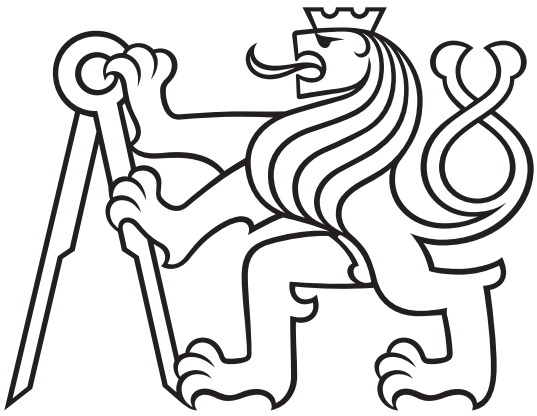
České vysoké učení technické v Praze
Fakulta stavební
program geodézie a kartografie
obor geomatika



konvexní obálky

Autoři: Bc. Linda Kladivová, Bc. Jana Špererová

Předmět: Algoritmy digitální kartografie a GIS

Úloha č. 2: Konvexní obálky a jejich konstrukce

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

Výstup: $\mathcal{H}(P)$.

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

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

Vstupní množiny bodů včetně vygenerovaných konvexních obálek vhodně vizualizujte. Pro množiny $n \in < 1000, 1000000 >$ vytvořte grafy ilustrující doby běhu algoritmů pro zvolená n. Měření proveďte pro různé typy vstupních množin (náhodná množina, rastr, body na kružnici) opakovaně (10x) a různá n (nejméně 10 množin) s uvedením rozptylu. Naměřené údaje uspořádejte do přehledných tabulek.

Zamyslete se nad problematikou možných singularit pro různé typy vstupních množin a možnými optimalizacemi. Zhodnoťte dosažené výsledky. Rozhodněte, která z těchto metod je s ohledem na časovou složitost a typ vstupní množiny P nejvhodnější.

Hodnocení:

Krok	**Hodnocení**
Konstrukce konvexních obálek metodami Jarvis Scan, Quick Hull, Seep Line.	15b
Konstrukce konvexní obálky metodou Graham Scan	*+5b*
Konstrukce striktně konvexních obálek pro všechny uvedené algoritmy.	*+5b*
Ošetření singulárního případu u Jarvis Scan: existence kolineárních bodů v datasetu.	*+2b*
Konstrukce Minimum Area Enclosing box některou z metod (hlavní směry budov).	*+5b*
Algoritmus pro automatické generování konvexních/nekonvexních množin bodů různých tvarů (kruh, elipsa, čtverec, star-shaped, popř. další).	*+4b*
Max celkem:	**36b**

Čas zpracování: 3 týdny.

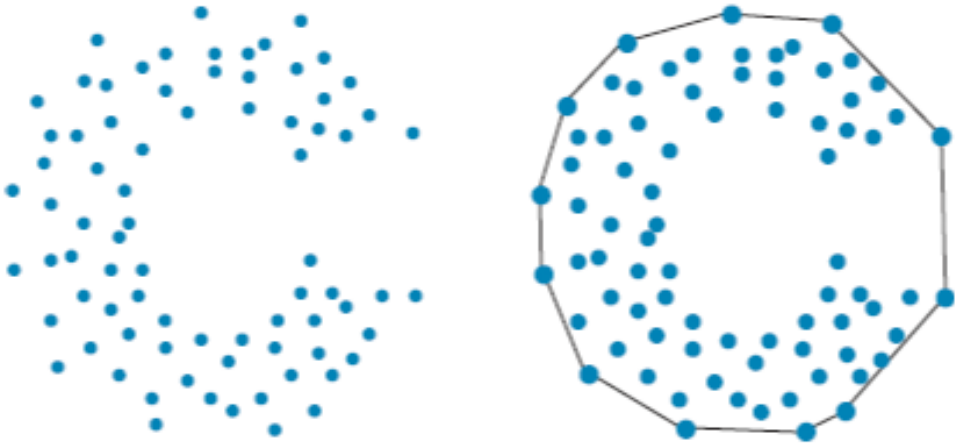
Obsah

Seznam obrázků

1 Popis a rozbor problému

Záměrem této úlohy je vytváření konvexních obálek pro libovolné množství vygenerovaných bodů. Konvexní obálka je uzavřená hranice spojující konkrétní body množiny tak, že každý bod množiny se nachází uvnitř nebo na hranici obálky (bod obálky). A také pokud spojíme dva libovolné body množiny úsečkou, tak daná úsečka leží v konvexní obálce nebo je s ní totožná.

Konvexní obálky se využívají například v kartografii (natočení budov), v geodézii (3D modelování), u počítačových her (prostředí), v designu či grafice a ve spoustě dalších oborů.



Obrázek 1: Ukázka konvexní obálky nad vstupní množinou bodů [2]

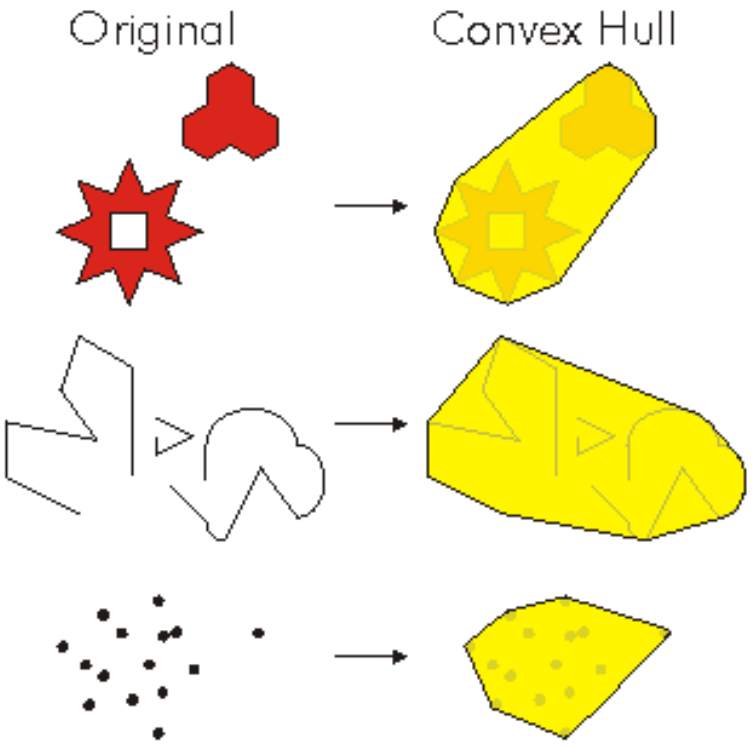
1.1 Údaje o bonusových úlohách

Byla snaha o to splnit všechny bonusové úlohy. Kromě konstrukce konvexních obálek metodami Jarvis Scan, Quick Hull a Sweep Line, byla naprogramována i konstrukce metodou Graham Scan. Vzhledem k tomu, že bylo zadáním změřit doby trvání algoritmů pro různé typy vstupních množin, byl vytvořen Combo Box s výběrem různých typů množin (kruh, elipsa, grid, čtverec, náhodná množina, star-shaped polygon), jejichž konstrukce byla zvlášť definována v samostatné třídě Generator,

která združuje několik funkcí vytvářejících výsledný obrazec na základě předem vybraného počtu bodů a typu množiny.

Kromě konstrukce konvexní obálky byl napsán algoritmus pro konstrukci Minimum Area Enclosing Box, který nezobrazuje pouze výsledný box, ale také hlavní směr budovy.

V rámci algoritmu Jarvis Scan byla ošetřena existence kolineárních bodů v datasetu. Aby byla výsledná obálka opravdu správně zkonstruovaná, bylo nutné vynechat body, které leží na přímce mezi 2 body konvexní obálky a dále smazat body, které jsou identické s body obálky. K tomuto účelu byla napsána funkce CorrectCH, která je implementována na závěr každého ze zmíněných algoritmů na generování konvexních obálek.



Obrázek 2: Ukázka konvexní obálky nad různými vstupními množinam [3]

2 Popis použitých algoritmů

Cílem této úlohy je vytvořit aplikaci, která po výběru různého typu vstupní množiny (náhodná množina, rastr, kružnice, star-shaped polygon, elipsa) vygeneruje automaticky daný obrazec s předem definovaným počtem bodů. Nad tímto obrazcem poté pomocí různých metod zkontruuje konvexní obálku nebo Minimum Area Enclosing Box. V praxi se ke konstrukci konvexních obálek využívá několik různých metod. V této práci byly naprogramovány metody Jarvis Scan, Quick Hull, Sweep Line (zametací přímka) a metoda Graham Scan. Dalšími používanými metodami jsou Divide and Conquer či inkrementální kontrukce. Na dalších řádcích jsou podrobně popsány využité algoritmy pro generování konvexních obálek a také algoritmus hlavních směrů budov, který je využíván při konstrukci Minimum Area Enclosing Box.

2.1 Jarvis Scan

Jarvis Scan je metoda, která vyhledává body konvexní obálky na základě hledání maximálního úhlu, tedy předpokládá, že tři body neleží na jedné přímce. Tento algoritmus lze snadno i využít pro prostorové body. Algoritmus pro sestavení metody Jarvis Scan je snadný, avšak časová náročnost tohoto algoritmu je jeho nevýhodou. Jeho rychlost je O(n2) pro body ležící na kružnici, nejběžněji pak O(n*h), kde n je počet bodů vstupní množiny a h je počet bodů konvexní obálky.

První bod, ze kterého celý algoritmus vychází, je bod s nejmenší y souřadnicí, někdy také označován jako pivot. Víme, že tento bod bude ležet na konvexní obálce. Tímto bodem povedeme rovnoběžku s osou x a následně projdeme všechny body množiny [i] a změříme úhel mezi rovnoběžkou s osou x a přímkou určenou bodem p (pivot) a i. Tyto úhly mezi sebou porovnáváme a největší z těchto úhlů nám označí následující bod konvexní obálky. V dalším kroku je ošetřena singulární situace, kdy bod s dosavadním maximálním úhlem a současný bod i, leží na jedné přímce, tedy úhly jsou téměř totožné. Pokud se tak stane, je jako dosavadní bod s největším úhlem přídán ten bod, který je vzdálenější od bodu p_j.

Nyní se nám bod i stává novým pivotem a bod p se stává bodem určující přímku, od které se měří úhly. Takto pokračujeme stále znovu, dokud se bodem i nestane opět bod p.

2.1.1 Slovní zápis algoritmu s ošetřením kolineárních bodů

1. Setřídění vstupní množiny bodů podle souřadnice y.
2. Nalezení pivota q: $q = min(y_i)$. Pokud nalezeny dva body, vezmi ten s maximální souřadnicí x.
3. Přidej bod q do konvexní obálky: $q \rightarrow CH$
4. Nalezení bodu r ve směru osy x (levé rameno úhlu): $r = (q.x() - 1, q.y())$.
5. Inicializuj: $p_j = q; p_{j-1} = r$
6. Opakuj, dokud: $p_j \neq q$

   Opakuj pro všechny body vstupní množiny:

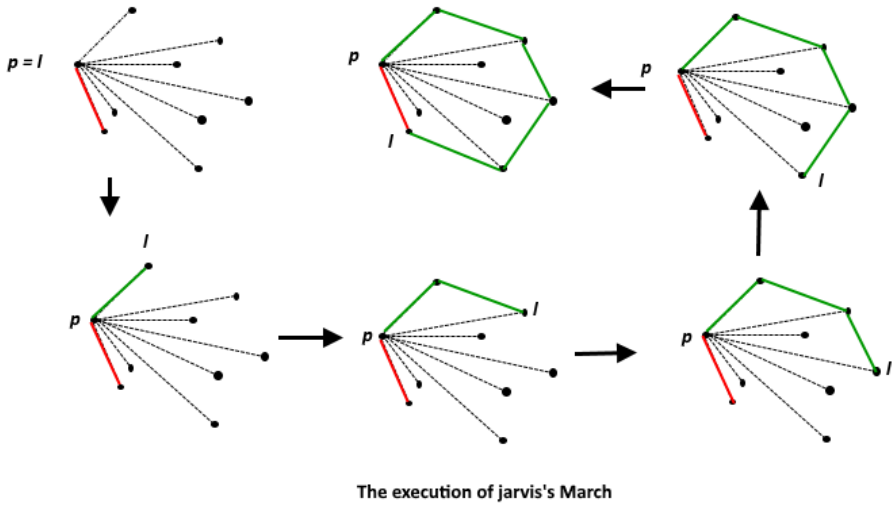
   Nalezni p_i: $p_i = argmax_{\forall p_i \in P} \angle(p_{j-1}, p_j, p_i)$

   Vypočti $dist_i, dist_{imax}$ pokud $|angle_{max} - angle_{current}| < eps$

   Pokud je $dist_i > dist_{imax}$, prohlaš bod i, že má maximální úhel

   Přidej vybraný bod p_i: $p_i \rightarrow CH$

   Přeindexuj body: $p_{j-1} = p_j; p_j = p_i$



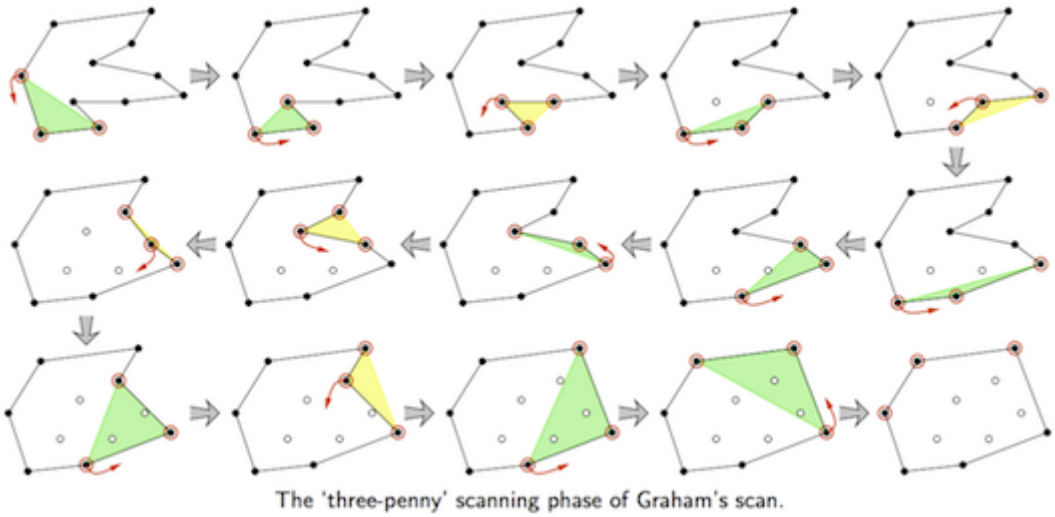
Obrázek 3: Princip Jarvis Scan algoritmu [1]

2.2 Graham Scan

Algorithmus Graham Scan funguje na principu zjišťování CCW (levotočivé) orientace trojúhelníku. Algoritmus Graham Scan je velmi rychlý, podobně, jak algoritmus metody Sweep Line, či Quick Hull metody. Jeho rychlost je O(n*log(n)), kde n je počet bodů vstupní množiny.

Na počátku algoritmu setřídíme body podle Y souřadnice a bod s nejmenší Y souřadnicí označíme jako pivot (p). Poté se vypočítá směrnice vzhledem k ose x ke všem bodům množiny. Tyto body pak následně seřadíme podle velikosti směrnice. Následně vždy testujeme CCW orientaci na posledních 2 bodech přidaných do množiny konvexní obálky a poslední bod s největší směrnicí. Takto pak postupujeme analogicky, dokud nenalezneme konvexní obálku.

Tato metoda je pojmenována nikoli po tom, jak funguje, ale po Ronaldu Grahamovi, který tuto metodu publikoval roku 1972.



Obrázek 4: Princip Graham Scan algoritmu [1]

2.2.1 Slovní zápis algoritmu

1. Setřídění vstupní množiny bodů podle souřadnice y.

2. Nalezení pivota q: $q = min_{\forall p_i \in S}(y_i), q \in CH$ (pokud jsou souřadnice y u bodů stejné, vybere toho s nejvyšším indexem).

3. Nalezení bodu s ve směru osy x (levé rameno úhlu): $s = (q.x() - 1, q.y())$.
4. Opakuj pro všechny body vstupní množiny:

   Spočti $omega_i$: $omega_i = \angle(s, q, p_i)$

   Spočti $dist_i$: $dist_i = |qp_i|$
5. Setřídění bodů dle úhlu s osou x: $\forall p_i \in S$ sort by $\omega_i = \angle(s, q, p_i)$
6. Při nalezení stejného úhlu: $\omega_k = \omega_l \rightarrow$ neuvažovat bližší bod
7. Vložení prvního bodu s nejmenším úhlem do množiny: $CH \leftarrow p_1$
8. Opakuj pro všechny setříděné body: $j < n$

   If p_j vpravo od předešlých bodů $\rightarrow$ odeber poslední prvek CH

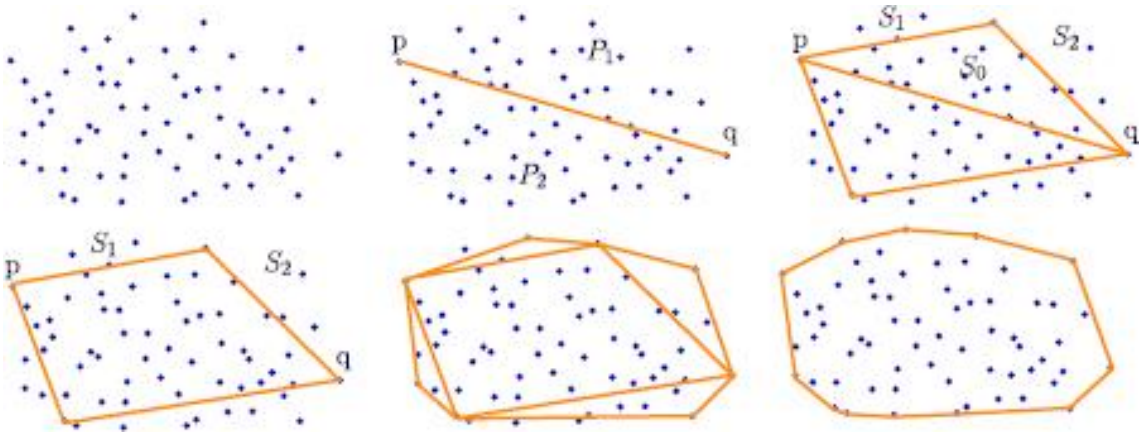
   V opačném případě přidej bod do CH: $p_j \rightarrow H$

2.3 Quick Hull

Quick Hull je metoda, která vyhledává body konvexní obálky, na základě vyhledávání nejvzdálenějšího bodu od přímky, která je určena 2 body množiny bodů. Rychlost algoritmu při použití metody Quick Hull je rychlá O(n*log(n)), ale někdy může dosáhnout i stejné rychlosti, jako při použití algoritmu Jarvis Scan, avšak to nastává jen v nejhorším případě a to O(n2), kde n je počet bodů vstupní množiny.

Algoritmus vychází z přímky, která je určena dvěma body množiny bodů. Tyto body, jsou body s nejmenší a největší x souřadnicí. Tato přímka nám množinu bodů rozdělí na 2 poloviny a my v každé polovině procházíme všechny body a zjišťujeme jejich vzdálenost od přímky Xmin Xmax. Když najdeme nejvzdálenější bod od přímky, přidáme ho do konvexní obálky a spojíme s ho s body Xmin Xmax. Tím nám vzniknou další dvě přímky a od nich pokračujeme stále analogicky stejně. Tento postup děláme pro horní množinu bodů i dolní množinu bodů, dokud nevytvoříme konvexní obálku.

Tato metoda je také jinak známá jako metoda Divide and Conquer, což v překladu znamená, rozděl a panuj. Naše přímka rozdělí množinu bodů na dvě části a v nich pak hledáme „panující“ bod, ten nejvíce vzdálený.



Obrázek 5: Princip dělení u Quick Hull algoritmu [5]

2.3.1 Slovní zápis algoritmu

1. Vytvoření horní a dolní množiny: $upoints = 0; lpoints = 0$
2. Nalezení extrémních hodnot: $q_1 = min_{\forall p_i \in S}(x_i); q_3 = max_{\forall p_i \in S}(x_i)$
3. Přidání extrémních bodů do horní a dolní množiny: $upoints \leftarrow q_1; upoints \leftarrow q_3; lpoints \leftarrow q_1; lpoints \leftarrow q_3$
4. Pro všechny body množiny: $\forall p_i \in S$:

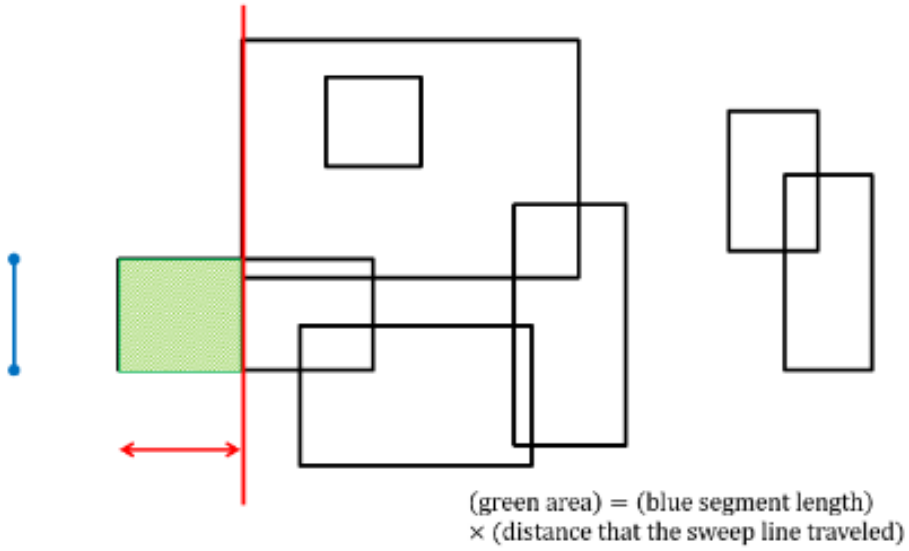
   Patří bod do horní množiny? $if(p_i \in \sigma_l(q_1, q_3))upoints \leftarrow p_i$

   V opačném případě: $lpoints \leftarrow p_i$
5. Přidání krajního bodu do konvexní obálky: $CH \leftarrow q_3$
6. Nalezení nejvzdálenějšího bodu c v horní části od přímky, přidání do množiny konvexní obálky a opakování vůči nově vzniklé přímce (rekurze)
7. Přidání krajního bodu do konvexní obálky: $CH \leftarrow q_1$
8. Opakování hledání nejvzdálenějšího bodu v dolní části (rekurze)

2.4 Metoda zametací přímky (Sweep Line)

Sweep Line je metoda, která funguje na principu inkrementální konstrukce, tedy přírůstkové, jelikož mám množinu rozdělenou na část již zpracovanou a tu doplňuji o nové body z nezpracované části, dokud nezpracuji celou množinu bodů. Algoritmus Sweep Line je velmi rychlý, podobně, jak algoritmus metody Quick Hull. Jeho rychlost je O(n*log(n)), kde n je počet bodů vstupní množiny.

Algoritmus pracuje s množinami předchůdců a následníků. Na počátku máme seřazené body podle X souřadnic a první dva mezi sebou propojíme, pak se přesuneme na další bod a ten propojujeme s naší úsečkou a určíme jeho předchůdce a jeho následníky. Takto pokračujeme stále dál, dokud nevytvoříme kompletní konvexní obálku.Tato metoda je také známá pod názvem zametací přímka.



Obrázek 6: Princip zametací přímky [6]

2.4.1 Slovní zápis algoritmu

1. Setřídění bodů množiny podle osy x
2. Odstranění dupliciticních vstupních bodů

3. Vytvoření vektoru předchůdců a vektoru následovníků

4. Vytvoření počáteční aproximace pomocí dvojúhelníku: $n[0] = 1; n[1] = 0; p[1] = 0; p[0] = 1$

5. Vyhodnocování následujících bodů: $for\ p_i \in P_S, i{>}3$

   Porovnání hodnoty souřadnice y: $if(y_i{>}y_{i-1})$

   Změna indexů při splnění podmínky: $p[i] = i - 1; n[i] = n[i - 1]$

   V opačném případě: $p[i] = p[i - 1]; n[i] = i - 1$

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

   Opravení horní tečny $while(n[n[i]]) \in \sigma_R(i, n[i])$:

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

   Opravení dolnií tečny $while(p[p[i]]) \in \sigma_L(i, p[i])$:

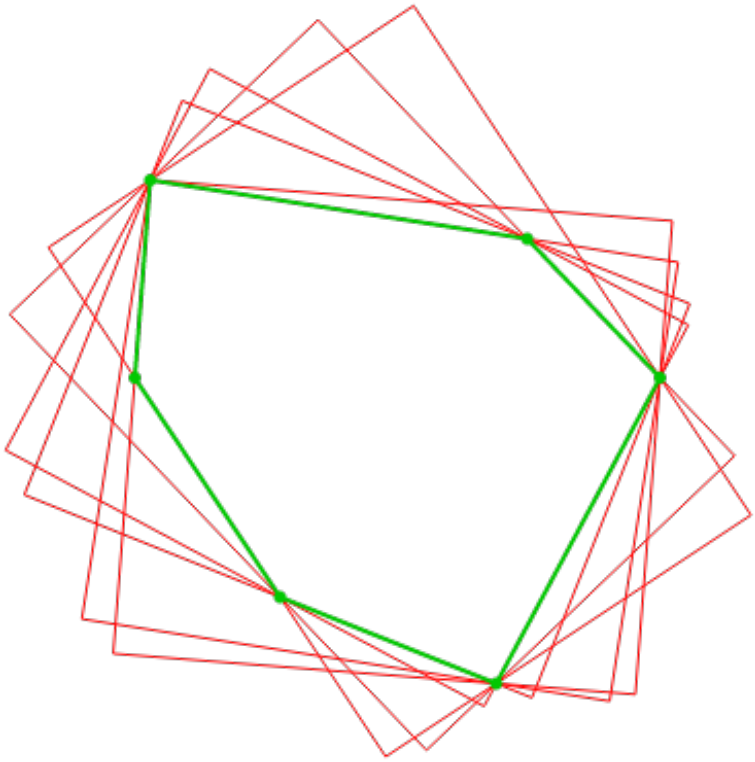
   Změna indexů: $n[p[p[i]]] = i; p[i] = p[p[i]]$

6. Podle vektoru následníků sestav polygon CH

2.5 Metoda hlavních směrů budov

Tato metoda vytváří obdélník okolo množiny bodů, který má nejmenší obsah. Princip této metody spočívá v tom, že porovnáváme obdélníky takové, které byly vytvořeny tak, aby minimálně jedna strana konvexní obálky byla totožná se stranou obdélníku. Časová náročnost této metody je stejná, jako u vytváření konvexní obálky metodou Jarvis Scan. Její rychlost je O(n), kde n je počet bodů vstupní množiny.

Na počátku máme konvexní obálku okolo množiny bodů. My nyní vybereme jednu stranu konvexní obálky a tou proložíme první stranu obdélníka. Následně najdeme levé maximum, pravé maximum a horní maximum a vytvoříme obdélník, kde jedna strana je totožná s hranou konvexní obálky a zbylé hrany jsou totožné s minimálně 1 bodem konvexní obálky. Vypočítáme obsah tohoto obdélníka. Takto pokračujeme analogicky, dokud nebudeme mít vypočítané obsahy všech obdélníků nad všemy hranami konvexní obálky. Obsahy pak porovnáme a obdélník o nejmenším obsahu je náš hledaný minimální obdélník.



Obrázek 7: Obdélníky při konstrukci Minimum Area Enclosing Box [4]

2.5.1 Slovní zápis algoritmu

1. Inicializuj minimální plochu na velkou hodnotu: $A_{min} = limit_{double}$
2. Inicializace místního souřadnicového systému a inicializace min a max souřadnic pootočené CH a min max souřadnic Enclosing Boxu
3. Pro všechny vstupní body CH:

   Převeď do místního systému: $points_i = p; points_{i+1} = k$

   Určií minimální a maximální souřadnice otočené konvexní obálky

   Nalezni obdélník, který se dotýká konvexní obálky v extrémních bodech

   Porovnej s minimální plochou If $A < A_{min}$:

   $A_{min} = A; Save(minX, maxX, minY, maxY); SaveAngle$
4. Vytvoř obdélník z (min X, max X, min Y, max Y).
5. Nalezni hlavního směru obdélníku v místní soustavě.
6. Transformuj obdélník i hlavní směr budovy do globálního systému o úhel Angle.

2.6 Generátor konvexních/nekonvexních množin bodů různých tvarů

Generátory jsou naimpementovány jako jednotlivé metody ve třídě Generator. Smyslem aplikace je, aby si mohl uživatel vybrat příslušný tvar - kruh, čtverec, elipsu, grid, náhodné rozmístění či star shaped obrazec, a teprve nad tímto obrazcem mohl konstruovat konvexní obálky, případně Minimum Area Enclosing Box. Náhodné generování bylo implementováno skrze funkce *rand* a *modulo*, aby bylo dosaženo správné zobrazení vygenerovaných bodů uvnitř zobrazovacího okna.

Pro tvorbu kruhu byl nejprve náhodně vygenerován střed kružnice a její poloměr. V závislosti na počtu zadaných bodů byl určen středový úhel dvou po sobě jdoucích bodů. Pokud je vstupní počet bodů menší než tři, je zobrazena chybová hláška (ze dvou dobů kružnici nevytvořím). Body na kružnici byly poté vygenerovány ze známých vzorců:

$$X = X_0 + r.cos(\phi)$$

$$Y = Y_0 + r.sin(\phi)$$

Elipsa byla generována obdobně jako kruh s tím rozdílem, že nebyl náhodně generován poloměr, ale hodnoty hlavní poloosy (a) a vedlejší poloosy (b). Body na elipse byly poté vygenerovány ze známých vzorců:

$$X = X_0 + a.cos(\phi)$$

$$Y = Y_0 + b.sin(\phi)$$

Při tvorbě čtverce je nejprve kontrolováno, zda je počet zadaných bodů opravdu dělitelný čtyřmi. Pokud je menší než čtyři, je zobrazena chybová hláška. Pokud není zadaný počet dělitelný čtyřmi je zobrazena upozorňující hláška a počet generovaných bodů mezi vrcholy čtverce je upraven tak, aby v rámci každé strany čtverce byl generován stejný počet bodů. Prvním vygenerováným náhodně umístěným bodem je levý horní roh. Poté je náhodně určena délka hrany, díky které může jsou dopočteny souřadnice zbývajících vrcholů a také délka krátké úsečky mezi body na hraně.

Při tvorbě Star Shaped množiny je automaticky vygenerován střed obou kružnic a poloosy a, b. Pro polygony typu star shaped je typické, že z jeho středu lze vidět do všech vrcholů. Polovina bodů je tedy vygenerována na kružnici s poloměrem a,

druhá polovina na kružnici s poloměrem b, tak aby dva po sobě jdoucí body byly vytvořeny s různou vzdáleností od středu. Pokud je vstupní počet bodů menší než šest, je zobrazena chybová hláška. (Pro tvorbu dvou kružnic se stejným středem a různým poloměrem potřebuji nejméně šest bodů). Body byly vygenerovány ze vzorců:

$$X_i = X_0 + a * cos(\phi)$$

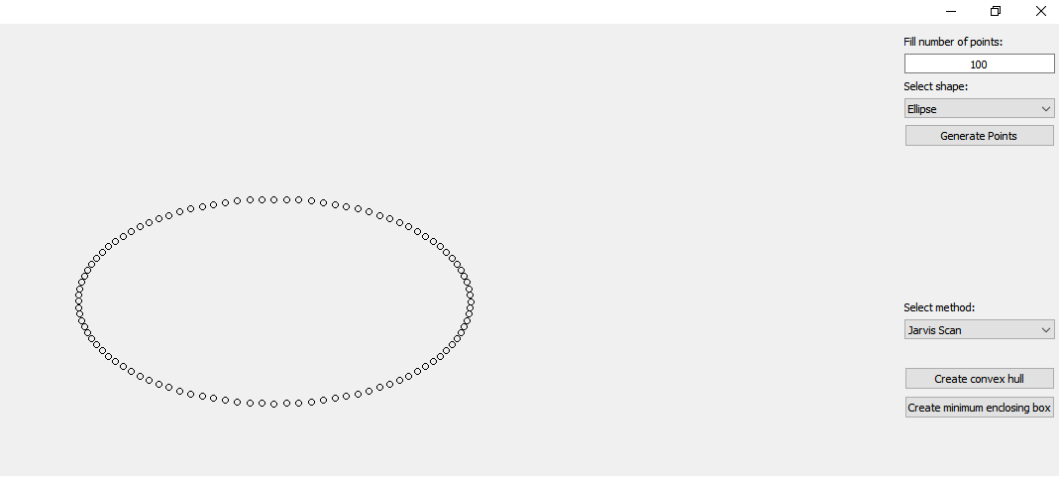
$$Y_i = Y_0 + a * sin(\phi)$$

$$X_{i+1} = X_0 + b * cos(\phi)$$

$$Y_{i+1} = Y_0 + b * sin(\phi)$$

Nejjednodušší implementací bylo vytvoření náhodně rozmístěných bodů v rámci okna. Body byly náhodně vygenerovány pomocí funkcí *rand* a *modulo* .

Algoritmus pro tvorbu pravidelné mřížky je v mnoha ohledech podobný algoritmu pro generování množiny čtvercového tvaru. Vzhledem k tomu, že generujeme pravidelnou mřížku, je zapotřebí, aby vstupní počet bodů byl roven nejméně čtyřem a také aby byl druhou mocninou nějakého přirozeného čísla. Pokud druhá podmínka není splněna, je počet bodů redukován pomocí přetypování na integer a uživatel je na změnu upozorněn. Takovýto případ může nastat například při zadání 17 bodů. Odmocnina (v tomto případě rovna 4.12) je přetypována na integer, tedy na číslo 4. Vytvořený grid má tedy velikost 4 x 4. Levý horní roh mřížky je nastaven napevno na souřadnice (10,10), ostatní body jsou dopočteny podle předem náhodně vygenerovaných délek mezer mezi body v ose x i v ose y.



Obrázek 8: Generování elipsy

3 Problematické situace a jejich rozbor

3.1 Singulární případy u Jarvis Scan algoritmu

U tohoto algoritmu dochází k určování maximálního úhlu ve vrcholu konvexní obálky. Podle tohoto kritéria je pak přidán další bod konvexní obálky. Je proto důležité, aby bod s tímto úhlem vždy jednoznačně nalezen. U pravidelných tvarů (např. mřížka) je ale mnoho bodů, ke kterým z daného bodu určíme stejný úhel, jelikož leží na stejné přímce.

V rámci bonusové úlohy byla odstraněna tato singularita způsobena kolineárními body. Pokud je vypočtený úhel totožný nebo téměř totožný s aktuálně největším nalezeným úhlem, je vypočtena jednak vzdálenost od bodu konvexní obálky k bodu s maximálním úhlem, a jednak vzdálenost od bodu konvexní obálky k bodu s počítaným úhlem. Pokud je vzdálenost k počítanému bodu větší (je tedy dále na přímce), je tento bod označen jako bod s maximálním úhlem. Kolineární bod je tedy přeskočen a není zahrnut do polygonu konvexní obálky. Proto například polygon konvexní obálky zkontruovaný nad mřížkou, která se skládá se 100 bodů, bude mít pouze 4 body (ty, které ohraničují danou mřížku). Tato problematika je ošetřena ve zdrojovém souboru *algorithms.cpp* na řádcích 118-126.

3.2 Konstrukce striktně konvexních obálek

U výsledných obálek můžou nastat následující singulární situace:

1. bod konvexní obálky leží na úsečce mezi dvěma body konvexní obálky – takový bod je nutné odstranit,

2. na sobě leží dva identické body konvexní obálky – jeden z nich je rovněž nutné odstranit.

Oba tyto případy jsou řešeny na závěr každé z výše zmíněných funkcí na konstrukci konvexní obálky ve funkci *correctCH*. K vymazání identických nebo téměř identických bodů byla definována třída *isPointIdentical*, která definuje operátor totožnosti *()*. Tento operátor pak vstupuje do funkce *std::unique*, která vybere iden-

tické body, jež jsou v dalším kroku pomocí funkce *erase* smazány z polygonu konvexní obálky.

Name	Value	Type
ch	<13 items>	QPolygon &
[0]	(185, 50)	QPoint
[1]	(675, 50)	QPoint
[2]	(699, 53)	QPoint
[3]	(699, 617)	QPoint
[4]	(698, 628)	QPoint
[5]	(691, 640)	QPoint
[6]	(593, 649)	QPoint
[7]	(125, 649)	QPoint
[8]	(104, 647)	QPoint
[9]	(100, 619)	QPoint
[10]	(100, 51)	QPoint
[11]	(185, 50)	QPoint
[12]	(185, 50)	QPoint
unique_end	(185, 50)	QPoint
xp	185	int
yp	50	int

Obrázek 9: Případ dvou totožných bodů č. 11, 12 u konvexní obálky o 13 vrcholech

Name	Value	Type
ch	<12 items>	QPolygon
[0]	(185, 50)	QPoint
[1]	(675, 50)	QPoint
[2]	(699, 53)	QPoint
[3]	(699, 617)	QPoint
[4]	(698, 628)	QPoint
[5]	(691, 640)	QPoint
[6]	(593, 649)	QPoint
[7]	(125, 649)	QPoint
[8]	(104, 647)	QPoint
[9]	(100, 619)	QPoint
[10]	(100, 51)	QPoint
[11]	(185, 50)	QPoint
index	0	unsigned int
pj	(185, 50)	QPoint
pjj	(100, 51)	QPoint
points	<10000 items>	std::vector<QPoint> &
q	(185, 50)	QPoint
r	(184, 50)	QPoint

Obrázek 10: Situace po odstranění jednoho z identických bodů pomocí funkce *std::unique*

Kolinearita bodů na stejné linii je řešena pomocí funkce *getPointLinePosition*. Pokud je výstupem této funkce *-1*, je daný bod na linii vymazán z polygonu konvexní obálky.

3.3 Duplicity u Sweep Line algoritmu

Problémy v této metodě mohou nastat v případě, že na vstupu budou duplicitní body. Tyto duplicity je tedy nutné ošetřit hned na začátku algoritmu:

1. Nalezení duplicitního bodu: $if((points[j].x == points[i].x)\&\&(points[j].y == points[i].y))$
2. Odstranění bodu z množiny: $Delete\ points[i]$

Tato problematika je ošetřena ve zdrojovém souboru *algorithms.cpp* na řádcích 227-236.

3.4 Singulární případy u Graham Scan algoritmu

Podobně jako u Jarvis Scan algoritmu může nastat případ, že úhly k některým bodům jsou téměř totožné. Po setřídění bodů podle velikosti úhlů je tedy nutné spočítat rozdíl mezi počítaným úhlem a předchozím úhlem. Pokud je menší než zadaná tolerance, jsou počítány obě vzdálenosti od pivota směrem k bodu, který určuje pravé rameno počítaného úhlu, a od pivota směrem k bodu, který určuje pravé rameno předcházejícího úhlu. Pokud je vzdálenost k počítanému bodu větší (je tedy dále na přímce), je vzdálenost k předchozímu bodu nastavena na vzdálenost k počítanému bodu. Předcházející bod (ten s kratší vzdáleností) do polygonu konvexní obálky přidán není a do dalšího výpočtu není uvažován. Tato problematika je ošetřena ve zdrojovém souboru *algorithms.cpp* na řádcích 473-490.

4 Vstupní data

Vstupní data tvoří analyzovaná skupina bodů datového typu std::vector¡QPointF¿, která je vygenerována po stisku tlačítka *Generate Points*. Tento generátor vytváří pomocí funkce *rand* množiny s předem definovaným počtem vrcholů a s předem definovaným typem (náhodná množina, rastr, kružnice, star-shaped polygon, elipsa).

5 Výstupní data

Hlavním výstupem této úlohy je grafická aplikace, která umožňuje nad vygenerovaným polygonem zkonstruovat konvexní obálku. Ke konstrukci je možné přistupovat pomocí čtyř různých metod – Jarvis Scan, Quick Hull, Sweep Line či Graham Scan. Po stisku tlačítka *Create convex hull* je výsledná obálka typu QPolygon zobrazena fialovou barvou a nad zmíněným tlačítkem je zobrazena doba běhu algoritmu.

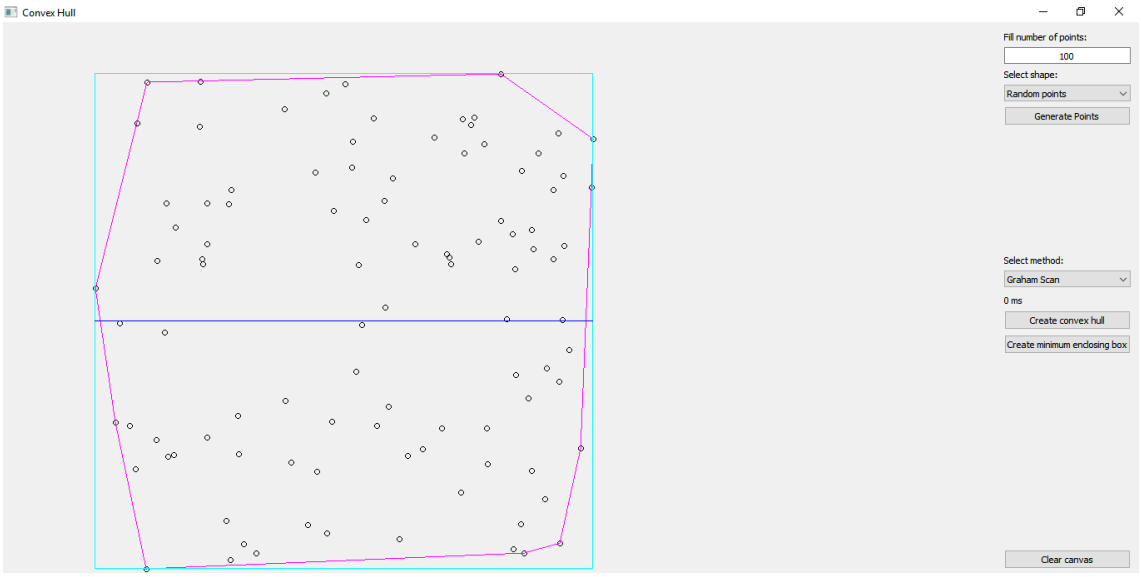
Ve chvíli, kdy je vytvořena obálka, může být po stisku tlačítka *Create minimum enslosing box* vytvořen nejmenší možný ohraničující obdélník typu QPolygon. Ten je vykreslen tyrkysovou barvou. Hlavní směrová úsečka budovy typu QLineF je zobrazena tmavě modře.

6 Ukázka aplikace

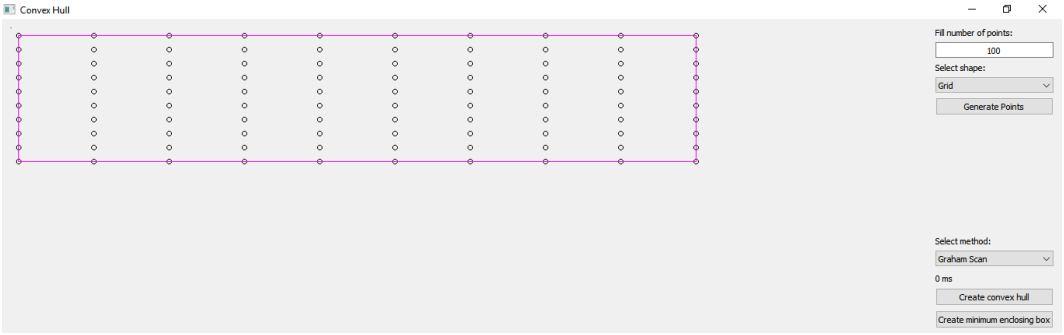
Do této kapitolky je zahrnuto několik ukázek vytvořené aplikace. Zároveň s konstrukcí konvexních obálek a Minumum Enclosing Bounding Box jsou ukázany možnosti generátoru konvexních a nekonvexních množin.

Obrázek 11: Konstrukce konvexní obálky metodou Jarvis Scan na elipse o 100 bodech

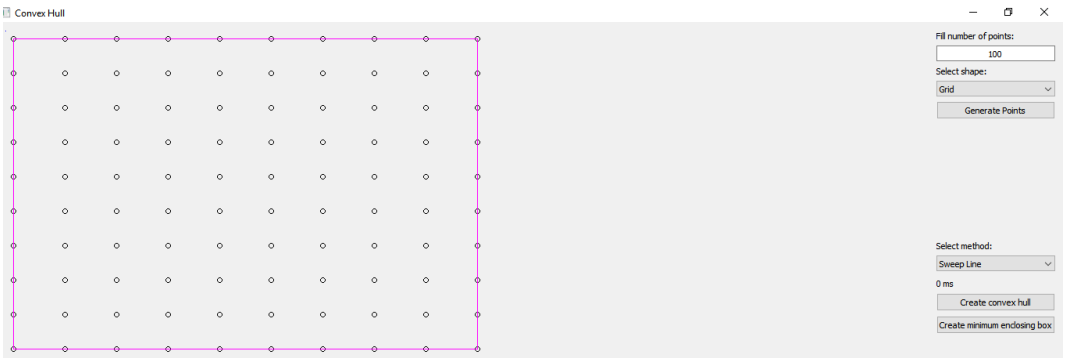
Obrázek 12: Konstrukce konvexní obálky metodou Jarvis Scan na obdélníku o 100 bodech



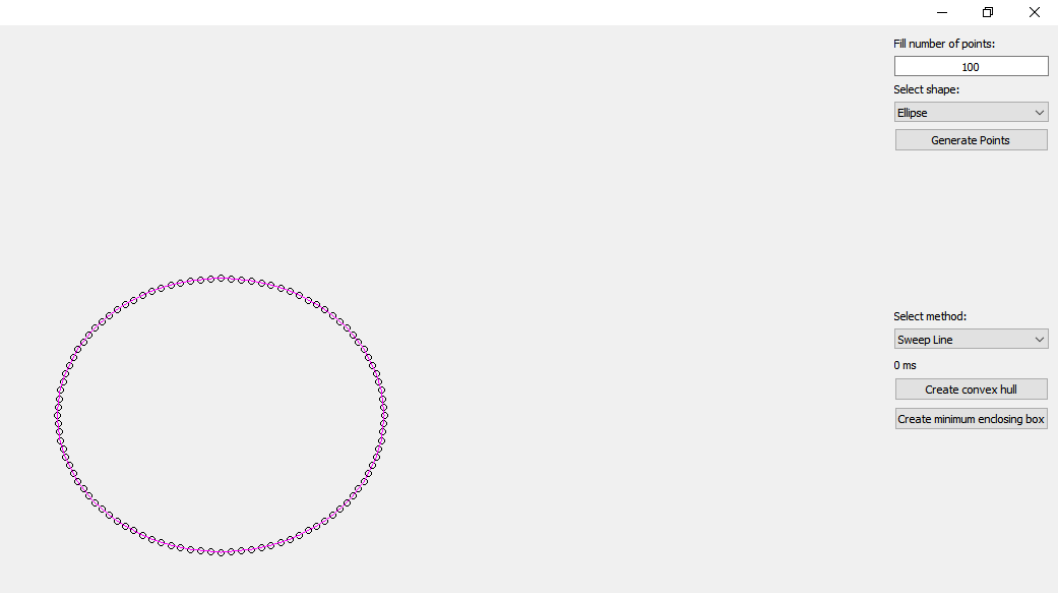
Obrázek 13: Konstrukce konvexní obálky metodou Graham Scan a Minimum Enclosing Box na náhodné množině o 100 bodech



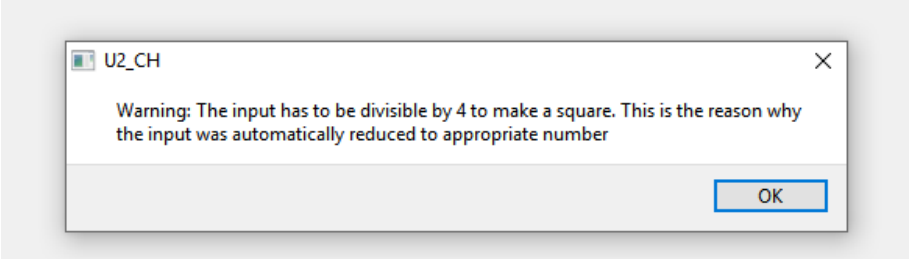
Obrázek 14: Konstrukce konvexní obálky metodou Graham Scan na pravidelné mřížce



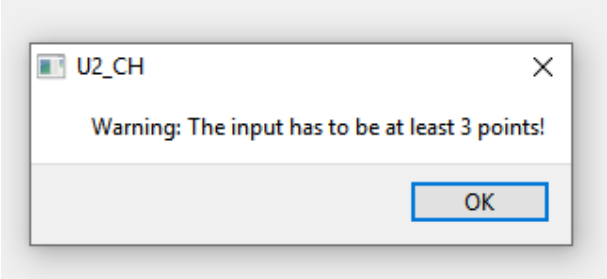
Obrázek 15: Konstrukce konvexní obálky metodou Sweep Line na pravidelné mřížce



Obrázek 16: Konstrukce konvexní obálky metodou Quick Hull na elipse



Obrázek 17: Varování u generování čtvercové množiny



Obrázek 18: Varování u generování kruhu

7 Technická dokumentace

7.1 Struktury

Ve zdrojovém souboru *algorithms.cpp* je definována struktura, která přehledněji manipuluje s bodem, úhlem a vzdáleností.

7.1.1 Angle

Jedná se o strukturu, která definuje datový typ Angle, který je využíván v metodě Graham Scan. Skládá se z bodu q typu QPoint, z úhlu od rovnoběžky s osou x k tomuto bodu (double) a ze vzdálenosti mezi počátečním bodem a bodem q.

7.2 Třídy

V aplikaci se nachází celkem šest tříd - Algorithms, Draw, Generator, SortbyX, SortbyY a Widget.

7.2.1 Algorithms - pomocné metody

Třída Algorithms je nejobsáhlejší. Obsahuje konstruktor a dalších několik pomocných i hlavních metod, které jsou určeny pro výpočet algoritmů používaných v digitálním GIS.

int getPointLinePosition(QPoint &q, QPoint &p1, QPoinF &p2)
Tato funkce má za úkol určit polohu bodu *q* vůči přímce zadané dvěma body *p1* a *p2*. Z vektorů je vypočten determinant. Pokud je determinant větší než tolerance (1.0e-6), bod se nechází v levé polorovině a funkce vrací hodnotu *1*. Pokud je menší než -tolerance, bod se nachází v pravé polorovině a funkce vrací hodnotu *0*. Pokud nenastane ani jeden z výše uvedených případů, tedy bod leží na linii, výstupem je hodnota *-1*.

double length2Points(QPoint p, QPoint q)
Vrací vzdálenost dvou bodů vypočtenou z Pythagorovy věty.

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

V této funkci je pomocí norem a skalárního součinu počítán úhel mezi dvěma hranami zadanými čtyřmi body typu QPoint. Úhel je vypočten jako arcus cosinus poměru skalárního součinu a součinu obou velikostí. Defaultně se v prostředí počítá v radiánech, což bylo ponecháno.

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

Vrací vzdálenost bodu a přímky zadané dvěma body vypočtenou ze známých geometrických vzorců.

void rotateByAngle(QPolygon &points, double angle)

Vrací vstupní polygon pootočený o daný úhel. Tato metoda je využívána při konstrukci Minimum Area Enclosing Box.

void rotateByAngle(QLine &points, double angle)

Vrací vstupní linii pootočenou o daný úhel. Tato metoda je využívána při konstrukci Minimum Area Enclosing Box.

QPolygon correctCH(QPolygon)

Kontroluje, zda ve výsledné konvexní obálce nejsou téměř totožné body nebo kolineární body. Pokud ano, ošetří a vrátí opravenou konvexní obálku.

7.2.2 Algorithms - hlavní metody

QPolygon jarvisScan(std::vector<QPoint> &points)

Metoda pro výpočet konvexní obálky nad vstupní množinou bodů typu *std::vector< QPoint >* algoritmem Jarvis Scan. Metoda vrací konvexní obálku s typem QPolygon.

QPolygon qHull(std::vector<QPoint> &points)

Metoda pro výpočet konvexní obálky nad vstupní množinou bodů typu *std::vector< QPoint >* algoritmem Quick Hull. Metoda vrací konvexní obálku s typem QPolygon.

QPolygon qh(unsigned int s, unsigned int e, std::vector<QPoint> &points, QPolygon &ch)
Pomocná metoda pro rekurzi v metodě qHull. Na vstupu jsou indexy bodů s a e, které určují přímku, podle níž se určí, zda bod p patří do konvexní obálky H. Metoda nic nevrací, jen ukládá body, které patří do konvexní obálky.

QPolygon grahamScan((std::vector<QPoint> &points))
Metoda pro výpočet konvexní obálky nad vstupní množinou bodů typu *std::vector< QPoint >* algoritmem Graham Scan. Metoda vrácí konvexní obálku s typem QPolygon.

QPolygon sweepLine((std::vector<QPoint> &points))
Metoda pro výpočet konvexní obálky nad vektorem bodů metodou Sweep Line. Metoda vrácí konvexní obálku s typem QPolygon.

QPolygon minimumAreaEnclosingRectangle(QPolygon &ch, QPolygon &rectangle, QLine &direction)
Metoda pro výpočet hlavních směrů budovy. Na vstupu je konvexní obálka ch a prázdné proměnné rectangle a direction. Do rectangle se uloží minimální ohraničující obdélník a do direction hlavní směr budovy.

7.2.3 Generator

Třída Generator obsahuje různé metody, které automatizovaně vrací různé konvexní či nekonvexní množiny bodů různých tvarů.

std::vector<QPoint >generateCircle(int n)
Metoda generuje výstupní množinu bodů ve tvaru kruhu. Na vstupu je počet generovaných bodů. Návratová hodnota je vektor bodů.

std::vector<QPoint >generateEllipse(int n)
Metoda generuje výstupní množinu bodů ve tvaru elipsy. Na vstupu je počet generovaných bodů. Návratová hodnota je vektor bodů.

std::vector<QPoint >generateSquare(int n)
Metoda generuje výstupní množinu bodů ve tvaru čtverce. Na vstupu je počet generovaných bodů, pokud není dělitelný čtyřmi je počet upraven. Návratová hodnota metody je vektor bodů.

std::vector<QPoint >generateStarShape(int n)
Metoda generuje výstupní množinu bodů ve tvaru hvězdy. Na vstupu je počet generovaných bodů. Návratová hodnota je vektor bodů.

std::vector<QPoint >generateRandomPoints(int n)
Metoda generuje množinu náhodně rozmístěných bodů. Na vstupu je počet generovaných bodů. Návratová hodnota je vektor bodů.

std::vector<QPoint >generateGrid(int n)
Metoda generuje pravidelnou mřížku. Na vstupu je počet generovaných bodů. Pokud není odmocnina celé číslo, je počet upraven, tak aby odmocninou celé číslo bylo. Návratová hodnota metody je vektor bodů.

7.2.4 SortbyX

Třída, která definuje operátor přetížení *()*, který třídí body podle X souřadnice. Pokud je X souřadnice stejná, rozhoduje se podle souřadnice Y.

7.2.5 SortbyY

Třída, která definuje operátor přetížení *()*, který třídí body podle Y souřadnice. Pokud je Y souřadnice stejná, rozhoduje se podle souřadnice X.

7.2.6 SortbyAngle

Třída, která definuje operátor přetížení *()*, který třídí úhly typu Angle (samostatně definovaná struktura) podle velikosti. Pokud je úhel stejný, rozhoduje vzdálenost mezi datovými prvky typu Angle.

7.2.7 Draw

Třída Draw dědí od třídy QWidget.

void mousePressEvent
V této funkci je vykreslen bod q a je přidán do vektoru bodů *points*

void paintEvent
Tato metoda slouží k vykreslení bodů *points*. Zároveň je díky této metodě vykreslena konvexní obálka a minimum enclosing box spolu s hlavním směrem budovy.

void clearCanvas
Metoda sloužící k vymazání všech polygonů i bodů a k překreslení. Volá se před automatickým generováním vstupní množiny.

void clearCanvas
Metoda sloužící k vymazání všech polygonů i bodů a k překreslení. Volá se před automatickým generováním vstupní množiny.

QPoint getPoints
Metoda, která vrací privátní body q třídy Draw.

QPoint getConvexHull
Metoda, která vrací privátní konvexní obálku P třídy Draw, aby bylo možné ji vracet ve třídě Widget.

QPoint setCH
Metoda, která naopak vektor konvexní obálky, který je vytvořen ve třídě Widget, uloží do privátních datových typů ve třídě Draw. Obdobně pracují i další veřejné metody **QPoint setPoints**, **QPoint setBox** a **QPoint setDirection**.

7.2.8 Widget

void on_create_CH_clicked

Při stisknutí tohoto tlačítka se v závislosti na vybrané variantě algoritmu v Comboboxu, provede daný výpočet. A výsledek se předá třídě Draw pro vykreslení. Je zde také měřen čas jednotlivých algoritmů pomocí funkce *std::clock*.

void on_quit_clicked

Při stisknutí tlačítka Quit je uživateli vyzván, zda si přeje ukončit aplikaci. Pokud stikne OK, aplikace se ukončí.

void on_clear_clear_clicked

Při stisknutí tlačítka Clear se zavolá metoda třídy Draw clearCanvas.

void on_generate_clicked

Při stisknutí tohoto tlačítka přivolána metoda generatePolygon, která podle zadaného počtu vrcholů vykreslí nekonvexní polygon.

void on_minimum_enclosing_box_clicked

Při stisknutí tlačítka se nad konvexní obálkou zavolá metoda třídy Algorithms minimumAreaEnclosingRectangle.

8 Testování

Testování bylo prováděno v režimu Release a bylo testováno chování všech čtyřech algoritmů. Testování je k dispozici v přiloženém souboru **testování.xlxs**. V tomto souboru jsou u všech algoritmů voleny typy vstupních množin kruh, mřížka a náhodné rozmístění. V každé ze záložek je vždy rozebrán jeden konrétní algoritmus a doby běhu jsou jednak vyplněny v tabulce, a jednak znázorněny do příslušného grafu. Testování pro daný počet bodů bylo provedeno vždy desetkrát. Výsledná doba běhu algoritmu je vypočtena jako průměr. Jelikož máme více měření, bylo možné určit i varianci. Poslední záložka obsahuje porovnání časové náročnosti všech algoritmů pro jednotlivé typy vstupních množin.

9 Závěr

V rámci úlohy byla vytvořena aplikace, která je schopna na vygenerovaných bodech zkonstruovat konvexní obálku. Zároveň je v rámci aplikace možné náhodně vygenerovat vstupní množiny bodů a to do šesti různých tvarů, což uživatel jistě ocení.

Součástí úlohy bylo i testování doby výpočtu jednotlivých konvexních obálek. U všech algoritmů je zahrnuta funkce, která opravuje singulární případy, tuto funkci tedy musíme do výsledné body běhu připočíst.

Nyní je na čase shrnout výsledky testování. Algoritmus Jarvis Scan měl znatelně nejhorší čas, a to zejména pro vstupní množinu ve tvaru kruhu, kdy se průměrná doba běhu pro množinu o milionu bodech pohybovala kolem 25000 ms. Naopak metoda Sweep Line se zdá býti nejlepším řešením, a to pro všechny typy vstupních množin. Sweep Line má velmi krátkou dobu běhu i pro kruhovou množinu, kdy Quick Hull poněkud zaostává. Jinak jsou algoritmy Sweep Line a Quick Hull srovnatelné. Metoda Graham Scan je o trochu pomalejší, ale má vyrovnanou dobu běhu pro všechny vyzkoušené typy vstupních množin.

9.1 Náměty na vylepšení

- Celý skript by mohl být převeden do floating point number datových typů, jelikož přetypovávání na integery v rámci odstranění warningů je poněkud krkolomné. Bohužel prvotním nápadem bylo používat datové typy QPointF, QLineF, QPolygonF, funkce Quick Hull však padala z důvodu špatné segmentace. Tuto chybu se vyřešit nepodařilo, pravděpodobně souvisí se složitostí rekurzivní metody. Body ze vstupní množiny byly tedy generovány s datovým typem QPoint.

- V rámci úlohy bylo provedeno mnoho testování pro různý počet bodů, různý typ vstupní množiny a různý algoritmus. Za úvahu by určitě stálo, popřemýšlet o vhodné automatizaci. Autorce se bohužel nedaří spustit výsledný exe soubor, aplikace hlásí chybu Kód nelze spustit, protože se nenašel Qt5Cored.dll... I když byla snaha tuto chybu opravit, nepodařilo se. Volání takového exe souboru odkudkoli by kvůli této chybě také zřejmě nefungovalo. Proto žádná automatizace nakonec implementována nebyla.

Nicméně, pokud by bylo zaručeno, že exe soubor lze bezchybně spustit a zavolat s příslušnými parametry, určitě by časová úspora testování byla velmi výrazná. Vzhledem k tomu, že autorka má zkušenosti se psaním Powershell skriptů, nebyl by problém v rámci takového skriptu nadefinovat tři argumenty - typ vstupní množiny, počet generovaných bodů vstupní množiny a typ algoritmu ke konstrukci konvexní obálky. V Powershell skriptu by se poté v několika for cyklech volal .exe soubor s příslušnými argumenty.

Autorka má již zkušenosti s voláním Python skriptu s argumenty v Powershellu. V Pythonu je možné celkem jednoduše nadefinovat pomocí knihovny argparse argumenty, které je pak možné Python skriptu předat právě například pomocí Powershellu nebo jen v obyčejném .bat souboru.

V Qt by bylo možné vytvořit něco obdobného a to pomocí třídy QCommandLineParser, kterou je třeba přidat do zdrojového souboru main.cpp klauzulí $\#include < QCommandLineParser >$. Uvnitř funkce main je poté nutné nadefinovat vstupní parametry a předat je třídě widget. Výsledné doby běhu jednotlivých algoritmů by mohly být dále vyexportovány spolu s označením použitých argumentů do textového souboru.

Literatura

[1] Graham Scan). [online], Naposledy navštíveno 28. 10. 2019, dostupné z: https://www.webdepot.umontreal.ca/Usagers/p1038321/MonDepotPublic/ift3150.

[2] Convex hull. [online], Naposledy navštíveno 30. 10. 2019, dostupné z: https://csdl-images.computer.org.

[3] ITopologicalOperator.ConvexHull Method. [online], Naposledy navštíveno 30. 10. 2019, dostupné z: http://resources.esri.com/help/9.3/arcgisengine/arcobjects/esriGeometry.

[4] Minimum Enclosing Bounding Box. [online], Naposledy navštíveno 30. 10. 2019, dostupné z: https://geidav.files.wordpress.com/2014/01/ombb_candidates.png?w=368&zoom=2.

[5] Quick Hull Algorithm. [online], Naposledy navštíveno 30. 10. 2019, dostupné z: https://media.springernature.com/original/springer-static/image.

[6] Park, J.: Shortest Path Algorithms. [online], Naposledy navštíveno 28. 10. 2019, dostupné z: https://web.stanford.edu/class/cs97si/09-computational-geometry.pdf.