

České vysoké učení technické v Praze
Fakulta stavební

155ADKG: Konvexní obálky

Michael Kala
Anna Zemánková

1 Zadání

Navrhněte aplikaci s grafickým rozhraním, která určí a vizualizuje konvexní obálku množiny bodů. Pro výpočet konvexní obálky použijte algoritmy Jarvis Scan, Quick Hull a Sweep Line.

Pro testování algoritmů užíjte rozložení bodů:

- 1) náhodné
- 2) mřížka
- 3) kružnice

2 Údaje o bonusových úlohách

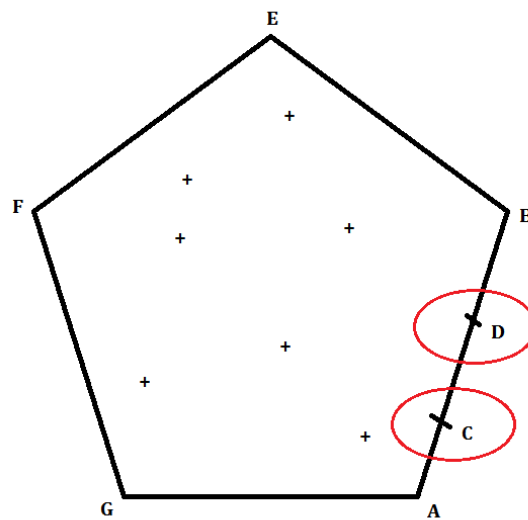
V rámci této úlohy byly zpracovány všechny bonusy kromě generování star-shaped množiny bodů.

3 Popis a rozbor problému

Mějme množinu S bodů $P_i[x_i, y_i]$, naším úkolem je nalézt konvexní obálku κ této množiny.

Konvexní obálka κ je hranice nejmenší konvexní množiny obsahující všechny body. Pokud mezi počátečním a koncovým bodem přímého segmentu konvexní obálky leží více bodů, jde o konvexní obálku, pokud jsou odstraněny, jde o striktně konvexní obálku. Např.:

- Konvexní obálka: A,C,D,B,E,F,G
- Striktně konvexní obálka: A,B,E,F,G



Obrázek 1: Konvexní obálka

4 Popis algoritmů

Pro tuto úlohu byly pro výpočet konvexní obálky vybrány algoritmy Jarvis Scan, Graham Scan, Quick Hull a Sweep Line

4.1 Jarvis Scan

Tento algoritmus nejprve vybere pivota $q : y_{min}$ a následně jej přidá do konvexní obálky, poté je vybrán bod $p_j - 1$ tak, aby spojnice pivota a $p_j - 1$ byla rovnoběžná s osou X $p_j - 1 : x_{min}, y_{min}$. Kvůli numerické stabilitě je vybrána minimální souřadnice X z celé množiny bodů.

Poté je v cyklu vybírán a následně přidáván do konvexní obálky bod $p_j + 1$ tak, aby byl $sphericalangle(p_j - 1, p_j, p_j + 1)$ největší (v prvním kroku $p_j = q$). Cyklus skončí, jakmile dojde opět k pivotovi.

4.1.1 Implementace metody

1. Nalezení pivota q : $q = y_{min}$
2. Přidej $q \rightarrow \kappa$
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 $p_j + 1 = argmax_i p_i \in P \angle (p_j - 1, p_j, p_i)$
6. Přidej $p_j + 1 \rightarrow \kappa$
7. $p_j - 1 = p_j; p_j = p_j + 1$

4.2 Graham Scan

Nejprve je vybrán pivot q tak, aby z něj bylo vidět na všechny zbylé body. Bodem q je vedena rovnoběžka s osou X a jsou určeny úhly ω od osy X po všechny ostatní body množiny. Následně jsou seříděny podle velikosti ω a pospojovány v tomto pořadí, Vznikne tak star-shaped polygon.

Do zásobníku je přidán bod q a první následující bod, tedy druhý. Poté je testován třetí bod - pokud je splněna podmínka levotočivosti, je přidán do zásobníku. Následně je testován čtvrtý bod. Je-li splněna podmínka levotočivosti, je přidán do zásobníku. Není-li podmínka splněna, je poslední bod v zásobníku smazán a je testován čtvrtý bod z druhého bodu atd. Po skončení cyklu je v zásobníku konvexní obálka množiny bodů.

4.2.1 Implementace metody

1. Nalezení pivotu q : $q = y_{min}$
2. Seřídění $p_i \in S$ dle $\omega_i = \angle(p_i, q, x)$, index j odpovídá jejich seříděnému pořadí
3. Pokud $\omega_k = \omega_l$, vymaž bod p_k, p_l bližší ke q
4. Inicializuj $j = 2, S = \emptyset$
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 $p_t - 1, p_t$:
 8. $S \leftarrow p_j$
 9. $j = j + 1$
10. else pop S

4.3 Quick Hull

Algoritmus Quick Hull je rekurzivní a skládá se tedy s lokální a globální procedury.

V lokální proceduře je hledán nejvzdálenější bod od dělicí přímky, který zároveň leží v její pravé polorovině. Spojnici tohoto bodu a koncového bodu dělicí přímky nazýváme první segment a spojnici vyhledaného bodu a počátečního bodu dělicí přímky druhý segment. Aby byly body do konvexní obálky přidány ve správném pořadí, je nejprve provedena rekurze pro první segment, následně přidán vyhledaný bod a poté provedena rekurze pro druhý segment.

V globální proceduře je z množiny S nalezen počáteční p_s (x_{min}) a koncový p_e (x_{max}) bod, které tvoří dělicí přímku. Nejprve je do konvexní obálky přidán koncový bod dělicí přímky, následně je provedena lokální procedura pro horní polorovinu dělicí přímky, přidán počáteční bod dělicí přímky a provedena lokální procedura pro dolní polorovinu. Takto dostaneme body konvexní obálky ve správném pořadí.

4.3.1 Implementace metody - lokální procedura

1. Najdi bod $p = \operatorname{argmax}_{p_i \in S} ||p_i - (p_s, p_e)||, p \in \sigma_r(p_s, p_e)$
2. If $p \neq \emptyset$:,
3. QuickHull (s, i, S, κ) // Upper Hull
4. $\kappa \leftarrow p$
5. QuickHull (i, e, S, κ) // Lower Hull

4.3.2 Implementace metody - globální procedura

1. $\kappa = \emptyset, S_U = \emptyset, S_L = \emptyset$
2. $q_1 = x_{min}, q_3 = x_{max}$
3. $S_U \leftarrow q_1, S_U \leftarrow q_3$
4. $S_L \leftarrow q_1, S_L \leftarrow q_3$
5. for $p_i \in S$
6. if $(p_i \in \sigma_l(q_1, q_3))$ $S_U \leftarrow p_i$
7. else $S_L \leftarrow p_i$
8. $\kappa \leftarrow q_3$
9. QuickHull ($1, 0, S_U, \kappa$) // Upper Hull
10. $\kappa \leftarrow q_1$
11. QuickHull ($0, 1, S_L, \kappa$) // Lower Hull

4.4 Sweep Line

Metoda sweep line pracuje s "přepisováním" předchůdců a následníků a jelikož by se tu až nezdravě opakovala slova přechůdce a následník, samotná implementace metody bude srozumitelnější (jako ostatně vždy).

4.4.1 Implementace metody

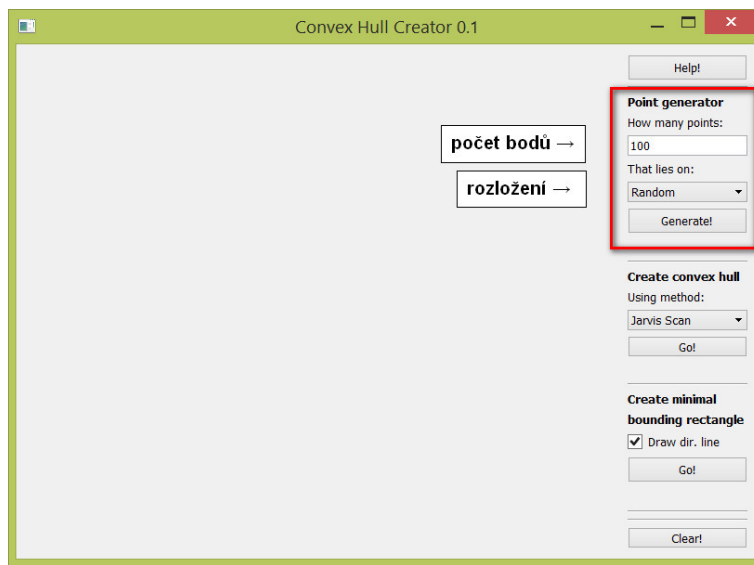
1. Sort $P_s = \text{sort}(P)_{byx}$
2. if $p_3 \in \sigma_L(p_1, p_2)$
3. $n[1] = 2; n[2] = 3; n[3] = 1$
4. $p[1] = 3; p[2] = 1; p[3] = 2$
5. else
6. $n[1] = 3; n[3] = 2; n[2] = 1$
7. $p[1] = 2; p[3] = 1; p[2] = 3$
8. for $p_i \in P_s, i > 3$
9. if $(y_i > y_i - 1)$
10. $p[i] = i-1; n[i] = n[i-1]$
11. else
12. $n[i] = i-1; p[i] = p[i-1]$
13. $n[p[i]] = i; p[n[i]] = i;$
14. while $(n[n[i]]) \in \sigma_R(i, n[i])$
15. $p[n[n[i]]] = i; n[i] = n[n[i]];$
16. while $(p[p[i]]) \in \sigma_L(i, p[i])$
17. $n[p[p[i]]] = i; p[i] = p[p[i]];$

5 Vstupní data

Vstupními daty je množina bodů, kterou lze zadat dvěma způsoby:

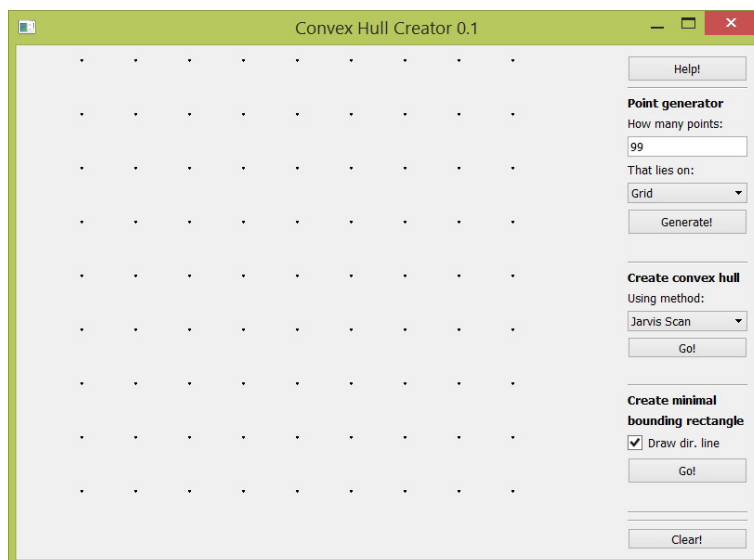
- 1) klikáním na canvas
- 2) použitím generátoru bodů, který je součástí aplikace, přičemž se body generují dle zadaných kritérií.

Do generátoru bodů je třeba zadat počet bodů a následně vybrat v rozbalovacím menu požadované rozmístění - Random/Grid/Circle/Elipse/Square



Obrázek 2: Generátor bodů

V případě, že je zadán nekorektní počet bodů pro dané rozmístění - např. pro čtverec nebo grid, je obrázen vytvořen z nejbližšího menšího možného počtu prvků.

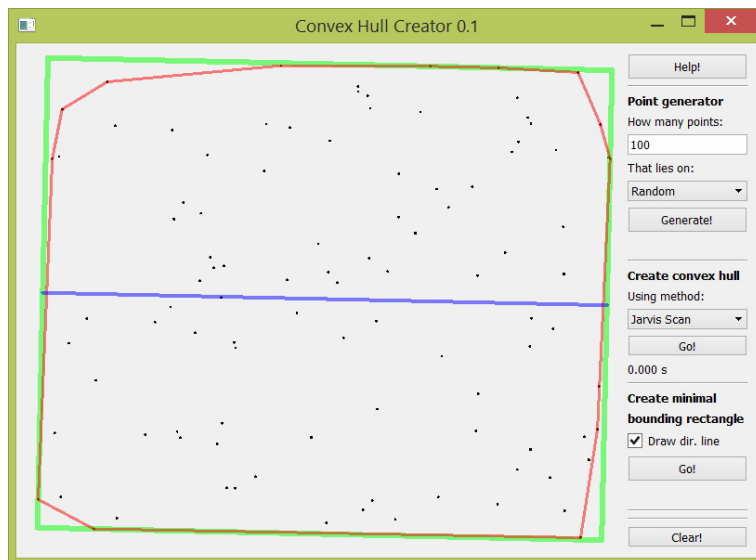


Obrázek 3: Ukázka nekorektního počtu bodů pro zadaný tvar

Pokud jsou body vygenerovány pomocí generátoru, je možné přidat další klikáním na canvas. Při spuštění generátoru jsou všechny dosud existující body vymazány.

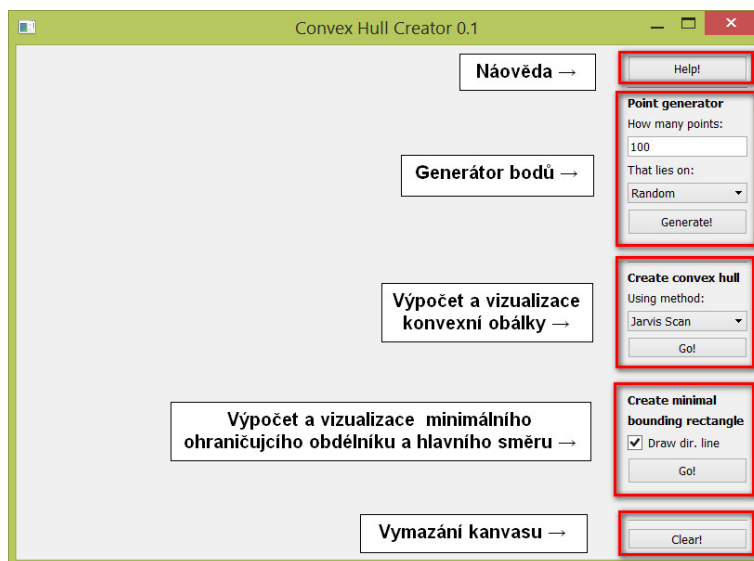
6 Výstupní data

Výsledná data (konvexní obálka/minimální ohraňující obdélník/hlavní směr útvaru) jsou vizualizována v kanvasu aplikace.



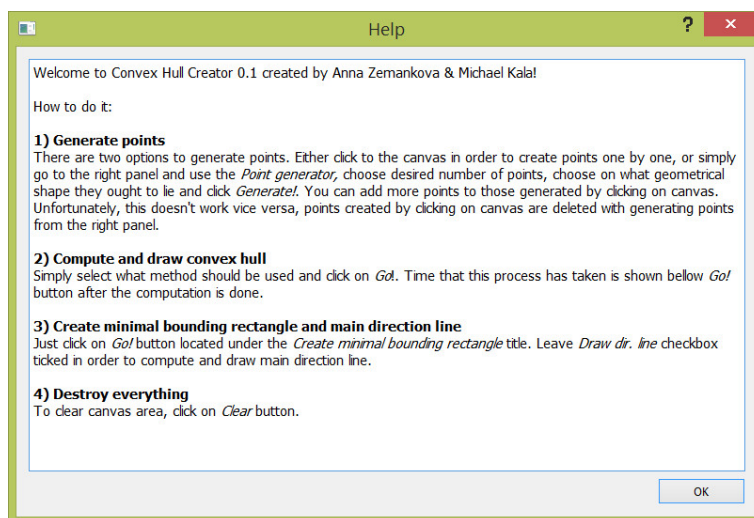
Obrázek 4: Výstupní data

7 Ukázka vytvořené aplikace



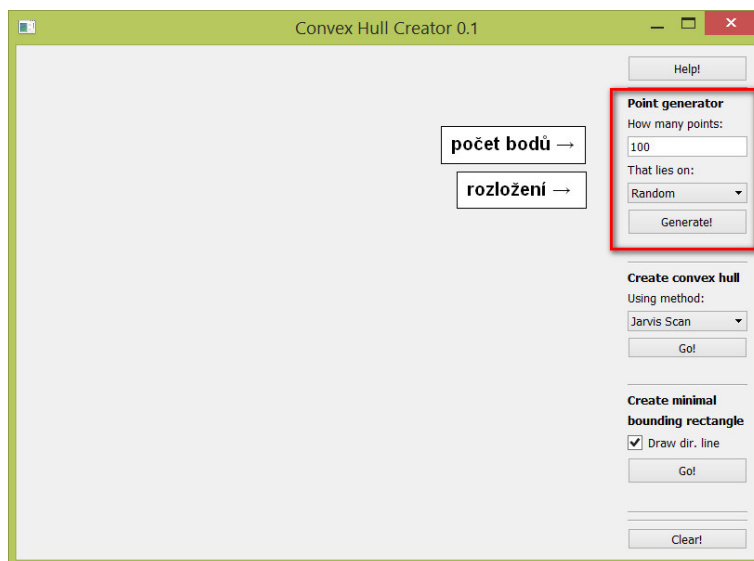
Obrázek 5: Okno aplikace

V Pravé horní části je možnost kliknutí na nápovědu "Help", poté se otevře okno s nápovědou, kde je popsán způsob zadávání vstupních dat a funkcionality aplikace.



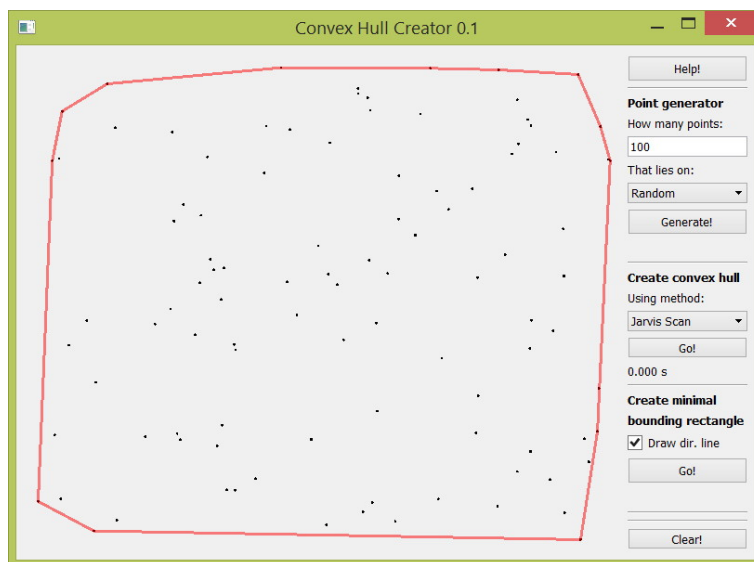
Obrázek 6: Nápověda

Nejprve je třeba zadat vstupní data - množinu bodů.



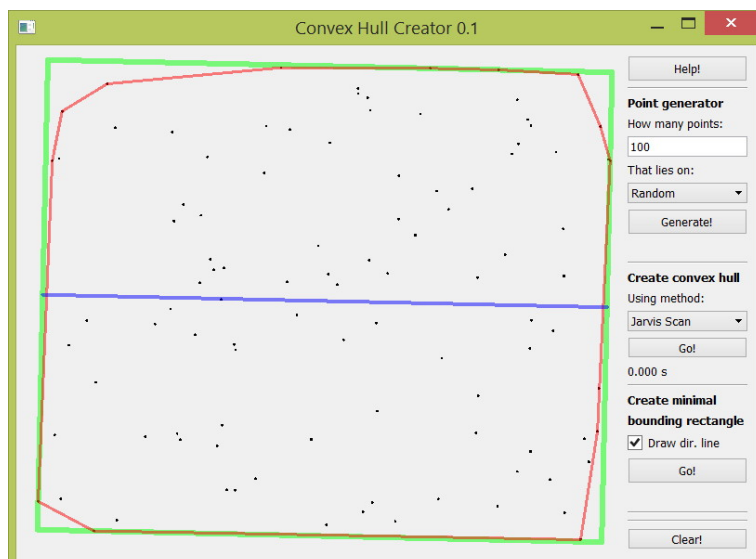
Obrázek 7: Vstupní data

Následně je možné vybrat metodu výpočtu konvexní obálky a kliknutím na tlačítko GO v sekci Create convex hull zahájen výpočet a vizualizace konvexní obálky zadané množiny bodů. Pod tlačítkem Go je vypsán čas, jak dlouho trval výpočet.



Obrázek 8: Výpočet konvexní obálky

Pokud chceme spočítat minimální ohraničující obdélník, lze tak učinit kliknutím na tlačítko Go! v sekci Create inimal bounding rectangle. Pokud je v této sekci zaškrtnuta možnost draw dir. line, vykreslí se také hlavní směr útvaru.



Obrázek 9: Výpočet hlavního směru a nejmenšího ohraňujícího obdélníku

Vše je uvedeno do původního stavu (smazány body i vypočtené výsledky) kliknutím na tlačítko Clear.

8 Dokumentace

9 Přílohy

10 Závěr

Aplikace byla vzhledem ke zpracování bonusů programována tak, že vstupní i výstupní data jsou typu float, při testování doby běhu algoritmů však bylo zjištěno, že při použití datového typu float je výpočet velmi VELMI pomalý, naopak při přepsání kódu do verze pracující s integrem byl výpočet výrazně rychlejší. Bohužel už ale byly vyřešeny bonusy za předpokladu, že vstupní data jsou typu float, a proto jsou nakonec odevzdávány dvě verze - float pro funkcionality aplikace pro menší množiny bodů a integer pro testování doby běhu algoritmů.

11 Zdroje

1. BAYER, Tomáš. Konvexní obálky [online][cit. 21.10.2018].
Dostupné z: <https://web.natur.cuni.cz/~bayertom/images/courses/Adk/adk4.pdf>