Zadání semestrální práce číslo 19
Přibližné vyhledávání k nejbližších sousedů pomocí BBD stromů

Petr Šádek[1]
Katedra počítačové grafiky a interakce,
Fakulta elektrotechnická, ČVUT Praha

Abstract

Implementujte stavbu a vyhledávání pomocí dekompozičních stromů hierarchie obálek (BBD-strom). Porovnejte přesné a přibližné vyhledávání pomocí BBD-stromů pro různé hodnoty epsilon. Vyzkoušejte pro různé rozložení bodů ve 2D, 3D a 4D a porovnejte s naivní implementací. Použijte datové sady různých velikostí od 10^3 do 10^8. Otestujte výkon pro 2 nebo 3 různé implementace prioritní fronty.

Keywords: nejbližší soused, přibližný nejbližší soused, BBD strom

1. Úvod

Cílem této práce bylo vytvořit implementaci hledání k přibližných nejbližších sousedů, konkrétně pomocí BBD stromu a analyzovat její efektivitu. Za pomoci této hierarchické datové struktury můžeme hledání nejbližších sousedů výrazně zrychlit oproti naivnímu řešení lineárním průchodem. K tomu můžeme toto hledání parametricky zrychlovat podle zvolené přesnosti nalezených nejbližších sousedů. Tento algoritmus má mnoho praktických využití, jako jsou například fotonové mapy, nebo způsob klasifikace ve strojovém učení.

2. Stavba BBD stromu

BBD strom se skládá ze 2 typů vnitřních uzlů: uzlů dělících rovinou a smršťujících uzlů. Uzly dělící rovinou dělí prostor rovinou kolmou na jednu souřadnicovou osu. Smršťující uzly jsou specifikované obálkou a dělí prostor na část uvnitř obálky a část vně.

Stavba BBD stromu je realizována pomocí midpoint algoritmu. Ten opakovaně dělí prostor vždy v nejdelší dimenzi dokud $n_i > \frac{2}{3}n$, kde n je původní počet bodů a n_i je aktuální počet. V každé této iteraci se zanořuje do části s větším počtem. Pokud proběhne pouze jedna tato iterace, pak je vytvořen uzel dělící rovinou. V opačném případě se vytvoří smršťující uzel reprezentovaný obálkou, která vznikla po již zmíněné sekvenci dělení.

Po vytvoření nového uzlu tímto způsobem se rekurzivně vykoná stejný postup pro potomky daného uzlu. Body rozdělíme mezi potomky pomocí algoritmu podobnému dělení pole na 2 části v quick sortu. U uzlu dělícího rovinou stačí postupně projíždět body a porovnávat jejich pozici vůči pozici dělící roviny. Pokud je nalevo, tak prohodíme s prvkem za prvky již

[1] A4M39DPG – Petr Šádek, zimní semestr 2022/23

přidanými nalevo. Obdobně to je pro druhou stranu. Pro smršt'ující uzel je to skoro stejné, jen místo pozici vůči rovině zjišt'ujeme zda je bod uvnitř či vně obálky.

V případě že počet bodů klesne na nebo pod maximální počet bodů obsažených v listu, pak vytvoříme list který uchovává pouze indexy na začátek a konec v rámci pole bodů. To můžeme udělat díky dělení bodů popsaném v předchozím odstavci, protože body obsažené v nějakém uzlu jsou potom vždy za sebou v paměti.

3. Popis algoritmu

Algoritmus pro hledání k přibližných nejbližších sousedů prochází jednotlivé uzly pomocí prioritní fronty podle jejich vzdálenosti od dotazovaného bodu. Pokud narazí na list, tak přidá jeho body do další prioritní fronty, která je ale limitovaná maximálním počtem prvků k. Obsah této fronty po doběhnutí průchodu je požadovaný výsledek.

V případě přibližných nejbližších sousedů lze výpočet ukončit dříve pokud narazíme na uzel pro který platí $d > d_l/(1+\epsilon)$, kde d je vzdálenost uzlu od dotazovaného bodu a d_l je bod z konce prioritní fronty k nejbližších sousedů.

Je více možností jak implementovat frontu pro k nejbližších sousedů. Součástí zadání tohoto projektu je vytvořit 2-3 implementace této fronty. Byly zvoleny následující implementace: lineární fronta, vlastní implementace haldy a implementace pomocí knihovní C++ haldy.

Lineární fronta jednoduše prochází celé pole a kontroluje zda je prvek menší. Implementace pomocí vlastní haldy funguje podobně, ale vykonává jenom logaritmus kroků, protože má prvky srovnané tak aby dodržovaly pravidla haldy. Varianta pomocí knihovní C++ std haldy je implementována jednoduše pomocí volání heap_push a heap_pop.

4. Potíže při implementaci

Nejvíce problémové asi bylo dát dohromady celkovou strukturu projektu. Několikrát jsem kompletně předělával některé části projektu, protože jsem později zjistil že by to bylo nevhodné například kvůli kódu na měření statistik nebo vizualizaci. Celkově vygenerovat data, naměřit statistiky a udělat vizualizaci (tak aby vypadala rozumně) bylo časově náročné (nejen jejich samotné vykonání, ale psaní kódu na jejich automatizaci). Možná mi to zabralo i výrazně více času než samotná implementace BBD stromu a vyhledávání nejbližších sousedů.

Také jsem se v jednu chvíli dlouho zasekl kvůli chybě ve stavbě stromu. Nakonec šlo jen o malou chybu v podmínce, která způsobovala že se prakticky nikdy nevytvářely uzly dělící rovinou a většina uzlů byla smršt'ovacích, což není požadované.

Řekl bych že jsem celkově na projektu strávil kolem 150 h.

5. Naměřené výsledky

Pro měření byly použity datové sady se 3 různými distribucemi: uniformní, normální a clustery. Clustery jsou složeny z několika normálních distribucí o různých rozptylech a průměrech. Každá tato distribuce má varianty ve 3 velikostech: 10^3, 10^5 a 10^7. Zároveň jsou zde všechny varianty pro dimenze 2, 3 a 4. Pro všechna měření byla maximální velikost listu v BBD stromu nastavena na 10.

Měřený kód byl kompilován pomocí MSVC v Release módu. Statistiky byly naměřeny na zařízení s následujícími parametry:

- AMD Ryzen 5 3600 CPU 3.6GHz, využito 1 jádro

  L1 cache: 64 kB (na jádro)

  L2 cache: 512 kB (na jádro)

  L3 cache: 32 MB

- 16GB paměti RAM
- OS Windows 64-bit

Nyní následují 3 tabulky statistik stavby BBD stromu pro dimenze 2, 3 a 4. První sloupec značí typ distribuce. N_P je počet bodů. T_B - čas stavby stromu. M - paměťové nároky uzlů. N_I - počet vnitřních uzlů. N_L - počet listů. N_S - poměr smršťovacích uzlů vůči celkovému početu vnitřních uzlů. N_{AP} - průměrný počet bodů v listu. D_{MAX} - maximální hloubka. D_{AVG} - průměrná hloubka.

Z tabulek je vidět že celkový počet smršťovacích uzlů je většinou poměrně malý. To je pozitivní, protože smršťovací uzly jsou poměrně paměťově náročné a jsou i náročnější z hlediska výpočtu vzdálenosti. Dále paměťová náročnost uzlů je vždy výrazně nižší než paměťová náročnost samotných vstupních bodů. Listy jsou v průměru zaplněné ze 70%, tedy body jsou mezi ně rozděleny relativně efektivně.

Distrib. 2D	N_P	T_B	M	N_I	N_L	N_S(%)	N_{AP}	D_{MAX}	D_{AVG}
clusters	10^3	81.0 μs	3 kB	137	138	41.6	7.3	9	7.2
normal	10^3	88.0 μs	2 kB	137	138	32.8	7.3	9	7.3
uniform	10^3	58.0 μs	2 kB	137	138	7.3	7.3	8	7.2
clusters	10^5	9.8 ms	190 kB	14012	14013	11.8	7.1	17	14.0
normal	10^5	9.7 ms	183 kB	14011	14012	8.7	7.1	16	13.9
uniform	10^5	9.0 ms	178 kB	14158	14159	5.5	7.1	15	13.9
clusters	10^7	1.3 s	17 MB	1403474	1403475	5.9	7.1	24	20.7
normal	10^7	1.3 s	17 MB	1402612	1402613	5.8	7.1	23	20.7
uniform	10^7	1.3 s	17 MB	1398605	1398606	5.8	7.1	22	20.5

Distrib. 3D	N_P	T_B	M	N_I	N_L	N_S(%)	N_{AP}	D_{MAX}	D_{AVG}
clusters	10^3	88.0 μs	3 kB	137	138	53.3	7.2	9	7.3
normal	10^3	89.0 μs	3 kB	138	139	44.2	7.2	10	7.4
uniform	10^3	62.0 μs	2 kB	136	137	5.9	7.3	8	7.2
clusters	10^5	11.0 ms	246 kB	14052	14053	24.7	7.1	17	14.1
normal	10^5	11.5 ms	218 kB	14085	14086	15.9	7.1	16	14.0
uniform	10^5	9.2 ms	184 kB	14156	14157	5.3	7.1	15	13.9
clusters	10^7	1.4 s	19 MB	1402959	1402960	8.0	7.1	25	20.8
normal	10^7	1.4 s	18 MB	1402984	1402985	6.7	7.1	24	20.7
uniform	10^7	1.2 s	18 MB	1398423	1398424	5.8	7.2	22	20.5

Distrib. 4D	N_P	T_B	M	N_I	N_L	N_S(%)	N_{AP}	D_{MAX}	D_{AVG}
clusters	10^3	124.0 μs	5 kB	141	142	66.7	7.0	9	7.3
normal	10^3	97.0 μs	4 kB	141	142	44.0	7.0	9	7.3
uniform	10^3	79.0 μs	2 kB	140	141	7.9	7.1	8	7.2
clusters	10^5	14.8 ms	323 kB	14033	14034	36.1	7.1	17	14.1
normal	10^5	13.6 ms	278 kB	14104	14105	25.7	7.1	17	14.0
uniform	10^5	11.0 ms	190 kB	14074	14075	5.7	7.1	15	13.9
clusters	10^7	1.9 s	22 MB	1403850	1403851	13.4	7.1	25	20.9
normal	10^7	1.7 s	20 MB	1403210	1403211	9.5	7.1	24	20.8
uniform	10^7	1.5 s	18 MB	1398463	1398464	5.8	7.2	22	20.5

V následujících tabulkách jsou uvedeny statistiky dotazů pro různé nastavení parametrů. Byla použita fronta implementovaná pomocí haldy. Každá tabulka je pro různé k, postupně 1, 10 a 100. Měřeno pro data o dimenzi D a počtu bodů N. Jsou zde 3 sloupce pro různé hodnoty parametru ϵ nabývající 0, 1 a 10. Pro každý je zde trojce veličin T_R, N_{TR} a S. T_R je průměrný čas dotazu. N_{TR} je počet navštívených uzlů při průchodu datovou strukturou. S je zrychlení vůči naivní implementaci.

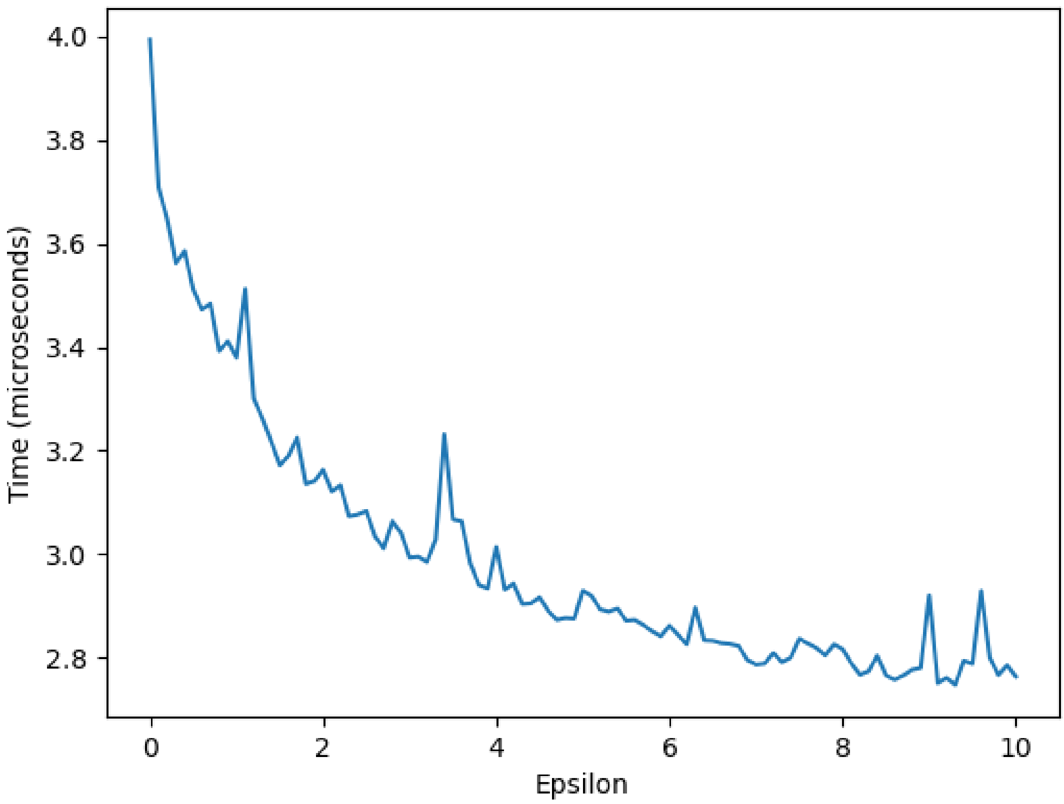
Je vidět že pro větší objemy dat se výrazně projevuje zrychlení oproti naivní variantě. Zároveň je vidět značný dopad nastavení parametru ϵ. Už při $\epsilon = 1$ je zrychlení i víc jak dvojnásobné.

k = 1		$\epsilon = 0$			$\epsilon = 1$			$\epsilon = 10$		
D	N	$T_R(\mu s)$	N_{TR}	S	$T_R(\mu s)$	N_{TR}	S	$T_R(\mu s)$	N_{TR}	S
2	10^3	2.4	17.2	2.3	1.2	15.5	4.7	1.2	15.2	4.7
2	10^5	6.1	46.9	113.5	3.7	45.2	187.2	3.7	43.8	187.2
2	10^7	18.2	59.1	3108.7	5.0	57.6	11315.6	4.6	56.1	12299.5
3	10^3	2.1	20.7	3.0	1.4	17.2	4.4	1.0	13.0	6.2
3	10^5	6.5	54.2	91.8	3.9	42.4	153.0	3.2	38.2	186.5
3	10^7	17.5	67.1	3387.5	6.5	61.5	9120.3	5.5	59.5	10778.5
4	10^3	3.9	30.0	1.6	1.5	18.5	4.1	1.0	13.7	6.2
4	10^5	13.2	79.9	63.9	6.4	55.0	131.9	5.0	41.8	168.8
4	10^7	28.7	106.4	2336.5	12.9	96.0	5198.3	6.7	70.4	10008.7

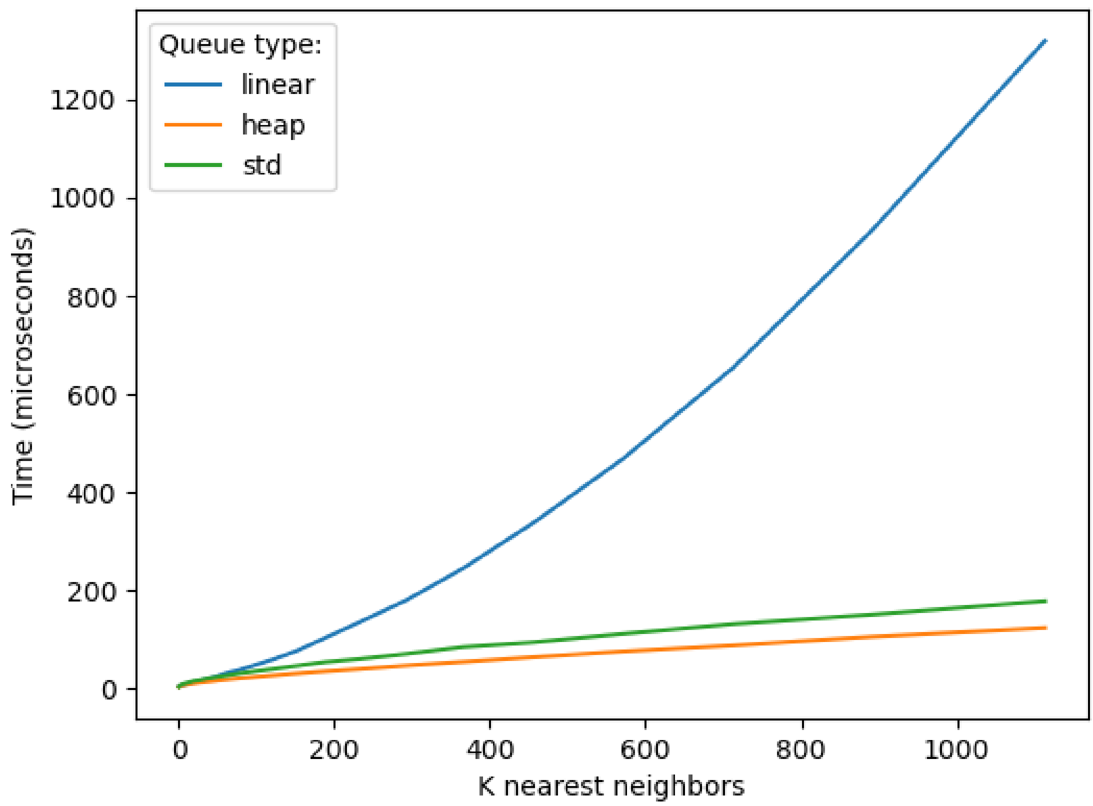
k = 10		$\epsilon = 0$			$\epsilon = 1$			$\epsilon = 10$		
D	N	$T_R(\mu s)$	N_{TR}	S	$T_R(\mu s)$	N_{TR}	S	$T_R(\mu s)$	N_{TR}	S
2	10^3	4.0	25.4	2.3	4.2	22.1	2.2	2.1	18.6	4.3
2	10^5	7.5	53.0	74.9	6.3	52.5	89.2	5.4	48.1	104.0
2	10^7	12.9	75.1	4391	7.5	69.4	7554	6.3	61.6	8993
3	10^3	6.2	43.7	1.5	3.8	31.4	2.4	2.2	19.0	4.1
3	10^5	11.9	90.2	49.2	8.7	72.9	67.3	5.7	46.5	102.6
3	10^7	19.4	102.1	3038	10.1	88.8	5836	7.9	70.8	7462
4	10^3	8.9	70.0	1.0	4.7	35.7	2.0	2.4	17.4	3.8
4	10^5	21.0	141.9	30.4	12.3	100.8	52.0	7.1	53.2	90.0
4	10^7	42.5	190.0	1535	13.9	115.6	4695	10.3	83.4	6337

k = 100		$\epsilon = 0$			$\epsilon = 1$			$\epsilon = 10$		
D	N	$T_R(\mu s)$	N_{TR}	S	$T_R(\mu s)$	N_{TR}	S	$T_R(\mu s)$	N_{TR}	S
2	10^3	9.6	60.3	7.6	7.2	47.5	10.1	6.1	43.4	11.9
2	10^5	15.7	105.6	49.3	11.4	83.2	67.8	10.1	78.2	76.6
2	10^7	20.0	115.8	2979.7	12.7	100.3	4692.4	11.9	97.1	5007.9
3	10^3	19.0	92.1	5.4	13.3	61.1	7.7	8.8	45.0	11.6
3	10^5	31.7	193.7	25.8	18.9	131.7	43.3	12.3	89.0	66.6
3	10^7	39.0	221.9	1519.9	19.8	144.2	2993.8	13.7	106.3	4326.8
4	10^3	19.2	121.0	3.6	13.3	81.3	5.2	7.4	49.4	9.3
4	10^5	50.4	345.2	17.0	33.4	187.8	25.6	19.8	105.4	43.2
4	10^7	103.5	467.8	628.0	39.1	244.2	1662.4	19.1	136.6	3403.1

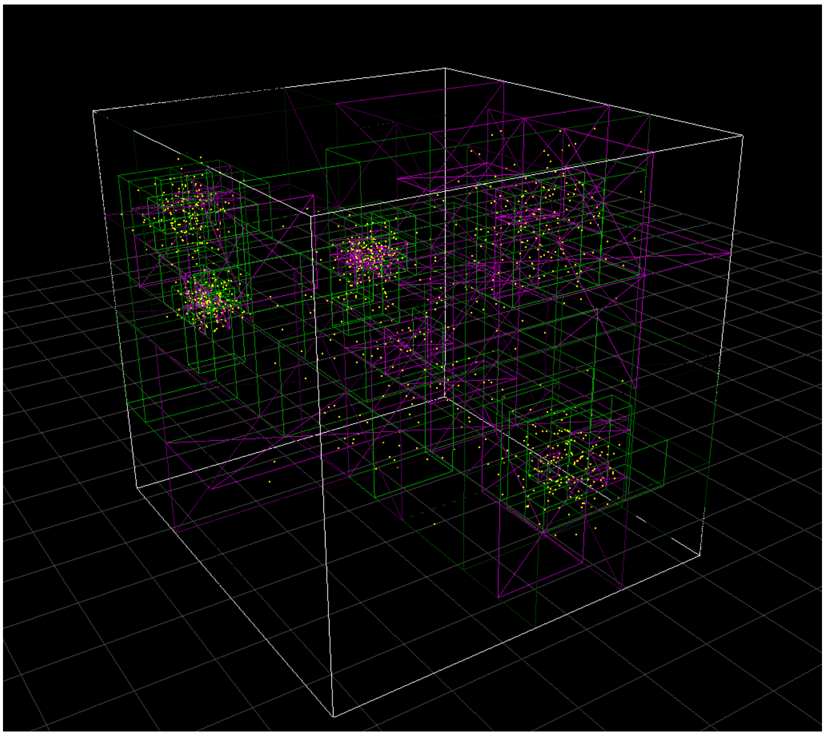
Na následujícím obrázku je vyznačena závislost času dotazu na parametru ϵ. Měřeno pro clusterová data, dimenze 3, počet bodů 10^5, k = 1. Podobný trend je zachován i pro jiná nastavení parametrů. Tyto výsledky se shodují s výsledky zdrojového článku, avšak jsou zde menší odchylky, nejspíš způsobené malým počtem vzorků (použito 1000 vzorků na každou hodnotu ϵ).

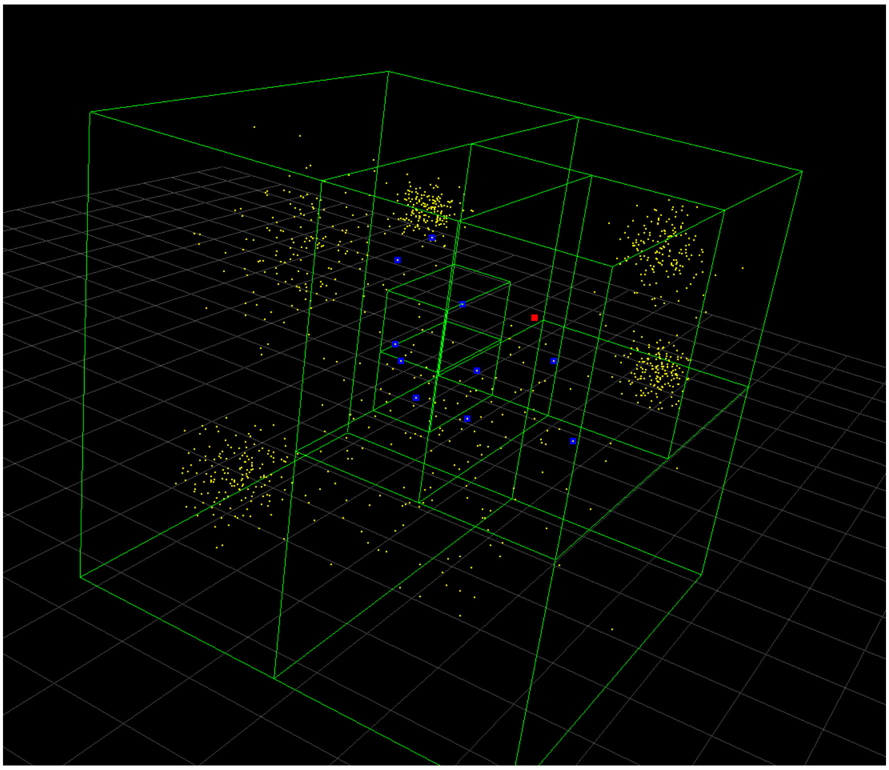


Další graf porovnává efektivnost různých implementací prioritních front pro hledání k nejbližších sousedů. Měřeno pro clusterová data, dimenze 3, počet bodů 10^5, $\epsilon = 0$. Modře je lineární implementace fronty, žlutě vlastní implementace pomocí haldy, zeleně C++ std implementace pomocí haldy. Nejlépe nakonec vyšla vlastní implementace pomocí haldy, std implementace jen o něco hůř. Lineární implementace nebyla lepší ani pro malá k, byla pouze srovnatelná. Zde by člověk očekával lepší výsledky.



Nakonec jsou zde 2 obrázky vizualizace. První znázorňuje uzly v BBD stromu. Zeleně jsou smršťující uzly, růžově uzly dělící rovinou, žlutě body. Druhý obrázek znázorňuje dotaz na 10 přibližných nejbližších sousedů s $\epsilon = 1$. Červeně je dotazovaný bod, modře jsou nejbližší sousedé, zeleně jsou uzly které dotaz navštívil, žlutě všechny ostatní body.





6. Závěr

Hlavní cíle projektu byly splněny. Je zde však množství nedokonalostí a nedotaženostÍ. Bylo by například vhodné zjistit proč lineární fronta není rychlejší pro malá data. Také by bylo dobré zjistit příčinu výkyvů u ϵ grafu, zda se skutečně jedná o nedostatečný počet vzorků. Vizualizace stromu by mohla být přehlednější. Bylo by také zajímavé prozkoumat efektivitu jiných algoritmů pro stavbu BBD stromu kromě midpoint algoritmu.

References

[2] S. Arya and D. M. Mount, "Approximate Nearest Neighbor Searching", Proc. 4th Ann. ACM-SIAM Symposium on Discrete Algorithms (SODA'93), 1993, 271-280

[2] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu: "An Optimal Algorithm for Approximate Nearest Neighbor Searching", Journal of the ACM, 45 (1998), 891-923