Izrada K-D stabla za pronalazak najbližeg susjeda u 3D oblaku točaka

Radošević, Stefan

Undergraduate thesis / Završni rad

2016

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Faculty of Electrical Engineering, Computer Science and Information Technology Osijek / Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:200:141782

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-04-19**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

SIPA JURJA STROSSMAYERA U OSIJEKU FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Sveučilišni studij

IZRADA K-D STABLA ZA PRONALAZAK NAJBLIŽEG SUSJEDA U 3D OBLAKU TOČAKA

Završni rad

Stefan Radošević

Osijek, 2016.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

Obrazac Z1P - Obrazac za ocjenu završnog rada na preddiplomskom sveučilišnom studiju

Osijek, 23.09.2016.

Odboru za završne i diplomske ispite

Prijedlog ocjene završnog rada

Ime i prezime studenta:	Stefan Radošević
Studij, smjer:	Preddiplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	R3524, 02.08.2013.
OIB studenta:	48283494050
Mentor:	Doc.dr.sc. Ivan Aleksi
Sumentor:	
Naslov završnog rada:	Izrada K-D stabla za pronalazak najbližeg susjeda u 3D oblaku točaka
Znanstvena grana rada:	**Obradba informacija (zn. polje računarstvo)**
Predložena ocjena završnog rada:	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 Postignuti rezultati u odnosu na složenost zadatka: 3 Jasnoća pismenog izražavanja: 1 Razina samostalnosti: 3
Datum prijedloga ocjene mentora:	23.09.2016.
Datum potvrde ocjene Odbora:	28.09.2016.

Potpis mentora za predaju konačne verzije rada u Studentsku službu pri završetku studija:	Potpis:
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA
I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O ORIGINALNOSTI RADA

Osijek, 28.09.2016.

Ime i prezime studenta:	Stefan Radošević
Studij:	Preddiplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	R3524, 02.08.2013.
Ephorus podudaranje [%]:	12

Ovom izjavom izjavljujem da je rad pod nazivom**: Izrada K-D stabla za pronalazak najbližeg susjeda u 3D oblaku točaka**

izrađen pod vodstvom mentora Doc.dr.sc. Ivan Aleksi

i sumentora

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

SADRŽAJ

1. UVOD 1
1.1.Zadatak završnog rada 1
2. KORIŠTENI ALATI I METODOLOGIJE 2
2.1 Programski jezik C# 2
2.2. Metoda izrade K-D stabla 4
3. ZAKLJUČAK 17
LITERATURA 18
SAŽETAK 19
ŽIVOTOPIS 20
PRILOG 21

1. UVOD

Cilj ovog završnog rada je izrada K – dimenzijskoga stabla. K-D stablo je struktura podataka koja omogućuje ubrzano pronalaženje susjedne točke za nejednoliko razmještene točke u oblaku (skupu) točaka. Naposljetku je na primjeru pokazan postupak pronalaženja najbližega susjeda. K-D stablo je primjenjivo u postupku praćenja zrake, kod uklanjanja geometrije s obzirom na volumen pogleda, detekciju kolizije i u drugim područjima. Glavni dio rada podijeljen je u dva poglavlja: *Programski jezik C#* i *K-D stablo.* U poglavlju *Programski jezik C#* ukratko je opisan jezik C#, njegov razvoj, sintaksa te platforma .NET Framework bez koje je nemoguće pokrenuti programe pisane u C# jeziku. U poglavlju *K-D stablo* opisano je K-dimenzijsko stablo, što je ono i kako se konstruira. Također, objašnjen je algoritam pronalaska najbližega susjeda i prikazan je postupak izrade istog algoritma primijenjenoga u nejednolikom oblaku točaka.

1.1.Zadatak završnoga rada

U ovom završnom radu potrebno je izraditi računalnu aplikaciju za učitavanje i prikaz oblaka točaka te izračun k-dimenzijskoga stabla. K-D stablo je struktura podataka koja omogućuje ubrzano pronalaženje susjedne točke za nejednoliko razmještene točke u oblaku (skupu) točaka.

2. KORIŠTENI ALATI I METODOLOGIJE

C# je jedan od mlađih programskih jezika. Nastao je 2002. godine kao sastavni dio Microsoft razvojnog okruženja .NET Framework 1.0. C# je objektno orijentirani programski jezik kao i većina modernih viših programskih jezika poput C++ ili Java. Jezik je opće primjene i namijenjen je izradi aplikacija za .NET Framework platformu.

C# u kombinaciji s .NET nudi mogućnost pisanja aplikacija za različite platforme kao što su mobilni uređaji, desktop računala ili web servere. C# je pogodan za pisanje aplikacija za središnje i ugrađene sustave, od veoma velikih koji koriste sofisticirane operativne sustave, do veoma malih koji imaju posvećenu funkciju. C# je dobio ime koje znači povišeni C kao analogija na tonske povisilice čime se htjelo ukazati na napredne mogućnosti u odnosu na C i C++.

2.1 Programski jezik C#

Tokom razvoja .NET Framework, klasne biblioteke bile su originalno pisane korištenjem sustava kompajlera upravljanog koda zvanog Simple Managed C (SMC). U siječnju 1999. Anders Hejlsberg je formirao tim za razvoj novoga jezika svojevremeno zvanog Cool, koji je bio skraćenica za "C-like Object Oriented Language". Microsoft je namjeravao zadržati ime "Cool" kao krajnje ime jezika, ali ga nije izabrao iz pravnih razloga. Vremenom .NET projekt je javno objavljen u julu 2000. na stručnoj razvojnoj konferenciji, ime je promijenjeno u C#, a klasne biblioteke su, kao i ASP.NET runtime, ubačene u C#.

Tab. 2.1 Razvoj C#

Verzija	**C# 1.0**	**C# 2.0**	**C# 3.0**	**C# 4.0**	**C# 5.0**	**C# 6.0**
Datum objavljivanja	Siječanj 2002.	Studeni 2005.	Studeni 2007.	Travanj 2010.	Kolovoz 2012.	Bit će objavljeno

Sržna sintaksa C# jeste slična onima iz ostalih C-stilskih jezika kao što su C, C++ i Java. U biti:

- Točka-zarez se koristi da se opiše kraj tvrdnje
- Zaobljene zagrade se koriste da grupiraju tvrdnje. Tvrdnje su obično grupirane u metode, metode u klase, a klase u imenski prostor (namespaces)
- Varijable su pridružene korištenjem znaka jednakosti, ali u usporedbi korištenjem dva uzastopna znaka jednakosti
- Oštre zagrade se koriste s nizovima, da bi ih deklarirale i da bi se uzele vrijednosti danog indeksa jednoga od njih

Microsoft .NET Framework je softverska platforma koja može biti instalirana na računalima koje pokreće Microsoft Windows operacijski sustav. On uključuje veliki broj gotovih biblioteka i virtualni stroj koji upravlja izvršavanjem programa pisanih specijalno za .NET Framework. .NET podržava više programskih jezika tako što omogućava interoperabilnost pri čemu svaki jezik mora biti napisan na drugom. Dostupan je na svim programskim jezicima koje .NET Framework obuhvaća. Kako bi aplikacije mogle biti pisane potrebno je, osim .NET Framework, instalirati Microsoft SDK (Software development kit) i Visual studio. Bazne klase pružaju širok spektar mogućnosti, uključujući korisničko sučelje, pristup podatcima, bazama, kriptografiju, razvoj web - aplikacija, numeričke algoritme i mrežne komunikacije. Biblioteke klasa se koriste od strane programera koji ga kombiniraju sa svojim kodom za izradu aplikacija. Programi pisani za .NET Framework izvršavaju se u specifičnom softverskom okruženju. To okruženje je poznato kao Common Languange Runtime (CLR). CLR osigurava izgled virtualnoga stroja ili aplikacije tako da programeri ne trebaju razmatrati mogućnosti specifičnih procesora koji će izvršiti program. On također pruža druge važne usluge kao što su sigurnost, upravljanje memorijom i rukovanje iznimkama. Biblioteke klasa (Framework Class Library) i CLR zajedno čine .NET Framework.

Tab. 2.2 Razvoj .NET Framework

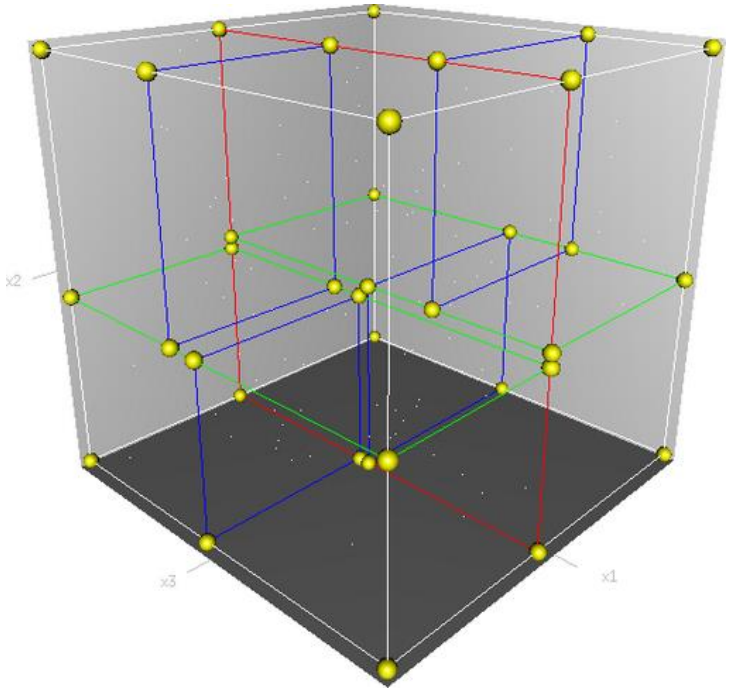
Verzija	1.0	1.1	2.0	3.0	3.5	4.0	4.5
Datum objavljivanja	Veljača 2002.	Travanj 2003.	Studeni 2005.	Studeni 2006.	Studeni 2007.	Travanj 2010.	Kolovoz 2012.

Windows Presentation Foundation (WPF) je grafički podsustav za pružanje korisničkih sučelja Windows-base aplikacija. Ranije je poznat kao Avalon i u početku je izdan kao dio .NET Framework 3.0. WPF koristi DirectX. WPF koristi XAML, što je XML-base jezik, da bi definirao i povezao različite elemente sučelja. WPF aplikacije se mogu pokrenuti kao samostalne desktop aplikacije ili kao ugrađeni objekt u web stranicu. WPF ima za cilj objediniti niz elemenata zajedničkih korisničkih sučelja, kao što su 2D / 3D renderiranje, fiksni i promjenjivi dokumenti, animacije, mediji... Ovi elementi se mogu povezati i manipulirati temeljeno na različitim događajima, interakcijama korisnika i podatkovnim vezama. WPF run-time biblioteke su uključene u svim verzijama Microsoft Windows od Windows Viste i Windows Server 2008. Nakon uspjeha markup jezika za izradu web stranica, WPF uvodi eXtensible Application Markup Language, koji se temelji na XML-u. XAML je osmišljen kao učinkovitija metoda za razvoj aplikacija korisničkih sučelja. Specifična prednost koju XAML donosi WPF je ta da je XAML potpuno deklarativni jezik. Korištenjem XAML za razvoj korisničkih sučelja omogućeno je odvajanje modela i pogleda, što se smatra dobrim arhitektonskim principom.

2.2. Metoda izrade K-D stabla

K-D stablo (K-dimenzijsko stablo) u informatici predstavlja strukturu podataka koja dijeli prostor i organizira točke u K-dimenzijskom prostoru. K-D stabla su veoma korisna struktura podataka za mnoge aplikacije kao što su pretraživanja koja uključuju višedimenzijsku pretragu ključeva. K-D stablo je posebna vrsta binarnog stabla. K-D stablo je binarno stablo u kome je svaki čvor K-dimenzijska točka. Svaki čvor može se promatrati kao implicitno generirana podjela hiperravnine koja dijeli prostor na dva dijela, poznata kao polu-prostor. Točke sa lijeve strane ove hiperravnine

predstavljaju lijevo podstablo, a točke s desne desno podstablo K-D stabla. Pravac hiperravnine je izabran na sljedeći način: svaki čvor u stablu je povezan s jednom od K dimenzija, s hiperravnine koja je normalna na osu te dimenzije. Tako, na primjer, ako je za određenu podjelu uzeta osa „x“, u lijevom podstablu će se nalaziti čvorovi s manjom vrijednošću „x“, a u desnom podstablu čvorovi s većom vrijednošću „x“. U tom slučaju hiperravniva će biti postavljena od strane x-vrijednosti točke, a njena normala će biti x-osa.



Sl. 2.1 K-D stablo

Postoji mnogo različitih načina na koje se može napraviti K-D stablo. Kanonski način konstruiranja K-D stabla ima sljedeća ograničenja:

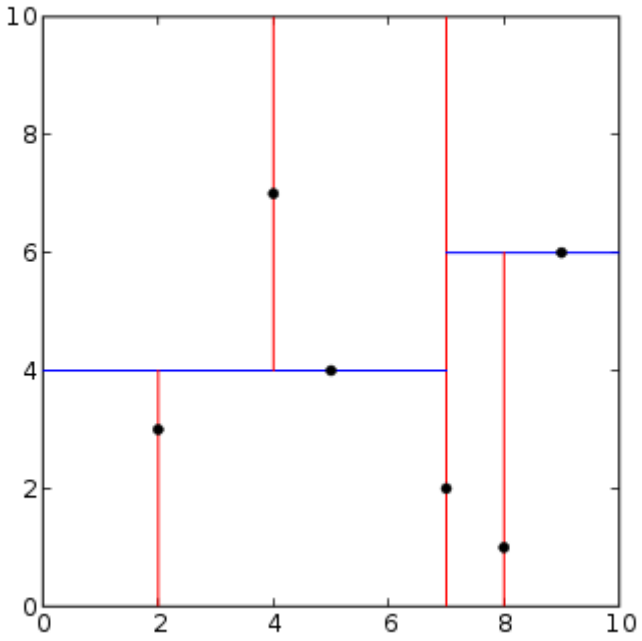
- Kako se krećemo niz drvo, jedna petlja se koristi za određivanje ravni
- Čvorovi se ubacuju u stablo izborom točke srednje vrijednosti korijena i zatim se točke ubacuju u lijevo, odnosno desno podstablo, uzimajući u obzir njihove vrijednosti.

Ova metoda dovodi do stvaranja balansiranog K-D stabla, u kojem je svaki list na istom rastojanju od korijena. Međutim, izbalansirano stablo ne mora biti optimalno za sve aplikacije. Treba imati na umu da nije potrebno izabrati srednju točku. U tom slučaju nema jamstva da će rezultat biti balansirano stablo. Najčešće se koristi jednostavan algoritam sortiranja (O(n logn)) radi nalaženja

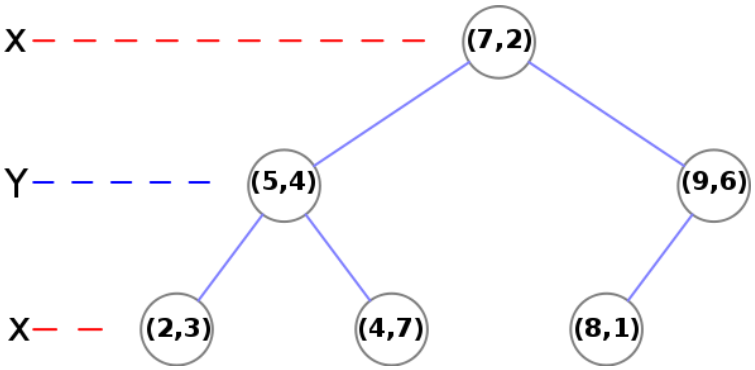
točke sa srednjom vrijednošću koja će služiti za dijeljenje ravnine. U primjeni ova tehnika često dovodi do stvaranja izbalansiranoga stabla.

Balansiranje K-D stabla zahtijeva pažnju jer su čvorovi sortirani u više dimenzija te se rotacija drveta ne može koristiti za balansiranje, jer može izazvati pucanje invarijante.

Postoji nekoliko načina balansiranja K-D stabla. Ona uključuje podijeljeno K-D stablo, pseudo K-D stablo, KDB stablo, HB stablo i BKD stablo. Mnogi od ovih načina su adaptacija K-D stabla.



Sl. 2.2 K-D stablo



Sl. 2.3 K-D stablo

Algoritam traženja najbližega susjeda ima za cilj pronaći čvor u drvetu koji je najbliži danom čvoru. Ova pretraga se može učinkovito uraditi uz pomoć osobine stabla da se brzo eliminiše velike dijelove prostora za pretragu.

Algoritam se obavlja na sljedeći način:

- Počevši od korijena algoritam se pomjera na dole u stablu rekurzivno, na isti način kao kod pretrage toka unosa novoga čvora u stablo
- Kada algoritam dostigne list, pamti taj list kao „trenutni najbolji“
- Algoritam se rekurzivno vraća kroz stablo izvršavajući sljedeće korake na svakom čvoru:
    - Ako je trenutni čvor bliži zadanom čvoru od „trenutno najboljeg“, on postaje „trenutni najbolji“
    - Zatim algoritam provjerava postoji li čvor bliži traženomu čvoru tako što formira sferu oko trenutnog čvora. Ako postoji točka u toj sferi, onda ona postaje „trenutno najbolja“
    - Kada se algoritam završi za korijeni čvor, proces pretrage je gotov

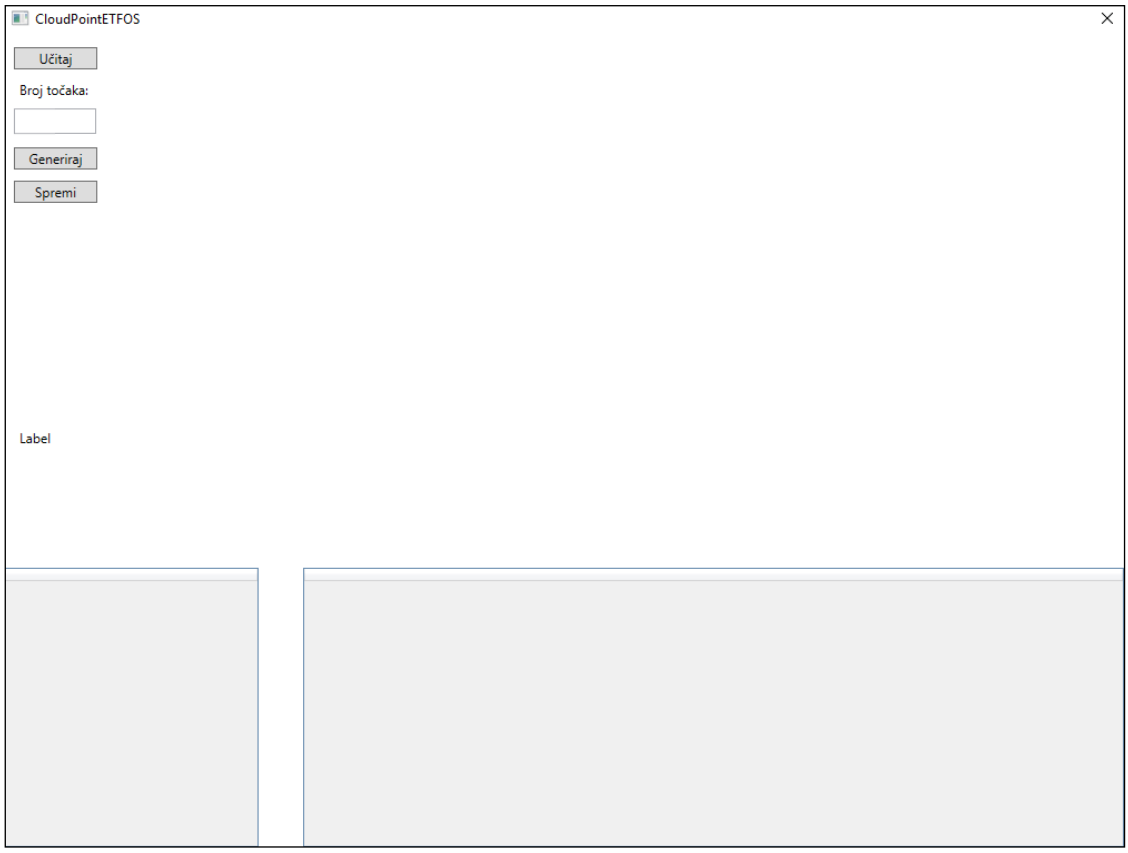
Generalno ovaj algoritam koristi kvadrat rastojanja za usporedbu, da bi se izbjeglo računanje kvadratnog korijena. Pored toga, može se uštedjeti i čuvajući kvadratno rastojanje „trenutnog najboljeg“, koje se koristi za usporedbu. Pronalaženje najbližeg čvora ima O(log N) operacija u slučaju raspoređenih točaka. Međutim, tvrdi se da algoritam daje složenost O(log N) u bilo kojem slučaju.

Analiza binarnog stabla pretrage je utvrdila da je najgori slučaj vremena obim pretrage u K-D stablu koje sadrži N čvorova dat sljedećom jednadžbom:

$$\boldsymbol{t = O(kN^{1-\frac{1}{k}})} \tag{1}$$

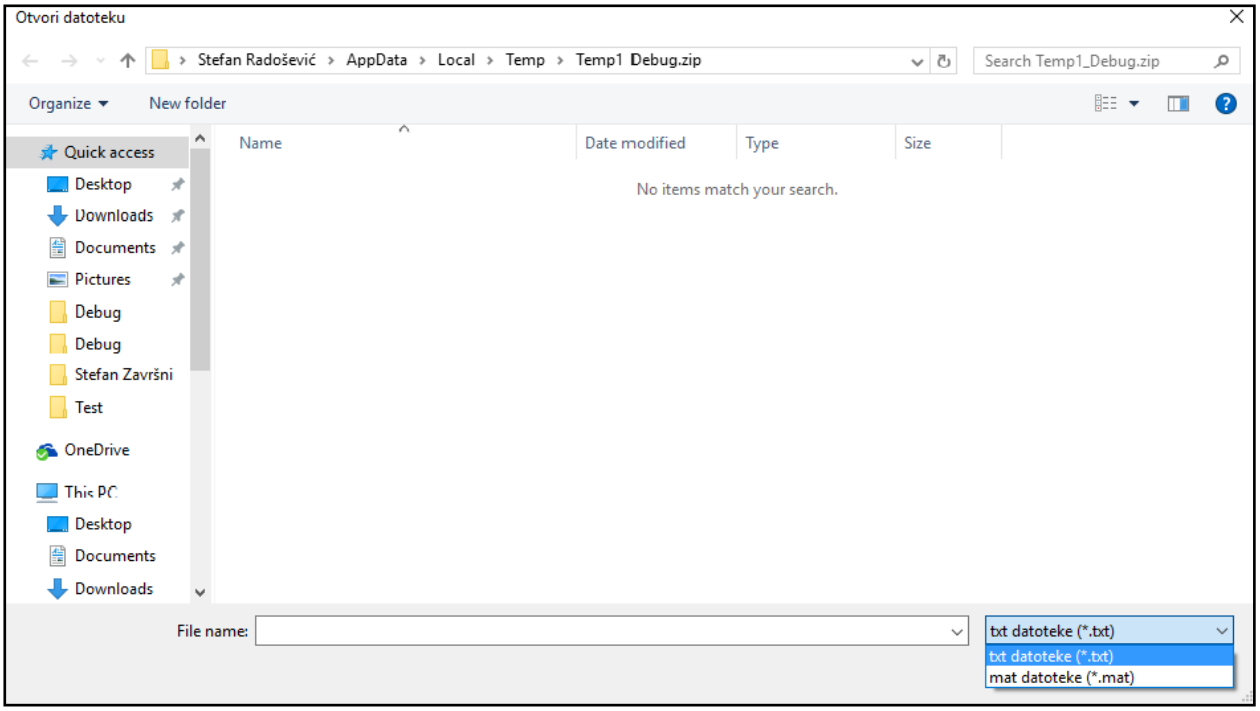
Po pravilu, ukoliko je prostor K-dimenzijski, a broj čvorova u stablu N, trebalo bi da bude $N>>2^k$.

Pomoću C# programskog jezika i WPF grafičkog podsustava realiziran je program. Izgled početnog prozora programa prikazan slikom 2.4.



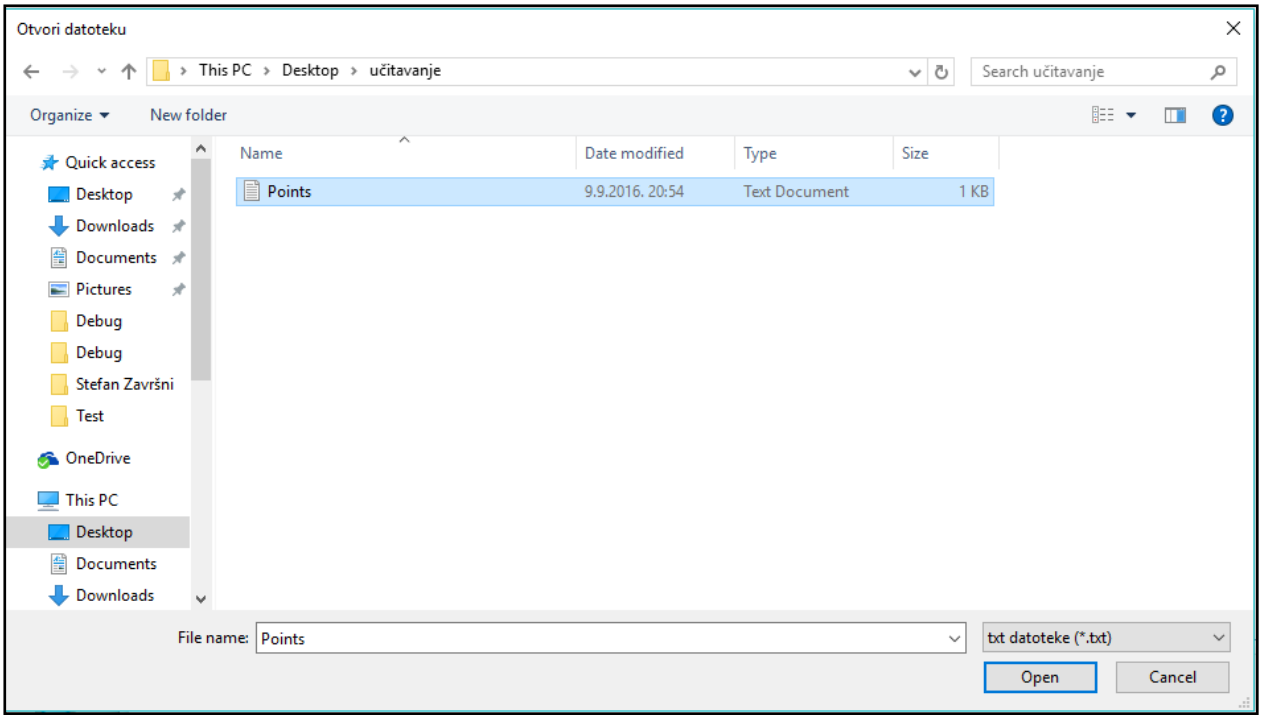
Sl. 2.4.Početni prozor programa

Postoje tri gumba. Gumb Učitaj, Generiraj i Spremi. Pritiskom na gumb Učitaj otvara se novi prozor prikazan slikom 2.5.

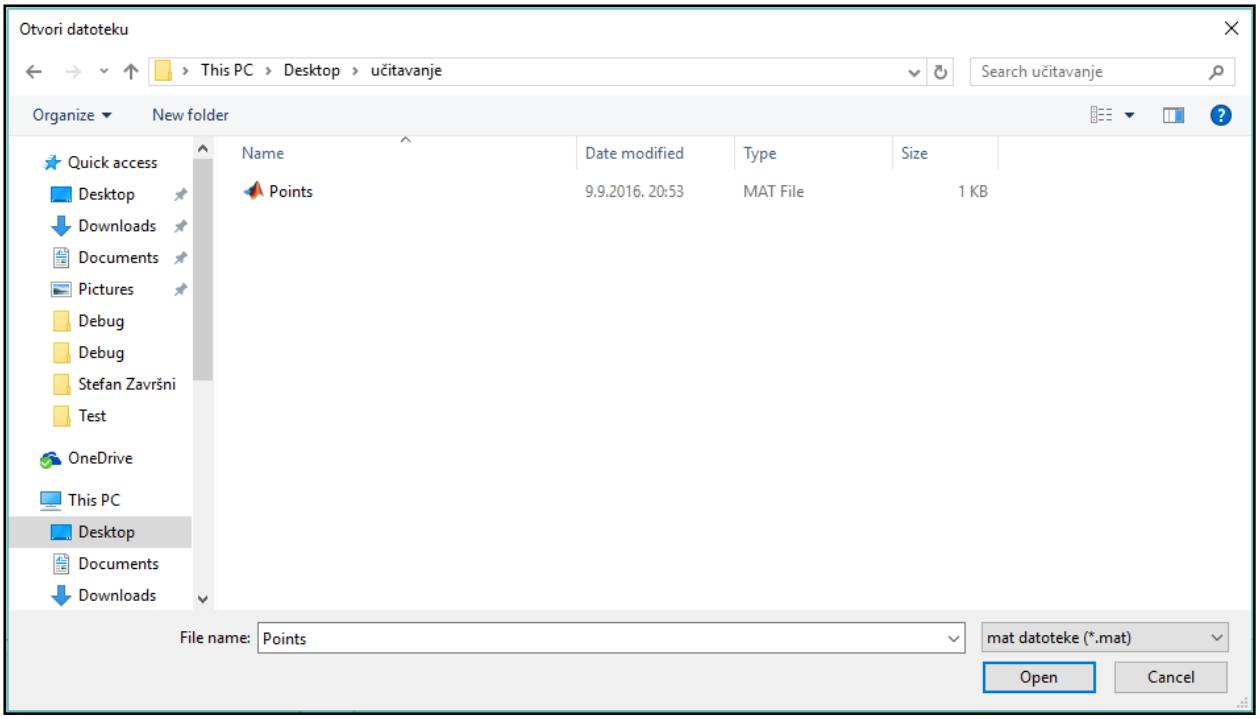


Sl. 2.5.Prozor pokrenut pritiskom na gumb Učitaj

Gumb Učitaj omogućuje učitavanje podataka iz tekstualne datoteke prikazane na slici 2.6. ili iz programskog jezika matlab prikazanoga na slici 2.7. (.txt ili .mat) u kojima se nalaze unaprijed poznate koordinate točaka oblaka točaka.

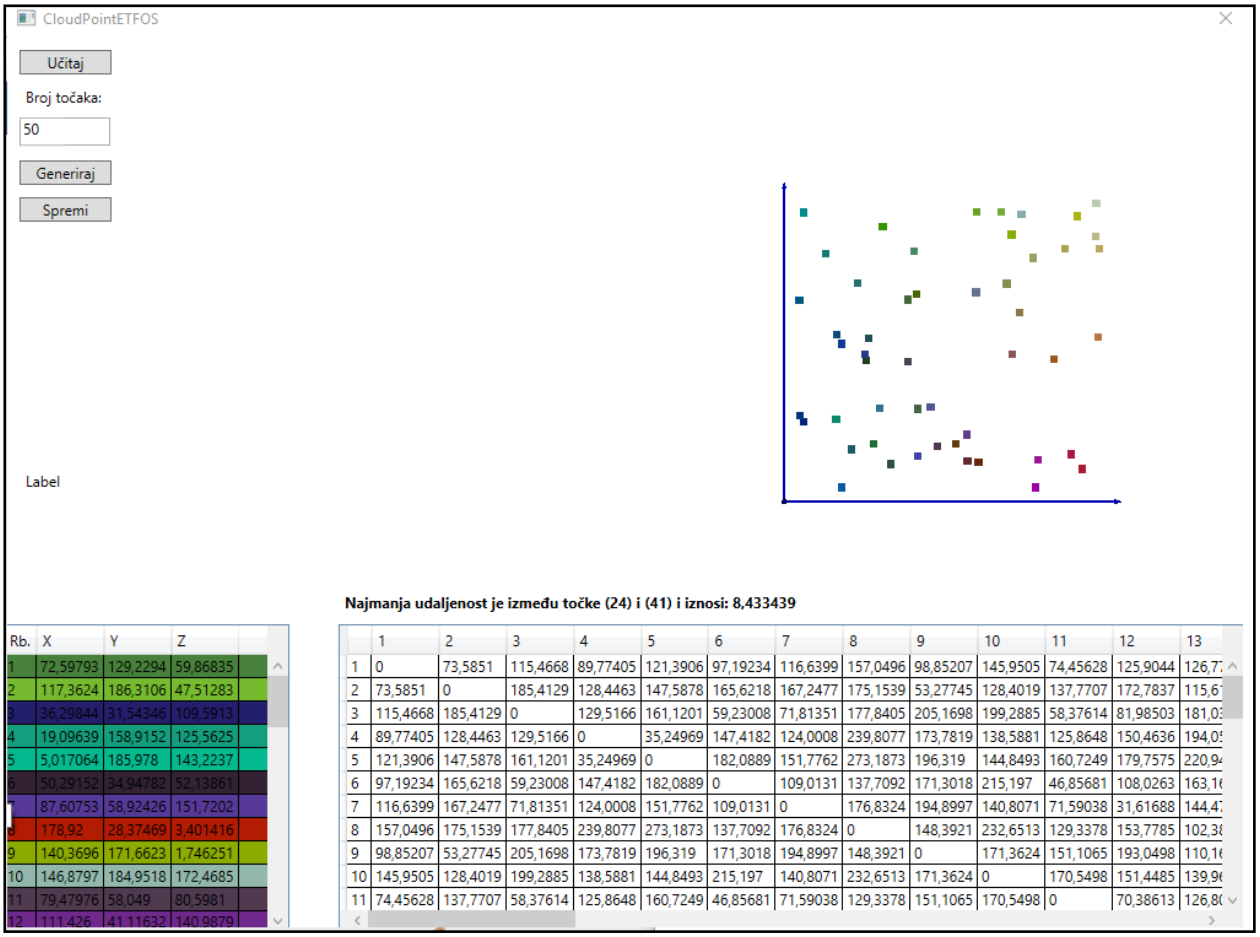


Sl. 2.6. Odabrane točke spremljene u .txt



Sl. 2.7. Odabrane točke spremljene u .mat

Ukoliko ne postoje podatci koje je moguće učitati, moguće je unijeti željeni broj točaka i pritiskom na gumb Generiraj pojavit će se željeni broj točaka.

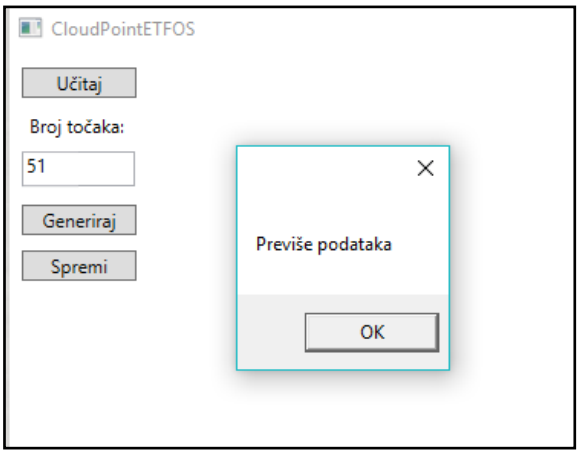


Rb.	X	Y	Z
1	72,59793	129,2294	59,86835
2	117,3624	186,3106	47,51283
3	36,29844	31,54346	109,5913
4	19,09639	158,9152	125,5625
5	5,017064	185,978	143,2237
6	50,29152	34,94782	52,13861
7	87,60753	58,92426	151,7202
8	178,92	28,37469	3,401416
9	140,3696	171,6623	1,746251
10	146,8797	184,9518	172,4685
11	79,47976	58,049	80,5981
12	111,426	41,11632	140,9879

	1	2	3	4	5	6	7	8	9	10	11	12	13
1	0	73,5851	115,4668	89,77405	121,3906	97,19234	116,6399	157,0496	98,85207	145,9505	74,45628	125,9044	126,7
2	73,5851	0	185,4129	128,4463	147,5878	165,6218	167,2477	175,1539	53,27745	128,4019	137,7707	172,7837	115,6
3	115,4668	185,4129	0	129,5166	161,1201	59,23008	71,81351	177,8405	205,1698	199,2885	58,37614	81,98503	181,0
4	89,77405	128,4463	129,5166	0	35,24969	147,4182	124,0008	239,8077	173,7819	138,5881	125,8648	150,4636	194,0
5	121,3906	147,5878	161,1201	35,24969	0	182,0889	151,7762	273,1873	196,319	144,8493	160,7249	179,7575	220,9
6	97,19234	165,6218	59,23008	147,4182	182,0889	0	109,0131	137,7092	171,3018	215,197	46,85681	108,0263	163,1
7	116,6399	167,2477	71,81351	124,0008	151,7762	109,0131	0	176,8324	194,8997	140,8071	71,59038	31,61688	144,4
8	157,0496	175,1539	177,8405	239,8077	273,1873	137,7092	176,8324	0	148,3921	232,6513	129,3378	153,7785	102,3
9	98,85207	53,27745	205,1698	173,7819	196,319	171,3018	194,8997	148,3921	0	171,3624	151,1065	193,0498	110,1
10	145,9505	128,4019	199,2885	138,5881	144,8493	215,197	140,8071	232,6513	171,3624	0	170,5498	151,4485	139,9
11	74,45628	137,7707	58,37614	125,8648	160,7249	46,85681	71,59038	129,3378	151,1065	170,5498	0	70,38613	126,8

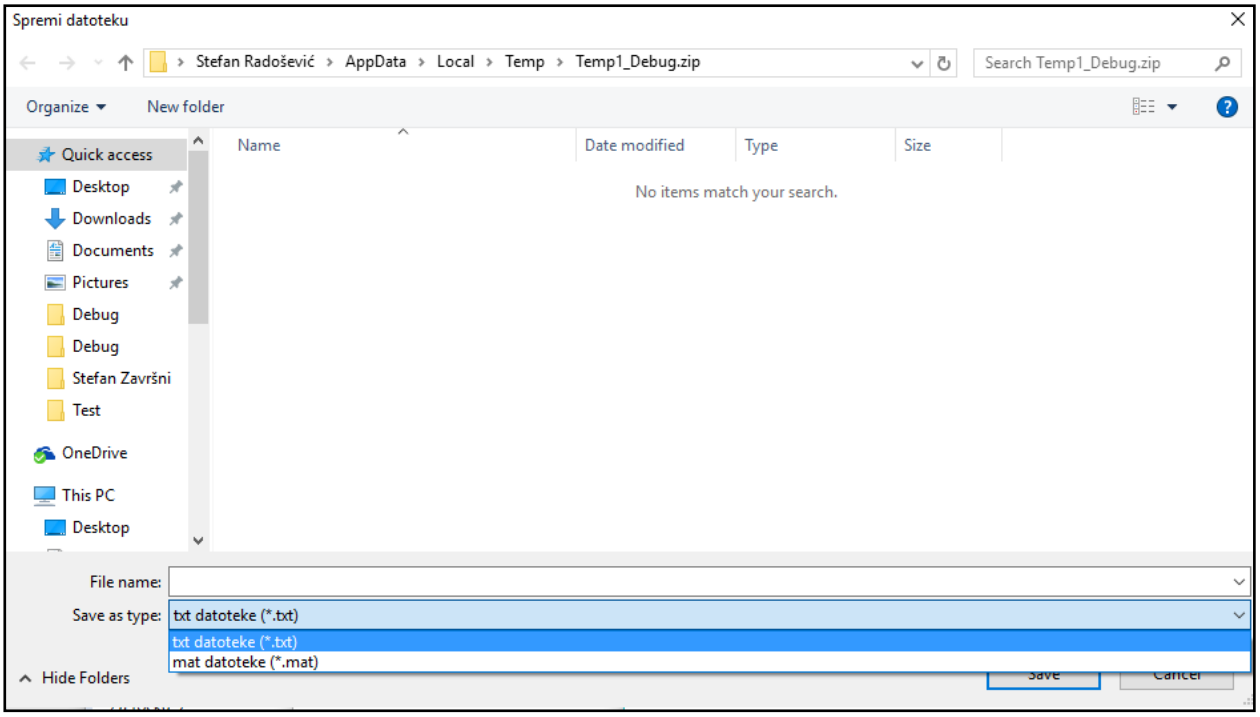
Sl. 2.8. Promjena početnog prozora nakon generiranja točaka

U donjem, lijevom dijelu slike 2.8. nalazi se tablica s koordinatama generiranih točaka, dok je u gornjem, desnom dijelu slike 2.8. prikazan oblak generiranih točaka. Svaka točka obojana je drugom bojom. Broj točaka u ovom programu ograničen je na 50. U slučaju unošenja većeg broja od 50 prikazat će se poruka da željena radnja nije moguća kao na slici 2.9.



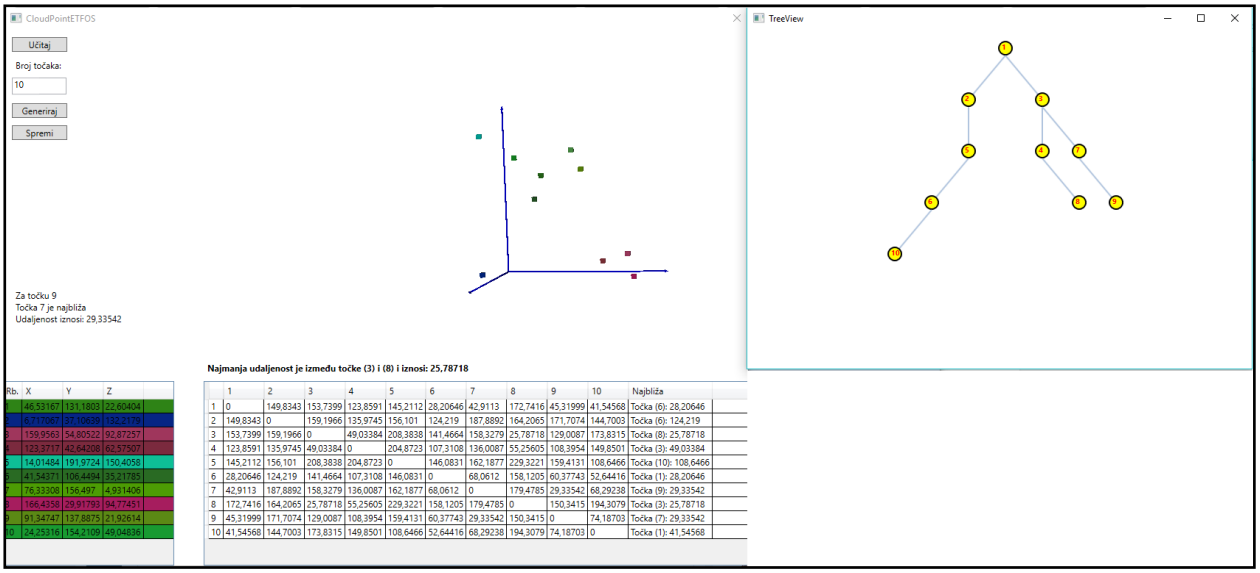
Sl. 2.9. Prozor upozorenja u slučaju prelaska ograničenja

Ukoliko je potrebno spremiti generirane točke dovoljno je pritisnuti na gumb Spremi. Pritiskom na gumb Spremi otvara se novi prozor prikazan slikom 2.10.



Sl. 2.10. Odabir formata za spremanje točaka

Prilikom spremanja generiranih točaka postoje dvije opcije. Prva opcija je željene točke spremiti kao .txt datoteku ili kao .mat datoteku. Spremljene je podatke, po potrebi, moguće ponovno učitati pomoću gumba Učitaj. Učitavanjem ili generiranjem točaka popuni se tablica s koordinatama točaka, tablica s udaljenostima između pojedinih točaka, oblak točaka te K-D stablo.



Sl. 2.11.

Svaka točka obojana je drugom bojom radi lakšeg prepoznavanja u oblaku točaka. Redni broj i boja pojedine točke, kao i njene koordinate prikazane su u tablici prikazanoj slikom 2.12.

Rb.	X	Y	Z
1	46,53167	131,1803	22,60404
2	6,717067	37,10639	132,2179
3	159,9563	54,80522	92,87257
4	123,3717	42,64208	62,57507
5	14,01484	191,9724	150,4058
6	41,54371	106,4494	35,21785
7	76,33308	156,497	4,931406
8	166,4358	29,91793	94,77451
9	91,34747	137,8875	21,92614
10	24,25316	154,2109	49,04836

Sl. 2.12. Tablica točaka

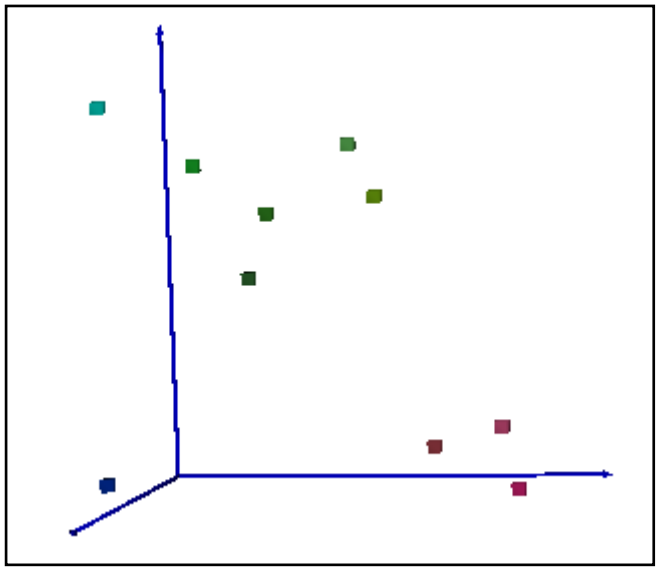
Desno od tablice točaka nalazi se tablica s udaljenostima točaka međusobno.

Najmanja udaljenost je između točke (3) i (8) i iznosi: 25,78718

	1	2	3	4	5	6	7	8	9	10	Najbliža
1	0	149,8343	153,7399	123,8591	145,2112	28,20646	42,9113	172,7416	45,31999	41,54568	Točka (6): 28,20646
2	149,8343	0	159,1966	135,9745	156,101	124,219	187,8892	164,2065	171,7074	144,7003	Točka (6): 124,219
3	153,7399	159,1966	0	49,03384	208,3838	141,4664	158,3279	25,78718	129,0087	173,8315	Točka (8): 25,78718
4	123,8591	135,9745	49,03384	0	204,8723	107,3108	136,0087	55,25605	108,3954	149,8501	Točka (3): 49,03384
5	145,2112	156,101	208,3838	204,8723	0	146,0831	162,1877	229,3221	159,4131	108,6466	Točka (10): 108,6466
6	28,20646	124,219	141,4664	107,3108	146,0831	0	68,0612	158,1205	60,37743	52,64416	Točka (1): 28,20646
7	42,9113	187,8892	158,3279	136,0087	162,1877	68,0612	0	179,4785	29,33542	68,29238	Točka (9): 29,33542
8	172,7416	164,2065	25,78718	55,25605	229,3221	158,1205	179,4785	0	150,3415	194,3079	Točka (3): 25,78718
9	45,31999	171,7074	129,0087	108,3954	159,4131	60,37743	29,33542	150,3415	0	74,18703	Točka (7): 29,33542
10	41,54568	144,7003	173,8315	149,8501	108,6466	52,64416	68,29238	194,3079	74,18703	0	Točka (1): 41,54568

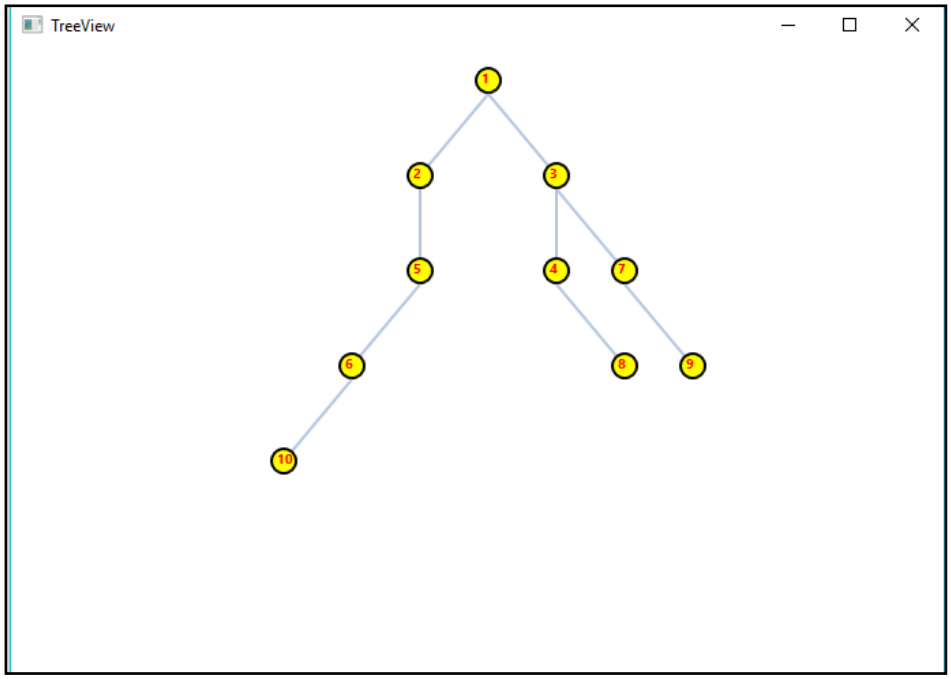
Sl. 2.13. Tablica udaljenosti

Iznad tablice udaljenosti nalazi se 3D prikaz oblaka točaka koji je moguće rotirati po potrebi.



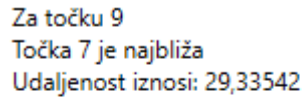
Sl. 2.14. Oblak točaka

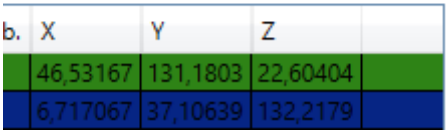
U posebnom prozoru prikazano je K-D stablo.



Sl. 2.15. K-D stablo učitanih točaka

Klikom na željenu točku u K-D stablu pokreće se algoritam pronalaska najbližeg susjeda. Najbliži susjed željene točke te udaljenost između te dvije točke prikazana je u početnom prozoru.





Sl. 2.16. Najbliža točka i udaljenost između najbližeg susjeda i odabrane točke

Na K-D stablu prikazanom na slici 2.15. odabrana je točka pod rednim brojem 9 čije se koordinate nalaze u tablici na slici 2.12. (x=91,34747 ; y=137,8875 ; z=21,92614) gdje također možemo vidjeti njenu boju te odrediti položaj u oblaku točaka prikazanom na slici 2.14. Točki pod rednim brojem 9 najbliža je točka pod rednim brojem 7 (x=76,33308 ; y=156,497 ; z=4,931406) i udaljenost između te dvije točke iznosi 29,33542 što je prikazano na slici 2.16. Slikom 2.13. prikazana je tablica s udaljenostima između točaka, stoga je moguće provjeriti dobiveni rezultat.

	1	2	3	4	5	6	7	8	9	10	Najbliža
7	42,9113	187,8892	158,3279	136,0087	162,1877	68,0612	0	179,4785	29,33542	68,29238	Točka (9): 29,33542

Sl. 2.17.

Usporedbom rezultata na slikama 2.16. i 2.17. vidljivo je da se rješenja poklapaju, stoga se izvodi zaključak da algoritam pronalaska najbližeg susjeda u K-D stablu funkcionira.

3. ZAKLJUČAK

U ovom završnom radu analizirano je K-D stablo, ostvaren je program za pronalazak najbližeg susjeda u oblaku točaka te izračunavanje njegove udaljenosti od zadane točke. Omogućeno je generiranje nasumičnih točaka kao i učitavanje već postojećih točaka. Postojeće točke moguće je učitati ukoliko su u .txt ili .mat formatu. Ukoliko je potrebno spremiti generirane točke, omogućena je opcija spremi. Generirane točke se mogu spremiti u formatu .txt i .mat. Samim generiranjem ili učitavanjem točaka program pravi K-D stablo te je moguće izabrati željenu točku i saznati njenog najbližeg susjeda. Također je zaključeno da K-D stablo nije pogodno za učinkovito traženje najbližega susjeda u prostoru s mnogo dimenzija.

LITERATURA

[1] C# Programming Guide, https://msdn.microsoft.com/en-us/library/67ef8sbd.aspx, rujan 2016.

[2] C Sharp, https://en.wikipedia.org/wiki/C_Sharp_(programming_language) , rujan 2016.

[3] WPF Tutorial, http://www.wpf-tutorial.com/about-wpf/what-is-wpf/, rujan 2016.

[4] WPF Tutorial, http://www.wpftutorial.net/, rujan 2016.

[5] Windows Presentation Foundation https://translate.google.hr/translate?hl=hr&sl=en&u=https://en.wikipedia.org/wiki/Windows_Presentation_Foundation&prev=search, rujan 2016.

SAŽETAK

Naslov: K-D stablo

U radu je glavni zadatak analizirati K- dimenzijsko stablo i za odabranu točku u oblaku točaka pronaći, bez obzira što je K- dimenzijsko stablo nejednoliko raspoređeno, najbližega susjeda. Opisan je program C# pomoću kojega je zadani zadatak ostvaren. Opisana je platforma .NET Framework koja u kombinaciji s C# nudi mogućnost pisanja aplikacija za različite platforme kao što su mobilni uređaji, desktop računala ili web serveri. Također je objašnjeno K-dimenzijsko stablo, što je ono u biti, način na koji je konstruirano i način na koji se ono pretražuje. Opisan je algoritam za pronalazak najbližega susjeda u K- dimenzijskom stablu te je isti ostvaren u C# jeziku tako što je učitana slika koja predstavlja oblak točaka i ta je slika obrađena.

Ključne riječi: K-D stablo, oblak točaka, .NET Framework, C#

ABSTRACT

Title: K-D tree

The paper's main taks is to analyze K-dimensional tree and for the selected point in point cloud, not matter if K-D tree is unevenly distributed, to find the nearest neighbor. Described a program, called C# which is used for realizing the given task. Then, described the platform .NET Framework, which in combination with C# provides the ability to write applications for various platforms such as mobile devices, desktop computers or web servers. Also, it is explained K-dimensional tree, what it is, the way it is constructed and the way in which can be searched. Descrobed the algorithm for finding nearest neighbors in the K-dimensional tree and that algorithm is realized in the C# programming languange in a way that the loaded image representing the point cloud and the image processing.

Keywords: K-D tree, point cloud, .NET Framework, C#

ŽIVOTOPIS

Stefan Radošević je rođen 26. ožujka 1995. u Vukovaru. Nakon završene osnovne škole (OŠ Siniše Glavaševića) 2009. godine upisao je Tehničku školu Nikole Tesle u Vukovaru. Maturirao je 2013. godine odličnim uspjehom. Nakon završetka srednje škole upisao je Elektrotehnički fakultet u Osijeku.

PRILOG

```
using KDTreeDLL;
using System.Windows;
using System;
using System.Windows.Shapes;
using System.Windows.Media;
using System.Windows.Controls;
using static KDTreeDLL.KDTree;
using System.Collections.Generic;
using WPFChart3D;
using System.Windows.Input;

namespace CloudPointETFOS
{
   class TreeView : Window
    {
        private KDTree kdTree;
        private List<ScatterPlotItem> listaTacaka;

        private int elipseWidth = 20;
        private int elipseHeight = 20;

        private int xOffset = 60;
        private int yOffset = 35;

        private Dictionary<KDNode, Point> alreadyDrawn = new Dictionary<KDNode,
Point>();

        public TreeView()
        {
            InitializeComponent();
        }

        public void setTreeAndDraw(KDTree tree, List<ScatterPlotItem> lista)
        {
            kdTree = tree;
            listaTacaka = lista;
            treeCanvas.Children.Clear();
            alreadyDrawn.Clear();
            drawTree();
        }

        private void drawTree()
        {
            var startingPoint = new Point(Width / 2, 25);
            var root = kdTree.getRootNode();
            alreadyDrawn.Add(root, startingPoint);
            drawNodeRecursive(root, startingPoint, null, 0);
        }

        private void drawNodeRecursive(KDNode node, Point point, bool? left, int
depth)
        {
            Ellipse elipsa = new Ellipse()
            {
                Fill = new SolidColorBrush() { Color = Color.FromArgb(255, 255, 255,
0) },
```

```
                StrokeThickness = 2,
                Stroke = Brushes.Black,
                Width = elipseWidth,
                Height = elipseHeight
            };

            TextBlock tekst = new TextBlock()
            {
                Foreground = new SolidColorBrush(Colors.Red),
                Text = (listaTacaka.IndexOf((ScatterPlotItem)node.v) + 1).ToString(),
                FontSize = 10,
                FontWeight = FontWeights.Bold
            };

            var linijaLevo = new Line()
            {
                Stroke = Brushes.LightSteelBlue,
                StrokeThickness = 2
            };

            var linijaDesno = new Line()
            {
                Stroke = Brushes.LightSteelBlue,
                StrokeThickness = 2
            };


            int pomakPoDubini = Math.Abs(depth * 22);
            int xPoint = (int)point.X;
            int yPoint = (int)point.Y;

            int xOffsetLeft =  (left != null && left == false ? 0 : xOffset -
pomakPoDubini);
            int xOffsetRight = (left != null && left == true  ? 0 : xOffset +
pomakPoDubini);

            Canvas.SetLeft(elipsa, xPoint - elipsa.Width / 2);
            Canvas.SetTop(elipsa, yPoint - elipsa.Height / 2);

            Canvas.SetLeft(tekst, xPoint - elipsa.Width / 2 + 7);
            Canvas.SetTop(tekst, yPoint - elipsa.Height / 2 + 5);

            linijaLevo.X1 = xPoint;
            linijaLevo.X2 = xPoint - xOffsetLeft;
            linijaLevo.Y1 = yPoint + elipsa.Height / 2;
            linijaLevo.Y2 = yPoint + yOffset;

            linijaDesno.X1 = xPoint;
            linijaDesno.X2 = xPoint + xOffsetRight;
            linijaDesno.Y1 = yPoint + elipsa.Height / 2;
            linijaDesno.Y2 = yPoint + yOffset;

            var pointLeftTree = new Point(xPoint - xOffsetLeft, yPoint + yOffset);
            var pointRightTree = new Point(xPoint + xOffsetRight, yPoint + yOffset);

            treeCanvas.Children.Add(elipsa);
            treeCanvas.Children.Add(tekst);

            if (node.left != null)
```

```
                treeCanvas.Children.Add(linijaLevo);
            if (node.right != null)
                treeCanvas.Children.Add(linijaDesno);

            if (node.left != null && !alreadyDrawnP(node.left, pointLeftTree))
                drawNodeRecursive(node.left, pointLeftTree, (left != null ? left :
true), depth++);
            if (node.right != null && !alreadyDrawnP(node.right, pointRightTree))
                drawNodeRecursive(node.right, pointRightTree, (left != null ? left :
false), depth++);
        }

        private bool alreadyDrawnP(KDNode left, Point point)
        {
            var contains = alreadyDrawn.ContainsKey(left);
            if(!contains)
                alreadyDrawn.Add(left, point);
            return contains;
        }

        private void treeCanvas_MouseLeftButtonUp(object sender, MouseButtonEventArgs
e)
        {
            if (e.ChangedButton != MouseButton.Left)
                return;
            KDNode tacka = null;
            Point p = Mouse.GetPosition((Canvas)sender);
            foreach (var k in alreadyDrawn.Keys)
            {
                Point item = new Point();
                alreadyDrawn.TryGetValue(k, out item);
                int x1 = (int)(item.X - elipseWidth / 2);
                int y1 = (int)(item.Y - elipseHeight / 2);
                int x2 = (int)(item.X + elipseWidth / 2);
                int y2 = (int)(item.Y + elipseHeight / 2);

                if(p.X > x1 && p.X < x2 && p.Y > y1 && p.Y < y2)
                {
                    tacka = k;
                    break;
                }
            }

            if (tacka != null)
            {
                var t = (ScatterPlotItem)tacka.v;
                ScatterPlotItem najmanjaUdaljenost =
(ScatterPlotItem)kdTree.nearest(new double[] { t.x, t.y, t.z }, 2)[1];
                float udaljen = MainWindow.distanceTwoPoints(t, najmanjaUdaljenost);
                int index = listaTacaka.IndexOf(najmanjaUdaljenost) + 1;
                txtNajbliza.Content = "Za točku " + (listaTacaka.IndexOf(t) + 1) +
"\nTočka " + index + " je najbliža\nUdaljenost iznosi: " + udaljen;
            }
        }
        Label txtNajbliza;
        internal void setTextField(Label label1)
        {
            txtNajbliza = label1;
        }
```

```
    }
}

using csmatio.io;
using csmatio.types;
using KDTreeDLL;
using Microsoft.Win32;
using System;
using System.Collections;
using System.Collections.Generic;
using System.IO;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Media3D;
using WPFChart3D;

namespace CloudPointETFOS
{
    public partial class MainWindow : Window
    {

        public WPFChart3D.TransformMatrix m_transformMatrix = new TransformMatrix();


        private WPFChart3D.Chart3D m_3dChart;
        public int m_nChartModelIndex = -1;
        public int m_nSurfaceChartGridNo = 100;
        public int m_nScatterPlotDataNo = 5000;


        ViewportRect m_selectRect = new ViewportRect();
        public int m_nRectModelIndex = -1;

        private int nDataRange = 200;
        private int defWidth = 5;
        private int defHeight = 5;

        private int maxTacaka = 50;

        private Random randomGenerator = new Random();

        private List<ScatterPlotItem> listaTacaka = new List<ScatterPlotItem>();

        private TreeView treeView = null;

        public MainWindow()
        {
            InitializeComponent();

            dataGridTacke.IsReadOnly = true;
            dataGridUdaljenosti.IsReadOnly = true;
        }

        private void UpdateModelSizeInfo(ArrayList meshs)
        {
            int nMeshNo = meshs.Count;
```

```
            int nChartVertNo = 0;
            int nChartTriangelNo = 0;
            for (int i = 0; i < nMeshNo; i++)
            {
                nChartVertNo += ((Mesh3D)meshs[i]).GetVertexNo();
                nChartTriangelNo += ((Mesh3D)meshs[i]).GetTriangleNo();
            }
        }

        private void TransformChart()
        {
            if (m_nChartModelIndex == -1) return;
            ModelVisual3D visual3d =
(ModelVisual3D)(mainViewport.Children[m_nChartModelIndex]);
            if (visual3d.Content == null) return;
            Transform3DGroup group1 = visual3d.Content.Transform as Transform3DGroup;
            group1.Children.Clear();
            group1.Children.Add(new
MatrixTransform3D(m_transformMatrix.m_totalMatrix));
        }

        public void OnViewportMouseDown(object sender, MouseButtonEventArgs args)
        {
            Point pt = args.GetPosition(mainViewport);
            if (args.ChangedButton == MouseButton.Left)
            {
                m_transformMatrix.OnLBtnDown(pt);
            }
            else if (args.ChangedButton == MouseButton.Right)
            {
                m_selectRect.OnMouseDown(pt, mainViewport, m_nRectModelIndex);
            }
        }

        public void OnViewportMouseMove(object sender, MouseEventArgs args)
        {
            Point pt = args.GetPosition(mainViewport);

            if (args.LeftButton == MouseButtonState.Pressed)
            {
                m_transformMatrix.OnMouseMove(pt, mainViewport);
                TransformChart();
            }
            else if (args.RightButton == MouseButtonState.Pressed)
            {
                m_selectRect.OnMouseMove(pt, mainViewport, m_nRectModelIndex);
            }
        }

        public void OnViewportMouseUp(object sender, MouseButtonEventArgs args)
        {
            Point pt = args.GetPosition(mainViewport);
            if (args.ChangedButton == MouseButton.Left)
            {
                m_transformMatrix.OnLBtnUp();
            }
            else if (args.ChangedButton == MouseButton.Right)
            {
                if (m_nChartModelIndex == -1) return;
```

```
                MeshGeometry3D meshGeometry =
WPFChart3D.Model3D.GetGeometry(mainViewport, m_nChartModelIndex);
                if (meshGeometry == null) return;


                m_3dChart.Select(m_selectRect, m_transformMatrix, mainViewport);


                m_3dChart.HighlightSelection(meshGeometry, Color.FromRgb(200, 200,
200));
            }
        }


        private void btnGeneriraj_Click(object sender, RoutedEventArgs e)
        {
            int nDataNo;
            if(!int.TryParse(txtBrojTocaka.Text, out nDataNo))
            {
                MessageBox.Show("Ne mogu pretvoriti \"" + txtBrojTocaka.Text + "\" u
broj!");
                return;

            }

            if (nDataNo < 2) return;

            if (nDataNo > maxTacaka)
            {
                MessageBox.Show("Previše podataka");
                return;
            }
            RandomPlotGraph(nDataNo);
        }

        private float NextFloat(Random random)
        {
            return (float)((random.NextDouble() * (nDataRange - 0) + 0) %
float.MaxValue);
        }

        private void RandomPlotGraph(int nDataNo)
        {
            Random randomObject = new Random();

            listaTacaka.Clear();

            for (int i = 0; i < nDataNo; i++)
            {
                ScatterPlotItem plotItem = new ScatterPlotItem();

                plotItem.w = defWidth;
                plotItem.h = defHeight;

                plotItem.x = NextFloat(randomObject);
                plotItem.y = NextFloat(randomObject);
                plotItem.z = NextFloat(randomObject);
```

```
                plotItem.shape = (int)Chart3D.SHAPE.BAR;

                addColor(plotItem);

                listaTacaka.Add(plotItem);
            }
            drawGraph();
        }

        private void drawGraph()
        {
            m_3dChart = new ScatterChart3D();
            m_3dChart.SetDataNo(listaTacaka.Count);

            int i = 0;
            foreach (var p in listaTacaka)
                ((ScatterChart3D)m_3dChart).SetVertex(i++, p);


            m_3dChart.GetDataRange();
            m_3dChart.SetAxes();


            ArrayList meshs = ((ScatterChart3D)m_3dChart).GetMeshes();


            UpdateModelSizeInfo(meshs);


            WPFChart3D.Model3D model3d = new WPFChart3D.Model3D();
            m_nChartModelIndex = model3d.UpdateModel(meshs, null, m_nChartModelIndex,
this.mainViewport);


            float viewRange = nDataRange;
            m_transformMatrix.CalculateProjectionMatrix(0, viewRange, 0, viewRange,
0, viewRange, 0.5);
            TransformChart();

            populateListAndCalculateDistance();
        }

        private void populateListAndCalculateDistance()
        {


            if (dataGridTacke.Columns.Count == 0)
            {
                dataGridTacke.Columns.Add(new DataGridTextColumn() { Header = "Rb.",
Binding = new Binding("[0]") });
                dataGridTacke.Columns.Add(new DataGridTextColumn() { Header = "X",
Binding = new Binding("[1]") });
                dataGridTacke.Columns.Add(new DataGridTextColumn() { Header = "Y",
Binding = new Binding("[2]") });
                dataGridTacke.Columns.Add(new DataGridTextColumn() { Header = "Z",
Binding = new Binding("[3]") });
            }

            var lista = new List<object>();
```

```
            int i = 1;
            foreach (var p in listaTacaka)
                lista.Add(new string[4] { i++.ToString(), p.x.ToString(),
p.y.ToString(), p.z.ToString() });

            dataGridTacke.ItemsSource = lista;


            int n = listaTacaka.Count;

            dataGridUdaljenosti.Columns.Clear();
            dataGridUdaljenosti.Columns.Add(new DataGridTextColumn() { Header = "",
Binding = new Binding("[" + n + "]") });
            for (i = 0; i < n; i++)
                dataGridUdaljenosti.Columns.Add(new DataGridTextColumn() { Header =
(i + 1).ToString(), Binding = new Binding("[" + i + "]") });
            dataGridUdaljenosti.Columns.Add(new DataGridTextColumn() { Header =
"Najbliža", Binding = new Binding("[" + (n + 1) + "]") });

            var udaljenosti = new List<object>();


            izracunajUdaljenostKD(udaljenosti);

            dataGridUdaljenosti.ItemsSource = udaljenosti;
        }


        private void izracunajUdaljenostKD(List<object> udaljenosti)
        {
            int n = listaTacaka.Count;
            var distance = new Dictionary<string, float>();
            float basNajmanjaUdaljenost = float.MaxValue;
            string[] tacke = new string[2];

            KDTree kdTree = new KDTree(3);
            foreach (var p in listaTacaka)
                kdTree.insert(new double[] {p.x, p.y, p.z }, p);

            for (int i = 1; i <= n; i++)
                for (int j = i; j <= n; j++)
                    if (!distance.ContainsKey(i.ToString() + "-" + j) && i != j)
                        distance.Add(i.ToString() + "-" + j,
distanceTwoPoints(listaTacaka[i - 1], listaTacaka[j - 1]));


            for (int i = 1; i <= n; i++)
            {
                var red = new string[dataGridUdaljenosti.Columns.Count];
                red[n] = i.ToString();

                for (int j = 1; j <= n; j++)
                {
                    if (i == j)
                    {
                        red[j - 1] = "0";
                        continue;
                    }
```

```
                    var udaljenost = (distance.ContainsKey(i.ToString() + "-" + j) ?
distance[i.ToString() + "-" + j] : distance[j.ToString() + "-" + i]);
                    red[j - 1] = udaljenost.ToString();
                }

                ScatterPlotItem p = listaTacaka[i - 1];
                ScatterPlotItem najmanjaUdaljenost =
(ScatterPlotItem)kdTree.nearest(new double[] { p.x, p.y, p.z }, 2)[1];
                float udaljen = distanceTwoPoints(p, najmanjaUdaljenost);
                int index = listaTacaka.IndexOf(najmanjaUdaljenost) + 1;

                if (udaljen < basNajmanjaUdaljenost)
                {
                    basNajmanjaUdaljenost = udaljen;
                    tacke[0] = i.ToString();
                    tacke[1] = index.ToString();
                }

                red[n + 1] = "Točka (" + index + "): " + udaljen;

                udaljenosti.Add(red);
            }

            if(treeView == null || !treeView.IsVisible)
            {
                treeView = new TreeView();
                treeView.Show();
                treeView.setTextField(label1);
                treeView.WindowStartupLocation = WindowStartupLocation.Manual;
                treeView.Left = Left + Width;
                treeView.Top = Top;
            }

            treeView.setTreeAndDraw(kdTree, listaTacaka);

            txtNajmanjaUdaljenost.Content = "Najmanja udaljenost je između točke (" +
tacke[0] + ") i (" + tacke[1] + ") i iznosi: " + basNajmanjaUdaljenost;
        }



        public static float distanceTwoPoints(ScatterPlotItem p1, ScatterPlotItem p2)
        {
            float deltaX = p2.x - p1.x;
            float deltaY = p2.y - p1.y;
            float deltaZ = p2.z - p1.z;

            float distance = (float)Math.Sqrt(deltaX * deltaX + deltaY * deltaY +
deltaZ * deltaZ);

            return distance;
        }

        private void addColor(ScatterPlotItem p)
        {
            byte nR = (byte)p.x;
            byte nG = (byte)p.y;
            byte nB = (byte)p.z;
```

```
    p.color = Color.FromRgb(nR, nG, nB);
}

private void btnUcitaj_Click(object sender, RoutedEventArgs e)
{
    OpenFileDialog theDialog = new OpenFileDialog()
    {
        Title = "Otvori datoteku",
        Filter = "txt datoteke (*.txt)|*.txt|mat datoteke (*.mat)|*.mat",
        DefaultExt = "*.mat",
        InitialDirectory = AppDomain.CurrentDomain.BaseDirectory
    };

    if (theDialog.ShowDialog() == true)
        ucitajTocke(theDialog.FileName);
}

private void ucitajTocke(string fileName)
{
    listaTacaka.Clear();

    if (fileName.Contains(".mat"))
    {
        MatFileReader mfr = new MatFileReader(fileName);
        MLDouble mlSquares = mfr.Content["Points"] as MLDouble;
        if (mlSquares != null)
            foreach (var point in mlSquares.GetArray())
            {
                var p = new ScatterPlotItem();

                p.w = defWidth;
                p.h = defHeight;

                p.shape = (int)Chart3D.SHAPE.BAR;

                p.x = (float)point[0];
                p.y = (float)point[1];
                p.z = (float)point[2];

                addColor(p);
                Console.WriteLine(p.x + ";" + p.y + ";" + p.z + ";");
                listaTacaka.Add(p);
            }
    }
    else if (fileName.Contains(".txt"))
    {
        var lines = File.ReadAllLines(fileName);
        if (lines.Length > 0)
        {
            foreach(var line in lines)
            {
                var point = line.Split(';');
                if(point.Length > 0)
                {
                    var p = new ScatterPlotItem();

                    p.w = defWidth;
                    p.h = defHeight;
```

```
                    p.shape = (int)Chart3D.SHAPE.BAR;

                    p.x = float.Parse(point[0]);
                    p.y = float.Parse(point[1]);
                    p.z = float.Parse(point[2]);

                    addColor(p);

                    listaTacaka.Add(p);
                }
            }
        }
    }
    else
    {
        MessageBox.Show("Nepodržana vrsta datoteke!");
        return;
    }

    if (listaTacaka.Count > 1)
    {
        drawGraph();
    }
    else MessageBox.Show("Nije učitana nijedna ili samo 1 točka!");
}

private void btnSpremi_Click(object sender, RoutedEventArgs e)
{
    if(listaTacaka.Count == 0)
    {
        MessageBox.Show("Lista ne sadrži nijednu točku!");
        return;
    }

    SaveFileDialog fileDialog = new SaveFileDialog()
    {
        Title = "Spremi datoteku",
        Filter = "txt datoteke (*.txt)|*.txt|mat datoteke (*.mat)|*.mat",
        DefaultExt = "*.mat",
        InitialDirectory = AppDomain.CurrentDomain.BaseDirectory
    };

    if (fileDialog.ShowDialog() == true)
        spremiTocke(fileDialog.FileName);
}

private void spremiTocke(string fileName)
{
    double [][] tacke = new double[listaTacaka.Count][];
    int i = 0;
    foreach(var point in listaTacaka)
    {
        tacke[i] = new double[3];
        tacke[i][0] = point.x;
        tacke[i][1] = point.y;
        tacke[i][2] = point.z;
        i++;
    }
```

```
            if (fileName.Contains(".txt"))
            {
                string [] zaSpremanje = new string[listaTacaka.Count];
                for (i = 0; i < listaTacaka.Count; i++)
                    zaSpremanje[i] = tacke[i][0] + ";" + tacke[i][1] + ";" +
tacke[i][2] + ";";
                File.WriteAllLines(fileName, zaSpremanje);
            } else if (fileName.Contains(".mat"))
            {
                MLDouble list = new MLDouble("Points", tacke );
                MatFileWriter writer = new MatFileWriter(fileName, new
List<MLArray>(){ list }, false);
            } else
            {
                MessageBox.Show("Nepodržana vrsta datoteke!");
                return;
            }
        }

        private void dataGridTacke_LoadingRow(object sender, DataGridRowEventArgs e)
        {
            double r = Convert.ToDouble(((string[])e.Row.DataContext)[1]);
            double g = Convert.ToDouble(((string[])e.Row.DataContext)[2]);
            double b = Convert.ToDouble(((string[])e.Row.DataContext)[3]);

            e.Row.Background = new SolidColorBrush(Color.FromRgb((byte)r, (byte)g,
(byte)b));
        }
    }
}
```