ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

FAKULTA STAVEBNÍ, OBOR GEODÉZIE A KARTOGRAFIE

KATEDRA GEOMATIKY

název předmětu

ALGORITMY DIGITÁLNÍ KARTOGRAFIE A GIS

číslo úlohy	název úlohy:			
2	**Konvexní obálky**			
školní rok:	semestr:	zpracovali:	datum:	klasifikace:
2019/20	zimní	**David Němec, Jan Šartner**	5. 11.	

Technická zpráva

Konvexní obálky

1. Zadání

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

Výstup: H(P)

Nad množinou P implementujte následující algoritmy pro konstrukci 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 (10×) 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.

2. Doplňující úlohy

2.1 Konstrukce konvexní obálky metodou Graham Scan

-řešeno

2.2 Konstrukce striktně konvexních obálek pro všechny algoritmy

-řešeno

2.3 Ošetření singulárního případu u Jarvis Scan: existence kolineárních bodů v datasetu

-řešeno

2.4 Konstrukce Minimum Area Enclosing box některou z metod

-řešeno

2.5 Algoritmus pro automatické generování konvexních/nekonvexních množin bodů různých tvarů (kruh, elipsa, čtverec, star-shaped, atd.)

-řešeno

3. Problematika nalezení konvexní obálky a minimálního ohraničujícího obdélníku - vzorce

3.1 Algoritmus Jarvis Scan

Výpočet úhlu mezi 2 vektory u, v:

$$\omega = \left|\text{acos}\left(\frac{u * v}{|u| \cdot |v|}\right)\right|$$

3.2 Algoritmus Quick Hull

Výpočet pozice bodu vzhledem k linii:

$$\vec{u} = |P_1P_2|$$

$$\vec{v} = |P_1Q|$$

$$t = u \times v$$

-na základě znaménka hodnoty t lze rozhodnout, ve které polorovině, určené analyzovanou hranou, bod Q leží.

3.3 Algoritmus Sweep line

Výpočet pozice bodu vzhledem k linii (viz. 3.2)

3.4 Algoritmus Graham Scan

Výpočet úhlu mezi 2 vektory u, v (viz. 3.1)
Výpočet pozice bodu vzhledem k linii (viz. 3.2)

3.5 Algoritmus Minimum Area Enclosing Box using Convex Hull

Výpočet úhlu mezi 2 vektory u, v (viz. 3.1)
Výpočet pozice bodu vzhledem k linii (viz. 3.2)

Rotace / transformace obdélníku (x,y → x‘,y‘) o úhel ω:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{bmatrix} cos\omega & sin\omega \\ -sin\omega & cos\omega \end{bmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix}$$

4. Problematika nalezení konvexní obálky a minimálního ohraničujícího obdélníku - popis

4.1 Algoritmus Jarvis Scan

Nejprve je vyhledán tzv „pivot“, neboli bod s nejmenší souřadnicí Y. Dále algoritmus využívá vzorce pro výpočet úhlu mezi dvěma přímkami, které jsou dané vektory. V první iteraci je přidán do konvexní obálky bod který maximalizuje úhel (bod, pivot, osa x). Následně jsou přidávány body maximalizující úhel (bod, poslední bod v obálce, předposlední bod v obálce), dokud se posledním bodem obálky nestane opět pivot.

Ve zdrojovém kódu je algoritmus napsán tak, aby výsledná obálka neobsahovala duplicitní body a obsahovala jinak všechny body, které se na ni nachází i přes existenci kolineárních bodů v datasetu (viz. 5.1).

4.2 Algoritmus Quick Hull

V datasetu jsou vyhledány body s nejnižší a nejvyšší souřadnicí X. Jejich spojnicí jsou rozděleny body do dvou skupin podle toho, ve které polorovině se nachází. Nad oběma polorovinami je provedena rekurzivní procedura. Ta je nejprve provedena nad původními dvěma body, přičemž vyhledá nejvzdálenější bod od jejich spojnice, nacházející se v pravé polorovině (vně dosavadní konvexní obálky), pokud existuje.

Ve zdrojovém kódu byl ponechán algoritmus, jenž v rekurzivní proceduře neuvažuje body na spojnici bodů, nad kterými je procedura prováděna. To zapříčiní, že výsledná konvexní obálka nebude obsahovat všechny body na ní ležící, nýbrž bude přímo striktní. Toto řešení bylo ponecháno z důvodu efektivity (především rychlosti) algoritmu. Výsledná obálka neobsahuje duplicitní body.

4.3 Algoritmus Sweep line

Nejprve jsou body v datasetu setříděny dle souřadnice X. V rámci prvních dvou bodů je inicializována „konvexní obálka“, resp. sekvence bodů. Dále je setříděný dataset postupně procházen a s přibývajícími body se aktualizuje konvexní obálka.

Na základě zdrojového kódu vykresluje aplikace konvexní obálku, která neobsahuje duplicitní body a obsahuje jinak všechny body na ní ležící. Sekvence bodů je řešena pomocí seznamu předchůdců a následovníků, které jsou aktualizovány s každým nově přidaným bodem. Seznamy se nejprve aktualizují na základě toho, zdali přidaný bod leží nad přímkou rovnoběžnou s osou x procházející posledním zařazeným bodem, nebo pod ní. Následně jsou aktualizovány ještě z hlediska úpravy horní / dolní tečny (zachování konvexnosti obálky)

4.4 Algoritmus Graham Scan

Nejprve je nalezen pivot. Body v datasetu jsou setříděny na základě úhlu (bod, pivot, osa x), takže vznikne tzv. „star-sahped polygon“. U bodů se stejným úhlem se maže ten, který je bližší pivotu. Body se procházejí postupně v rámci orientace CVV a rozhoduje se, v jaké polorovině vůči poslední hraně obálky jsou (při první iteraci je jako hrana brána osa x). V případě, že analyzovaný bod leží v levé polorovině, přidá se do konvexní obálky. V opačném případě jsou hrany (poslední body) z obálky odebírány, dokud neplatí první případ.

Ve zdrojovém kódu je algoritmus zkonstruován tak, že konvexní obálka neobsahuje duplicitní body a obsahuje jinak všechny body na ní ležící, protože analyzovaný bod je do obálky přidán i v případě, že leží na přímce dané poslední hranou k.o. Jedinou výjimkou mohou být kolineární body nacházející se poblíž pivotu. Pokud existovaly kolineární body, které byly bezprostřední předchůdci či následovníci pivotu, a měly být součástí konvexní obálky, pak tyto body nebyly uváženy z důvodu mazání bodů se stejným úhlem a menší vzdáleností k pivotu.

4.5 Algoritmus Minimum Area Enclosing Box using Convex Hull

Tento algoritmus pracuje s již vytvořenou konvexní obálkou a znalostí faktu, že nejméně jedna strana minimálního ohraničujícího obdélníku je kolineární s hranou konvexní obálky. Daný obdélník je natáčen pomocí transformace ve směru hran konvexní obálky. Výsledný obdélník je ten s nejmenší plochou.

Jelikož minimální ohraničující obdélník nemá souřadnice vrcholů v celých číslech, jsou ve zdrojovém kódu využívány pro tento případ funkce předchozích algoritmů upravené pro reálná čísla.

5. Problematické situace (+ řešení doplňkových úloh)

5.1 Existence kolineárních bodů v datasetu

V algoritmu Jarvis Scan je tento problém řešen tak, že při porovnávání úhlu je bližší bod se stejným úhlem přidán dříve. To zajistí přidání všech bodů konvexní obálky.

```
když (úhly bodu s max. úhlem a testovaného bodu jsou stejné a zároveň vzdálenost od
testovaného bodu je menší než vzdálenost od bodu s max. úhlem)
{
        aktualizuj hodnotu úhlu; //Není tolik nutné, zde hraje roli pouze tolerance
                                 //výpočtu.
        aktualizuj index bodu;
}
```

Z hlediska efektivity není u algoritmu Quick Hull tento problém řešen a je generována přímo striktní obálka.

Metody Sweep Line se tento problém netýká.

V případě Graham Scanu byl podobně jako u Jarvis Scanu tento problém vyřešen přidáním bodů, které se nacházely na linii, při vyhodnocování jejich polohy oproti poslední hraně konvexní obálky.

```
když (analyzovaný bod se oproti poslední hraně k.o. nenachází v levé polorovině)
{
        přidej bod do obálky;
        zvětši index j o 1;
}
```

5.2 Konstrukce striktních konvexních obálek

Do zdrojového kódu byla přidána funkce (strictCH, viz. 9), která odebere mezilehlé kolineární body z konvexní obálky. Funkce je vykonána po zaškrtnutí CheckBoxu „Strict convex hull“.

```
pro(všechny body v datasetu)
{
        když(3 po sobě jdoucí body jsou kolineární)
        {
                vymaž prostřední z nich;
                sniž index i o 1;  //pro případ více než 3 kolineárních bodů
        }
}

vrať konvexní obálku;
```

5.3 Generování bodů

V aplikaci je možné generovat různé typy množin bodů (random, square, circle, ellipse, star-shaped)

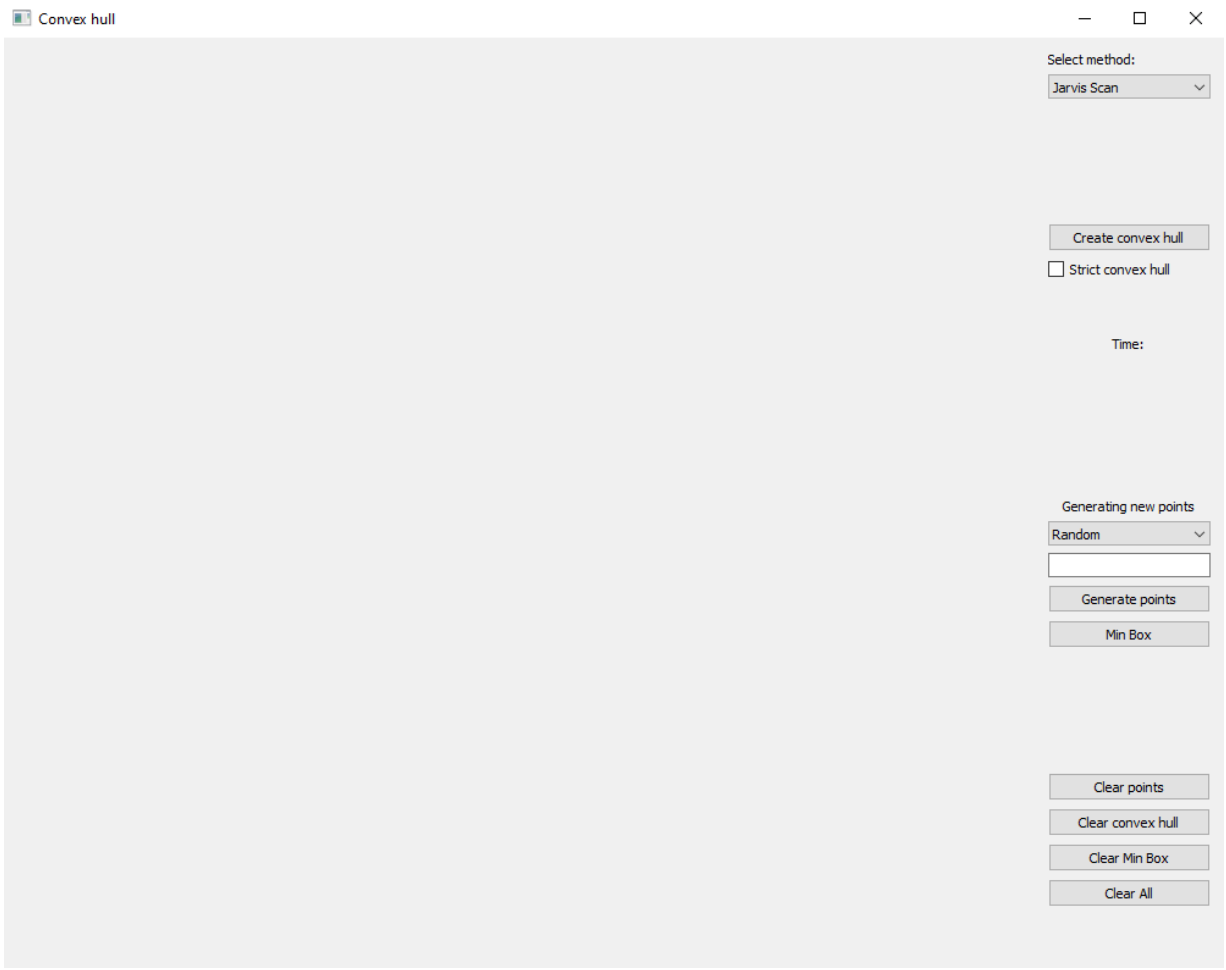
6. Vstupní data

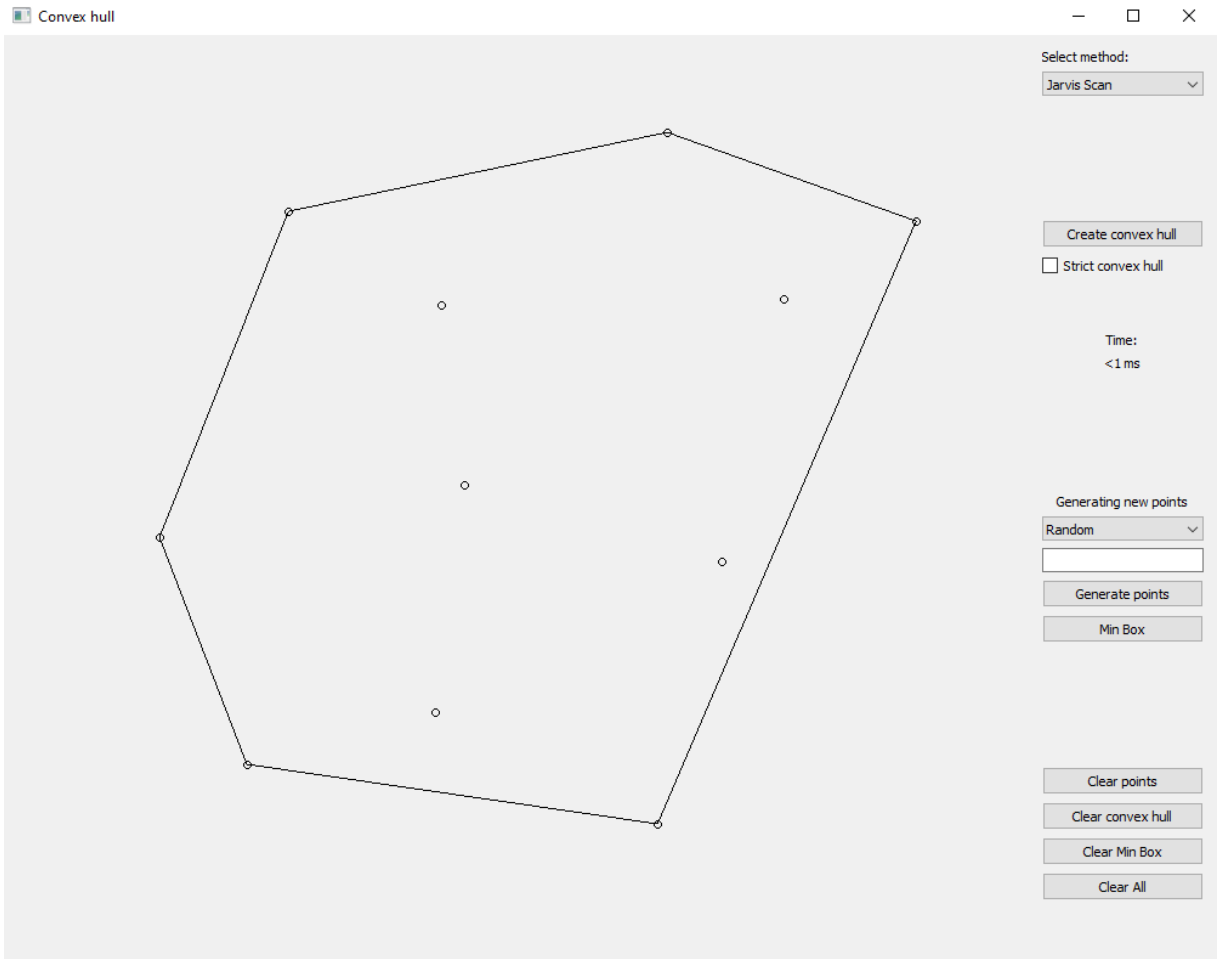
K aplikaci nejsou k dispozici žádná vstupní data ani možnost jejich načtení. K výpočtům se používá generování bodů dle zadaného tvaru, nebo lze body ručně zvolit klikáním levého tlačítka myši. Datový typ: std::vector<QPoint>

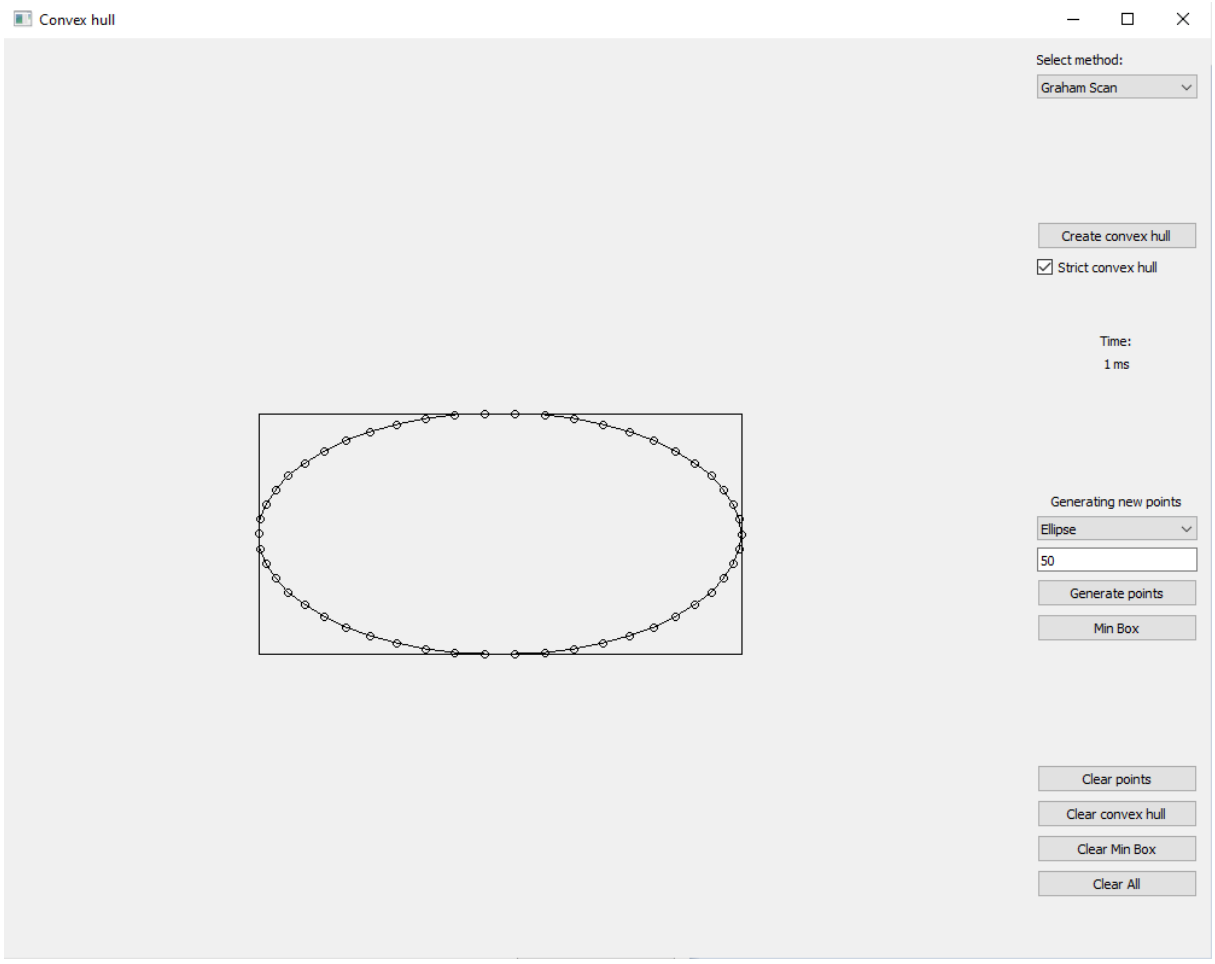
7. Výstupní data

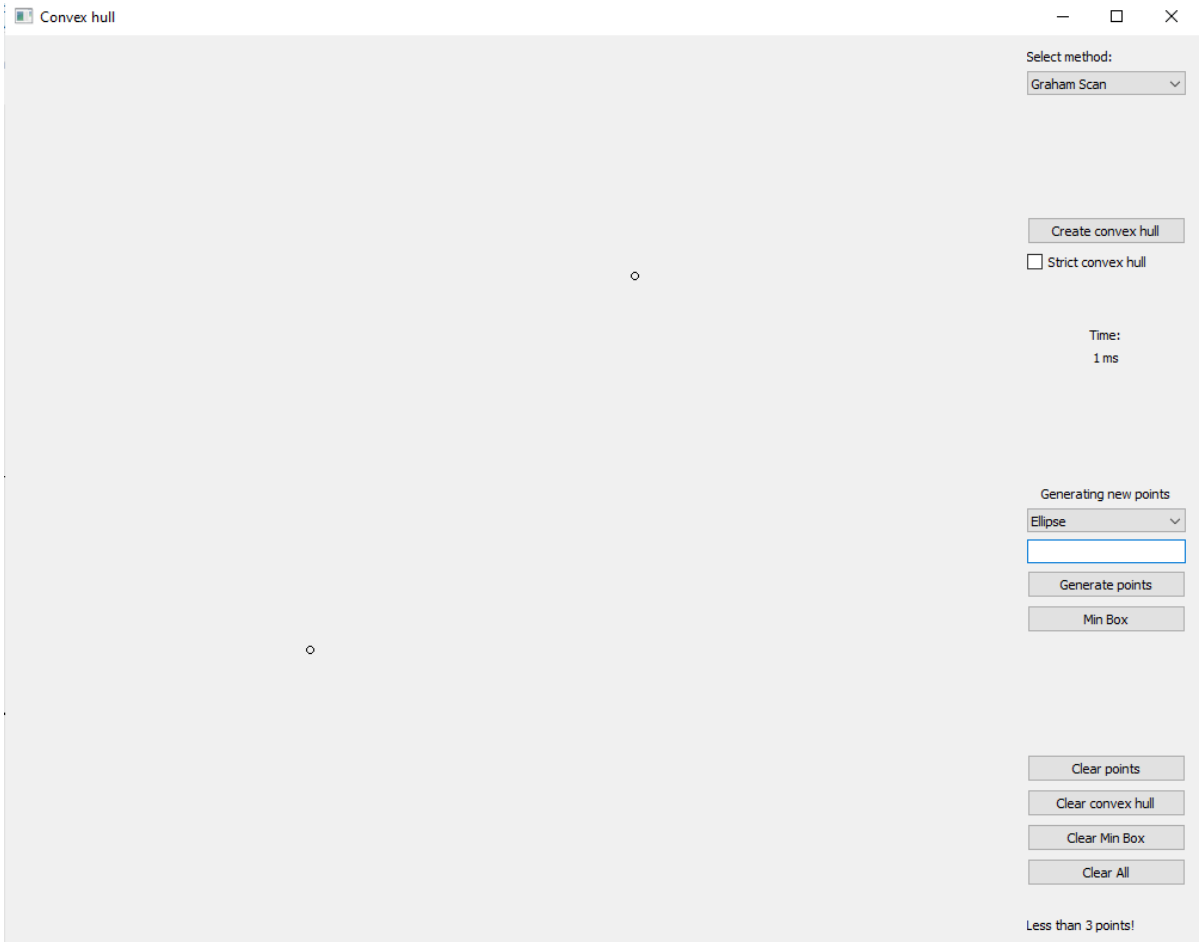
Aplikace neposkytuje žádná výstupní data. Výsledkem je vykreslení bodů, konvexní obálky a minimálního ohraničujícího obdélníku. Zároveň je zde uvedena doba trvání výpočtu. Datový typ: QPolygon

8. Vzhled aplikace




Convex hull
Select method:
Jarvis Scan
Create convex hull
Strict convex hull
Time:
<1 ms
Generating new points
Random
Generate points
Min Box
Clear points
Clear convex hull
Clear Min Box
Clear All


Convex hull
Select method:
Graham Scan
Create convex hull
Strict convex hull
Time:
1 ms
Generating new points
Ellipse
50
Generate points
Min Box
Clear points
Clear convex hull
Clear Min Box
Clear All



9. Dokumentace

Třídy:

Algorithms:

Třída obsahuje 12 funkcí, které se používají pro analýzu zadaných množin bodů:

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

Funkce zjišťuje, zda bod q leží vlevo či vpravo od vybrané úsečky spojující body p1 a p2. Také určuje, zda neleží přímo na úsečce. Funkce vrací hodnoty *int(1 pokud je bod vlevo, 0 pokud je bod vpravo, -1 pokud bod leží na přímce).*

static double getAngle2Vectors(QPoint &p1, QPoint &p2, QPoint &p3, QPoint &p4);

Funkce vrací velikost úhlu mezi dvěma vektory, které jsou určeny body p1,p2 a p3,p4. Úhel je vracen v radiánech.

static double getPointLineDistance(QPoint &q, QPoint &p1, QPoint &p2);

Funkce vrací vzdálenost bodu q od linie tvořené body p1 a p2.

static double getDistance2Points(QPoint &p1, QPoint &p2);

Funkce vrací vzdálenost mezi body p1 a p2.

static void rotatePolygon(QPolygonF &pol, double angle);

Funkce transformuje polzgon pol o úhel angle.

static QPolygon jarvisScan(std::vecto□<QPoint> &points);
Funkce vrací konvexní obálku typu QPolygon, kterou počítá pomocí metody Jarvis Scan.

static QPolygon qHull(std::vecto□<QPoint> &points);
Funkce vrací konvexní obálku typu QPolygon, kterou počítá pomocí metody Quick Hull

static QPolygon sweepLine(std::vector<QPoint> &points);
Funkce vrací konvexní obálku typu QPolygon, kterou počítá pomocí metody Sweep Line.

static QPolygon grahamScan(std::vecto□<QPoint> &points);
Funkce vrací konvexní obálku typu QPolygon, kterou počítá pomocí metody Graham Scan.

static void qh(int s, int e, std::vecto□<QPoint> &points, QPolygon &ch);
Rekurzivní funkce pro vyhledávání nejvzdálenějšího bodu v metodě Quick Hull.

static QPolygon strictCH(QPolygon &ch);
Funkce vrací konvexní obálku s odstraněnými body, které by jinak ležely na úsečce mezi jinými dvěma body.

static QPolygonF minimumRectanhle(QPolygon &ch);
Funkce vrací polygon opsaného obdélníka s nejmenší plochou. Používá přitom postupnou transformaci jednotlivých hran konvexní obálky a vyhledává nejmenší obsah. Jedná se tedy o metodu Brutal Force.

GeneratePoints:
Třída obsahuje 5 funkcí určené k automatickému generování množin bodů.

static std::vecto□<QPoint> generatingSquare(int &n);
Funkce generuje vektor bodů ve tvaru grid o velikosti n*n.

static std::vecto□<QPoint> generatingCircle(int &n);
Funkce generuje n-počet bodů na kružnici.

static std::vecto□<QPoint> generatingEllipse(int &n);
Funkce generuje n-počet bodů na elipse.

static std::vector<QPoint> generatingRandomField(int &n);
Funkce generuje n-počet náhodných bodů.

static std::vecto□<QPoint> generatingStar(int &n);
Funkce generuje n-počet bodů ve tvaru Star.

Draw:
Třída obsahuje 3 privátní proměnné a 10 funkcí. Třída se používá pro vykreslení bodů, konvexní obálky, obdélníka s minimální plochou a získání souřadnic určovaného bodu na obrazovce.

Proměnné v třídě draw:
std::vecto□<QPoint> points;
Vektor bodů, se kterými bude program dále pracovat.

QPolygon ch;

Polygon obsahující konvexní obálku.

QPolygonF minBounda□y;

Polygon obsahující obdélník o minimální ploše.

Funkce v třídě draw:

*void mousePressEvent(QMouseEvent *e);*

Funkce snímá pozici myši při kliknutí.

*void paintEvent(QPaintEvent *e);*

Funkce vykresluje generované a vytvořené body, konvexní obálku a obdélník o minimální ploše.

void clearCH() {ch.clea□(); □epaint();}

Funkce vymaže konvexní obálku a překreslí Canvas.

void clearPoints() {points.clea□(); □epaint();}

Funkce vymaže body v *points* a překreslí Canvas.

void clea□MinBounda□y() {minBounda□y.clea□(); □epaint();}

Funkce vymaže polygon obdélníka s minimální plochou a překreslí Canvas.

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

Funkce nastaví polygon konvexní obálky.

void setPoints(std::vecto□<QPoint> &pts) {points=pts;}

Funkce nastaví vektor bodů.

void setMinBoundary(QPolygonF &bound) {minBounda□y=bound;}

Funkce nastaví polygon obdélníka s minimální plochou.

QPolygon getCH() {□etu□n ch;}

Funkce vrací konvexní obálku.

std::vecto□<QPoint> getPoints() {□etu□n points;}

Funkce vrací vektor bodů.

SortbyY:

Operátor. Srovná QPointy podle velikosti Y. Pokud je Y stejné, řadí dle X.

SortbyYF:

Obdobný operátor jako SortbyY pro reálná čísla.

SortbyX:

Operátor. Srovná QPointy podle velikosti X. Pokud je X stejné, řadí dle Y.

SortbyXF:

Obdobný operátor jako SortbyX pro reálná čísla.

Widgets:

Třída obsahuje 7 funkcí, které rozhodují o následných procesech po stisknutí daného tlačítka.

void on_pushButton_clicked();

Funkce se spustí po stisknutí tlačítka Create convex hull a vytvoří konvexní obálku nad množinou bodů za pomocí metody, která se vybere v ComboBoxu nad tlačítkem.

void on_pushButton_2_clicked();

Funkce se spustí po stisknutí tlačítka Clear points a smaže vytvořené body.

void on_pushButton_3_clicked();

Funkce se spustí po stisknutí tlačítka Clear convex hull a smaže vytvořenou konvexní obálku.

void on_pushButton_4_clicked();

Funkce se spustí po stisknutí tlačítka Clear Min Box a smaže vytvořenou obdélník s minimální plochou.

void on_pushButton_4_clicked();

Funkce se spustí po stisknutí tlačítka Clear All a smaže vše, co bylo vytvořeno.

void on_pushButton_genPoints_clicked();

Funkce se spustí po stisknutí tlačítka Generate points a vygeneruje množinu bodů dle přednastavených parametrů. Ty jsou n: počet bodů a tvar množiny. Mřížka, kruh, elipsa, star shape, random.

void on_pushButton_minBox_clicked();

Funkce se spustí po stisknutí tlačítka Min Box a vytvoří obdélník s minimální plochou.

10. Závěr

Byla vytvořena aplikace podle zadání. Řešeny byly všechny doplňkové úlohy.

10.1 Naměřené hodnoty

Formát:	random [n]
	circle [n]
	ellipse [n]
	grid [n*n]

dle formátu:	Jarvis Scan t [ms]										Průměr [ms]	Rozptyl [ms]
1000	3	3	3	4	4	4	4	3	3	3	3,4	0,5
	30	30	31	30	30	30	29	30	30	30	30,0	0,5
	29	29	29	28	28	29	29	28	30	30	28,9	0,7
											Příliš dlouhý čas	
2000	8	8	8	7	9	7	9	7	8	9	8,0	0,8
	70	71	70	71	71	70	70	71	70	70	70,4	0,5
	63	61	61	60	60	61	60	62	61	61	61,0	0,9
5000	28	29	31	30	31	35	34	30	31	29	30,8	2,2
	185	172	175	180	173	171	174	175	173	174	175,2	4,2
	151	152	149	153	152	151	157	151	153	153	152,2	2,1
10000	82	104	89	100	99	97	90	109	96	96	96,2	7,7
	360	357	361	359	362	363	362	363	362	362	361,1	1,9
	304	302	299	304	329	303	302	302	304	300	304,9	8,6
50000	1747	1673	1844	1831	1700	1742	1860	1765	1762	1696	1762,0	64,8
	1786	1795	1790	1800	1786	1797	1804	1802	1799	1793	1795,2	6,4
	1508	1581	1507	1505	1507	1510	1511	1506	1522	1519	1517,6	23,0
100000	5593	5664	5567	5670	5913	5663	6150	5552	5775	5452	5699,9	202,7
	3632	3638	3742	3648	3736	3667	3745	3660	3706	3637	3681,1	46,4
	3001	3026	3034	3026	3017	3030	3003	3001	2998	3030	3016,6	14,4
250000	24343	24209	24465	24893	23810	25278	24859	24515	24765	24822	24595,9	416,0
	9166	9207	9167	9257	9146	9169	9279	9178	9252	9183	9200,4	46,1
	7615	7577	7602	7701	7578	7658	7581	7656	7593	5624	7418,5	631,9
500000	60098	59008	59598	60110	60032	59899	59147	60100	59547	59217	59675,6	432,3
	18327	18312	18434	18446	18396	18400	18425	18397	18355	18412	18390,4	45,0
	15250	15136	15200	15102	15266	15231	15190	15222	15147	15230	15197,4	53,5
750000	92585	91141	92122	91123	92648	91236	91470	92322	91954	91366	91796,7	600,3
	27537	27594	27686	27621	27600	27547	27592	27544	27609	27671	27600,1	50,5
	22900	22898	22949	22920	22890	22914	22900	22932	22888	22879	22907,0	21,7
1000000	126793	123569	125828	125012	124369	124110	125853	123659	125174	126030	125039,7	1092,4
	36870	36936	36983	36951	36944	36887	36891	36920	36936	36944	36926,2	34,4
	30555	30519	30506	30600	30502	30522	30589	30544	30527	30536	30540,0	33,0

dle formátu:	Quick Hull t [ms]										Průměr [ms]	Rozptyl [ms]
1000	1	1	1	1	1	1	1	1	1	1	1,0	0,0
	6	5	5	5	5	5	5	5	5	5	5,1	0,3
	4	4	4	4	4	4	4	4	4	4	4,0	0,0
	799	789	781	782	787	788	790	796	782	779	787,3	6,6
2000	2	3	2	2	2	2	2	2	2	2	2,1	0,3
	10	10	10	10	10	10	9	10	10	10	9,9	0,3
	8	9	8	8	9	8	8	8	8	9	8,3	0,5
	3384	3379	3420	3560	3350	3385	3357	3372	3425	3405	3403,7	60,2
5000	6	6	6	6	7	6	6	6	6	6	6,1	0,3
	25	25	24	25	25	25	26	25	25	25	25,0	0,5
	21	21	21	21	21	21	21	21	21	21	21,0	0,0
	23790	23870	23762	23900	23784	23824	23813	23800	23771	23849	23816,3	44,8
10000	11	11	11	10	12	11	10	11	11	11	10,9	0,6
	54	54	53	54	54	53	53	54	54	54	53,7	0,5
	43	42	43	43	43	43	43	42	43	43	42,8	0,4
											Nelze vygenerovat	
50000	50	54	47	50	51	50	50	52	53	54	51,1	2,2
	1793	1788	1790	1801	1830	1790	1812	1796	1795	1798	1799,3	12,8
	1513	1511	1513	1507	1510	1525	1540	1523	1504	1511	1515,7	10,7
100000	94	95	96	105	97	108	96	94	93	107	98,5	5,8
	3675	3666	3646	3652	3660	3728	3758	3677	3655	3672	3678,9	36,0
	3021	3030	3025	3015	3020	3037	3030	3030	3035	3023	3026,6	7,0
250000	241	239	247	241	247	244	234	250	293	245	248,1	16,4
	9245	9131	9222	9177	9183	9167	9194	9163	9109	9128	9171,9	42,4
	7605	7591	7680	7628	7623	7629	7583	7642	7592	7633	7620,6	29,1
500000	486	487	491	483	484	488	485	487	534	519	494,4	17,4
	18459	18375	18363	18400	18439	18328	18399	18409	18423	18380	18397,5	38,2
	15247	15189	15198	15223	15204	15181	15206	15250	15233	15197	15212,8	24,1
750000	738	773	773	751	745	735	738	766	746	746	751,1	14,4
	27616	27696	27641	27650	27714	27666	27620	27703	27654	27642	27660,2	34,1
	22811	22994	22748	22850	22795	22913	22856	22930	22814	22863	22857,4	72,3
1000000	987	994	992	980	996	998	997	1014	994	990	994,2	8,8
	36847	36741	36798	36822	36814	36793	36702	36755	36828	36781	36788,1	44,4
	30484	30579	30740	30600	30566	30497	30523	30580	30512	30493	30557,4	76,5

dle formátu:	Sweep Line t [ms]										Průměr [ms]	Rozptyl [ms]
1000	1	1	1	1	1	1	1	1	1	1	1,0	0,0
	1	1	1	1	1	1	1	1	1	1	1,0	0,0
	1	1	1	1	1	1	1	1	1	1	1,0	0,0
	769	778	779	773	780	769	799	779	776	777	777,9	8,4
2000	1	1	2	1	1	1	2	1	1	2	1,3	0,5
	2	1	1	1	1	2	2	1	2	1	1,4	0,5
	1	1	1	2	1	1	1	1	1	1	1,1	0,3
	3349	3343	3337	3322	3340	3341	3336	3351	3348	3331	3339,8	8,9
5000	4	4	3	4	4	4	4	4	3	4	3,8	0,4
	3	3	4	3	4	4	3	3	4	3	3,4	0,5
	3	3	3	3	3	4	3	3	3	3	3,1	0,3
	23453	23469	23598	23488	23501	23413	23466	23522	23468	23455	23483,3	49,9
10000	8	8	8	9	8	8	8	8	8	8	8,1	0,3
	6	6	6	6	6	6	6	7	6	6	6,1	0,3
	5	6	5	5	6	6	5	6	6	5	5,5	0,5
											Nelze vygenerovat	
50000	42	42	42	47	45	43	42	42	43	42	43,0	1,7
	29	29	28	29	29	29	28	29	29	29	28,8	0,4
	29	29	29	30	29	30	29	29	29	30	29,3	0,5
100000	85	85	85	87	85	85	86	89	87	84	85,8	1,5
	62	62	63	63	62	63	62	62	63	62	62,4	0,5
	62	61	62	62	62	61	61	61	61	62	61,5	0,5
250000	208	210	207	208	208	208	207	207	208	209	208,0	0,9
	158	159	156	160	157	157	159	156	162	158	158,2	1,9
	161	161	159	161	159	161	167	158	163	160	161,0	2,5
500000	408	407	407	410	409	407	408	408	404	411	407,9	1,9
	330	330	335	333	334	336	336	334	334	336	333,8	2,3
	335	335	332	337	350	345	335	336	337	347	338,9	6,1
750000	604	606	593	616	601	608	623	604	601	667	612,3	20,9
	517	510	526	517	515	533	514	515	518	515	518,0	6,6
	518	528	516	528	512	512	514	516	513	526	518,3	6,5
1000000	802	799	791	805	802	795	794	793	799	811	799,1	6,1
	688	697	697	688	688	692	699	696	694	695	693,4	4,2
	697	693	699	693	696	692	700	692	699	691	695,2	3,4

dle formátu:	Graham Scan t [ms]										Průměr [ms]	Rozptyl [ms]
1000	16	9	20	12	19	19	17	9	8	21	15,0	5,0
	13	13	21	14	14	13	14	16	18	13	14,9	2,7
	14	12	11	13	11	14	11	14	12	12	12,4	1,3
											Příliš dlouhý čas	
2000	71	69	64	66	64	68	76	56	73	63	67,0	5,7
	35	36	39	35	37	39	39	38	36	35	36,9	1,7
	26	27	30	27	28	26	26	29	29	30	27,8	1,6
5000	456	446	443	418	425	435	402	333	439	455	425,2	36,5
	93	99	95	100	95	93	94	102	95	94	96,0	3,2
	74	69	70	70	71	71	73	68	74	68	70,8	2,3
10000	1640	1665	1635	1662	1659	1655	1661	1669	1647	1649	1654,2	11,1
	194	203	193	191	201	195	200	194	190	201	196,2	4,6
	141	132	144	142	139	142	135	136	134	140	138,5	4,0
50000	31190	31316	31266	31278	31309	31208	31258	31194	31245	31291	31255,5	45,8
	960	1010	962	991	976	984	989	970	1004	1003	984,9	17,7
	673	670	692	711	682	697	705	672	686	712	690,0	16,0
100000	101845	101730	101691	101783	101801	101920	101776	101721	101753	101795	101781,5	65,9
	2074	1937	1973	2037	2021	1956	1984	2077	1995	1969	2002,3	48,3
	1364	1342	1349	1380	1366	1371	1360	1349	1355	1382	1361,8	13,4
250000											Příliš dlouhý čas	
	5117	4935	4846	4995	5025	4921	4984	5003	5087	4871	4978,4	87,1
	3516	3441	3354	3398	3410	3495	3547	3456	3473	3485	3457,5	58,3
500000												
	10053	9716	10264	9900	9983	9997	10105	9951	10047	9894	9991,0	145,1
	6932	7039	6955	7135	6982	7063	7049	7084	6991	7032	7026,2	61,8
750000												
	14914	15020	15530	15462	15332	15293	15051	15368	14975	15472	15241,7	229,8
	10340	10124	10271	10212	10356	10176	10077	10234	10263	10158	10221,1	90,1
1000000												
	20099	19995	19966	19987	20090	20103	20015	20039	20021	19981	20029,6	51,3
	13508	13734	13530	13657	13606	13525	13697	13644	13619	13740	13626,0	84,8

10.2 Zhodnocení měřených dat

Nejlépe z testovaných algoritmů dopadl algoritmus Sweep Line pro všechny generované množiny bodů.

U algoritmu Jarvis Scan je problém rychle narůstající doba běhu při velkém množství bodů.Obdobný problém má algoritmus Quick Hull, u kterého je ovšem nárůst doby běhu značně nižší než u Jarvis Scan.

Algoritmus Graham Scan má ještě větší nárůst doby běhu než Jarvis Scan u množiny náhodných bodů. Velmi efektivní je ovšem u kruhové nebo eliptické množiny.

10.3 Další poznámky

Zdrojový kód algoritmu Graham Scan byl testován ve více verzích Qt Creatoru a není se staršími verzemi kompatibilní. Byl shledán kompatibilním s verzí 4.10.0 založené na 5.13.1.

Porovnání Graham Scanu a ostatních algoritmů nemusí být přesné. Graham Scan pravděpodobně nebyl napsán efektivním způsobem (řazení dle úhlů) a jeho doba běhu byla měřena na jiném PC než ostatní algoritmy (z důvodu nekompatibility se staršími verzemi).

Do Aplikace byly přidány dodatečné funkce na mazání bodů, konvexní obálky a minimálního ohraničujícího obdélníku. Je zde také widget pro chybové hlášení, který je využit v případě nedostatečného počtu bodů pro tvorbu obálky nebo generování počtu bodů menšího než 1.

10.4 Náměty na zlepšení

Zdrojový kód Graham Scanu, především třídění dle úhlů, by se dal napsat efektivněji.
Zdrojový kód Quick Hull by mohl být upraven, aby do konvexní obálky řadil i kolineární body.

11. Přílohy

1) Zdrojový kód aplikace (algorithms.h / .cpp; draw.h / .cpp; generatepoints.h / .cpp; CH.pro; main.cpp; sortbyx.h/ .cpp; sortbyxf.h/ .cpp; sortbyy.h/ .cpp; sortbyyf.h/ .cpp; widgets.h / .cpp / .ui)

5. 11. 2019
David Němec, Jan Šartner