Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Bakalářská práce

Pomocné programové vybavení a experimenty pro vizualizaci modelů terénu

Plzeň 2017 Tomáš Šedivý

Místo této strany bude zadání práce.

Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 1. května 2017

Tomáš Šedivý

Poděkování

Tímto bych rád poděkoval prof. Dr. Ing. Ivaně Kolingerové a Ing. Ondřeji Kaasovi za odborné vedení a výpomoc při vypracovávání této práce.

Abstract

This thesis solves two separate tasks. The first task is to create a software that will modify the terrain model using centroidal Voronoi tesselation and to examine whether this adjustment improves the properties of automatically generated contour lines. The second part of the thesis consists of the implementation of several clustering algorithms, their comparison with an already implemented algorithm and the formulation of recommendations on the suitability of using these algorithms.

Abstrakt

Tato práce řeší dva samostatné úkoly. Prvním úkolem je vytvořit programové vybavení, které bude upravovat model terénu pomocí centroidní Voroného teselace, a prozkoumat, zda tato úprava přinese zlepšení vlastností automaticky generovaných vrstevnic. Druhá část práce sestává z implementace několika shlukovacích algoritmů, jejich srovnání s již implementovaným algoritmem a vytvoření doporučení o vhodnosti využití těchto algoritmů.

Obsah

1 Úvod

Automatické generování vrstevnic, využívá digitální reprezentaci terénu ve formě trojúhelníkové sítě. Tato trojúhelníková síť sestává ze sady bodů v prostoru a trojúhelníků, jejichž vrcholy jsou tvořeny právě těmito body.

Kvalita trojúhelníkové sítě má značný vliv na kvalitu softwarově generovaných vrstevnic. Takto vygenerované vrstevnice mohou obsahovat nežádoucí artefakty, které vyplynou z vlastností modelu terénu.

Prví část práce spočívá ve vytvoření programu, který pomocí tzv. centroidní Voroného teselace upraví model terénu, a v ověření, zda se na základě takovéto úpravy terénu zlepší kvalita generovaných vrstevnic.

Druhá část práce byla vytvářena v rámci projektu na katedře informatiky a výpočetní techniky. Mým úkolem bylo implementovat algoritmy K-Means a C-Means a porovnat je s již implementovaným algoritmem. Dalším úkolem v rámci této práce bylo vytvořit sadu iterátorů pro jednotlivé algoritmy, které budou pro vstupní data a daný algoritmus hledat takovou konfiguraci, které poskytne nejlépe ohodnocené řešení.

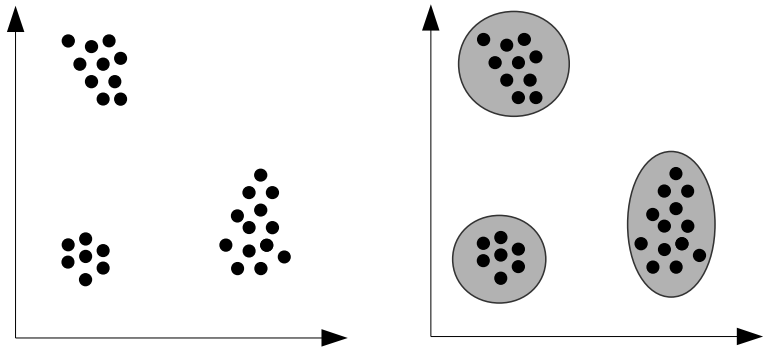
Práce sestává z následujících částí:

- Kapitola 2 - teoretická část práce, popisující problematiku shlukování, shlukovacích algoritmů, metod triangulace a problematiku centroidní Voroného teselace.
- Kapitola 3 - vytvořené řešení, tato kapitola se zabývá vytvořeným řešením daných problémů, experimenty a analýzou získaných výsledků
- Kapitola 4 - závěr, zabývající se dosaženými výsledky

2 Teoretická část

2.1 Shlukování

Shluková analýza, neboli shlukování, je proces rozdělení (organizování) množiny objektů na podmnožiny (shluky) tak, že prvky v rámci jednoho shluku si jsou co nejvíce podobné a objekty ze dvou různých shluků co nejméně [1].



Obrázek 2.1.1: Ukázka shlukování bodů

Na obrázku 2.1.1 je vidět ukázka shlukování ve dvourozměrném prostoru. V levém obrázku jsou umístěny tři skupiny bodů. V pravém obrázku je vidět jejich rozdělení na tři shluky na základě vzdálenosti bodů. Tento příklad demonstruje shlukování na základě vzdálenosti, je však možné body shlukovat pomocí dalších kritérií (viz 2.3).

Shlukování má své využití v mnoha oblastech. Příkladem může být použití při datové analýze, kdy jsou zpracovávána velká množství dat. Je tedy možné využít shlukování pro nalezení několika podmnožin dat, na základě tohoto rozdělení vytvořit prototyp každého shluku (prvek, který nejlépe daný shluk vystihuje) a provést analýzu na těchto prototypech [2].

2.2 Rozdělení metod shlukování

Metody shlukování lze třídit na základě různých kritérií, což bude ukázáno v následujících podkapitolách.

2.2.1 Hierarchické a nehierarchické

Kritériem pro toto rozdělení metod je, zda mohou být shluky vnořeny [3].

Při hierarchickém shlukování je vytvářen strom, kde je každý uzel (shluk), vyjma listů stromu, tvořen sjednocením svých potomků. Listy tohoto stromu jsou tedy elementární, dále nedělitelné shluky (nebo jednotlivé prvky) a kořen je tvořen množinou obsahující všechny prvky.

Hierarchické shlukování lze dále dělit podle směru shlukování, a to na [4]:

- **Aglomerativní** shlukování postupuje od počátečního rozkladu množiny objektů (elementární, jednoprvkové shluky) a na základě podobnosti mezi těmito prvky (viz 2.3) postupně po dvojicích slučuje jednotlivé shluky, dokud nevznikne jeden, kořenový shluk.

- **Divizivní** shlukování postupuje opačným směrem. Na začátku procesu je dán kořenový shluk obsahující všechny prvky množiny. Dalšími kroky je postupné rozdělování jednotlivých shluků na dvě podmnožiny tak, aby bylo splněno dané kritérium (aby si dané podmnožiny byly co „nejméně podobné“).

Nehierarchické shlukování (anglicky *partitional clustering*) je rozdělení objektů do shluků tím způsobem, že každý objekt je součástí právě jednoho shluku.

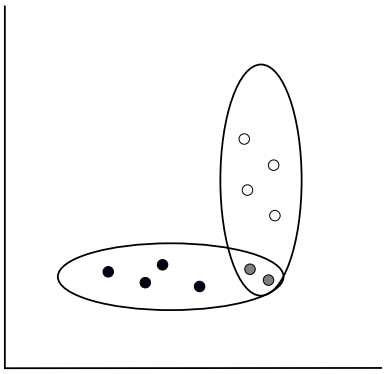
2.2.2 Podle příslušnosti do shluků

Jednotlivé objekty mohou příslušet do jednoho či více shluků. Pomocí tohoto kritéria se dají metody rozdělit na výlučné, překrývané a fuzzy.

Výlučné shlukovací metody přiřadí každý objekt do právě jednoho shluku. Takovéto rozdělení bylo ukázáno v obrázku 2.1.1.

Pokud je možné, aby objekt příslušel do více shluků než jednoho, lze toho docílit pomocí dvou způsobů.

1. **Překrývané shlukování** - využívá se v případě, kdy může objekt příslušet do jednoho i více shluků. Příkladem může být rozdělení studentů podle předmětů, které studují. Je možné, aby student studoval jen jeden předmět, nicméně zpravidla bude studovat předmětů více. V tomto případě by byl součástí několika shluků.



Obrázek 2.2.1: Ukázka překrývaného shlukování

2. **Fuzzy shlukování** - tyto metody jsou založeny na principu, že každý prvek přísluší do každého shluku, nicméně tato příslušnost je podmíněna váhou. Tato váha nabývá hodnot z intervalu $[0, 1]$, přičemž 1 odpovídá stavu, kdy objekt úplně přísluší do shluku, 0 naopak, že objekt do shluku nepřísluší vůbec.

2.3 Podobnost prvků

Jak již bylo zmíněno, základem pro metody shlukování je možnost vyjádřit míru podobnosti mezi dvěma prvky. Jakým stylem se toho docílí závisí buď na metodě shlukování, nebo na datech samotných.

Příkladem mohou být body v prostoru, kdy mírou podobnosti bude právě vzdálenost mezi těmito body. Při určování vzdálenosti mezi body lze použít různé metriky. Nejčastěji se používají:

1. **Eukleidovská metrika** - tato metrika vyjadřuje délku úsečky mezi dvěma body. Vzdálenost ϱ mezi body x a y v n-dimenzionálním prostoru se spočte:
$$\varrho = \sqrt[2]{\sum_{i=1}^{n} |x_i - y_i|^2}$$

2. **Manhattanská metrika**, neboli součtová, určuje vzdálenost mezi body jako vzdálenost, kterou je potřeba urazit mezi danými body tak, že se smí pohybovat jen po navzájem kolmých přímkách rovnoběžných s osami prostoru. Vzdálenost ϱ mezi body x a y v n-dimenzionálním prostoru se spočte:
$$\varrho = \sum_{i=1}^{n} |x_i - y_i|$$

3. **Maximová metrika** definuje vzdálenost mezi dvěma body jako maximální vzdálenost v jedné dimenzi:

$$\varrho = \max\{|x_1 - y_1|, |x_2 - y_2|, \ldots, |x_n - y_n|\}$$

2.4 Vybrané shlukovací algoritmy

2.4.1 K-Means

Algoritmus K-Means je jedním z nejjednodušších algoritmů pro řešení problému shlukování. Tento algoritmus rozděluje objekty do K shluků, přičemž tento počet je dopředu znám.

Cílem tohoto algoritmu je minimalizace účelové funkce [5]:

$$f = \sum_{j=1}^{K} \sum_{i=1}^{N} \left\| x_i^{(j)} - c_j \right\|$$

kde $\left\| x_i^{(j)} - c_j \right\|$ je vzdálenost mezi centrem shluku c_j a bodem $x_i^{(j)}$ (příslušícího do daného centra j). K vyjadřuje počet shluků, N počet bodů určených ke shlukování.

Výpočet center shluků:
Vstupem algoritmu jsou shlukované objekty a počet shluků, které mají být vytvořeny. Výstupem jsou vytvořená centra shluků a rozřazení objektů do těchto shluků.
Algoritmus probíhá následovně [6]:

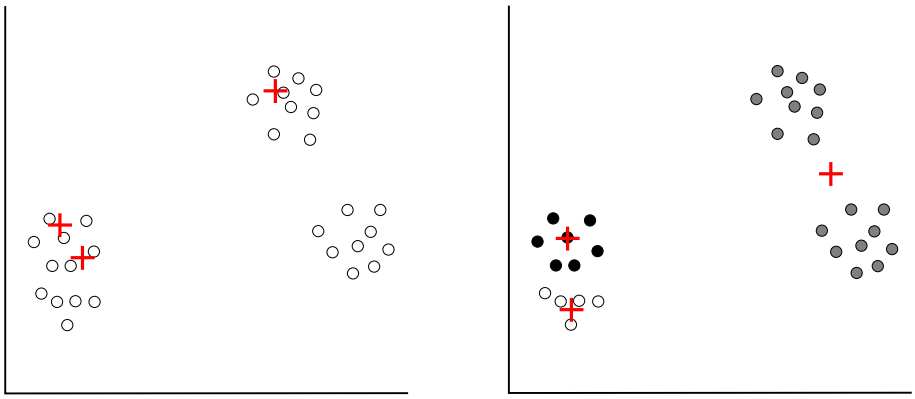
1. Náhodné umístění K počátečních center do prostoru tvořeného shlukovanými objekty.
2. Každý objekt se přiřadí k nejbližšímu centru.
3. Poté, co jsou všechny body přiřazeny, se spočítá těžiště každého takto vytvořeného shluku. Tato těžiště nahradí původní centra.
4. Kroky 2 a 3 se opakují, dokud dochází k přeřazení bodů do jiného centra.

Vzhledem k tomu, že se jako podobnost prvků v tomto algoritmu používá vzdálenost a určují se těžiště shluků, je pro tento algoritmus podmínkou

použití v prostoru s nějakou metrikou (příkladem může být eukleidovský prostor). Nicméně tento algoritmus konverguje v konečném počtu kroků k nějakému řešení. Toto řešení ovšem nemusí být globálním minimem účelové funkce, jelikož tento algoritmus má dva problémy [7].

1. **Počáteční rozdělení** center shluků má veliký vliv na výsledek shlukování. Je možné vyzkoušet několik iterací algoritmu s různými počátečními rozloženími center shluků a vybrat vhodnější řešení.

   Příkladem může být situace na obrázku 2.4.1. Z obrázku je patrné naznačení tří shluků, do nichž by se očekávalo, že budou body algoritmem rozděleny. Ovšem vzhledem k nevhodnému počátečnímu rozdělení nedošlo k vytvoření ideálních shluků.



Obrázek 2.4.1: Ukázka špatně zvoleného počátečního rozložení

2. **Zvolení parametru** K - jelikož není možné dopředu určit ideální počet shluků, je nutné, pokud chceme najít co nejlepší řešení, vyzkoušet několik průběhů algoritmu pro různé počty shluků. Příliš málo shluků způsobí, že jsou vzdálenosti bodů k centrům příliš velké.

Při implementaci algoritmu K-Means je nutné počítat s faktem, že je ve fázi přiřazování objektů ke shlukům možné, aby zůstalo centrum bez přiřazeného objektu. V tomto případě dojde ke zhoršení výsledku, je tedy nutné centrum vhodně přemístit. Jednou z možností je vybrat shluk, pro který je součet vzdáleností mezi body a centrem největší, určit konvexní obálku prostoru tvořenými body tohoto shluku, na náhodné místo tohoto prostoru centrum přemístit a opakovat krok s rozdělením bodů.

2.4.2 C-Means

Algoritmus C-Means patří do skupiny fuzzy shlukovacích algoritmů, což znamená, že každý objekt přísluší ke každému centru s danou pravděpodobností. Jako algoritmus K-Means má i tento algoritmus dopředu daný a fixní počet shluků.

Je založen na minimalizaci účelové funkce

$$f = \sum_{i=1}^{N} \sum_{j=1}^{C} u_{ij}^{m} \left\| x_i - c_j \right\|^2$$

kde:

- N je počet objektů a C je počet shluků
- u_{ij} je váha, s jakou bod x_i přísluší do shluku c_j. Tato váha nabývá hodnot z intervalu $[0, 1]$. S váhou 1 objekt plně přísluší do daného shluku, s váhou 0 vůbec.
- m je reálné číslo v intervalu $[1, \infty)$ a jeho hodnota ovlivňuje důležitost váhy u_{ij}. Čím větších hodnot m nabývá, tím rychleji klesá příslušnost objektu do daného shluku s jeho klesající vahou příslušnosti do tohoto shluku.
- x_i je i-tý objekt z množiny shlukovaných objektů
- c_j je j-té centrum shluku
- $\|x_i - c_j\|$ vzdálenost (podobnost) mezi měřeným bodem a daným centrem.

Jak již bylo řečeno, každý prvek přísluší do každého shluku na základě váhy (míry příslušnosti). Tyto hodnoty jsou zaneseny do matice vah U[i, j]. Počet sloupců matice (j) odpovídá počtu shluků, zatímco počet řádek (i) odpovídá počtu objektů, které jsou shlukovány. Každý řádek obsahuje hodnoty vah pro daný objekt ke všem centrům shluků.

Výpočet center shluků: Vstupem algoritmu jsou objekty, které mají být shlukovány, počet shluků, které mají být vytvořeny, parametr m a funkce, pomocí které lze spočítat váhu příslušnosti objektu ke shluku.

Algoritmus vypadá následovně:

1. Náhodně se umístí C počátečních center do prostoru tvořeného shlukovanými objekty. Provede se výpočet hodnot matice vah U^0.

2. V každém kroku se spočtou nová centra shuků (c_k, $k \in \{1, 2, \ldots, C\}$) se současnou maticí vah podle vzorce:

$$c_k = \frac{\sum_{i=1}^{n} u_{ij}^m \cdot x_i}{\sum_{i=1}^{n} u_{ij}^m}$$

3. Spočítá se nová matice vah U všechny body x_i, $i \in \{1, 2, \ldots, N\}$ a pro všechna centra c_j, $j \in \{1, 2, \ldots, C\}$. Hodnota u_{ij} odpovídá váze, s jakou bod na indexu i přísluší do shluku s indexem j a vypočítá se pomocí vzorce:

$$u_{ij} = \frac{1}{\sum_{l=1}^{C} \left(\frac{\|x_i - c_j\|}{\|x_i - c_l\|} \right)^{\frac{2}{m-1}}}$$

4. Kroky 2 a 3 se opakují, dokud největší změna ve váze mezi libovolným centrem a shlukem je větší, než předem dané ε.

Podobně jako u K-Means i u C-Means závisí na počátečním rozložení center a také na jejich počtu. Důležité je také zvolení parametru ε, protože při nízkých hodnotách nemusí algoritmus dojít k dobrému řešení a při příliš vysokých hodnotách naopak proběhne příliš mnoho iterací, což přidá na čase zpracování. Běžně lze jako ε použít hodnoty v řádu desetin až tisícin.

2.5 Geometrické základy

Konvexní obálka množiny bodů S je nejmenší konvexní polygon, který obsahuje všechny body množiny S. To znamená, že polygon je konvexní, jestliže každá úsečka, spojující libovolné dva body z tohoto polygonu, v něm je celá obsažena.

Pokud je množina objektů S rozdělena do K podmnožin s tou vlastností, že žádný objekt náležící do podmnožiny S_i nenáleží do žádné jiné množiny a každý objekt náleží do nějaké podmnožiny, nazývá se rozdělení $\{S_1, S_2, \ldots, S_K\}$ **teselací** množiny S [8]. Nejobvyklejší teselací je **Delaunayho triangulace**.

Triangulace T nad množinou bodů $P = \{p_1, p_2, \ldots, p_n\}$ v rovině představuje takové planární rozdělení bodů, které vytvoří soubor m trojúhelníků $T = \{t_1, t_2, \ldots, t_m\}$ tak, že platí [9]:

- libovolné dva trojúhelníky $t_i, t_j \in T, i \neq j$ mají společnou nejvýše hranu nebo vrchol
- sjednocení trojúhelníků je souvislá množina ve 2D
- uvnitř žádného z trojúhelníků není žádný další bod z P

Delaunayho triangulace je nejčastěji používaná triangulace. Má tu vlastnost, že žádný další bod z množiny P neleží v kružnici opsané libovolnému trojúhelníku. Tato triangulace maximalizuje minimální velikost úhlů v trojúhelnících.

Constrained Delaunay Triangulation (CDT, česky také Delaunayho triangulace s omezením) je taková Delaunayho triangulace, která vytváří triangulaci nejen ze vstupní množiny bodů, ale také hran. To znamená, že některé hrany jsou do triangulace vnuceny a musejí být obsažené ve výsledku. Vnucení takovýchto hran může být využito například pro zachování tvaru oblasti při triangulaci nekonvexní oblasti. Běžná triangulace by totiž do výsledku zahrnula i konvexní obálku vstupních bodů [10].

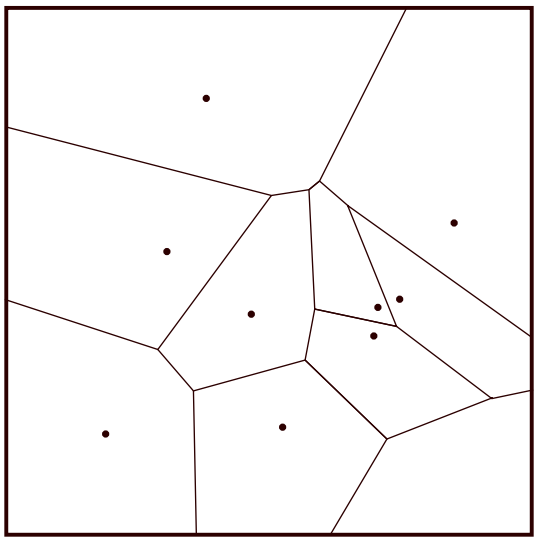
Pokud je množina S, obsahující K prvků x_i $(i = 1, 2, \ldots, K)$, rozložena na K podmnožin V_j tak, že:

$$V_j = \{w \in S \mid d(w, z_j) < d(w, z_i), i = 1, 2, \ldots, K, i \neq j\}$$

kde $d(w, z)$ je funkce vzdálenosti mezi prvky w a z, nazývá se rozdělení $\{V_1, V_2, \ldots, V_K\}$ **Voroného diagramem** (či Voroného teselací) množiny S.

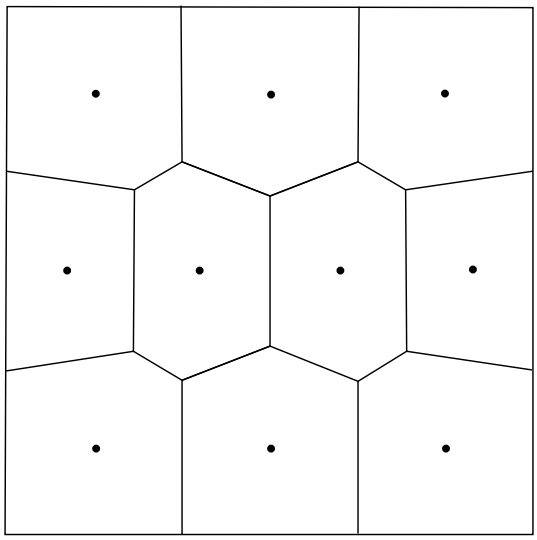
To znamená, že rozdělení objektů množiny S na podmnožiny V_j je provedeno tím stylem, že všechny body z podmnožiny V_j jsou blíže k bodu z_i než ke kterémukoli jinému bodu z množiny S.

Množina bodů $z_i, i = 1, 2, \ldots, K$ se nazývá generátory Voroného teselace. Podmnožiny V_j jsou také nazývány Voroného buňkami.



Obrázek 2.5.1: Ukázka výřezu Voroného diagramu (převzato z [8])

Centroidní Voroného teselace (CVT) je speciální případ Voroného teselace, kdy generátory Voroného buněk odpovídají centroidům (těžištím) daných buněk. Těžištěm buňky se myslí bod vytvoření průměrováním souřanic jednotlivých bodů tvořících danou buňku. Neuzavřené Voroného buňky se musí uměle uzavřít hranicemi prostoru, na kterém se Voroného diagram určuje (například rámeček na obrázku 2.5.1).



Obrázek 2.5.2: Ukázka výřezu centroidního Voroného diagramu (převzato z [8])

2.5.1 Algoritmy sestrojení CVT

Lloydův algoritmus

Lloydův algoritmus je iterativní metoda získání CVT. Sestává ze dvou kroků:

1. Určí se Voroného diagram pro celou množinu bodů.
2. Pro každou Voroného buňku se určí těžiště. Původní body se přesunou na místa těchto těžišť.

Tyto kroky se opakují tak dlouho, dokud největší změna (posunutí) bodů není menší než předem zvolené ε (typickou volbou bývají hodnoty v řádu tisícin jednotek).

McQueenův algoritmus

McQueenova metoda má oproti Lloydově tu výhodu, že nevyžaduje konstruování Voroného diagramu v každé iteraci. Namísto toho vypadá algoritmus následovně:

1. Do prostoru S tvořeného K body $z_1, z_2, \ldots, z_K$ se náhodně umístí bod w
2. Najde se bod z_i, který je nejblíže k bodu w

3. Vytvoří se bod, který je průměrem z_i a w, který nahradí původní bod z_i. Během jednotlivých iterací se zachovává informace o tom, kolikrát byl daný bod doposud posunut. Průměrování bodů w a z_i je děláno s váhou odpovídající počtu přesunů.

   Namísto průměrování

   $$z_i = \frac{w + z_i}{2}$$

   se tedy provádí

   $$z_i = \frac{w + n \cdot z_i}{n + 1}$$

   kde n je číslo udávající, kolikrát byl doposud daný bod posunut.

4. Tyto kroky se opakují tak dlouho, dokud není proveden předem daný počet iterací

McQueenova metoda má tu nevýhodu, že i přes to, že body tímto postupem konvergují ke generátorům CVT, je tato konvergence velmi pomalá. Pro velké počty bodů může být potřeba i několika milionů iterací.

Tento problém lze řešit tou úpravou, že se vygeneruje několik bodů najednou (klidně i tisíce), tyto body se poté rozdělí do shluků podle vzdáleností k jednotlivým původním bodům. Takto vzniklé zhluky bodů se poté zprůměrují a finální průměrování s bodem z_i se provede právě s tímto průměrem vygenerovaných bodů. Nejen, že se takto dojde ke konečnému řešení rychleji, ale je také možné tento výpočet paralelizovat [8].

CVT má širokou škálu využití, například při shlukování, segmentaci obrazu, kompresi dat, dá se i použít pro řešení optimálního rozmístění prostředků. CVT se dá využít i pro zlepšení výsledků triangulace. Rozprostřením bodů po prostoru tak, aby tvořily CVT, a dopočítáním nových z souřadnic, se docílí toho, že při triangulaci vzniknou trojúhelníky co nejvíce rovnostranné. To může být výhodou v mnoha aplikacích.

3 Vytvořené řešení

3.1 Úpravy modelu terénu pomocí CVT

Úkolem této části práce bylo zjistit, zda úprava rozprostření bodů v terénu pomocí centroidní Voroného teselace pomůže zlepšit kvalitu automaticky generovaných vrstevnic. Generování vrstevnic využívá triangulace daného terénu, kvalita trojúhelníkové sítě a samotných trojúhelníků by tedy mohla značně ovlivnit výsledek generování.

Za tímto účelem byl vytvořen program, který upraví model terénu na základě následujících kroků:

1. načtení vstupních dat
2. přemístění bodů terénu v x a y souřadnicích pomocí CVT
3. přepočítání z souřadnice, aby nedošlo ke znehodnocení modelu terénu po přesunech bodů
4. uložení nového modelu do souboru

Program byl napsán v jazyce C++, vzhledem k tomu, že to je překládaný jazyk a bylo zapotřebí maximálního výkonu. Pro vytvoření grafického uživatelského rozhraní jsem použil framework Qt, protože je multiplatformní a je tedy možné program přeložit a spustit na podporovaném operačním systému (macOS, Linux, Windows).

3.1.1 Vstupní data

Formát vstupních dat byl pevně dán požadavkem vedoucí práce a kompatibilitou s dalšími existujícími programy. Vstupní data jsou načítána ze souboru v textové formě. Soubor obsahuje následující data:

Seznam bodů terénu (vrcholů trojúhelníků)

Seznam bodů terénu je ve vstupním souboru uveden jako první. První řádek souboru obsahuje číselný údaj o počtu bodů. Po tomto řádku následují samotné řádky se souřadnicemi bodů terénu. Každý bod je na samostatném řádku ve formátu `x y z` souřadnic. Program pracuje s trojrozměrným modelem, je tedy nutné uvést alespoň 3 souřadnice. Jakékoli další hodnoty na daném řádku budou ignorovány. Příkladem dat může být:

```
3061
743187.365349 1044835.875050 195.294453
743187.914376 1044833.930995 195.277962
743186.205921 1044826.939328 195.499947
...
```

Seznam trojúhelníků

Trojúhelníky jsou ve vstupním souboru uloženy podobně jako body. Seznam trojúhelníků následuje za seznamem bodů a stejně jako seznam bodů začíná údajem o počtu trojúhelníků. Na každém dalším řádku jsou tři čísla – indexy vrcholů trojúhelníků orientovaných proti směru hodinových ručiček. Body jsou indexovány od nuly a indexy ukazují do předešlého seznamu bodů. Z toho vyplývá, že pořadí bodů ve vstupním seznamu je závazné a nelze prohodit jednotlivé řádky bez příslušné úpravy trojúhelníků. Příklad dat:

```
6055
268 266 269
266 256 269
256 257 269
...
```

Seznam sousednosti trojúhelníků

Z důvodu určení okrajů oblasti, která je programem načítána, je potřeba zadat i sousednost trojúhelníků. Jako v předchozích případech je na prvním řádku počet položek v tomto seznamu. Těchto položek musí být stejně jako trojúhelníků, jelikož řádky v tomto seznamu odpovídají jejich indexům. Každý řádek udává tři hodnoty – indexy trojúhelníků sousedících s aktuálním trojúhelníkem. Pokud je místo nějakého indexu udána hodnota `-1`, znamená to, že na této hraně trojúhelník nemá žádného souseda a daná hrana je hranicí oblasti. Příklad dat:

```
6055
1 4 2959
2 0 2963
3 -1 1618
...
```

Tři hodnoty sousednosti v řádku odpovídají hranám daného trojúhelníku. Pokud je tedy trojúhelník dán vrcholy p_1, p_2, p_3, pak první hodnota sousednosti odpovídá hraně mezi body p_1 a p_2, druhá hodnota hraně mezi body p_2 a p_3 a poslední hodnota odpovídá hraně mezi body p_3 a p_1.

Seznam úseček (povinných hran triangulace)

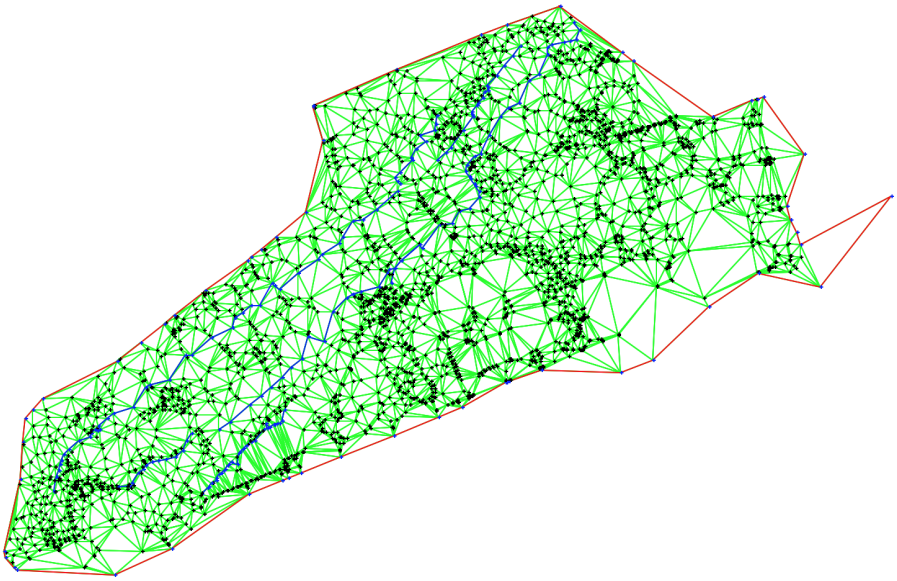
Poslední část vstupního souboru je seznam úseček – povinných hran. Tyto hrany nemusí být v triangulaci přítomny a nejsou pro běh programu nutné, jejich význam bude vysvětlen dále. První řádek opět udává počet řádků s úsečkami v souboru uložených. Následují řádky s indexy bodů, mezi kterými úsečky probíhají. Na jednom řádku může být libovolný počet bodů. Pokud tedy budou v souboru následující dva řádky:

```
1 4 8 34 387 5
908 876 39 75 1083
```

znamená to, že je zadáno 9 úseček – mezi body 1 a 4, 4 a 8, 8 a 34, 34 a 387, 387 a 5, 908 a 876, atd. Mezi posledním bodem jednoho řádku a prvním bodem řádku následujícího úsečka není (v tomto případě body na indexech 5 a 908).

3.1.2 Funkce programu

Po úspěšném načtení vstupních dat dojde k vykreslení načteného modelu na obrazovku. Vzhledem k tomu, že hlavní náplní práce s programem je posun bodů ve dvou souřadnicích (`x` a `y`), body se vykreslují jen v daných dvou dimenzích. K zobrazení dat ve 3D a ověření funkčnosti programu je potřebný jiný vizualizér podporující zobrazení ve 3D.



Obrázek 3.1.1: Ukázka grafického výstupu programu po načtení dat

V obrázku 3.1.1 jsou vidět 4 základní prvky, se kterými vytvořený program pracuje:

1. Jednotlivé body jsou vykresleny jako černé tečky.
2. Hrany trojúhelníků jsou vykresleny zelenými čarami.
3. Okraje oblasti, tedy hrany trojúhelníků, kde trojúhelníky neměly žádného souseda, jsou vykresleny červenými čarami
4. Modré čáry vyznačují zadané úsečky. Body, kterými úsečky procházejí, jsou také vykresleny modře

Přemisťování bodů

Po načtení terénu do programu je možné přemisťovat body. Pokud byly ve vstupních datech uvedeny povinné hrany, jsou body, které dané úsečky tvoří, statické. Tyto body není možné žádným způsobem přemisťovat. Význam úseček a těchto statických bodů bude vysvětlen dále.

Pro samotné přemisťování bodů v terénu byl implementován McQueenův algoritmus CVT (viz kap. 2.5.1). Tento algoritmus jsem si vybral pro implementaci ze dvou následujících důvodů:

- Jednoduchost – u tohoto algoritmu není nutné konstruovat Voroného diagram jako u Lloydova algoritmu, a je tedy jednoduší na implementaci. Přesto lze dosáhnout stejných výsledků.

- Očekávaná menší velikost vstupních dat. Vstupní data, pro která byl tento program připravován, obsahovala přibližně 3000 bodů, není proto potřeba příliš velkého množství iterací a výpočet není tak časově náročný, aby byl McQueenův algoritmus brzdou.

Standardní McQueenův algoritmus generuje body pro vytvoření CVT v konvexní obálce prostoru s body. Vzhledem k tomu, že je možné, aby byla oblast, kde se mají body přemísťovat, nekonvexní, bylo nutné algoritmus upravit. Pokud by byly body přemístěny mimo tuto oblast, nebylo by pro ně možné přepočítat `z` souřadnici. Důvodem je, jak bude vysvětleno dále, že k rekonstrukci `z` souřadnice je potřeba trojúhelníků definujících terén.

Z tohoto důvodu je nutné mít zadanou úplnou a uzavřenou hranici oblasti. Pokud by se body generovaly v konvexním polygonu definovaném vstupními body, dala by se konvexní obálka jednoduše zjistit a nebylo by potřeba hranici zadávat.

Skutečnost, že obálka prostoru je obecně nekonvexní polygon, ztížila samotný úkol generování bodů. Při generování bodů pro McQueenův algoritmus se musí dbát na to, aby každý takto vygenerovaný bod byl uvnitř nekonvexního polygonu. Toho bylo docíleno pomocí „Ray casting“ algoritmu [11].

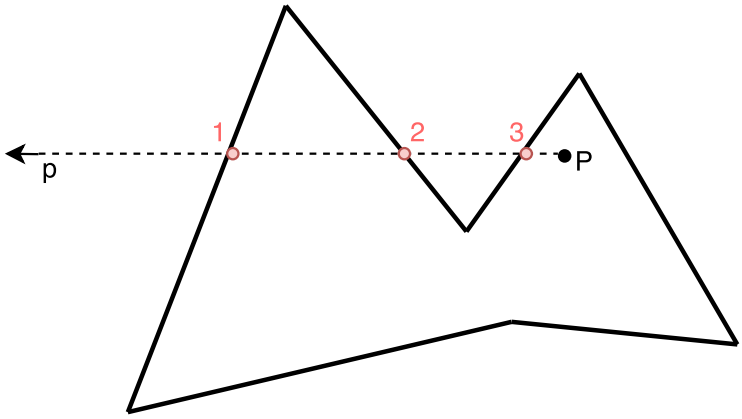
Ray casting algoritmus

Tento algoritmus je rychlý a efektivní způsob, jak určit, zda se bod nachází uvnitř obecného polygonu. Princip tohoto algoritmu je následující:

Zkoumají se průsečíky hran polygonu a polopřímky vedené z bodu P, u kterého zkoumáme, zda se uvnitř polygonu nachází, v libovolném směru.

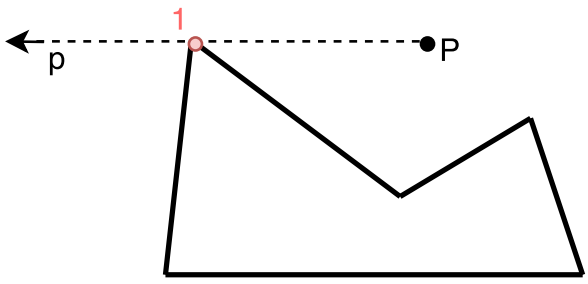
V mé implementaci jsem postupoval tím způsobem, že je vedena vodorovná polopřímka z levé hranice oblasti, kterou tvoří daný polygon, k danému bodu. V tuto chvíli je předpokládáno, že se bod uvnitř polygonu nenachází. Dále se zkoumá, zda má polopřímka průsečík s jednotlivými hranami. Každý průsečík „překlopí“ příznak, zda se bod nachází uvnitř polygonu. Tj. lichý počet průsečíků znamená, že bod je uvnitř polygonu, sudý počet průsečíků znamená, že bod uvnitř polygonu neleží.

Příkladem algoritmu je situace na obrázku 3.1.2. Polopřímka p, která je vedena z bodu P, protíná daný polygon celkem třikrát. Z toho vyplývá, že se bod P v daném polygonu nachází.



Obrázek 3.1.2: Ukázka funkce algoritmu

Tento algoritmus může způsobovat problém v situaci uvedené v obrázku 3.1.3. Pokud vedená polopřímka prochází přímo vrcholem polygonu, pak se podle definice algoritmu bod v daném polygonu nachází. Řešením tohoto problému může být vedení několika paprsků v různých směrech. V mé implementaci je problém vyřešen jednodušeji. Při určování průsečíku s jednotivými přímkami tvořící polygon dojde k detekci průsečíku i na okrajích přímek. V této situaci to znamená, že vrchol polygonu, který paprsek protíná, vygeneruje celkem dvě protnutí. To znamená, že se bod v polygonu nenachází, a potenciální problém je vyřešen.

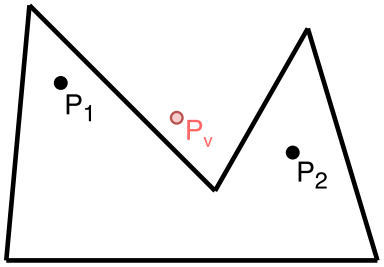


Obrázek 3.1.3: Problémovost algoritmu ray casting

Počet iterací přemisťování bodů je dopředu zadán uživatelem a poté probíhá generování bodů pro přesouvání (viz kap. 2.5.1). Vzhledem k tomu, že vygenerovaný bod nemusí být použit, počítá se za iteraci vygenerovaný bod, který splní následující podmínky:

1. Náhodně vygenerovaný bod se nachází uvnitř oblasti, ve které se body přesunují.

2. Pokud dojde k přesunu nejbližšího bodu váženým průměrováním (kap. 2.5.1), musí se i nově vzniklý bod vyskytovat v oblasti.

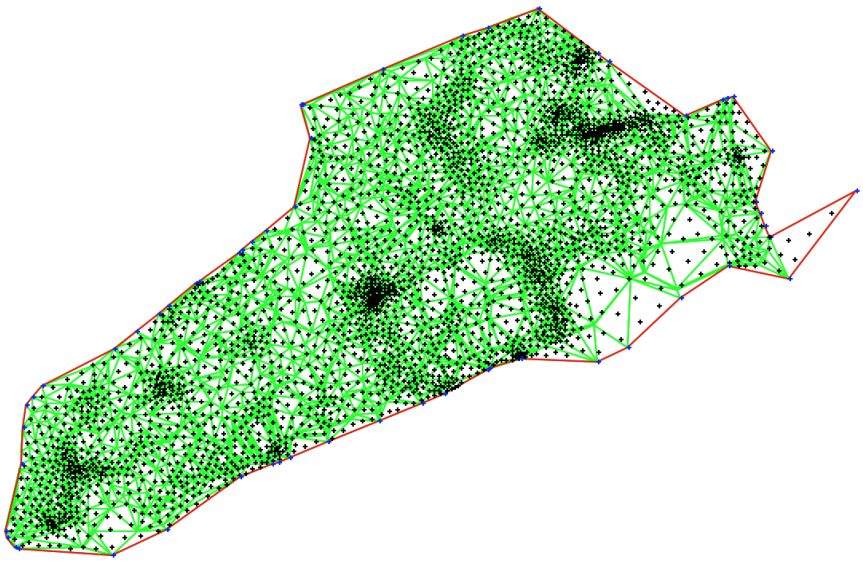


Obrázek 3.1.4: Problém vzniklý přesunem bodu

Druhou problémovou situaci ukazuje obrázek 3.1.4. Bod P_1 je náhodně vygenerovaný bod. Ten se uvnitř polygonu nachází. K němu nejbližší je bod P_2. Body se zprůměrují a výsledným bodem je bod P_v, který už ale uvnitř oblasti neleží – k této situaci nesmí dojít. Tento bod se tedy nepřesune a iterace se bude opakovat.

Po dokončení zadaného počtu iterací program vykreslí nové rozmístění bodů v terénu. Po přemístění bodů již původní triangulace neplatí, takže se trojúhelníky nemohou vykreslit v novém rozložení. Namísto toho jsou vykresleny trojúhelníky původní – pro představu, jak byly body rozloženy před přesunutím bodů.

Výsledná situace po dokončeném přesunu bodu může být viděna v obrázku 3.1.5 (načtená data jsou stejná jako v obrázku 3.1.1, kde je k vidění původní rozložení bodů). Body byly rozmístěny milionem iterací McQueenova algoritmu. V pozadí přemístěných bodů je možné vidět původní, zeleně vykreslené trojúhelníky. V pravé dolní části modelu je možné vidět původní veliké trojúhelníky. Rozprostření bodů po terénu způsobilo překrytí těchto trojúhelníků více body, což po další triangulaci bude znamenat, že si velikosti trojúhelníků budou velmi podobné.

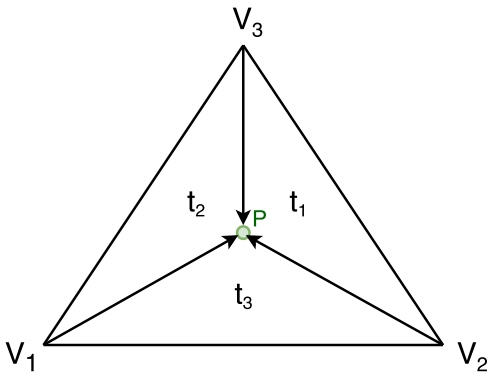


Obrázek 3.1.5: Ukázka přesunutých bodů

Samotné přemístění bodů ovšem není postačující, protože si body zachovávají svoji z souřadnici. To při jejich přesunu způsobí znehodnocení modelu terénu. Po dokončení přesouvání bodů je tedy nutné přepočítat z souřadnice jednotlivých bodů, aby odpovídaly původnímu modelu terénu.

Barycentrické souřadnice

Barycentrické souřadnice slouží k dopočtení z souřadnice bodu ležícího uvnitř trojúhelníku v prostoru [12]. Tato hodnota se dá zjistit za pomoci hodnot z souřadnic jednotlivých vrcholů daného trojúhelníku.



Obrázek 3.1.6: Princip výpočtu barycentrických souřadnic

z souřadnice bodu P se spočítá pomocí vzorce [12]:

$$z_P = w_1 \cdot z_1 + w_2 \cdot z_2 + w_3 \cdot z_3$$

Význam jednotlivých hodnot je následující:

- z_P je `z` souřadnice bodu, kterou chceme dopočítat
- z_1, z_2, z_3 jsou `z` souřadnice jednotlivých vrcholů trojúhelníku
- w_1, w_2, w_3 jsou váhy příslušné daným vrcholům. $w_1,\ w_2,\ w_3 \in [0;\ 1]$ a $w_1 + w_2 + w_3 = 1$

Váhy u `z` souřadnic jednotlivých bodů udávají, jak hodnota tohoto bodu ovlivní z_p. Tato váha se vypočítá jako podíl obsahu trojúhelníku protilehlého danému bodu, který je tvořen spojnicemi vrcholů trojúhelníku s bodem a obsahu celého trojuhelníku. Příkladem z obrázku může být váha w_1, která přísluší vrcholu V_1. Ta se spočítá jako podíl obsahu trojúhelníku t_1 a celého trojúhelníku $V_1V_2V_3$.

Z toho vyplývá, že čím dále bod leží od daného vrcholu, tím menší váha tomuto vrcholu přísluší. V krajním případě, kdy bod leží na hraně protilehlé tomuto vrcholu, je příslušná váha nulová. Poté tento bod nehraje ve výsledku žádnou roli.

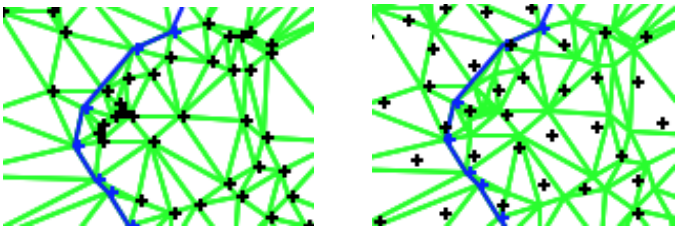
Tímto způsobem tedy program přepočítá `z` souřadnice všech bodů. V případě, že nějaký z bodů bude ležet mimo trojúhelníky definující terén, což je možné, pokud se ve vstupních datech objeví „díra“ mezi trojúhelníky, tj. pokud mezi nějakými třemi body nebude nadefinovaný trojúhelník, zůstane bodu přidělena jeho původní `z` souřadnice. Důsledkem je, že do výsledného modelu bude zanesena chyba. Proto by se takovýmto chybám mělo při vytváření vstupních dat předejít.

Doplňkovou funkcí programu je práce se zadanými povinnými hranami a statickými body. Tyto hrany, které procházejí body, jež byly zadány, slouží ke dvěma účelům.

Může se stát, že terén bude obsahovat hrany, které by uživatel rád zachoval a nebylo by vhodné jejich vrcholy přemístit. Důvodem může být důležitost daných hran pro danou aplikaci a posunutí jejich vrcholů a přepočítání `z` souřadnic by mohlo vést ke ztrátě informace o těchto hranách. To je prvním důvodem, proč byly tyto úsečky do programu zavedeny. Slouží k označení hran, které se v terénu vyskytují, a jejichž vrcholy nebude možné přemístit.

Tato funkce je vidět v obrázku 3.1.7, kde jsou statické body vykresleny modře a probíhá jimi modrá čára. Obrázek vlevo ukazuje situaci před

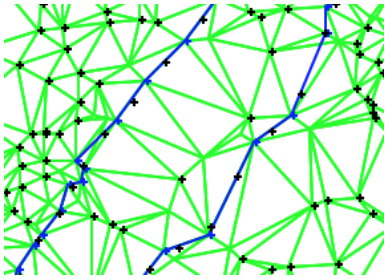
přemisťováním bodů. Na obrázku vpravo je možno vidět, že statické body nebyly přesunuty a zachovaly si svou původní pozici.



Obrázek 3.1.7: Ukázka zachování polohy bodů

Druhou možností využití úseček je možnost nechat okolní body se po nich rozprostřít. Toho je docíleno upravením McQueenova algoritmu tak, že se body, které se náhodně umísťují, budou generovat na daných úsečkách. Tyto body se rovnoměrně generují po celé délce zadaných úseček.

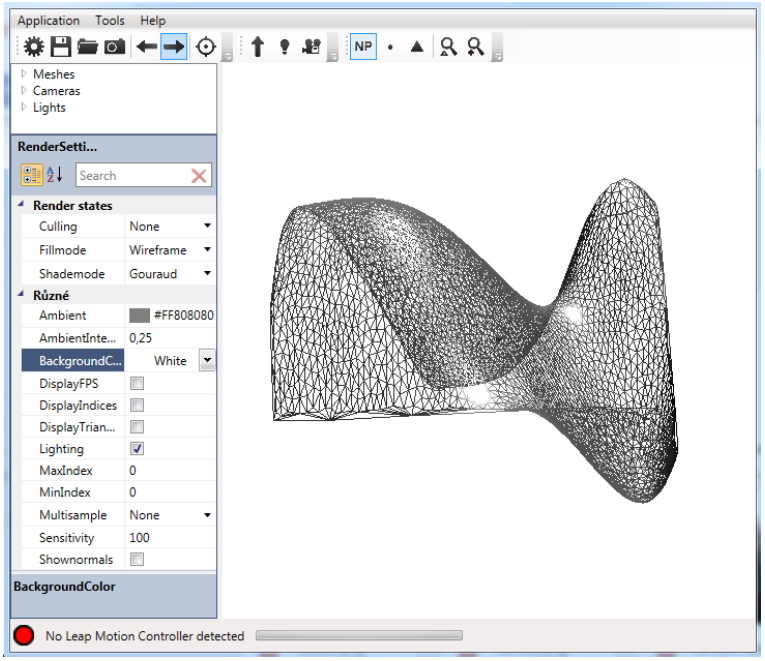
Výsledek takového přesunu je možné vidět na obrázku 3.1.8. Na modře vykreslených úsečkách se rovnoměrně rozprostřely okolní body terénu. Toho lze využít například při zkvalitnění triangulace kolem hran.



Obrázek 3.1.8: Přesun bodů na úsečku

3.1.3 Vizualizace modelu terénu

Jak již bylo zmíněno, mnou vytvořený program nepodporuje trojrozměrné zobrazení terénu. K tomu je potřeba jiný vizualizér. Veškeré trojrozměrné vizualizace, které budou použity, byly vytvořeny programem, který poskytl Doc. Ing. Libor Váša, Ph.D. (ukázka na obr. 3.1.9). Program pracuje s různými vstupními formáty dat, nicméně formát, který používá můj program a který byl popsán v kapitole 3.1.1, podporován není. Bylo tedy potřeba provést konverzi do jiného, podporovaného formátu.



Obrázek 3.1.9: Ukázka 3D vizualizačního programu

Wavefront OBJ formát

Wavefront .obj soubor je textový formát popisu trojrozměrných modelů [13]. Svou strukturou byl velmi podobný vstupním datům, které používá můj program, proto jsem si vybral právě tento. Formát podporuje širokou škálu prvků, které mohou být popsány, pro účely práce však byly důležité jen dva:

1. **Vrcholy**, které tvoří daný model. V souboru jsou uloženy ve formátu:

   $$\texttt{v}\ x\ y\ z$$

   Písmeno `V` značí, že se jedná o vrchol (anglicky Vertex). Za tímto znakem následují souřadnice daného vrcholu.

2. **Stěny** modelu tvořené jednotlivými vrcholu. Jsou uloženy ve formátu:

   $$\texttt{f}\ v_1\ v_2\ v_3, \ldots$$

   Písmeno `F` značí, že se jedná o stěnu (anglicky Face). Poté následují indexy jednotlivých bodů, které danou stěnu tvoří a byly v souboru uvedeny. Těch může být libovolný počet, musí však být minimálně tři, což tvoří základní prvek stavby modelu – trojúhelník. Body jsou indexovány od jedné, ne od nuly, jako tomu je ve vstupním formátu pro mnou vytvořený program.

Vedle samotné práce jsem tedy vytvořil program, který konvertuje data popsaná v kapitole 3.1.1 do .obj formátu. K tomuto účelu je potřeba pouze seznamu bodů a trojúhelníků další seznamy jsou ignorovány.

3.1.4 Výstupní data

Po dokončení práce s vytvořeným programem je možné uložit výstup programu. Vzhledem k tomu, že po přemístění bodů není původní triangulace platná (body se mohou navzájem promíchat), je jediným výstupem seznam bodů. Ten je, jako tomu bylo ve vstupních datech, uložen do textového souboru. První řádek tohoto souboru obsahuje údaj o počtu bodů, každý další řádek poté obsahuje jednotlivé, mezerou oddělené souřadnice daných bodů.

3.2 Experimenty a výsledky k tématu CVT

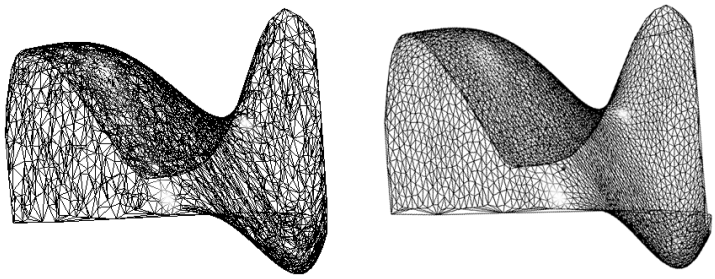
Experimenty s programem byly prováděny na následujících dvou počítačích:

1. MacBook Pro, macOS Sierra, Intel Core i7 2,2 GHz (4 jádra), 16 GB RAM
2. PC, Windows 7 Professional, Intel Xeon X3360 2,83 GHz (4 jádra), 8 GB RAM

Nejprve bylo potřeba ověřit, že vytvořený program funguje správně. Přesouvání bodů bylo možné vidět v programu, který body přesouval, bylo ovšem potřeba ověřit správnost přepočítání `z` souřadnice. Toho bylo docíleno ručním přepočítáním a ověřením s výstupem programu.

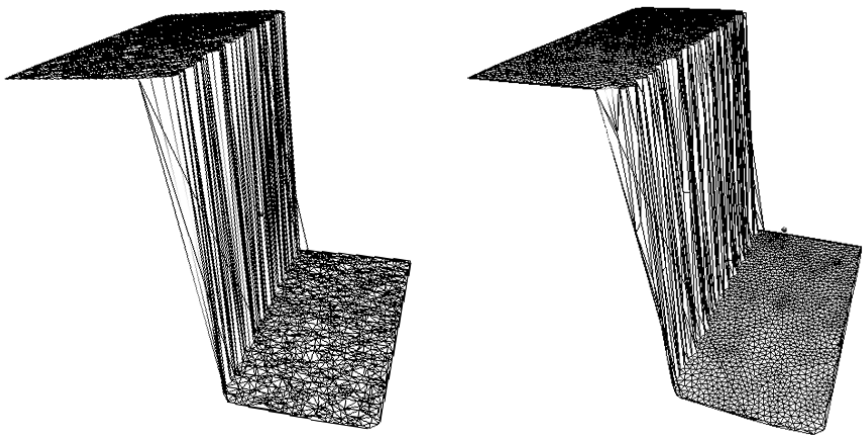
Výsledek přepočtu je demonstrován na sadě umělých testovacích dat.

První ukázka testovacích dat je vidět na obrázku 3.2.1. Jedná se o testovací data ve tvaru „sedla“ o počtu 3000 bodů. Vlevo je původní model, vpravo model programem upravený. Je vidět, že došlo ke zrovnoměrnění rozložení vrcholů triangulace a samotná triangulace modelu vypadá lépe než na modelu původním, protože trojúhelníky jsou tvarově blíže k rovnostranným. Zůstal také zachován tvar oblasti, což demonstruje správnost přepočtu `z` souřadnice.



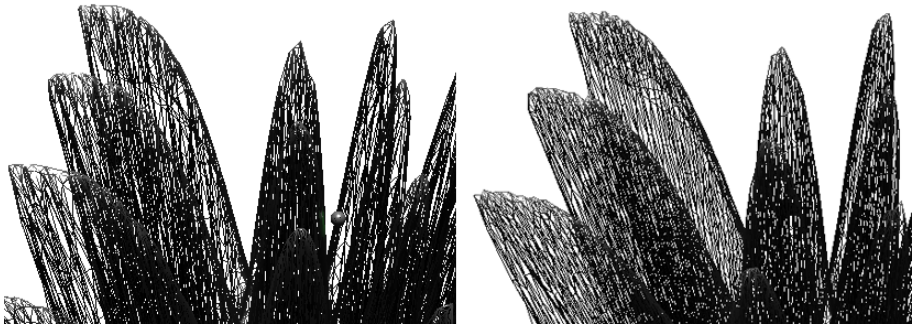
Obrázek 3.2.1: První ukázka testování

Druhá testovací data, která je možné vidět na obrázku 3.2.2, obsahují veliký schod, který je složen ze 3000 bodů. Stejně jako u prvních testovacích dat je možné vidět zrovnoměrnění trojúhelníkové sítě na modelu po úpravě (obrázek vpravo).



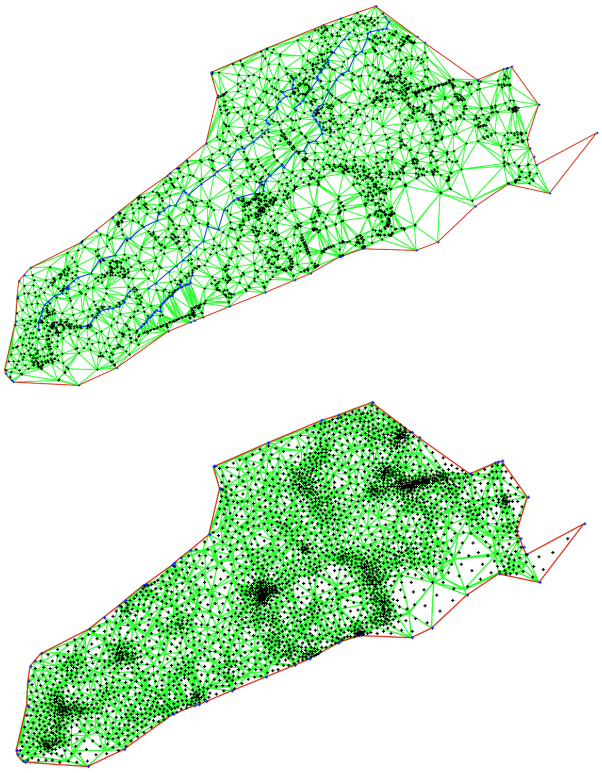
Obrázek 3.2.2: Druhá ukázka testování

Poslední testovací data je možné vidět v obrázku 3.2.3. Tento model obsahoval 30000 bodů, z toho důvodu je na obrázku hůře viditelná trojúhelníková síť.



Obrázek 3.2.3: Výřez třetí sady dat

Další experiment demonstruje vliv bodů na kvalitu vrstevnic u reálných dat. Testování zlepšení kvality vrstevnic po úpravě modelu terénu bylo prováděno na modelu botanické zahrady (data získána z PřF UK Praha). Původní a upravená data je možné vidět na obrázku 3.2.4 Z rozhodnutí vedoucí práce bylo ověření zlepšení vrstevnic provedeno pouze vizuálně. Výsledné triangulace upravených modelů a jednotlivé vizualizace generovaných vrstevnic byly poskytnuty vedoucí práce.



Obrázek 3.2.4: Skutečná data – model botanické zahrady

Vygenerované vrstevnice na Delaunayově triangulaci původních, neupra-

vených bodů je možné vidět v obrázku 3.2.5. Ve vrstevnicích jsou vidět artefakty, které zhoršují jejich kvalitu. Tyto artefakty jsou například malé ostrůvky mezi jednotlivými čárami, smyčky nebo ostré výběžky na vrstevnicích.

Obrázek 3.2.5: Vrstevnice původního modelu

V modelu terénu se místy vyskytují hrany, které jsou součástí schodovitého terénu v horní části modelu. Tyto hrany byly dohledány a byla vytvořena CDT, kde byly tyto hrany do triangulace zařazeny jako hrany povinné. Výsledkem generování souřadnic potom byl obrázek 3.2.6.

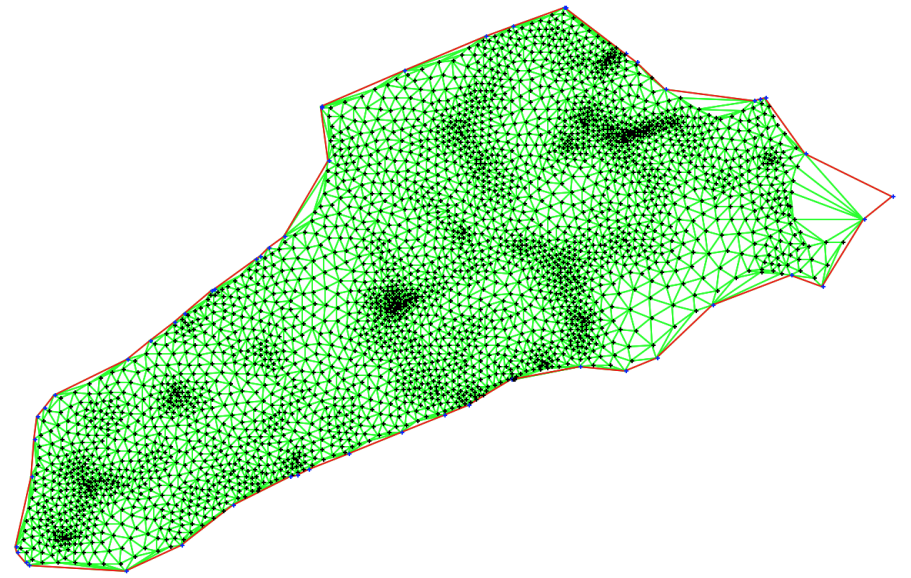
Obrázek 3.2.6: Vrstevnice původního modelu triangulovaného pomocí CDT

Z porovnání obou obrázků je patrné, že samotné zavedení povinných hran

sledujících „schody“ v terénu do Delaunayovy triangulace nepřineslo žádné zlepšení.

Tyto dva obrázky jsou tedy předlohou pro experimenty s CVT a snahou bylo zjistit, zda a jak výrazně se zlepší výsledné vrstevnice po úpravě terénu pomocí CVT.

Prvním experimentem bylo upravení terénu pomocí dvou milionů iterací CVT. Výsledný model, který byl po úpravě triangulován pomocí Delaunayovy triangulace, je možné vidět na obrázku 3.2.7.



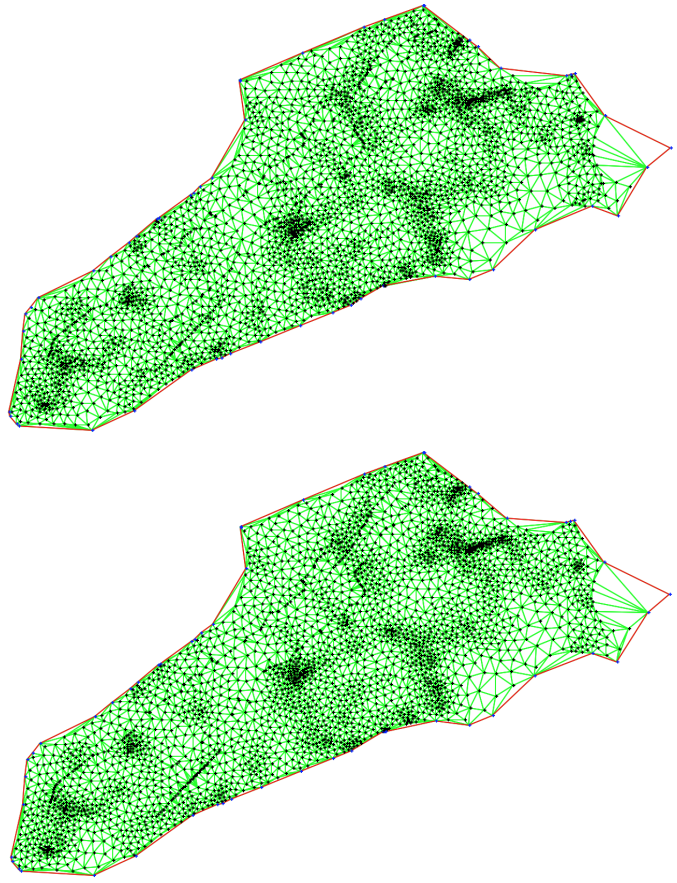
Obrázek 3.2.7: Model terénu upravený pomocí CVT

Výsledné vrstevnice, které byly získány z tohoto modelu, je možné vidět na obrázku 3.2.8. Je patrné, že došlo ke značnému zlepšení oproti původnímu modelu (srovnej obr. 3.2.5). Vrstevnice ve svahu (část obrázku, kde jsou vrstevnice hustější) byly narovnány a nejsou v nich patrné tak velké výkyvy, jako u původního modelu. V pravé, pozvolnější části terénu je také možné sledovat zlepšení – značná část malých ostrůvků a nežádoucích ostrých špiček zmizela.

Obrázek 3.2.8: Vygenerované vrstevnice z upraveného terénu

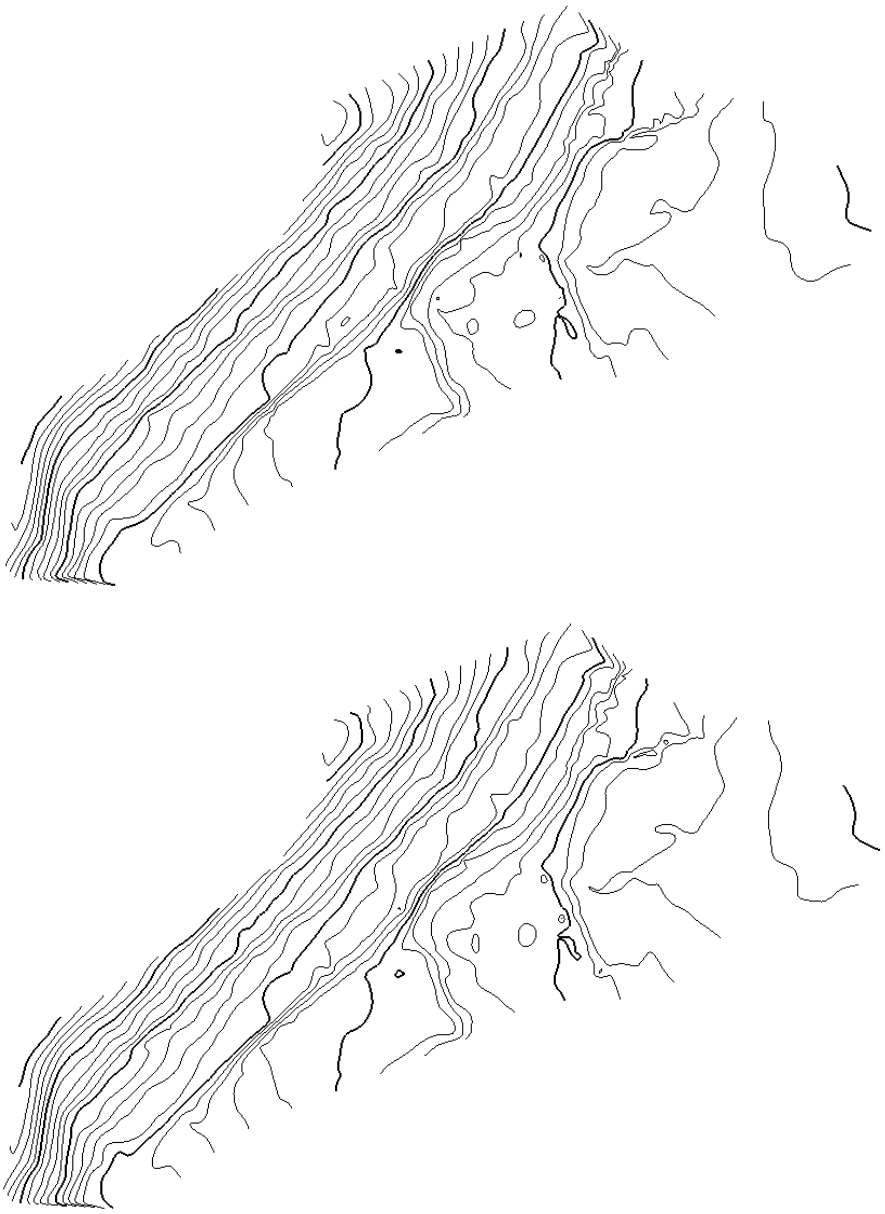
Úmyslem dalších experimentů bylo ověření, zda přidání povinných hran v do modelu nepřinese větší zlepšení. Protože šlo o pilotní experiment, na doporučení vedoucí práce bylo hledání hran v terénu provedeno manuálně.

Nalezené hrany jsou patrné z obrázku 3.2.9. S povinnými hranami byly provedeny 2 experimenty. Tyto dva experimenty jsou velmi podobné, jediný rozdíl je v tom, že ve druhém experimentu (spodní obrázek) bylo „více naléháno“ na okolní body přemístit se na dané hrany.



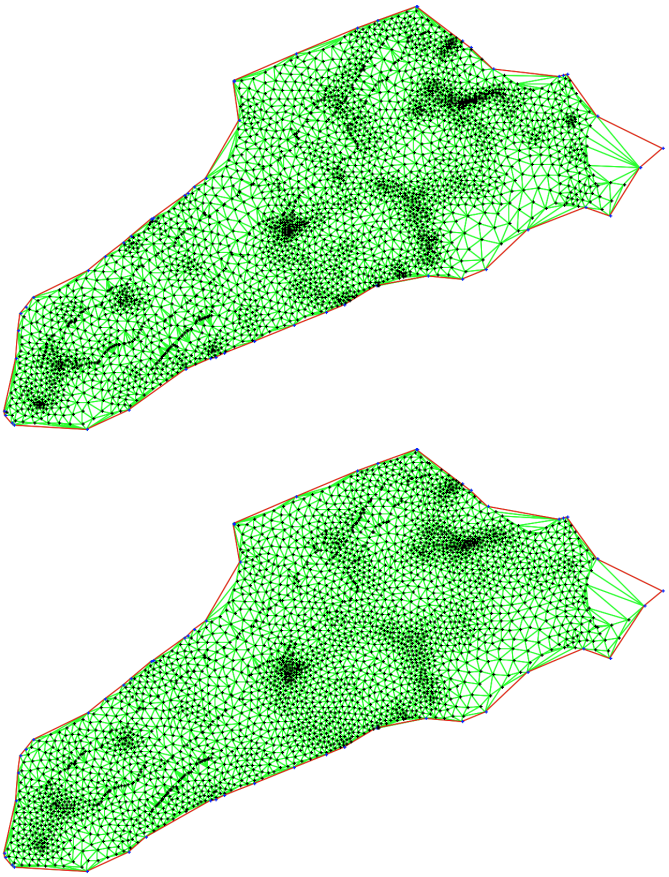
Obrázek 3.2.9: Upravený terén pomocí CVT s povinnými hranami

V obrázku 3.2.10 jsou zobrazené výsledné vrstevnice daných dvou pokusů. V porovnání s pokusem na obrázku 3.2.8 je patrné, že zakomponování hran do vstupu nepřineslo pozorovatelné zlepšení. V těchto dvou pokusech byly použity hrany v terénu, které byly nejvýraznější. Proto byly dohledány další hrany, méně výrazné a byly do vstupních dat také přidány.



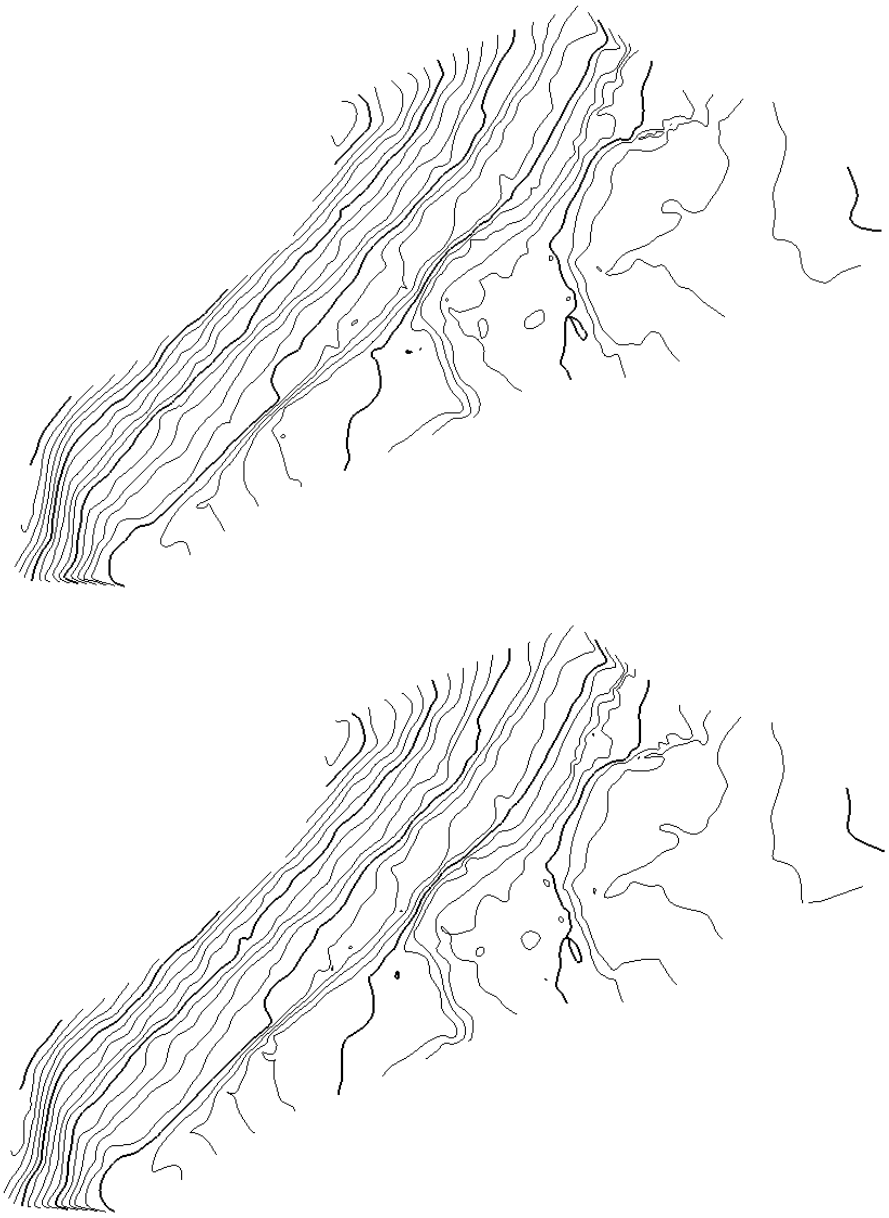
Obrázek 3.2.10: Vygenerované vrstevnice z upravených terénů s hranami

V obrázku 3.2.11 jsou vidět provedené experimenty s více hranami. Vstupním souborem byla data, která je možné vidět na obrázku 3.1.1 na začátku kapitoly 3.1.2. Stejně jako u předchozích dvou pokusů, liší se tyto dva „úrovní naléhaní“ na přesun na hrany – tedy kolik iterací přemístění bodů na hrany bylo provedeno. Tento druhý pokus byl navíc proveden vyšším množstvím iterací CVT (kolem 4 milionů), kdy už změny v pozicích bodů byly minimální (pod 0,001 jednotky vzdálenosti).



Obrázek 3.2.11: Upravený terén pomocí CVT s více hranami

Jak je vidět na obrázku 3.2.12, ani přidání dalších hran nepřineslo výrazné zlepšení oproti předchozím pokusům. Z toho vyplývá, že zachování pozic bodů na hraně, či nucení okolních bodů se na danou hranu přesunout nepřináší zlepšení oproti čistému přesunu bodů pomocí CVT.



Obrázek 3.2.12: Vygenerované vrstevnice z upravených terénů s více hranami

Závěrem těchto experimentů tedy je, že úprava terénu pomocí CVT přináší zlepšení generovaných vrstevnic. Triangulace provedené z upravených dat vedly k lepším výsledkům, než Delaunayho triangulace s vnucenými hranami dat původních. Vnucení těchto hran do upravování terénu pomocí CVT nepřináší pozorovatelné zlepšení kvality vrstevnic a proto nemusí být prováděno.

3.3 Shlukování

Tato část práce byla vypracovávána jako součást interního projektu KIV. Úkolem bylo vypracovat implementace algoritmů K-means a C-means a porovnat je s již existující implementací algoritmu *local search*, který řeší tzv. *facility location* problém. Ten spočívá v tom, že pro danou cenu za vytvoření centra shluku a cenu za jednotku vzdálenosti mezi centrem a bodem k němu příslušejícím, najde optimální počet a rozmístění shluků.

Vše bylo vypracováváno jako součást již existujícího programu autora Ing. Onřeje Kaase a byl implementován v jazyce C#.

3.3.1 Vstupní data

Vstupní data tohoto programu byla jednodušší, než v předchozím případě. Vstup sestával z pouhého seznamu bodů. Tyto body ovšem mohly mít libovolný počet dimenzí.

Na prvním řádku souboru se vyskytují dva, mezerou oddělené, číselné údaje. První číslo udává počet bodů, které jsou v souboru uložené. Druhé číslo udává počet dimenzí.

Od druhého řádku začíná samotný seznam bodů. Každý řádek představuje souřadnice jednoho bodu a musí na něm být tolik mezerou oddělených čísel, kolik bylo uvedeno v prvním řádku. Příkladem dat může být:

```
1519 4
-0.70740 -0.82523 -0.05 2.08
-0.76430 0.946348 -0.11 0.12
 0.28603 -0.54246 -0.69 1.84
```

Pro experimenty se shlukovacími algoritmy byla k dispozici reálná data bodů v terénu, byla také ovšem připravena data umělá.

3.3.2 Hodnocení vytvořeného řešení

Vizuální porovnání výsledných experimentů nebylo v tomto případě dostačující. Pouhý pohled na rozložení shluků nenapoví, jaká je výsledná cena shlukování. Aby bylo možné porovnávat jednotlivé algoritmy, bylo potřeba určit, jak tuto cenu pro jednotlivá řešení spočítat.

První složkou tohoto hodnocení je součet vzdáleností jednotlivých bodů k jim příslušejícím centrům. Příliš dlouhé vzdálenosti bodů k centrům jsou nežádoucí. Je lepší, když se vytvoří více shluků s body blízko u nich, než aby bylo center v prostoru málo a vzdálenosti mezi body a centry byly veliké.

Tento přístup ovšem vede k tomu, že se za ideální řešení bude považovat, když každý prvek vstupní množiny bodů bude označen za centrum shluku. V tom případě bude celkový součet vzdáleností roven nule – bude vytvořeno řešení, jehož ohodnocení nelze překonat. Z tohoto důvodu bylo do hodnocení vytvořeného řešení nutné zavést i cenu za vytvoření centra shluku. Výsledná cena řešení se tedy spočítá jako součet cen všech center a všech vzdáleností mezi jednotlivými body a centry shluků, ke kterým přísluší. Vzorec pro výpočet ceny řešení tedy vypadá následovně:

$$C_s = n_c \cdot C_c + \sum_{i=1}^{n_p} \|p_i - c_{pi}\|$$

C_s je výsledná cena řešení, n_c je počet center shluků, C_c je cena za vytvoření jednoho shluku, n_p počet bodů a $\|p_i - c_{pi}\|$ je míra vzdálenosti mezi bodem a jemu příslušejícím centrem.

Výslednou cenu, a tedy i kvalitu, samozřejmě ovlivní cena za vytvoření centra. Příliš vysoká cena povede k řešení, kde bude méně shluků (v krajním případě jen jeden), příliš nízká cena zase povede k řešení, kdy se vyplatí pro každý bod vytvořit samostatný shluk. V reálném případě, kdy se určuje, jak například rozmístit sklady tak, aby byla distribuce zboží co nejefektivnější, je cena za otevření centra daná.

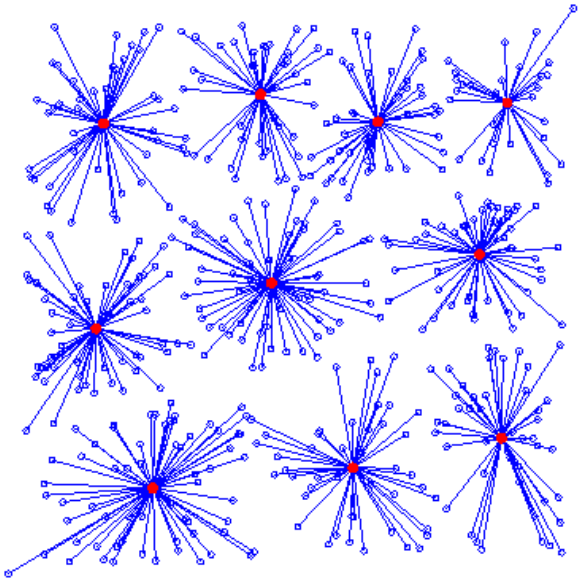
Tyto aplikace odráží vlastnosti již zmíněného *facility location* problému. Nyní popíšeme experimenty s algoritmy K-Means, C-Means a local search. Algoritmy C-means a K-means mají pevně daný počet shluků, při hledání optimálního řešení je tedy nutné algoritmus provádět několikrát pro různé počty shluků. Algoritmus local search s cenou za vytvoření shluku již při výpočtu počítá. Vstupem tohoto algoritmu není počet shluků, jen cena za vytvoření centra – počet shluků se v algoritmu určí tak, aby řešení bylo co nejblíže optimálnímu řešení.

Nejprve si ukážeme typické výsledky jednotlivých algoritmů pro různá vstupní data. Veškeré experimenty probíhaly na počítači následující konfigurace: operační systém Windows 7 Professional, Intel Xeon X3360 2,83 GHz (4 jádra), 8 GB RAM

3.3.3 K-means

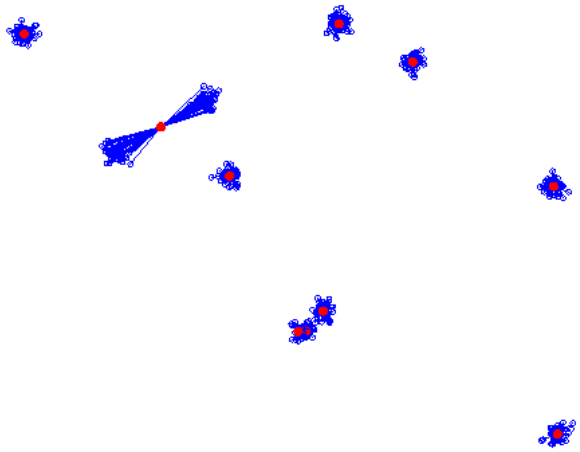
Jak již bylo rozebráno v kapitole 2.4.1, K-means rozdělí vstupní množinu do předem daného počtu shluků. Tento počet shluků je tedy vstupním parametrem algoritmu a kromě vstupních bodů algoritmus žádné jiné vstupy nevyžaduje. Úkolem je najít optimální počet shluků.

Na obrázku 3.3.1 je možné vidět ukázku shlukování 500 uniformně rozložených bodů ve dvourozměrném prostoru do deseti shluků. Vzhledem k tomu, že byla vstupní množina bodů rozložena uniformně, je možné pozorovat i přibližné uniformní rozložení shluků. Rychlost, s jakou algoritmus konverguje k výslednému řešení, je poměrně vysoká – stačilo v průměru 15 až 20 iterací pro 500 vstupních bodů. Rychlost konvergence závisí na zvolení počátečních center.



Obrázek 3.3.1: Výsledek shlukování algoritmu K-means

Počáteční rozložení neovlivní jen rychlost konvergence algoritmu. Jak bylo zmíněno v kapitole 2.4.1, počáteční rozložení bodů ovlivní i kvalitu výsledného řešení. Příkladem může být situace na obrázku 3.3.2. Vstupními daty bylo 500 bodů, tentokrát rozdělených do deseti oblastí. Data byla vytvořena tak, že se náhodně zvolilo deset bodů prostoru, které sloužily jako centra pro gaussovsky rozmístěných 50 bodů.



Obrázek 3.3.2: Problém počtečního rozložení center shluků

Tato data jsou na první pohled rozdělena do deseti shluků. Očekávaným (a také ideálním) výsledkem shlukování do deseti shluků by tedy bylo vytvoření shluků obsahujících každý právě jednu skupinu bodů ve vstupních datech.

Na obrázku je možné vidět, že tomu tak ve výsledku není. V jedné skupině bodů (na obrázku vlevo dole, jeden ze dvou navzájem blízkých shluků) došlo k vytvoření dvou shluků, což muselo být vynahrazeno jinde, kdy jedno centrum připadá na na dvě skupiny bodů.

Pro nalezení lepšího řešení je tedy nutné algoritmus opakovat s různými výběry počátečních center.

3.3.4 C-means

C-means, stejně jako K-means, rozděluje prvky do předem daného počtu shluků. Narozdíl od K-means přísluší každý prvek do každého shluku, kdy příslušnost je podmíněna váhou, se kterou do daného shluku prvek patří.

Oproti K-means jsou vstupem tohoto algoritmu další dvě hodnoty:

1. Hodnota m, určující rychlost klesání příslušnosti ke shluku s rostoucí vzdáleností od centra. Tato hodnota byla popsána v kapitole 2.4.2.

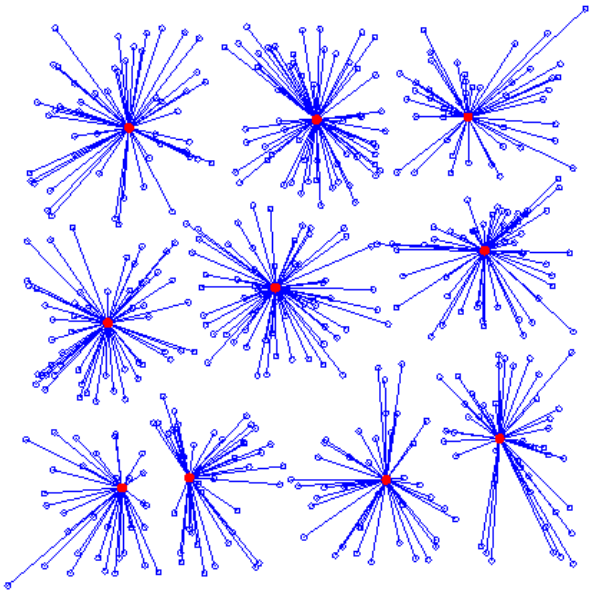
2. Práh p, který umožňuje určit, jakou minimální váhu příslušnosti k centru shluku má prvek mít, aby byl započítán do výpočtu těžiště shluku.

Hodnota prahu byla přidána, jelikož by v některých aplikacích mohla mít

přínos.

Algoritmus C-Means, jak již bylo řečeno, spočívá v principu, že každý bod přísluší do každého shluku. Pro účely aplikace a výsledné vizualizace rozložení do shluků bylo potřeba určit příslušnost do právě jednoho shluku. Toho bylo docíleno výběrem centra, ke kterému měl daný bod největší hodnotu příslušnosti.

Výsledek shlukování pomocí algoritmu C-means je možné vidět na obrázku 3.3.3. Práh byl v tomto případě nastaven na hodnotu 0, aby byly započítávány všechny body. Hodnota m byla nastavena na 10.



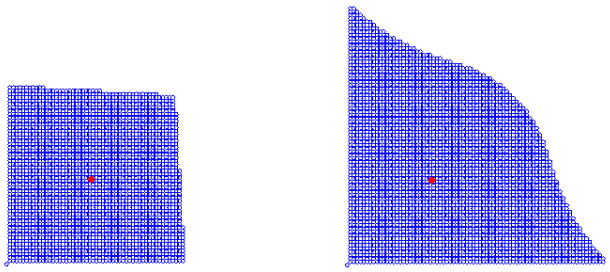
Obrázek 3.3.3: Výsledek shlukování algoritmu C-Means

Je vidět, že výsledek je značně podobný výsledku algoritmu K-means. V tomto případě bylo ovšem potřeba desetinásobného počtu iterací, a to v průměru 150. Počet iterací v tomto případě totiž závisí na největší velikosti změny ve váhové matici v dané iteraci. Algoritmus se opakuje, dokud je tato změna větší, než dopředu daná hodnota ε. V tomto případě byla hodnota ε nastavena na 0.001.

V této aplikaci se bod přiřadí k tomu shluku, ke kterému má největší váhu příslušnosti. V případě, že by se zkoumaly jednotlivé shluky, bylo by možné zařazení do daných shluků provést ne na základě nejvyšší hodnoty

příslušnosti, ale na základě nenulové hodnoty příslušnosti větší než je daná zadaná hodnota. Toto je výhoda algoritmu C-Means oproti K-Means.

Podobná situace je zobrazena na obrázku 3.3.4, kdy bylo prováděno testování tohoto přístupu na datech, která sestávala z uniformně rozložených bodů v prostoru. Mimo první dvě souřadnice však data obsahovala další dvě – vektor, který mířil ve směru od středu ven. Ve středu prostoru byla velikost těchto vektorů nulová a s rostoucí vzdáleností bodů od středu rostla i velikost tohoto vektoru. Obrázek vlevo demonstruje zobrazení bodů, které přísluší do jednoho vybraného shluku s vyšší váhou, obrázek vpravo zase ukazuje příslušnost, kdy váha pro přiřazení do daného shluku nabývala menší hodnoty.



Obrázek 3.3.4: Zařazení bodů do shluku s různými prahy hodnot příslušnosti

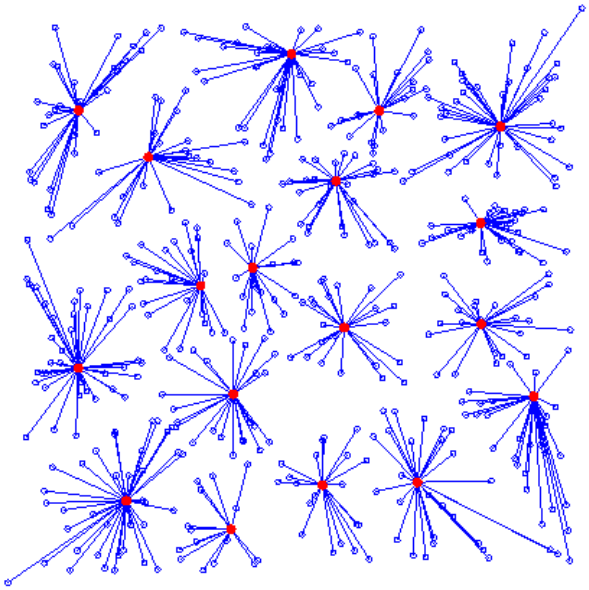
3.3.5 Local search

Tento algoritmus byl v projektu již naimplementovaný jako prostředek řešení *facility location* problému.

Jediným vstupním parametrem tohoto algoritmu je tedy multiplikátor ceny za vytvoření centra shluku (C_m). Výsledná cena za vytvoření shluku se poté určí jako hodnota multiplikátoru ceny vynásobené velikostí diagonály ohraničujícího obdélníku oblasti, kterou určuje vstupní množina bodů.

Při určování ceny za vytvoření shluku je totiž nutné brát v potaz velikost prostoru – je nutné cenu přizpůsobit vzdálenostem bodů od shluků, aby vzdálenosti značně nepřevýšily cenu vytvoření shluků (to by vedlo k příliš mnoho vytvořeným shlukům), nebo aby naopak nebyla cena značně větší – to by v extrémním případě vedlo k jednomu vytvořenému shluku. Pro představu, uniformní vstupní data použitá v obrázku 3.3.5 byla generována ve čtverci o hraně velikosti 2.

Počet vytvořených shluků je součástí výstupu algoritmu a závisí na vlastnostech vstupní množiny bodů a na ceně za vytvoření nového shluku.

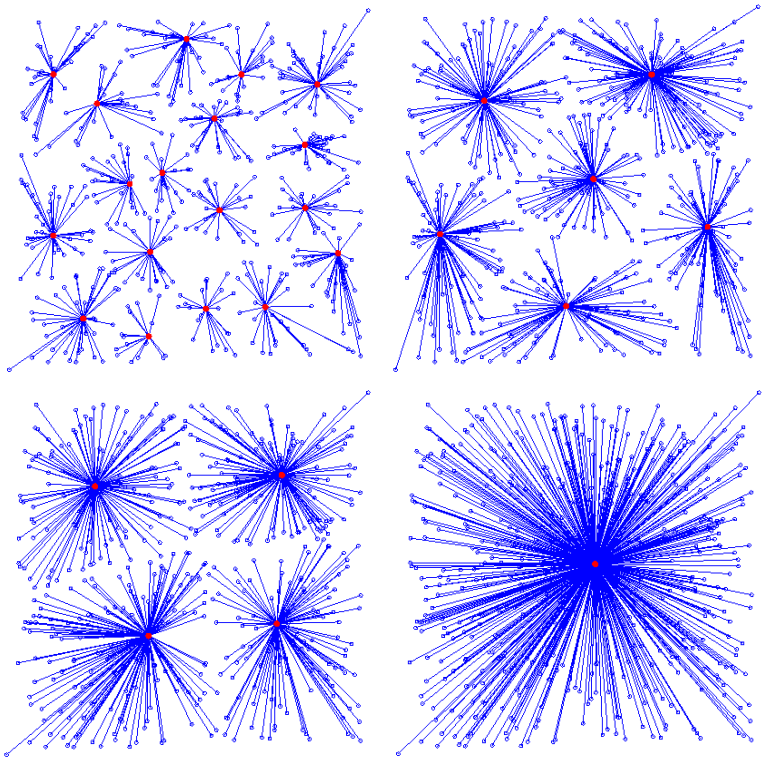


Obrázek 3.3.5: Výsledek shlukování algoritmu local search

Na obrázku je možné vidět výsledek shlukování na množině uniformně rozložených bodů. Jedná se o stejná testovací data jako v předchozích případech. Tento výsledek byl vytvořen s multiplikátorem ceny za vytvoření centra o hodnotě 1 a algoritmus vstupní množinu bodů rozdělil do 18 shluků.

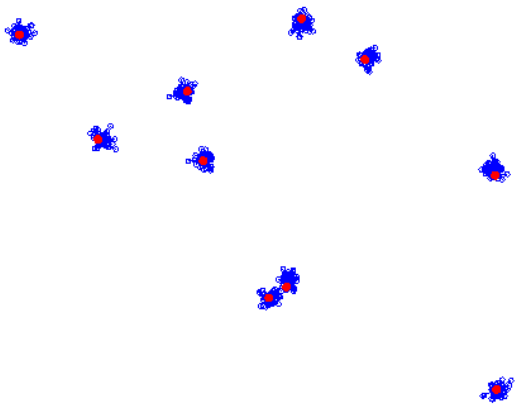
Na obrázku 3.3.6 je možné vidět výsledky shlukování pro různé hodnoty multiplikátoru ceny za vytvoření center. Vlevo nahoře je výsledek pro hodnotu multiplikátoru 2. Toto řešní se příliš neliší od hodnoty rovné 1, došlo ovšem k menšímu snížení počtu shluků. Vpravo nahoře je výsledek pro hodnotu 5, kdy už je možné pozorovat počet shluků značně menší – stejně jako vlevo dole, kde je výsledek pro hodnotu 10.

Při hodnotě 30, jejíž výsledek je vidět vpravo dole, došlo k vygenerování řešení s jedním jediným shlukem. V tomto případě je vytvoření jediného centra uprostřed výhodnější, než vytvoření druhého sluku, právě kvůli převyšující ceně za vytvoření centra.



Obrázek 3.3.6: Výsledeky shlukování pro různé váhy

Na obrázku 3.3.7 je vidět výsledek shlukování pro stejná shlukovaná data jako v předchozím případě. Shlukování proběhlo s hodnotou multiplikátoru ceny vytvoření centra o hodnotě 1. Narozdíl od předchozích algoritmů, local search nemá problém s počátečním rozložením center a výsledek shlukování dopadl tak, jak bylo očekáváno (pro tuto hodnotu ceny). V tomto ohledu je tedy daný algoritmus lepší, než předchozí dva.



Obrázek 3.3.7: Výsledeky shlukování pro skupiny bodů

3.4 Hledání optimálního nastavení shlukovacích algoritmů

Nyní ukážeme výsledky experimentů hledajících optimální nastavení parametrů jednotlivých algoritmů. Z důvodu problematiky počátečního rozložení u algoritmů K-Means a C-Means byly tyto experimenty prováděny opakovaně a jednotlivá ohodnocení byla zprůměrována.

Vzhledem k množství vstupních parametrů algoritmů a možnostem, které lze prozkoumat, bylo potřeba vytvořit systém iterátorů, které budou hledat optimální nastavení jednotlivých algoritmů. Výsledek shlukování závisí samozřejmě i na povaze vstupních dat, cílem těchto iterátorů bylo tedy zjistit, jaká nastavení algoritmů poskytnou nejlepší řešení shlukování (tedy C_s bude co nejmenší) pro dané množiny vstupních dat.

Pro algoritmy, u kterých bylo potřeba nastavit jediný parametr, bylo hledání jednoduché. V tomto případě se jednalo o K-means a local search algoritmy. U K-means byl jediným parametrem počet shluků a proto stačilo prozkoumat výsledky shlukování pro různé počty shluků. Pro local search byl zase parametrem multiplikátor ceny za vytvoření centra (C_m, viz kap. 3.3.5). Stejně jako u algoritmu K-means byla zkoumána různá nastavení pro hodnotu multiplikátoru.

Porovnávání algoritmů probíhalo na několika sadách dat:

- Uniformní data (500 uniformně rozložených bodů)
- Shlukovaná data (500 bodů rozmístěných do deseti shluků)
- Tři sady reálných dat - body v terénu. První data obsahovala 154 bodů, druhá 729 bodů a třetí 1519 bodů.

Umělá data vygeneroval Ing. Ondřej Kaas, reálná data byla poskytnuta PřF UK praha.

3.4.1 Výsledky K-means

Na obrázku 3.4.1 jsou vidět průměrné hodnoty C_s na 500 uniformně rozložených bodech. Iterátor řešení zkoumal cenu pro počet shluků rovný čtyřem až třiceti. V grafu je možné vidět, že zpočátku cena řešení klesala, až došla do svého minima. Poté začala cena zase růst – počet shluků byl příliš veliký a načítala se za ně cena. Výsledkem je, že optimálním počtem shluků pro

tato data je 16, kdy hodnota C_s získaná algoritmem dosahovala hodnoty 133,21.

Obrázek 3.4.1: K-Means - graf závislosti C_s na počtu shluků

Na obrázku 3.4.2 je vidět graf pro shlukovaná data. Nejlepšího výsledku algoritmus dosáhl pro 16 shluků, kdy byla průměrná hodnota C_s rovna 35,44.

Obrázek 3.4.2: K-Means - graf závislosti u shlukovaných dat

Obrázek 3.4.3 ukazuje graf výsledků pro první reálná data. Nejlepšího řešení algoritmus dosáhl pro 7 shluků, kdy byla přůměrná hodnota C_s rovna 71763,58.

Obrázek 3.4.3: K-Means - graf závislosti u prvních reálných dat

Pro druhá reálná data dosáhl algoritmus nejlepšího hodnocení 434969,52 pro 17 shluků. Graf výsledků je možné vidět na obrázku 3.4.4.

Obrázek 3.4.4: K-Means - graf závislosti u druhých reálných dat

Výsledek u třetích reálných dat byl odlišný než u předchozích (obrázek 3.4.5). V tomto případě průměrná C_s klesala s rostoucím počtem shluků až do nastavení 20 shluků, kdy byla průměrná cena řešení rovna 825848,62. Poté hodnocení začlo znovu růst.

Obrázek 3.4.5: K-Means - graf závislosti u třetích reálných dat

3.4.2 Výsledky C-Means

Iterování nastavení algoritmu C-Means je značně složitější, než u algoritmu K-Means. Kromě samotného počtu shluků se iteruje i přes hodnotu prahu a hodnoty m (viz kap. 2.4.2).

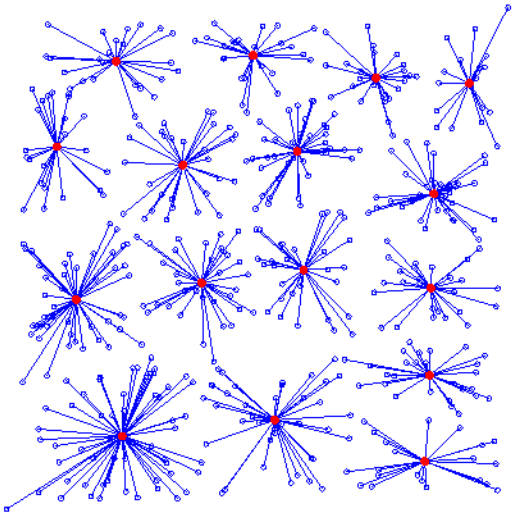
Počet shluků byl, narozdíl od K-Means, v tomto případě testován od 4 do 20. Hodnota prahu byla zkoumána od 0,0 do 1,0 s krokem 0,1. Hodnota m byla testována od 2 do 15. Vzhledem k množství kombinací, která lze těmito nastaveními vytvořit, nebylo možné nakreslit graf či zobrazit tabulku s jednotlivými nastaveními řešení. Hodnoty nastavení a výsledky jednotlivých kroků jsou proto uloženy v souboru na přiloženém CD.

Výsledné hodnoty, se kterými algoritmus C-Means poskytl nejmenší C_s pro uniformní data, jsou:

- počet shluků: 16, práh p: 1,0, hodnota m: 2

Hodnota C_s takto vytvořeného řešení dosahovala 133,71, což je výsledek téměř totožný s algoritmem K-Means.

Shlukování bodů odpovídající zjištěnému nastavení iterátoru je zobrazeno na obrázku 3.4.6.



Obrázek 3.4.6: Ukázka výsledku iterace algoritmu C-Means

Výsledky pro ostatní sady dat (data shlukovaná a reálná) jsou také na CD přiloženy. Zde budou znovu uvedeny jen nejlépe ohodnocené konfigurace.

Shlukovaná data:

- počet shluků: 15, práh p: 1,0, hodnota m: 2

Hodnota C_s pro toto nastavení byla 36,40.

První reálná data:

- počet shluků: 7, práh p: 1,0, hodnota m: 3

Hodnota C_s byla 71318,57.

Druhá reálná data:

- počet shluků: 15, práh p: 1,0, hodnota m: 3

Hodnota C_s byla 420163,94.

Třetí reálná data:

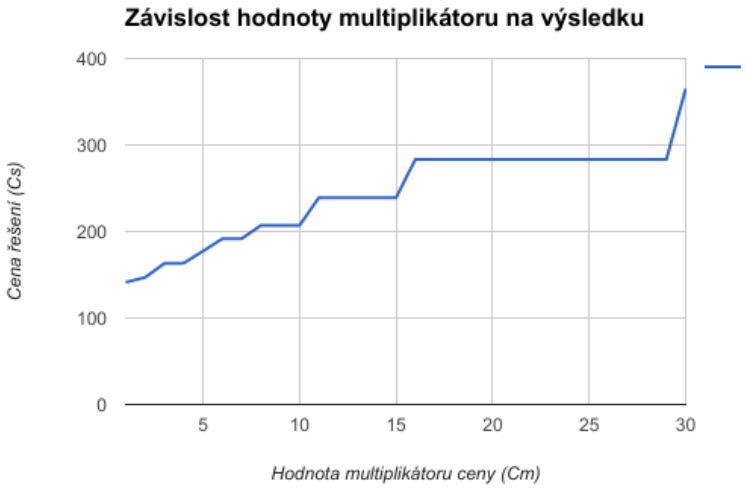
- počet shluků: 20, práh p: 1,0, hodnota m: 2

Cena takovéhoto řešení byla 811902,40.

Z výsledků je patrné, že algoritmus C-Means pro data v prostoru poskytl nejlepší výsledky, pokud byla hodnota prahu nastavena na 1,0. Při takovéto konfiguraci má algoritmus téměř totožné vlastnosti, jako algoritmus K-Means a poskytuje téměř shodné výsledky.

3.4.3 Výsledky local search

Stejně jako u K-Means byl jen jediný parametr pro iterování. Tím byl multiplikátor ceny za vytvoření shluku. Při iterování byl zkoumán parametr multiplikátoru od jedné do třiceti. Výsledek pro uniformní data je možné vidět v grafu 3.4.7:

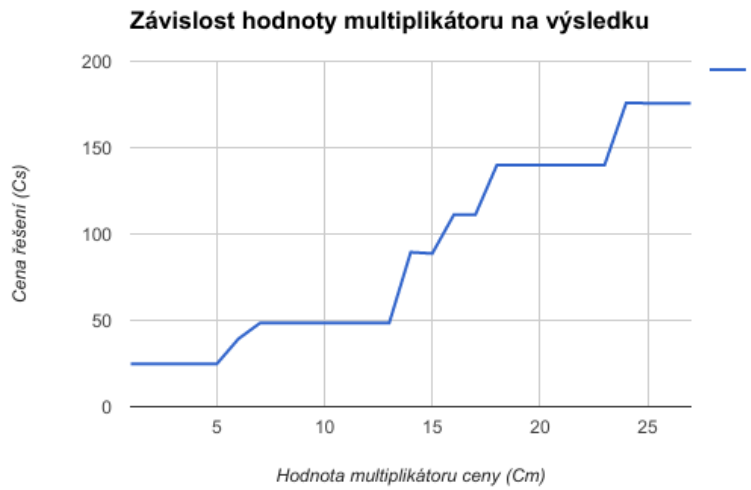


Obrázek 3.4.7: Local search - graf závislosti ceny řešení na multiplikátoru ceny

Nejlepší řešení algoritmus poskytl pro hodnotu multiplikátoru 1, kdy algoritmus vytvořil 18 shluků. Cena výsledného řešení byla 141,27.

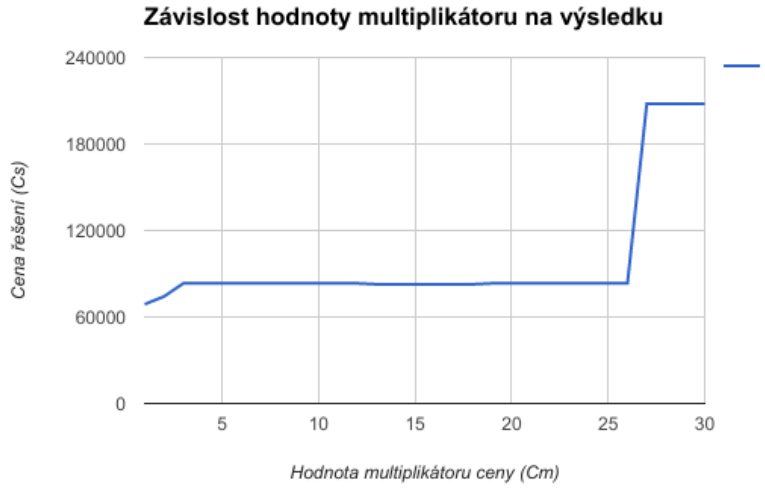
Vzhledem k tomu, že tento algoritmus pro stejné vstupní parametry vyprodukuje pokaždé stejné řešení, vizualizaci nejlepšího nalezeného řešení je možné vidět na obrázku 3.3.5 v kapitole 3.3.5.

Graf hodnocení pro experimenty se shlukovanými daty je vidět na obrázku 3.4.8. Nejlepšího hodnocení se dosáhlo pro hodnotu multiplikátoru rovnou 1, a to s deseti shluky a s hodnotou C_s rovnou 24,69.



Obrázek 3.4.8: Local search - výsledky nastavení pro shlukovaná data

Graf výsledků pro první reálná data je vidět na obrázku 3.4.9. Nejlepší řešení vyšlo, když byla hodnota multiplikátoru nastavena na 1, kdy algoritmus vytvořil řešení s pěti shluky s cenou 68850,62.



Obrázek 3.4.9: Local search - výsledky nastavení pro první reálná data

Graf výsledků pro druhá reálná data je vidět na obrázku 3.4.10. Nejlepší řešení bylo vytvořeno s hodnotou multiplikátoru nastavenou na 1, kdy algoritmus vytvořil řešení s patnácti shluky s cenou 414450,36.

Obrázek 3.4.10: Local search - výsledky nastavení pro druhá reálná data

Poslední sada experimentů probíhala na třetí sadě reálných dat. Graf je vidět na obrázku 3.4.11. Nejlepší řešení, vytvořené s cenou 773352,31 a devatenácti shluky, vytvořil algoritmus s nastavením multiplikátoru na hodnotu 1.

Obrázek 3.4.11: Local search - výsledky nastavení pro třetí reálná data

Z experimentů s tímto algoritmem vyplývá, že nemá cenu algoritmu vnucovat vyšší cenu za vytvoření shluku. Vzhledem k tomu, že algoritmus pro danou cenu nalezne nejlepší možné řešení, nastavení jiné ceny jen zhorší

získaný výsledek.

3.4.4 Závěr z experimentů

Po provedení všech testů byla vyhotovena tabulka výsledků všech algoritmů pro různá data (tab. 3.1). Pro každý algoritmus a sadu dat bylo provedeno několik průběhů algoritmu a výsledky byly zprůměrovány. Testy byly prováděny se záměrem najít takovou konfiguraci algoritmu, která u něj přinese co nejlepší ohodnocení.

	Uniformní	Shlukovaná	1. reálná	2. reálná	3. reálná
K-Means	133,21	35,44	71763,58	434969,52	825848,62
C-Means	133,71	36,40	71318,57	420163,94	811902,40
Local search	141,27	24,69	68850,62	414450,36	773352,31

Tabulka 3.1: Tabulka výsledků pro jednotlivá data

Na uniformních datech se ukázalo, že algoritmy K-means a C-means poskytují lepší řešení než algoritmus local search.

U shlukovaných dat zase algoritmus local search poskytl řešení o čtvrtinu lépe hodnocené. Toto řešení algoritmus nalezl se stejným počtem shluků, jako bylo shluků bodů ve vstupních datech (tedy 10). Tato data a řešení algoritmu local search je možné vidět na obrázku 3.3.7. K-Means a C-means našly jako nejlepší řešení 16 shluků. To je způsobeno problémem počátečního rozložení bodů (viz kap. 2.4). Algoritmus local search tento problém nemá, proto dokáže takto dopředu shlukovaná data dobře rozdělit.

U reálných dat není rozdíl mezi poskytnutými řešeními tak velký. Nejlepší hodnocení, stejně jako u shlukovaných dat, poskytl algoritmus local search. Výsledky algoritmů C-Means a K-Means jsou, stejně jako u předchozích dat, velmi podobné. Tato podobnost je způsobena tím, že nejlepší řešení pro algoritmus C-Means bylo nalezeno pro konfiguraci, kdy se chování algoritmu velmi podobá chování algoritmu K-Means.

Z těchto pokusů je možné vyvodit závěr, že pro reálné využití je z těchto tří testovaných algoritmů nejlepší local search. Nejen, že poskytl nejlepší řešení pro reálná data, narozdíl od ostatních dvou algoritmů nevyžaduje zadání počtu shluků – počet shluků pro nalezení nejlepšího řešení sám najde. Tento algoritmus se tedy hodí pro aplikace, kdy se při dané ceně vytvoření centra shluku a vzdálenosti bodů shluku od jejich centra hledá nejlepší počet a rozmístění center. V praxi může jít například o řešení logistických problémů – jedním z těchto problémů může být nalezení vhodného rozmístění skladů

pro zásobování, rozmístění elektrických stanic apod.

To ale neznamená, že by algoritmy K-Means a C-Means neměly svoje využití. K-Means se hodí pro aplikace, kdy je potřeba najít rozdělení do daného počtu shluků. Pokud by tedy cílem práce nebylo najít nejlepší řešení celkově, nýbrž nejlepší řešení pro daný počet shluků, bylo by nutné využití jiného algoritmu, než je facility location. Vzhledem k tomu, že K-Means a C-Means poskytly velmi podobné řešení, je i přes to, že byly výsledky získané algoritmem C-Means o trochu lepší, vhodnější použít algoritmus K-Means. Důvodem je menší počet iterací, který algoritmus k nalezení řešení potřeboval, a tudíž i menší časová náročnost. Vzhledem k tomu, že je nutné shlukování těmito algoritmy opakovat kvůli problému počátečního rozložení, mohl by tento výkonový rozdíl hrát roli.

Výhodou algoritmu C-Means je fuzzy logika, která je jeho podstatou. V této práci byla potřeba striktního rozdělení bodů do shluků, což tuto vlastnost potlačilo. Fuzzy logika má své využití v mnohých aplikacích, jako je umělá inteligence, medicína, předpověď počasí atd. [14].

4 Závěr

Výsledkem první části práce je vytvořené programové vybavení, které pomocí centroidní Voroného teselace upravuje rozmístění bodů v prostoru. Demonstrace funkce programu byla ukázána na několika sadách testovacích dat. U reálných dat poté proběhly experimenty s generováním vrstevnic. Bylo zjištěno, že takovéto upravení terénu zlepší kvalitu vrstevnic, a je tedy vhodnou alternativou k CDT, která, jak se ukázalo, pro daný typ dat téměř žádné zlepšení nepřinesla.

Možným rozšířením programu do budoucna by mohla být automatická detekce hran v terénu, která v rámci této práce nebyla nebyla realizována.

V rámci druhé části práce byly naprogramovány dva shlukovací algoritmy a bylo vypracováno srovnání s již implementovaným algoritmem. Jako součást řešení byla vytvořena sada iterátorů hledající ideální konfiguraci jednotlivých algoritmů. Na základě testovacích dat jak umělých, tak reálných bylo vytvořeno doporučení pro jednotlivé algoritmy.

Jako další práce na daném projektu by bylo možné implementovat další shlukovací algoritmy, zahrnout je do porovnání a podrobněji prozkoumat chování fuzzy metod shlukování.

Literatura

[1] Rajaraman, A. – Ullman, J. D. *Mining of Massive Datasets.* Cambridge University Press, 2011. ISBN 1107015359, 9781107015357.

[2] Tan, P.-N. – Steinbach, M. – Kumar, V. *Introduction to Data Mining, (First Edition).* Addison-Wesley Longman Publishing Co., Inc., 2005. ISBN 0321321367.

[3] Murphy, K. P. *Machine Learning: A Probabilistic Perspective.* MIT Press, 2012. ISBN 978-0-262-01802-9.

[4] Palumbo, F. – Lauro, C. – Greenacre, M. *Data Analysis and Classification: Proceedings of the 6th Conference of the Classification and Data Analysis Group of the Società Italiana di Statistica.* Studies in Classification, Data Analysis, and Knowledge Organization. Springer Berlin Heidelberg, 2009. ISBN 9783642037382.

[5] Duda, R. O. – Hart, P. E. – Stork, D. G. *Pattern Classification.* Wiley-Interscience, 2001. ISBN 0-471-05669-3.

[6] Bishop, C. *Pattern Recognition and Machine Learning.* Springer Verlag, 2006. ISBN 0-387-31073-8.

[7] *K-Means* [online]. 2016. [cit. 2017/04/09]. Dostupné z: `https://home.deib.polimi.it/matteucc/Clustering/tutorial_html/kmeans.html`.

[8] Gunzburger, M. Centroidal Voronoi tesselations. *CCS-2/CNLS Seminar.*

[9] Surynková, P. *Prezentace k přednáškám Počítačová geometrie (přednáška 10)* [online]. 2016. Dostupné z: `http://www.surynkova.info/dokumenty/mff/PG/Prednasky/prednaska_10.pdf`.

[10] Fuksa, M. Delaunayova triangulace s omezením (CDT). Diplomová práce, Západočeská univerzita v Plzni, Plzeň, 2006.

[11] Schneider, P. J. – Eberly, D. H. *Geometric tools for computer graphics.* Morgan Kaufmann Publishers, 2003. ISBN 1-55860-594-0.

[12] Floater, M. S. Generalized barycentric coordinates and applications. *Acta Numerica (2016).* s. 1–50.

[13] Murray, J. D. – vanRyper, W. *Encyclopedia of Graphics File Formats (2Nd Ed.).* O'Reilly & Associates, Inc., 1996. ISBN 1-56592-161-5.

[14] Singh, H. et al. Real-Life Applications of Fuzzy Logic. *Advances in Fuzzy Systems*. doi: 10.1155/2013/581879.