<table>
<tr><td colspan="5">ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA STAVEBNÍ, OBOR GEODÉZIE A KARTOGRAFIE
KATEDRA GEOMATIKY</td></tr>
<tr><td colspan="5">název předmětu
ALGORITMY V DIGITÁLNÍ KARTOGRAFII</td></tr>
<tr><td>číslo úlohy

2</td><td colspan="4">název úlohy

Konvexní obálka a její konstrukce</td></tr>
<tr><td>školní rok

2020/21</td><td>studijní skupina

60</td><td>zpracovali:
Michal Zíma,
Tomáš Lauwereys</td><td>datum

15.11. 2020</td><td>klasifikace</td></tr>
</table>

Zadání

Anotace:

V rámci úlohy byla vytvořena aplikace v prostředí QT, která umí vygenerovat a vizualizovat body o zadaném počtu na kružnici, mřížce nebo náhodně podle volby uživatele. Dále má uživatel možnost si zvolit jeden ze tří algoritmů (Jarvis Scan, Quick Hull, Swep Line) pomocí nichž dojde k vytvoření konvexní obálky. Aplikace vždy vypíše kolik ms trvalo algoritmu pro zvolený počet bodů vytvořit konvexní obálku a zároveň ji vykreslí.

Přesné zadání:

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

Výstup: $\mathcal{H}(P)$.

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

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

Vstupní množiny bodů včetně vygenerovaných konvexních obálek vhodně vizualizujte. Pro množiny $n \in < 1000, 1000000 >$ 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, dosažené výsledky zhodnoťte. Rozhodněte, která z těchto metod je s ohledem na časovou složitost a typ vstupní množiny P, nejvhodnější.

Bonusové úlohy

1. Konstrukce konvexní obálky metodou Graham Scan.
2. Konstrukce striktně konvexních obálek pro všechny uvedené algoritmy.
3. Ošetření singulárního případu u Jarvis Scan: existence kolineárních bodů v datasetu.
4. Algoritmus pro automatické generování nekonvexních polygonů

Vytvořený program neobsahuje řešení ani jedné bonusové úlohy.

Zvolené algoritmy

Jarvis Scan

Jedná se o metodu, která umí vyhledávat body konvexní obálky za pomoci hledání maximálního úhlu. Musí ovšem platit podmínka, že tři body neleží na jedné přímce. Výhodou algoritmu je jeho jednoduchost. Na druhou stranu je jeho nevýhodou časová náročnost.

Rychlost algoritmu je O(n2) pro body, které leží na kružnici. Nejčastěji pak O(n*h), kde *n* je počet bodů vstupní množiny a *h* je počet bodů konvexní obálky.

První bod, ze kterého Jarvis Scan vychází je bodem s nejmenší y souřadnicí, protože u tohoto bodu máme jistotu, že bude ležet na konvexní obálce. Dále tímto bodem vedeme rovnoběžku s osou x a následně projdeme všechny body množiny [*i*]. Poté měříme úhel mezi rovnoběžkou s osoou x a přímkou, kterou nám určí bod p (tedy první bod) a *i*. Pokud tyto úhly vzájemně porovnáme, tak největší úhel nám značí následující bod námi hledané konvexní obálky. Poté je zapotřebí ušetřit tzv. singulární situaci, kdy bod s dosavadním maximálním úhlem a současný bod i leží na jedné přímce (úhly jsou stejné). Pokud nastane taková situace, je jako dosavadní bod s největším úhlem přidán takový bod, který je vzdálenější od bodu p_j.

V tento moment se bod i stane novým "prvním bodem" a z bodu p se stane bod určující přímku, od které se měří úhly. Takto pokračuje proces stále dokola do momentu, dokud se bodem i nestane opět bod p.

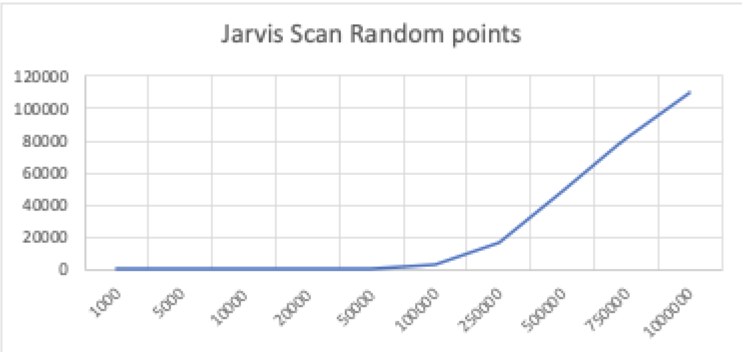
Algoritmus je založen na hledání průsečíků polopřímky p s hranami polygonu P. Polopřímka p je vedena bodem Q - tvoří tzv. paprsek (Ray). Počet průsečíků následně určuje výsledek algoritmu.

1. *Nalezení pivota q,* $q = min\,(y_i)$*,*
2. *Přidej* $q \rightarrow H$*,*
3. *Inicializuj* $p_{j-1} \in X,\ p_j = q,\ \ p_{j+1} = p_{j-1}$,
4. *Opakuj, dokud* $p_{j+1} \neq q$:
5. *Nalezni* $p_{j+1} = arg\ max \forall\ p_i \in P < (p_{j-1},\ p_j,\ p_i)$,
6. *Přidej* $p_{j+1} \rightarrow H$.
7. $p_{j-1} = p_j;\ \ p_j = p_{j+1}$.

Časová náročnost algoritmu Jarvis Scan podle počtu zvolených bodů a typu obrazce:

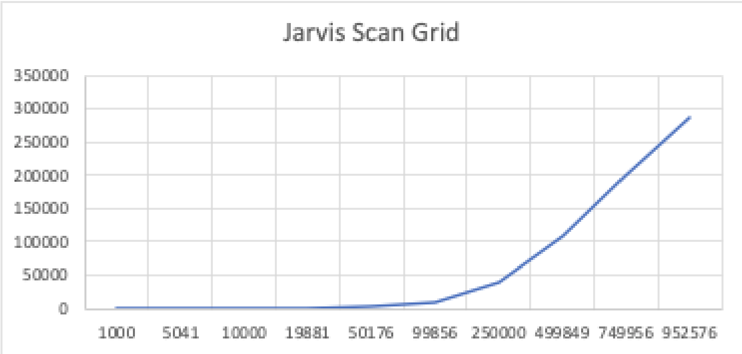
Random

počet bodů	n1	n2	n3	n4	n5	n6	n7	n8	n9	n10	průměr	rozptyl
1000	3	2	2	3	2	3	2	3	2	3	2,5	0,3
5000	21	20	21	17	18	21	21	21	26	23	20,9	6,1
10000	52	53	47	57	56	44	68	58	51	61	54,7	48,0
20000	167	176	168	171	191	167	176	178	155	158	170,7	107,1
50000	1041	882	889	860	1038	850	983	1001	917	934	939,5	5155,8
100000	3441	3131	3217	3415	3472	3636	3436	3394	3395	3081	3361,8	28573,5
250000	17254	17651	17051	18221	17079	16046	17711	16356	16677	16662	17070,8	442980,0
500000	49540	49840	46495	50930	47851	47201	46791	49880	46863	49023	48441,4	2517196,3
750000	82357	82204	79836	83550	81056	79343	80902	82840	79298	79206	81059,2	2596960,4
1000000	111033	110894	110100	110168	110866	108754	109359	109670	109090	108046	109798	996108,7



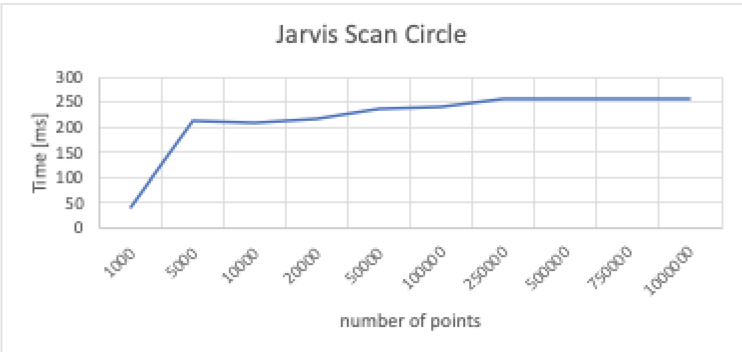
Grid

počet bodů	n1	n2	n3	n4	n5	n6	n7	n8	n9	n10	průměr	rozptyl
1000	0	0	0	0	0	0	0	0	0	0	0	0,0
5041	5	3	3	3	3	5	4	5	5	3	3,9	1,0
10000	311	308	312	310	319	313	312	310	309	308	311,2	10,4
19881	866	870	866	865	866	869	864	865	866	867	866,4	3,4
50176	3481	3465	3471	3461	3475	3460	3465	3486	3477	3482	3472,3	86,0
99856	9730	9762	9733	9717	9729	9740	9737	9763	9744	9723	9737,8	230,8
250000	38535	38494	38568	38471	38497	38489	38560	38513	38492	38566	38518,5	1291,4
499849	108785	109035	108851	108845	108862	108869	108669	108859	108705	108939	108841,9	11123,7
749956	199907	199486	199634	200244	199858	200160	199955	200328	200082	199921	199957,5	68512,5
952576	286732	286421	286064	286056	286922	281799	287395	291864	293929	293561	288074,3	14729095,6



Circle

počet bodů	n1	n2	n3	n4	n5	n6	n7	n8	n9	n10	průměr	rozptyl
1000	43	42	43	47	42	42	42	43	42	44	43	2,4
5000	212	212	210	210	215	211	211	210	210	210	211,1	2,5
10000	207	208	213	205	208	207	207	206	206	205	207,2	5,3
20000	221	216	217	216	216	216	218	218	224	221	218,3	7,8
50000	236	235	235	236	236	239	237	235	235	235	235,9	1,7
100000	240	239	239	242	239	239	238	241	238	244	239,9	3,7
250000	254	262	256	254	254	254	255	256	256	255	255,6	5,8
500000	256	258	254	254	256	257	256	255	254	254	255,4	2,0
750000	256	256	256	255	257	256	255	255	256	254	255,6	0,7
1000000	255	257	255	257	255	255	255	257	255	255	255,6	0,9



Graham Scan

Tento algoritmus funguje na principu zjišťování CCW orientace trojúhelníku (levotočivý směr). Výhodou algoritmu je vysoká rychlost O(n*log(n)), kde *n* je počet bodů vstupní množiny.

U Graham Scan algoritmu je nejprve zapotřebí seřadit body podle y souřadnice a bod, který má nejmenší souřadnici označíme jako *p*. Následně vypočteme směrnici vůči ose x pro všechny body množiny. Poté body seřadíme podle velikosti směrnice.

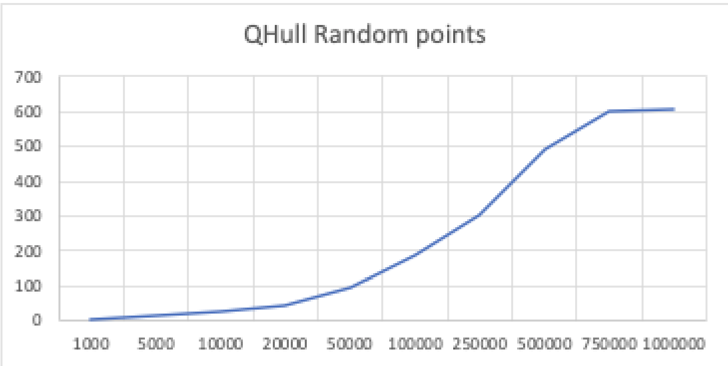
V dalším kroku probíhá testování CCW orientace na posledních dvou bodech přidaných do množiny konvexní obálky včetně posledního bodu s největší orientací. Tento postup probíhá do stavu, dokud nenalezneme námi hledanou konvexní obálku.

1. *Nalezení pivota* $q = min_{\forall p_i \in S}(y_i)$, $q \in H$.
2. *Setřídění* $\forall p_i \in S$ *dle* $\omega_i = \angle(p_i, q, x)$, *index j odpovídá setříděnému pořadí.*
3. *pokud* $\omega_k = \omega_l$, *vymaž bof* p_k, p_l *blíže ke q.*
4. *Inicializuj j = W; S = 0.*
5. $S \leftarrow q$, $S \leftarrow p_1$, *(indexy posledních dvou prvků* p_t, p_{t-1}*)*
6. *Opakuj pro j < n :*
7. *if* p_j *vlevo od* p_t, p_{t-1}
8. $S \leftarrow p_1$
9. *j= j+1*
10. *else pop S.*

Časová náročnost algoritmu Graham Scan podle počtu zvolených bodů a typu obrazce:

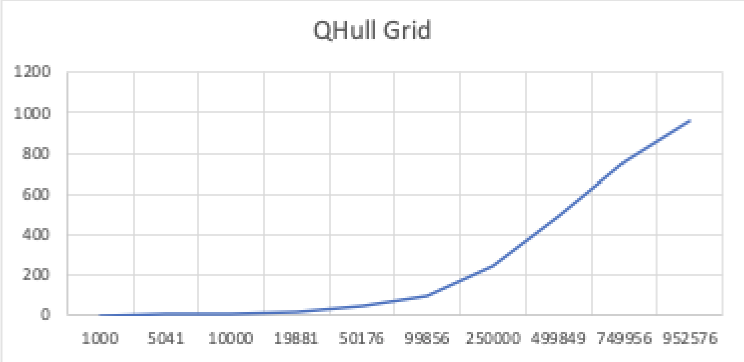
Random

počet bodů	n1	n2	n3	n4	n5	n6	n7	n8	n9	n10	průměr	rozptyl
1000	3	3	3	3	2	3	2	4	5	3	3,1	0,7
5000	13	14	13	13	11	14	15	13	14	13	13,3	1,0
10000	26	20	24	27	30	26	27	29	26	29	26,4	7,4
20000	54	48	50	44	51	38	46	44	45	47	46,7	17,8
50000	73	91	109	104	118	93	97	84	86	98	95,3	152,4
100000	242	192	179	167	177	197	153	166	189	196	185,8	538,2
250000	290	264	264	293	321	237	231	410	378	358	304,6	3324,8
500000	556	408	509	507	508	512	407	510	408	607	493,2	4013,8
750000	594	529	658	532	659	594	663	532	594	658	601,3	2855,8
1000000	616	689	618	542	705	551	695	540	544	546	604,6	4375,6



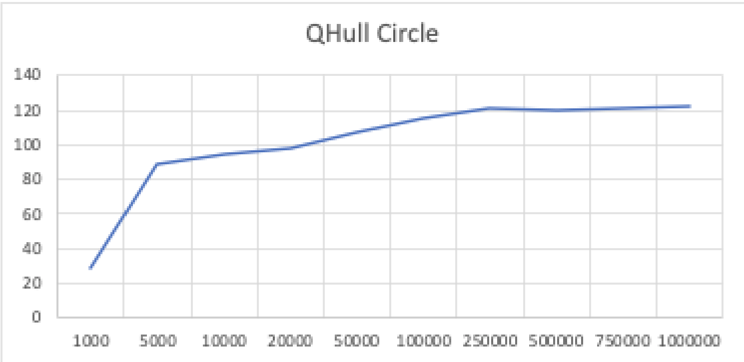
Grid

počet bodů	n1	n2	n3	n4	n5	n6	n7	n8	n9	n10	průměr	rozptyl
1000	0	0	0	0	0	0	0	0	0	0	0	0,0
5041	4	5	4	4	5	6	4	4	6	4	4,6	0,6
10000	9	12	12	9	9	13	9	9	10	9	10,1	2,3
19881	18	18	23	24	18	18	19	24	18	20	20	6,2
50176	48	53	48	49	52	47	48	52	47	47	49,1	4,9
99856	95	95	98	96	95	99	99	96	98	98	96,9	2,5
250000	250	246	250	245	242	241	248	247	241	241	245,1	12,1
499849	501	490	493	496	496	491	497	494	491	491	494	11,0
749956	760	758	765	756	760	756	753	758	756	760	758,2	9,8
952576	962	959	960	961	960	957	960	963	957	966	960,5	6,7



Circle

počet bodů	n1	n2	n3	n4	n5	n6	n7	n8	n9	n10	průměr	rozptyl
1000	28	28	27	28	28	28	27	27	28	32	28,1	1,9
5000	89	87	88	91	89	87	87	88	91	88	88,5	2,1
10000	95	93	95	99	94	93	94	94	95	95	94,7	2,6
20000	101	98	97	96	97	97	98	97	96	98	97,5	1,9
50000	106	106	111	106	105	108	106	106	107	106	106,7	2,6
100000	115	115	114	115	116	117	115	116	115	114	115,2	0,8
250000	122	122	121	121	122	121	121	120	121	121	121,2	0,4
500000	120	120	119	120	121	120	121	119	120	121	120,1	0,5
750000	120	121	120	121	121	123	120	122	121	121	121	0,8
1000000	121	124	123	122	120	122	121	127	125	120	122,5	4,7



Quick Hull

Algoritmus vyhledává body konvexní obálky na základě vyhledávání nejvzdálenějšího bodu od přímky, která je určena 2 body množiny bodů. Rychlost algoritmu je obdobná předchozímu algoritmu. V ojedinělých případech může být dokonce stejně rychlý jako u Jarvis Scan.

V tomto případě vycházíme z přímky, kterou nám určují dva body s nejmenší a největší souřadnicí x. Definovaná přímka rozdělí množinu bodů na dvě poloviny a v každé polovině je zapotřebí projít všechny body. U bodů zjišťujeme jejich vzdálenost od přímky (x_min a x_max). Jakmile nalezneme nejvzdálenější bod od přímky, přidáme tento bod do konvexní obálky a spojíme ho s body x_min a x_max. Díky tomu vzniknou další přímky a od nich se postupuje obdobně znova. Postup se vykonává pro dolní i horní množinu bodů do situace, dokud nevytvoříme konvexní obálku.

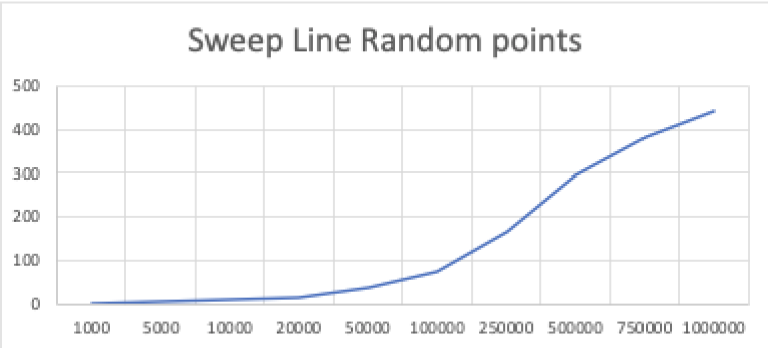
1. $H = \emptyset,\ S_U = \emptyset,\ S_L = \emptyset$
2. $q_1 = min_{\forall p_i \in S}(x_i),\ q_3 = max_{\forall p_i \in S}(x_i)$
3. $S_U \leftarrow q_1,\ S_U \leftarrow q_3$
4. $S_L \leftarrow q_1,\ S_L \leftarrow q_3$
5. $for\ \forall p_i \in S$
    a. $if\ (p \in \sigma_l(q_1, q_3))\ S_U \leftarrow q_i$
    b. $else\ S_L \leftarrow q_i$

6. $H \leftarrow q_3$ *// Přidej bod c do H*
7. *Quick Hull (1, 0,S_U, H) //Upper Hull*
8. $H \leftarrow q_1$ *// Přidej bod c do H*
9. *Quick Hull (1, 0,S_L, H) //Lower Hull*

Časová náročnost algoritmu Quick Hull podle počtu zvolených bodů a typu obrazce:

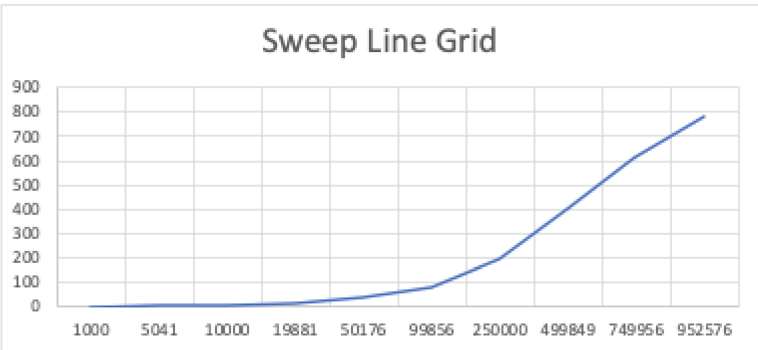
Random

počet bodů	n1	n2	n3	n4	n5	n6	n7	n8	n9	n10	průměr	rozptyl
1000	0	0	0	0	0	0	0	0	0	0	0	0,0
5000	5	3	3	5	4	3	5	5	5	3	4,1	0,9
10000	6	10	7	7	7	9	7	7	10	9	7,9	1,9
20000	14	14	16	14	17	14	14	19	19	15	15,6	3,8
50000	36	36	38	36	37	36	36	40	36	37	36,8	1,6
100000	71	73	72	72	74	71	71	76	71	73	72,4	2,4
250000	168	169	168	167	166	166	167	165	167	166	166,9	1,3
500000	298	294	292	293	293	298	293	298	297	293	294,9	5,7
750000	388	386	382	380	386	379	384	378	380	384	382,7	10,4
1000000	444	440	442	438	446	441	445	439	441	438	441,4	7,2



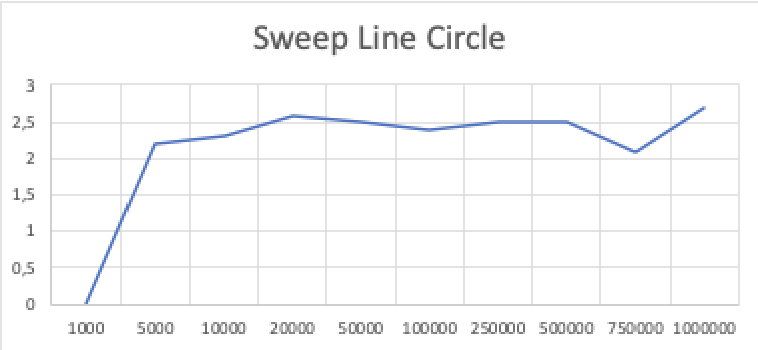
Grid

počet bodů	n1	n2	n3	n4	n5	n6	n7	n8	n9	n10	průměr	rozptyl
1000	0	0	0	0	0	0	0	0	0	0	0	0,0
5041	5	3	3	3	3	4	5	4	3	5	3,8	0,8
10000	8	10	7	7	7	10	7	7	7	10	8	1,8
19881	14	18	14	16	14	19	17	15	14	19	16	4,0
50176	39	38	37	38	44	38	39	41	43	37	39,4	5,4
99856	78	79	77	79	78	79	78	77	81	77	78,3	1,4
250000	197	201	196	201	196	197	201	203	198	195	198,5	6,9
499849	404	406	403	400	403	401	400	401	402	410	403	8,6
749956	618	615	617	617	628	619	617	618	621	617	618,7	11,8
952576	778	781	777	780	780	778	780	780	778	780	779,2	1,6



Circle

počet bodů	n1	n2	n3	n4	n5	n6	n7	n8	n9	n10	průměr	rozptyl
1000	0	0	0	0	0	0	0	0	0	0	0	0,0
5000	2	2	3	2	2	3	2	2	2	2	2,2	0,2
10000	2	3	2	2	2	3	3	2	2	2	2,3	0,2
20000	2	3	2	3	2	3	2	3	3	3	2,6	0,2
50000	3	3	2	2	2	3	2	3	3	2	2,5	0,3
100000	3	3	2	3	3	2	2	2	2	2	2,4	0,2
250000	2	3	3	3	3	2	3	2	2	2	2,5	0,3
500000	3	2	3	3	3	2	3	2	2	2	2,5	0,3
750000	2	2	2	2	3	2	2	2	2	2	2,1	0,1
1000000	3	2	2	2	3	3	3	3	4	2	2,7	0,4



Problematické situace a jejich rozbor

V průběhu nastala pouze jedna problémová situace a to taková, kdy QT nezvládlo vytvořit mřížku o 1000 x 1000 bodů. Příčinu tohoto chování se nám nepodařilo identifikovat.

Vstupní data

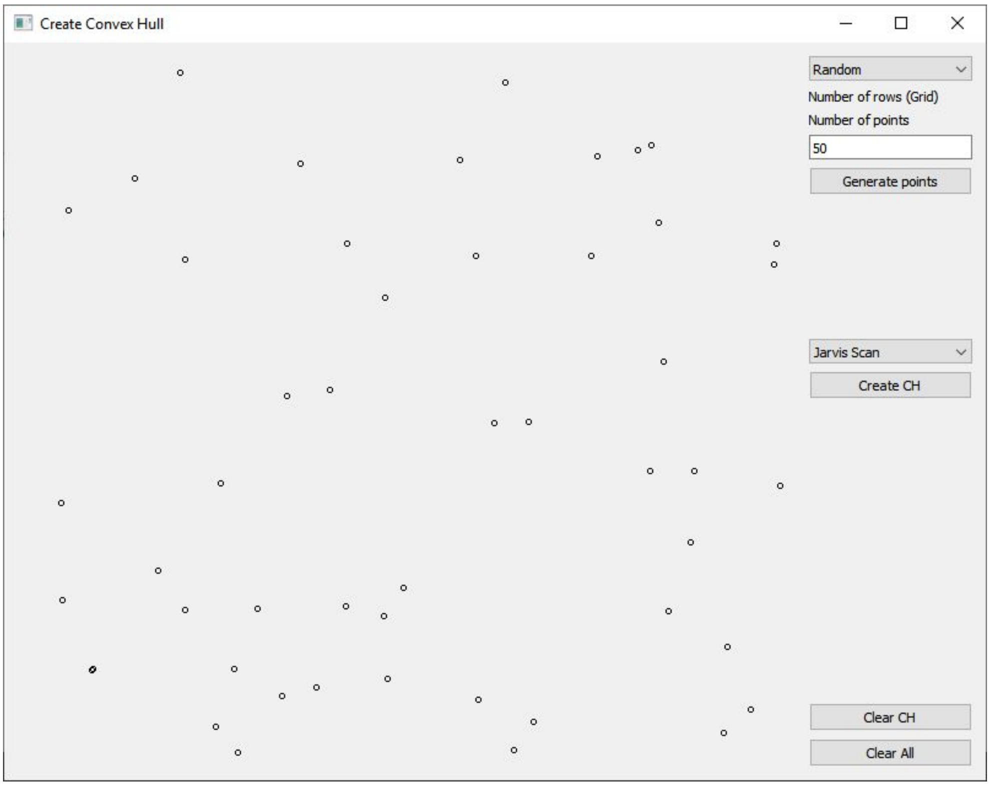
Vstupní data vytvoří uživatel v aplikaci. Do pole napíše počet bodů a zvolí metodu vygenerování (náhodné body, body v mřížce a body na kružnici).

Výstupní data

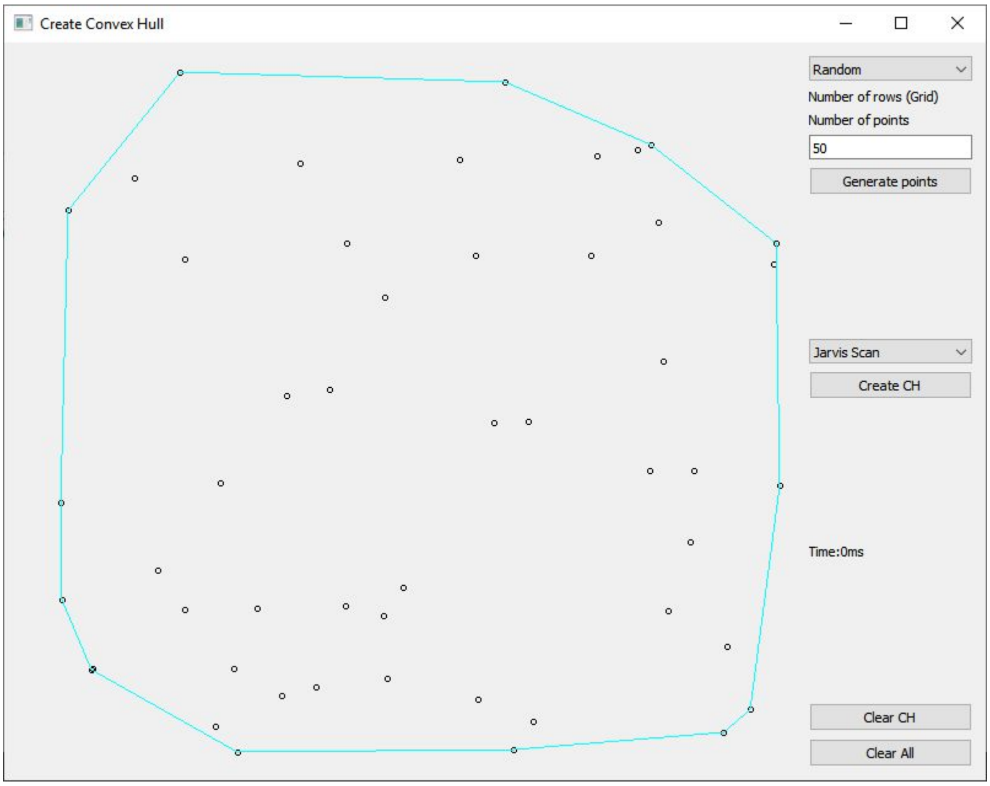
Výstupem je grafická aplikace, která graficky určí polohu konvexní obálky okolo vygenerovaných bodů. V aplikaci uživatel definuje způsob generování bodů a vybere typ algoritmu, kterým se vytvoří konvexní obálka. Po vygenerování konvexní obálky je zobrazen délka procesu v ms.

Vytvořená aplikace

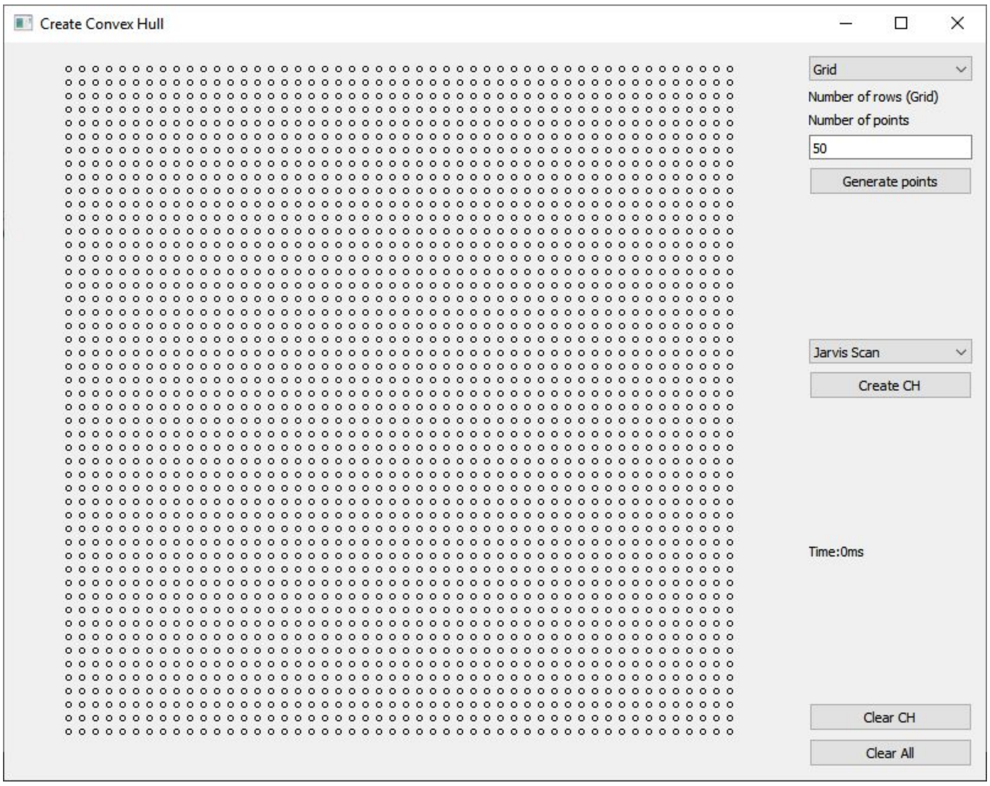
Následující snímky vytvořené aplikace zobrazují řešení daných situací:



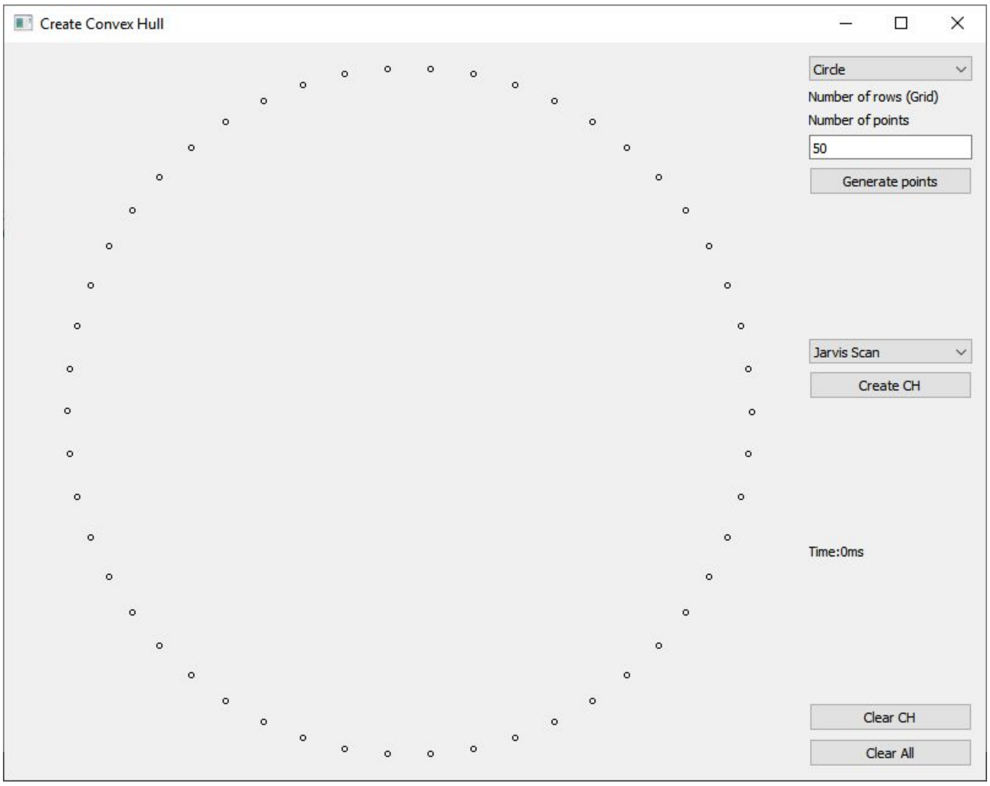
Vygenerování 50 náhodných bodů



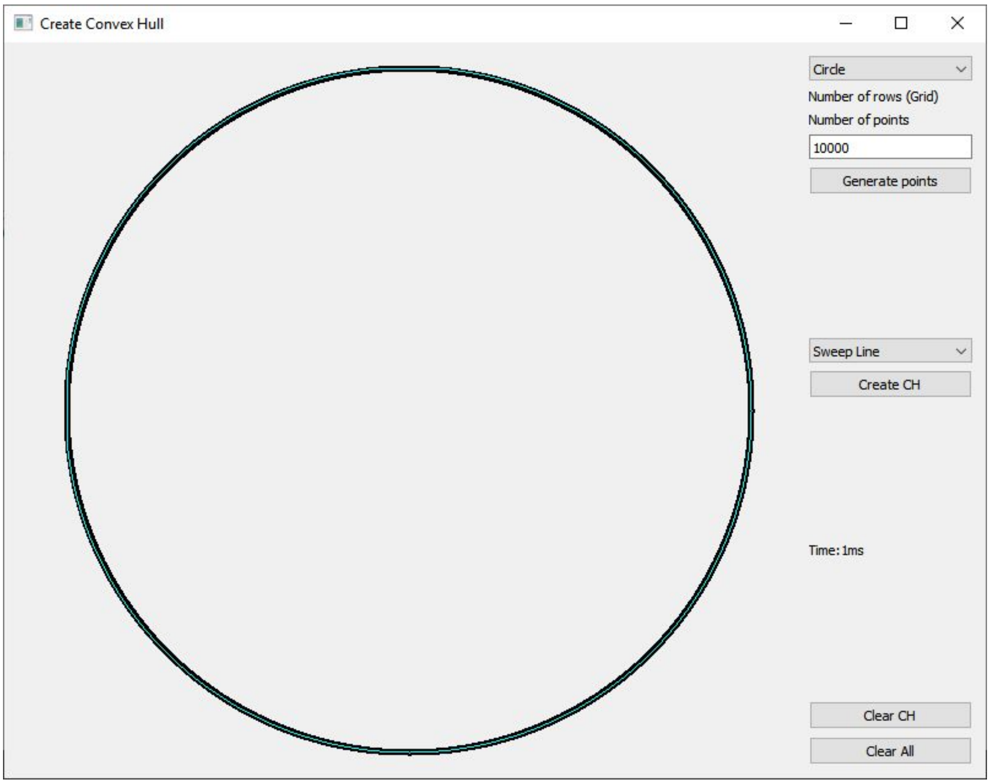
Vytvoření komplexní obálky za použití Jarvis Scan algoritmu



Vygenerování bodů v mřížce a rozměrech 50 bodů x 50 bodů



Vygenerování 50 bodů na kružnici



Vytvoření konvexní obálky algoritmem Sweep Line

Dokumentace

Třídy, datové položky a metody

Aplikace obsahuje pět tříd - Algorithms, Draw, sortByX, sortByY a Widget. Každá třída je zastoupena hlavičkovým souborem a zdrojovým souborem. V hlavičkových souborech jsou definovány společně se třídou její proměnné a metody.

- **Třída Algorithms**

  Třída Algorithms obsahuje celkem čtyři metody, které jsou použity pro vyřešení zadaného problému. Datovými typy metod byly QPointF a QPolygonF, oba s plovoucí desetinnou čárkou.

  double getAngle(QPointF &p1,QPointF &p2,QPointF &p3,QPointF &p4);

  Metoda vrací hodnotu úhlu mezi dvěma vektory.

  int getPointLinePosition(QPointF &q, QPointF &p1, QPointF &p2);

  Tato metoda určuje pozici bodu *q* vůči zadané hraně polygonu *P*. Vrací hodnoty 1 (bod leží v levé polorovině), 0 (bod leží v pravé polorovině) a -1 (bod leží na hraně). Ošetření případu, že bod leží na hraně, bylo vytvořeno na základě podmínky sestrojení trojúhelníku. Trojúhelník lze sestrojit tehdy pokud součet délek dvou stran je větší než délka třetí strany. Pokud podmínka neplatí, body leží v rovině. Jelikož uživatel kliká myší a né vždy klikne na hranu byla zvolena tolerance 0,2. Pokud podmínku, že leží na hraně nesplňuje, je následně vypočítán determinant vektorů p1p2 a qp1. Pokud je determinant větší než 0 funkce vrací hodnotu 1, pokud je záporný, vrací hodnotu 0.

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

  Metoda, která vrací hodnotu vzdálenosti mezi bodem a vektorem.

QPolygon jarvis(QPolygon &points);

Metoda, která vrací konvexní obálku určenou pomocí Jarvis algoritmu.

QPolygon qhull(QPolygon &points);

Metoda, která vrací konvexní obálku určenou pomocí algoritmu QHull

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

Metoda, která ukládá body, které patří do konvexní obálky. Je to pomocná metoda k metodě *qhull.*

QPolygon sweepLine(QPolygon &points);

Metoda, která vrací konvexní obálku určenou pomocí Sweep Line algoritmu.

static QPolygon removeDuplicate(QPolygon &points);

Metoda, která kontroluje a odstraňuje případnou duplicitu ve vygenerovaných bodech.

- **Třída Draw**

void *mousePressEvent*(QMouseEvent *e);

Tato metoda ukládá souřadnice bodu *q*, které uživatel zadá kliknutím myši do kreslícího pole.

void *paintEvent*(QPaintEvent *e);

Touto metodou se vykreslují vygenerované body a vytvořena konvexní obálka. Konvexní obálka je navíc vykreslena tirkisově.

QPolygon getPoints(){return points;}

Metoda (getter) vrací polygon bodů.

QPolygon getCH() {return ch;}

Metoda (getter) vrací body konvexní obálky.

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

Metoda, která ukládá body konvexní obálky do datových typů třídy Draw.

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

Metoda, která ukládá vygenerované body do datových typů třídy Draw.

QPolygon generateRandom(int n, int height, int width);

Metoda, která generuje náhodné body. Počet náhodných bodů definuje uživatel v aplikaci.

void clearCH(){ch.clear(); repaint();}

Metoda, která maže konvexní obálku.

void clearPoints(){points.clear(); repaint();}

Metoda, která maže načtené body.

QPolygon generateGrid(int n, int height, int width);

Metoda, která generuje body v mřížce. Počet bodů jedné strany mřížky definuje uživatel v aplikaci.

QPolygon generateCircle(int n, int height, int width);

Metoda, která generuje body na kružnici. Počet bodů na kružnici definuje uživatel v aplikaci.

- **Třída sortByX**

  Tato třída seřadí vstupní body podle souřadnice X.

- **Třída sortByY**

  Tato třída seřadí vstupní body podle souřadnice Y.

- **Třída Widget**

  void on_pushButton_clicked**();**

  Po kliknutí na tlačítko *pushButton* (Create CH) dojde k vytvoření konvexní obálky algoritmem, který uživatel vybere ve výše umístěném comboBoxu_1.

  void on_pushButton_2_clicked**();**

  Po kliknutí na tlačítko *pushButton_2* (Clear CH) dojde ke smazání vytvořené konvexní obálky.

  void on_pushButton_3_clicked**();**

  Po kliknutí na tlačítko *pushButton_3* (Clear all) dojde ke smazání vygenerovaných a vytvořené konvexní obálky.

  void on_pushButton_4_clicked**();**

  Po kliknutí na tlačítko *pushButton_4* (Generate points) dojde k vygenerování bodů na základě výběru ve výše zvoleném comboBoxu_2 v počtu, který uživatel definuje v *lineEdit.*

Závěr

V rámci této úlohy byla vytvořena aplikace, která je schopna na základě vygenerovaných bodů zkonstruovat konvexní obálku pomocí tří různých algoritmů s výpisem délky řešení.

Z přiložených grafů a tabulek je dobře vidět, jaká je časová náročnost jednotlivých algoritmů při určitém množství bodů a typu obrazce. Jako nejpomalejší se jeví algoritmus jarvis Scan. Naopak nejrychlejší algoritmus byl z naměřených hodnot určen algoritmus Sweep Line.

V Praze dne 15. 11. 2020

Bc. Michal Zíma,
Bc. Tomáš Lauwereys

Seznam literatury a zdrojů