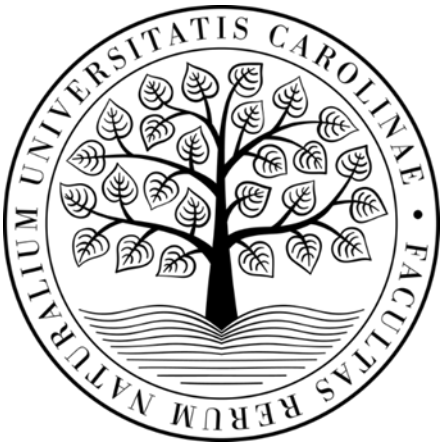
Univerzita Karlova

Přírodovědecká fakulta

Studijní program: Geoinformatika, kartografie a dálkový průzkum Země



Karolína Drápelová

1. ročník studijního programu Geoinformatika, kartografie a dálkový průzkum Země

Prostorová indexace

Geoinformatika

Akademický rok 2025/2026

Praha, 2025

1 Zadání

Pro zadané bodové mračno tvořené body pi = [xi, yi, zi] určete níže uvedené charakteristiky:

- prostorovou hustotu,
- křivost k v každém bodě.

Pro výpočet prostorové hustoty mračna využijte průměrnou vzdálenost d_{aver} k nejbližšímu bodu

$$\rho = \frac{1}{d_{aver}^3}$$

Aproximovanou křivost určete metodou PCA v každém bodě mračna ze vztahu

$$\kappa = \frac{\lambda_1}{\lambda_1 + \lambda_2 + \lambda_3}$$

počet knn volte 30.

Prohledávání bodového mračna bude využívat následující metody:

1. naivní hledání,
2. akcelerované hledání s využitím voxelizace,
3. akcelerované hledání s využitím kd-tree.

Veškeré výše uvedené vyhledávací struktury implementujte samostatně (tj. bez použití knihoven).
Určení hustoty bodového mračna realizujte metodami 1-3, výpočet křivosti metodami 2-3. Hodnoty křivosti v každém z bodě mračna vizualizujte s využitím vhodné barevné škály. Vizualizujte výpočetní čas jako funkci velikosti vstupní množiny, u metody 2 také jako funkci velikosti voxelu.

Krok	**Hodnocení**
Prostorová indexace třemi metodami + výpočet charakteristik	20b
Akcelerované hledání s využitím Octree (vlastní implementace).	*+15b*
Akcelerované hledání s využitím Rtree.	*+5b*
Akcelerované hledání s využitím Rtree (vlastní implementace).	*+20b*
Max celkem:	**60b**

2 Data

Veškeré skripty byly spouštěny na poskytnutých datech tree_18.txt. Jedná se o 3D bodové mračno s přibližně 25 tisíci body. Dále byly využity menší části poskytnutého souboru pro testování a debuggování, jelikož pro původní mračno byly některé operace výpočetně náročné.

3 Naivní hledání

Algoritmus naivního vyhledávání *k* nejbližších sousedů funguje následujícím způsobem:

- Projde všechny body v bodovém mračnu,
- Vypočítá euklidovskou vzdálenost od každého bodu mračna k zadanému bodu (x, y, z),
- Uloží vzdálenost a index bodu do seznamu,
- Pokud je vzdálenost rovna 0, daný bod vyloučí,
- Seřadí seznam,
- Vybere prvních *k* bodů ze seznamu,
- Vrátí seznamy se souřadnicemi bodů.

Def *naive_search* (X, Y, Z, x, y, z, k):

    ….

    return Xn, Yn, Zn

Vstupem jsou tři seznamy obsahující souřadnice bodů (X, Y, Z), souřadnice dotazovaného bodu (x, y, z) a počet nejbližších sousedů (k).

Funkce vrací tři seznamy (Xn, Yn, Zn) souřadnic pro *k* nejbližších sousedů.

Pomocnou funkcí je výpočet euklidovské vzdálenosti, která bere na vstupu souřadnice dvou bodů a vrací float hodnotu vzdálenosti mezi body.

Def *euclid_distance* (x1, y1, z1, x2, y2, z2):

    return sqrt((x1 - x2)**2 + (y1 - y2)**2 + (z1 - z2)**2)

4 Voxelizace

Def *init_Spat_Index* (X, Y, Z, n_xyz):

return x_min, y_min, z_min, dx, dy, dz, bx, by, bz

Funkce init_Spat_Index spočítá minimální a maximální souřadnice bodů, a spočítá parametry pro voxelovou mřížku.

Def *get_3D_index* (x, y, z, dx, dy, dz, x_min, y_min, z_min, n_xyz):

return jx, jy, jz

Funkce převádí 3D souřadnice na celočíselné indexy bodu (jx, jy, jz).

Pomocí funkce níže jsou pak tyto indexy převedeny na 1D indexy (j).

Def *get_1D_index* (jx, jy, jz, n_xyz):

return jx + jy * n_xyz + jz * n_xyz**2

Def *create3Dindex* (X, Y, Z, x_min, y_min, z_min, dx, dy, dz, bx, by, bz, n_xyz):

return H, J

Tato funkce vytváří prostorový index H (defaultdict), klíčem je 1D index voxelů a hodnotou jsou seznamy indexů bodů, které se v daném voxelu nacházejí. Dále funkce vytváří seznam J, kde J[i] je 1D index voxelu pro i-tý bod v mračnu.

def *knn_search_voxel* (X, Y, Z, x, y, z, k, H, dx, dy, dz, x_min, y_min, z_min, n_xyz):

Tato funkce bere jako argument dotazovaný bod a prohledává jen ty body, které spadají do stejného voxelu jako dotazovaný bod.

- Vypočítá 1D index voxelu pro dotazovaný bod,
- Načte seznam indexů bodů z tohoto voxelu pomocí H,
- Prochází tyto body, počítá vzdálenosti,
- Setřídí seznam a vrátí *k* nejbližších sousedů.

Tato funkce využívá tzv. *approximate nearest neighbour search*, jelikož prochází jen jeden voxel. Pro přesnější výsledky by bylo nutné prohledávat i vedlejší voxely.

Pomocná funkce def *knn_search_voxel_wrapper* (X, Y, Z, x, y, z, k): předává globálně inicializované parametry funkci *knn_search_voxel*.

5 KD Tree

Class Node:

Reprezentuje „node“ ve stromu, obsahuje „splitting“ bod a odkazy na levý a pravý podstrom.

def *build_kdtree* (points, depth=0):

Pomocí rekurze sestaví strom z bodového mračna.

- Vybere osu, podle které se bodové mračno rozdělí. Na základě hloubky (zde 3) se osy střídají. První rozdělení je podél osy X, poté podle Y, pak Z, takto se stále iteruje.
- Seřadí se body podle vybrané osy,
- Vypočítá se medián jako „splitting“ bod pro aktuální „node“,
- Vytvoří se levý podstrom (body před mediánem) a pravá podstrom (body za mediánem), zvýší se hloubka,
- Pokud je bodové mračno prázdné, funkce vrátí None a rekurze se ukončí.

Funkce pro hledání nejbližšího souseda:

def *kd_tree_knn_search* (node, target, k, knn_list, depth=0):

Pomocí rekurze vyhledává *k* nejbližších sousedů ke zvolenému bodu.

- Vypočítá vzdálenost mezi zvoleným bodem a bodem v aktuální „node“, přidá ho do seznamu,
- Podívá se na souřadnici aktuální dělící osy (X, Y, Z),
- Pokud je souřadnice zvoleného bodu menší než souřadnice „node“ bodu, hledaný soused leží pravděpodobně vlevo (near = node.left), v opačném případě naopak.
- Zanoření do *near* větve, prohledání bodů,
- Pokud je potřeba, prohledává se i *far* větev.

Pomocné funkce:

def *knn_list_add* (knn_list, point, dist, k):

Tato pomocná funkce spravuje seznam *k* aktuálně nalezených nejbližších sousedů.

- Pokud seznam není plný, bod se přidá,
- Pokud je seznam plný, najde se bod s největší vzdáleností,
- Pokud je nový bod blíže než bod s největší vzdáleností, nahradí ho

def *kd_tree_search_wrapper* (X, Y, Z, x, y, z, k):

Pomocná funkce, která vrací seznamy souřadnic nejbližších sousedů.

6 Výpočet hustoty

def *compute_density* (X, Y, Z, knn_method):

Vstupy funkce pro výpočet hustoty bodového mračna jsou seznamy souřadnic bodů a vyhledávací metoda (naivní hledání, voxelizace, kd tree).

- Funkce projde všechny body v mračnu,
- Pomocí zadané metody nalezne pro každý bod jeho nejbližšího souseda,
- Vypočítá vzdálenost k nalezenému sousedovi a přidá ji do seznamu,
- Ze všech vzdáleností v seznamu se pak vypočítá průměrná hodnota,
- Hustota se pak spočítá pomocí vzorce:

$$\rho = \frac{1}{d_{aver}^3}$$

7 Výpočet křivosti

Def *curvature* (X, Y, Z, knn_method):

Tato funkce odhaduje křivost pro každý bod v bodovém mračnu. Jako vstupy bere seznamy souřadnic bodů a vyhledávací metodu (voxelizaci, kd tree).

- Pro každý bod se najde 30 nejbližších sousedů,
- Pro tuto oblast se vypočítá kovariační matice,
- Vypočítají se vlastní čísla kovariační matice, kde $\lambda_1 < \lambda_2 < \lambda_3$,
- Křivost se pak vypočítá pomocí vzorce:

$$\kappa = \frac{\lambda_1}{\lambda_1 + \lambda_2 + \lambda_3}$$

8 RTree

V souboru RTREE_library.py je implementováno hledání s využitím Rtree. Bylo využito knihovny rtree.

9 Výsledky

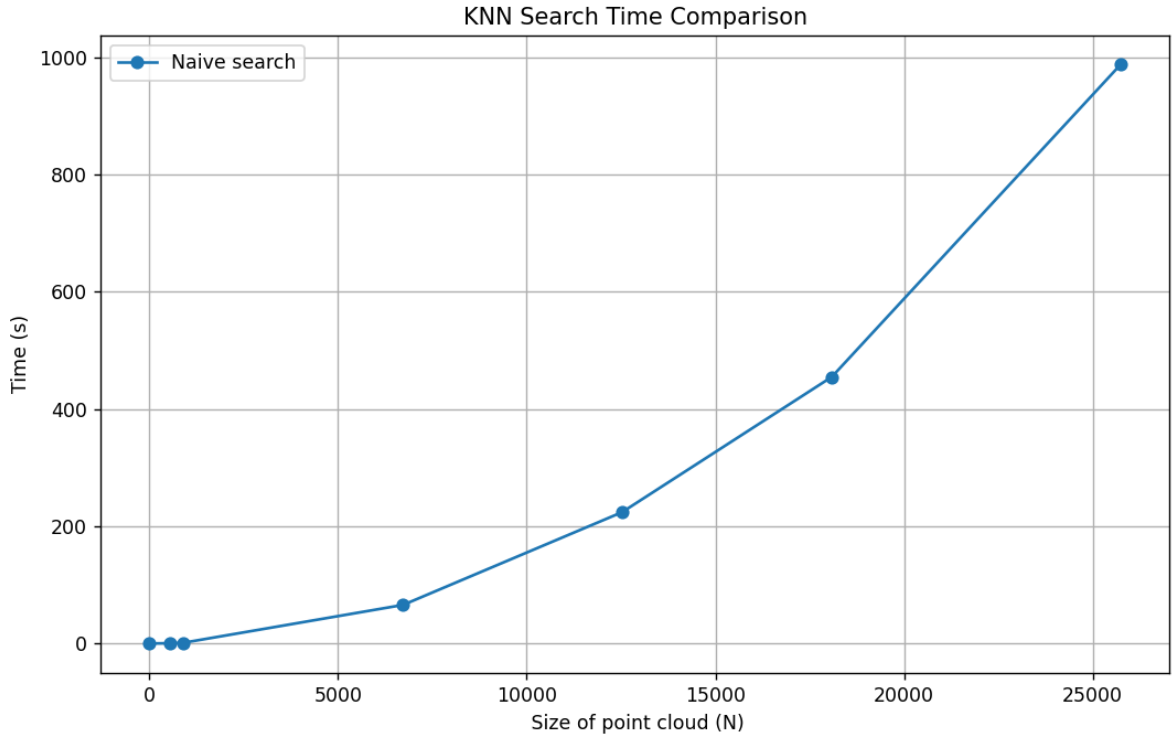
V tabulce jsou uvedeny hodnoty hustot vypočítané různými metodami.

Metoda	Hustota
Naive search	5881,44
Voxelization	5526,578
KD-Tree	5881,44

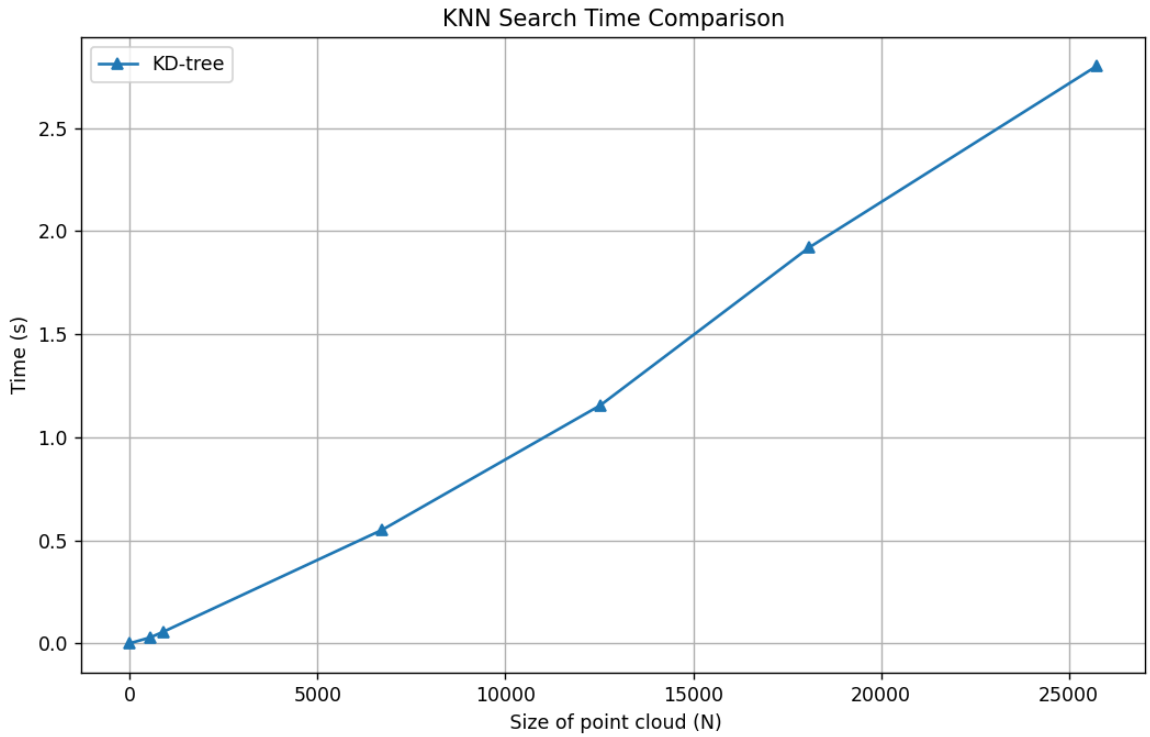
Tabulka 1: Výsledky hustoty.

Lze si všimnout, že hodnota u metody voxelizace se mírně liší, způsobeno to může být tím, že u této metody hledáme *approximate nearest neighbor* a né přesného nejbližšího souseda.

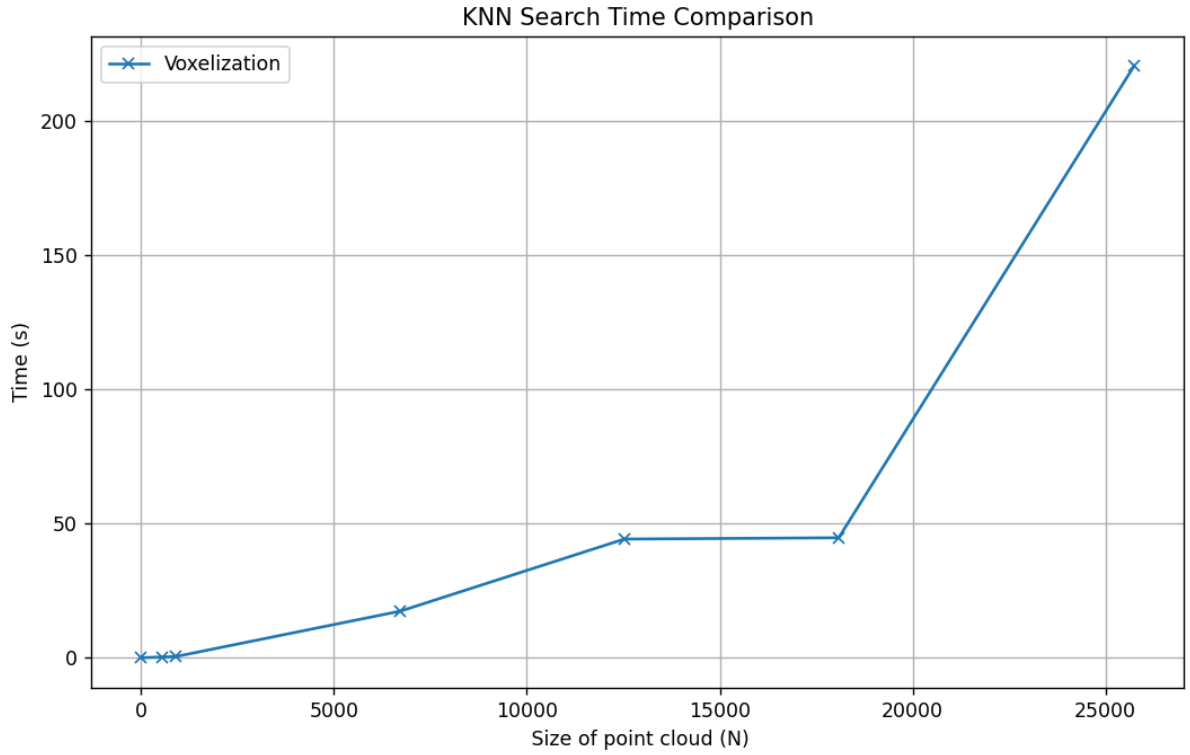
Na obrázcích níže jsou grafy zobrazující rychlost výpočtu hustoty na základě vybrané metody a velikosti vstupní množiny.



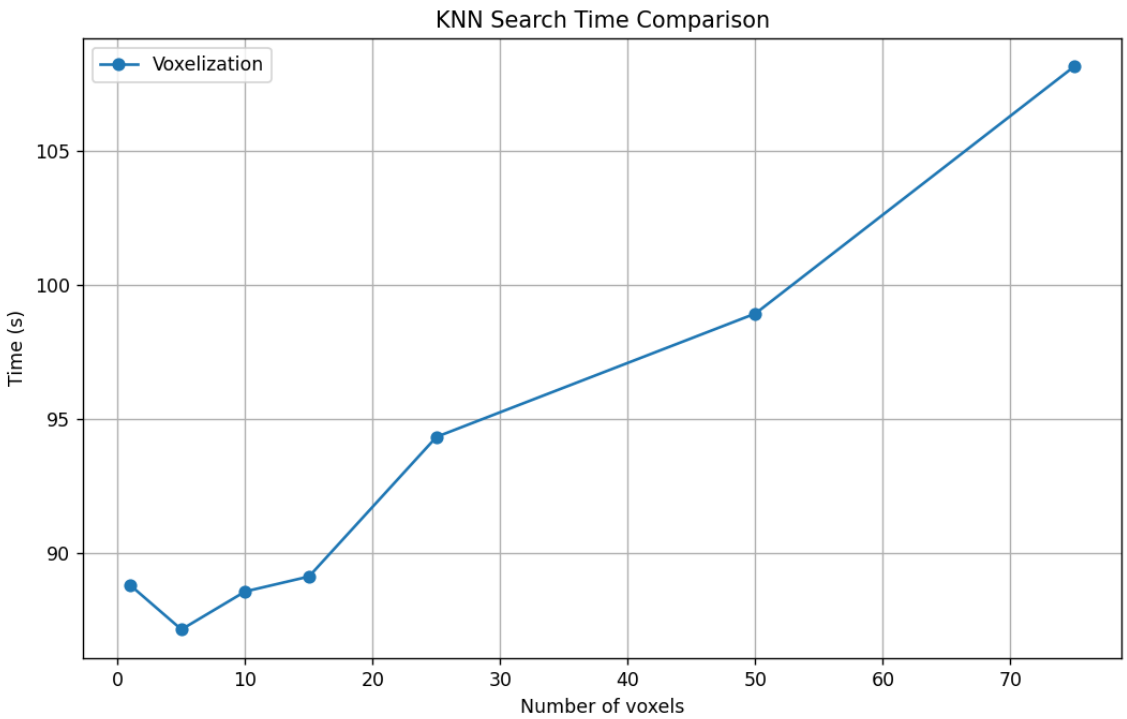
Obrázek 1: Výpočet hustoty s využitím Naive search.



Obrázek 2: Výpočet hustoty s využitím KD-Tree.



Obrázek 3:Výpočet hustoty s využitím voxelizace.



Obrázek 4:Výpočet hustoty s využitím voxelizace (mění se počet voxelů).

Výpočet křivosti je vizualizován v samostatných skriptech CURVATURE_VOXEL_PLOT.py a KDTREE_CURVATURE_PLOT.py.

10 Zdroje

WIKIPEDIA (2025): K-d tree. https://en.wikipedia.org/wiki/K-d_tree

Geeks for geeks (2025): Search and Insertion in K Dimensional tree. https://www.geeksforgeeks.org/dsa/search-and-insertion-in-k-dimensional-tree/

PyPi (2025): Rtree: Spatial indexing for Python. https://pypi.org/project/rtree/

Bayer, T. (2025): Prostorová indexace dat. https://web.natur.cuni.cz/~bayertom/images/courses/Geoinf/geoinf3_index.pdf