolygon strictlyConvex(QPolygon ch)
Návratová hodnota: *QPolygon*;
Metoda ze zadané obecné konvexní obálky ve formě polygonu vytvoří striktně konvexní obálku a vrátí jí ve formě polygonu.

QPolygon jarvisScan(std::vector$< QPoint >$ **points)**
Návratová hodnota: *QPolygon*;
Metoda z vektoru bodů vytvoří striktně konvexní obálku ve formě polygonu metodou Jarvis Scan.

QPolygon qHull(std::vector$< QPoint >$ **points)**
Návratová hodnota: *QPolygon*;
Metoda z vektoru bodů vytvoří striktně konvexní obálku ve formě polygonu metodou Quick Hull.

void qh(int s, int e, std::vector$< QPoint >$ **points, QPolygon ch)**
Návratová hodnota: *Void*;
Pomocná metoda k výpočtu konvexní obálky metodou Quick Hull.

QPolygon sweepLine(std::vector$< QPoint >$ **points)**
Návratová hodnota: *QPolygon*;
Metoda z vektoru bodů vytvoří striktně konvexní obálku ve formě polygonu metodou Sweep line.

QPolygon grahamScan(std::vector$< QPoint >$ **points)**
Návratová hodnota: *QPolygon*;
Metoda z vektoru bodů vytvoří striktně konvexní obálku ve formě polygonu metodou

Graham Scan.

7.1.2 SortbyX

Třída SortbyX slouží k porovnání souřadnic v ose x.

bool operator()(QPoint &p1, QPoint &p2)
Přetížený operátor () vrátí bod s větší souřadnicí x z dvojice bodů.

7.1.3 SortbyY

Třída SortbyY slouží k porovnání souřadnic v ose y.

bool operator()(QPoint &p1, QPoint &p2)
Přetížený operátor () vrátí bod s větší souřadnicí y z dvojice bodů.

7.1.4 sortPointsAngleQ

Třída sortPointsAngleQ obsahuje funkci pro seřazení bodů vzhledem k rovnoběžce s osou x procházející bodem q.

std::vector$< QPoint >$ **sortPointsByAngleQ (std::vector**$< QPoint >$ **points, QPoint q)**
Návratová hodnota: $vector< QPoint >$;
Metoda metoda vrátí vektor bodů, které jsou seřazeny podle velikosti úhlu mezi rovnoběžkou s osou x vedenou bodem q a přímkou danou bodem q a hodnoceným bodem.

8 Testování

Pro každou množinu bodů bylo provedeno 10 testů, výsledný čas uvedený v tabulkách je průměrem časů těchto testů.

8.1 Jarvis Scan

Jarvis scan

Circle	5 000	10 000	25 000	50 000	75 000	100 000	250 000	500 000	750 000	1 000 000
	[ms]									
1	2024	52121	-	-	-	-	-	-	-	-
2	2006	-	-	-	-	-	-	-	-	-
3	2003	-	-	-	-	-	-	-	-	-
4	1981	-	-	-	-	-	-	-	-	-
5	1980	-	-	-	-	-	-	-	-	-
6	1982	-	-	-	-	-	-	-	-	-
7	1970	-	-	-	-	-	-	-	-	-
8	2002	-	-	-	-	-	-	-	-	-
9	1981	-	-	-	-	-	-	-	-	-
10	2059	-	-	-	-	-	-	-	-	-
Průměr [ms]	**1998.8**	**52121**	**-**	**-**	**-**	**-**	**-**	**-**	**-**	**-**

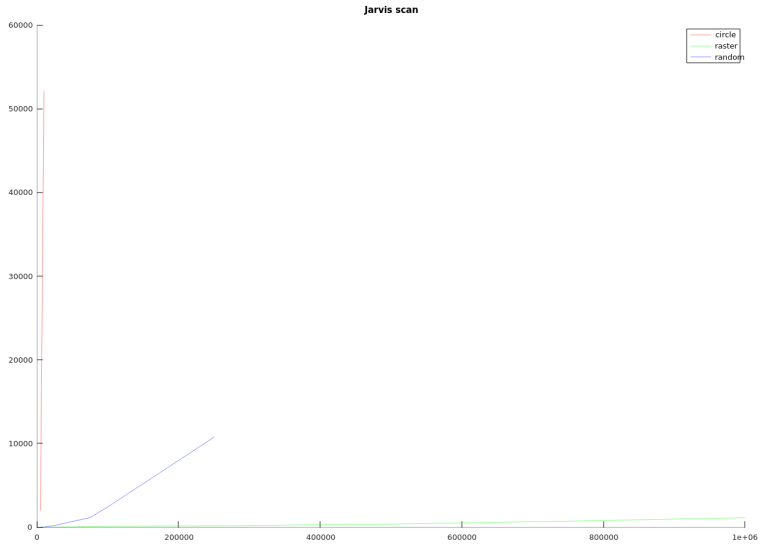
Raster	5 000	10 000	25 000	50 000	75 000	100 000	250 000	500 000	750 000	1 000 000
	[ms]									
1	1	6	13	33	68	103	142	444	731	1079
2	1	6	13	34	68	101	135	379	706	1081
3	1	6	13	35	68	107	138	351	715	1055
4	0	6	13	33	68	110	153	350	691	1030
5	1	6	13	38	67	103	139	361	732	1271
6	1	6	13	38	69	101	143	394	687	1225
7	1	6	13	39	68	102	138	348	684	1152
8	1	6	13	35	69	101	143	370	707	1052
9	1	6	13	34	68	100	178	340	751	1110
10	1	6	13	34	68	100	184	343	755	1176
Průměr [ms]	**0.9**	**6**	**13**	**35.3**	**68.1**	**102.8**	**149.3**	**368**	**715.9**	**1123.1**

Random	5 000	10 000	25 000	50 000	75 000	100 000	250 000	500 000	750 000	1 000 000
	[ms]									
1	12	42	197	660	1152	2438	10739	-	-	-
2	12	41	195	658	1142	2395	-	-	-	-
3	13	41	195	660	1143	2396	-	-	-	-
4	13	41	196	657	1136	2406	-	-	-	-
5	13	45	194	658	1152	2472	-	-	-	-
6	13	41	213	662	1141	2447	-	-	-	-
7	13	41	195	658	1153	2396	-	-	-	-
8	13	41	214	685	1137	2401	-	-	-	-
9	12	43	194	662	1132	2387	-	-	-	-
10	16	41	195	666	1143	2401	-	-	-	-
Průměr [ms]	**13**	**41.7**	**198.8**	**662.6**	**1143.1**	**2413.9**	**10739**	**-**	**-**	**-**

Obrázek 7: Testování časové náročnosti Jarvis Scan

Počet bodů	**Circle [ms]**	**Rastr [ms]**	**Random [ms]**
5000	79.1	0.9	13
10000	1998.8	6	13
25000	21482.2	13	198.8
50000	55892.4	35.3	662.6
75000	92519	68.1	1143.1
100000	133729	102.8	2413.9
250000	336309	149.3	10739
500000	610392	368	X
750000	X	715.9	X
1000000	X	1123.1	X

Tabulka 1: Jarvis Scan



Obrázek 8: Testování časové náročnosti Jarvis Scan - graf

8.2 Sweep Line

Sweep line

Circle	5 000	10 000	25 000	50 000	75 000	100 000	250 000	500 000	750 000	1 000 000
	[ms]									
1	0	0	2	5	10	13	18	51	111	177
2	0	0	1	4	10	13	18	49	117	178
3	0	0	1	4	9	12	18	48	105	158
4	0	0	1	4	10	12	19	49	98	190
5	0	0	1	4	9	12	17	47	103	175
6	0	0	1	4	10	12	18	52	99	183
7	0	0	1	4	9	12	18	48	102	160
8	0	0	1	4	8	12	19	48	99	167
9	0	0	1	4	8	12	17	47	103	172
10	0	0	1	4	8	13	18	49	99	156
Průměr [ms]	**0**	**0**	**1.1**	**4.1**	**9.1**	**12.3**	**18**	**48.8**	**103.6**	**171.6**

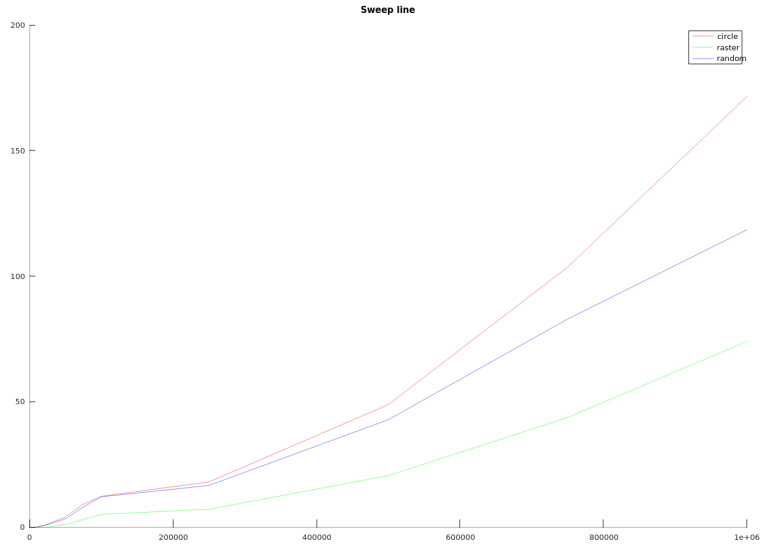
Raster	5 000	10 000	25 000	50 000	75 000	100 000	250 000	500 000	750 000	1 000 000
	[ms]									
1	0	0	0	1	3	5	7	21	42	64
2	0	0	0	1	3	5	7	20	42	66
3	0	0	0	1	3	5	7	20	44	74
4	0	0	0	1	3	5	7	20	44	69
5	0	0	0	1	3	5	8	20	43	85
6	0	0	0	1	3	5	7	21	42	73
7	0	0	0	1	3	5	7	19	42	78
8	0	0	0	1	3	6	7	21	42	67
9	0	0	0	1	3	5	7	22	52	73
10	0	0	0	1	3	5	7	21	44	91
Průměr [ms]	**0**	**0**	**0**	**1**	**3**	**5.1**	**7.1**	**20.5**	**43.7**	**74**

Random	5 000	10 000	25 000	50 000	75 000	100 000	250 000	500 000	750 000	1 000 000
	[ms]									
1	0	0	1	4	8	13	17	44	84	119
2	0	0	1	4	8	12	17	43	82	120
3	0	0	1	3	8	12	17	42	82	118
4	0	0	1	3	8	12	16	44	86	118
5	0	0	1	3	8	12	16	42	84	122
6	0	0	1	3	8	12	17	42	86	118
7	0	0	1	3	8	12	17	42	82	117
8	0	0	1	4	8	12	16	42	81	118
9	0	0	1	3	8	12	17	42	81	118
10	0	0	1	3	8	12	16	45	81	117
Průměr [ms]	**0**	**0**	**1**	**3.3**	**8**	**12.1**	**16.6**	**42.8**	**82.9**	**118.5**

Obrázek 9: Testování časové náročnosti Sweep Line

Počet bodů	Circle [ms]	Rastr [ms]	Random[ms]
5000	0	0	0
10000	0	0	0
25000	1.1	0	1
50000	4.1	1	3.3
75000	9.1	3	8
100000	12.3	5.1	12.1
250000	18	7.1	16.6
500000	48.8	20.5	42.8
750000	103.6	43.7	82.9
1000000	171.6	74	118.5

Tabulka 2: Sweep Line



Obrázek 10: Testování časové náročnosti Sweep Line - graf

8.3 Quick Hull

Quick hull

Circle	5 000	10 000	25 000	50 000	75 000	100 000	250 000	500 000	750 000	1 000 000
	[ms]									
1	7	178	861	2755	6080	-	-	-	-	-
2	7	178	711	2662	6183	-	-	-	-	-
3	7	176	678	2835	6488	-	-	-	-	-
4	7	178	621	2769	6468	-	-	-	-	-
5	7	189	617	2745	6389	-	-	-	-	-
6	7	191	618	2563	6551	-	-	-	-	-
7	7	178	648	2553	6568	-	-	-	-	-
8	7	179	674	2686	6228	-	-	-	-	-
9	8	190	654	2604	6387	-	-	-	-	-
10	8	244	656	2609	6313	-	-	-	-	-
Průměr [ms]	**7.2**	**188.1**	**673.8**	**2678.1**	**6365.5**	**-**	**-**	**-**	**-**	**-**

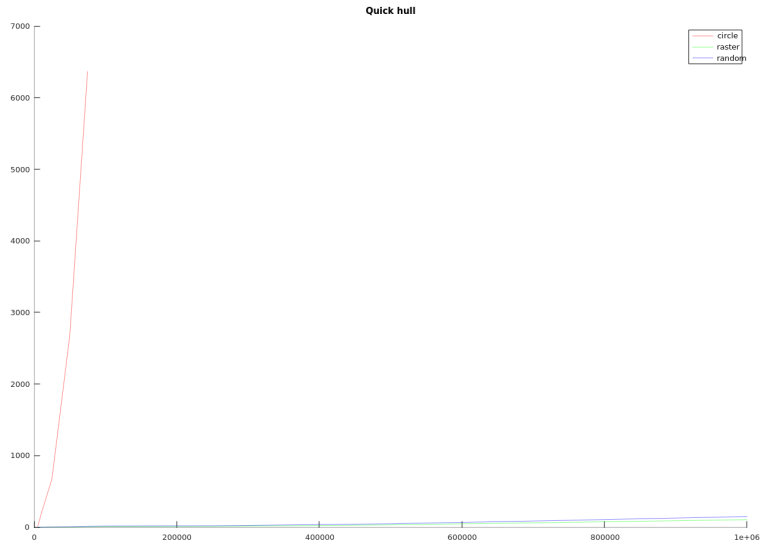
Raster	5 000	10 000	25 000	50 000	75 000	100 000	250 000	500 000	750 000	1 000 000
	[ms]									
1	0	0	1	2	6	9	12	34	69	111
2	0	0	1	2	5	8	12	31	67	113
3	0	0	1	2	5	8	12	32	70	103
4	0	0	1	2	5	8	12	31	66	101
5	0	0	1	2	5	9	12	31	65	108
6	0	0	1	2	5	9	12	31	65	101
7	0	0	1	2	5	8	12	30	65	101
8	0	0	1	2	5	9	11	32	66	102
9	0	0	1	2	5	8	12	31	65	101
10	0	0	1	2	5	9	12	32	69	100
Průměr [ms]	**0**	**0**	**1**	**2**	**5.1**	**8.5**	**11.9**	**31.5**	**66.7**	**104.1**

Random	5 000	10 000	25 000	50 000	75 000	100 000	250 000	500 000	750 000	1 000 000
	[ms]									
1	0	1	3	5	10	16	19	46	100	155
2	0	1	3	5	10	15	19	51	97	155
3	0	1	3	4	9	15	21	49	94	146
4	0	1	2	5	10	15	20	47	96	151
5	0	1	2	5	10	13	18	50	99	152
6	0	1	3	5	9	13	19	46	94	143
7	0	1	3	5	10	14	18	46	93	149
8	0	1	2	5	12	15	20	47	93	147
9	0	1	3	5	12	15	19	44	96	144
10	0	1	2	5	10	14	19	46	94	143
Průměr [ms]	**0**	**1**	**2.6**	**4.9**	**10.2**	**14.5**	**19.2**	**47.2**	**95.6**	**148.5**

Obrázek 11: Testování časové náročnosti Quick Hull

Počet bodů	**Circle [ms]**	**Rastr [ms]**	**Random[ms]**
5000	7.2	0	0
10000	188.1	0	0
25000	673.8	1	2.6
50000	2678.1	2	4.9
75000	6365.5	5.1	10.2
100000	X	8.5	14.5
250000	X	11.9	19.2
500000	X	31.5	47.2
750000	X	66.7	95.6
1000000	X	104.1	148.5

Tabulka 3: Quick Hull



Obrázek 12: Testování časové náročnosti Quick Hull - graf

9 Závěr

Pro tvorbu konvexních obálek byly vytvořeny tři základní metody (Jarvis Scan, Quick Hull, Sweep Line) a jedna bonusová (Graham Scan). Pro nízký počet vstupních bodů fungují všechny algoritmy bez problému. Problémové situace jsou z velké části ošetřeny (duplicitní body apod.). Pro větší množinu vstupních bodů se bez problému podařilo tvořit konvexní obálku pouze metodou Sweep Line. Metoda Jarvis Scan pro náhodně generované body na kružnici v počtu nad 100000 "běží"velmi dlouho, problém se nepodařilo odhalit. Metoda Quick Hull je ze své podstaty nevhodná pro tvorbu konvexní obálky s body na kružnici. Časová náročnost algoritmů pro zadané množiny bodů jsou obsahem předchozí kapitoly Testování. Algoritmus Graham Scan pro malé množství bodů funguje bezchybně, ovšem pro vygenerované větší množství bodů také kolabuje. Tato chyba je nejspíše důsledkem složitosti funkce pro řazení bodů podle úhlu.

10 Náměty pro vylepšení

Pro zlepšení testování algoritmů by bylo výhodné udělat aplikaci výcevláknovou aby bylo vidět že aplikace provádí výpočty a UI se nestalo neaktivní.

11 Reference

1. BAYER, Tomáš. Metody konstrukce konvexní obálky [online][cit. 5.11.2019]. Dostupné z: https://web.natur.cuni.cz/ bayertom/images/courses/Adk/adk4.pdf