SVEUČILIŠTE U ZAGREBU - GEODETSKI FAKULTET
UNIVERSITY OF ZAGREB - FACULTY OF GEODESY
Zavod za geomatiku; Katedra za geoinformatiku
Institute of Geomatics; Chair of Geoinformatics
Kačićeva 26; HR-10000 Zagreb, CROATIA
Web: www.geoinfo.geof.hr; Tel.: (+385 1) 46 39 222; Fax.: (+385 1) 48 28 081

***Usmjerenje: Geoinformatika, Diplomski studij geodezije i geoinformatike***

DIPLOMSKI RAD

# Obrada prostornih podataka pomoću programskog jezika Python

**Izradio:**

*Marino Čuljat*

*Fužinska 21.*

*51000 Rijeka*

*mculjat@geof.hr*

Mentor: prof. dr. sc. Damir Medak

Zagreb, rujan 2010.

**Zahvala:**

*Zahvaljujem se svome mentoru prof. dr. sc. Damiru Medaku i voditelju moga diplomskog rada asistentu Mariu Mileru dipl. ing. koji su svojim znanstvenim i stručnim savjetima oblikovali ideju i pomogli mi u izradi ovoga diplomskog rada. Također zahvalu upućujem i Geodetskom zavodu Rijeka na ustupljenim podacima korištenima pri izradi diplomskog rada.*

*Posebno se želim zahvaliti svojim roditeljima koji su me tokom čitavog mog školovanja podupirali i poticali moju težnju k ostvarivanju sve viših i viših ciljeva.*

*Želim se zahvaliti i svim djelatnicima Geodetskog fakulteta Sveučilišta u Zagrebu koji su svojim radom pomogli u stjecanju moga znanja o geodeziji i geoinformatici te životu u struci i oko nje.*

*I na kraju želim se zahvaliti svim kolegama koji su mi vrijeme provedeno na fakultetu uljepšali svojim prisustvom i pomogli da to vrijeme smatram najljepšim dijelom svoga života.*

# Obrada prostornih podataka pomoću programskog jezika Python

***Marino Čuljat***

***Sažetak:*** *U ovome radu teoretski i praktično prikazane su neke od mogućnosti obrade prostornih podataka pomoću programskog jezika Python i dodatno instaliranih biblioteka s procedurama za obradu i manipulaciju prostornim podacima. Prikazane su i neke od mogućnosti ArcGIS paketa te je na temelju nekoliko metode obrade (buffer, envelope, centroid) prostornih podataka uspoređena brzina izvođenja. Dobiveni rezultati usporedbe utrošenog vremena pokazuju kako su u sva tri navedena slučaja brže zadatak obavile skripte koje koriste Shapely module. Prezentirane su i osnove izrade skripte u Python programskom jeziku za automatizaciju procesa u ArcGIS paketu.*

***Ključne riječi:*** *prostorni podaci, Python, ArcGIS, Shapely*

## *Geoprocesing with Python*

***Abstract:*** In this paper, a theoretical and practical presents some of the features of spatial data using the Python programming language and further installed with a library of procedures for processing and manipulating spatial data. The paper also presents some features of the ArcGIS package and is based on several methods of treatment (buffer, envelope, centroid) of spatial data compared speed of execution. The results obtained show a comparison of time that in all three cases, the task done faster scripts that use the shapely modules. Presented are the basics of creating a script in Python programming language to automate processes in the ArcGIS package.

***Keywords:*** *geoprocesing, Python, ArcGIS, Shapely, spatial data*

# Obrada prostornih podataka pomoću programskog jezika Python

*Marino Čuljat*

## S A D R Ž A J

# 1. Uvod

U današnje vrijeme u kojem trenutno vlada globalna ekonomska recesija, u kojem zadnjih nekoliko desetaka godina svakim trenutkom raste broj informacija koje imaju prostornu komponentu i u doba kada jedino prodaja informacija unatoč lošim gospodarskim prilikama bilježi rast i profitabilno poslovanje potrebno je okrenuti se prikupljanju i obrađivanju informacija kako bi im se dodale nove vrijednosti te im se na taj način povećala vrijednost i vjerojatnost njihove prodaje. U današnje vrijeme kada geodetsku struku pritišće smanjenje investicija u građevinarske djelatnosti i kada je sve teže pronaći kvalitetan i ekonomski isplativ projekt potrebno se okrenuti besplatnim rješenjima i automatizaciji procesa koji će u konačnici dovesti do veće produktivnosti i donijeti će veće prihode. Potrebno je napomenuti kako je svakim danom sve lakše doći do podataka s prostornom komponentom. Ti podaci su određene točnosti i velika većina ne zadovoljava stroge geodetske uvjete točnosti i preciznosti, no potrebno je istaknuti kako postoji velika količina projekata koji ne zahtijevaju geodetsku preciznost, točnost i kvalitetu podataka već je dozvoljeno puno veće odstupanje. Također, oprema za mjerenje postaje svakim danom sve jeftinija pa tako danas za mjerenje satelitskim metodama uz korištenje CROPOS sustava ne trebate imati komplet za RTK mjerenje koji se sastoji od baze i rovera već vam je dovoljan samo jedan uređaj. S obzirom na količinu i cijenu prikupljanja podataka vidi se da je najskuplji dio izrade projekta radno vrijeme ljudstva koje na projektu radi i cijena nabavke komercijalnih softverskih paketa. Tema ovoga diplomskog rada je dati pregled nad jednim načinom automatizacije proizvodnog procesa u softverskom paketu ArcGIS, pokazati da je besplatnim softverskim rješenjem uz malo truda moguće izvesti veliki broj analiza nad podacima, utvrditi kako je korisnije podatke obrađivati vlastitim softverskim rješenjima jer je moguće dodavati dodatne parametre koji u pojedinim projektima mogu zatrebati, a moguće je da u komercijalnom rješenju nisu predviđeni. Za kraj ono što je središnji dio cijelog diplomskog rada je prikaz ubrzanja procesa upotrebom čistog vlastitog programskog koda naspram komercijalnog koda koji je prilagođen svim unaprijed predviđenim potrebama pa mu treba i višestruko više vremena da odradi jednako zahtjevnu zadaću i izvrši jednak analitički proces.

# 2. Osnovni pojmovi

## *2.1. GDAL*

GDAL - *Geospatial Data Abstraction Library* doslovnim prijevodom naziva radi se o Biblioteci za apstrakciju prostornih podataka. Ustvari GDAL je konverzijska biblioteka (biblioteka prevoditelj) za rasterske prostorne podatke. Izdana je u *Open Source*[1] licenci od strane Open Source Geospatial Foundation[2]. Kao biblioteka, GDAL predstavlja jedan apstraktni model podataka za pozivanje aplikacija za sve podržane formate. U većini slučajeva GDAL dolazi sa raznim korisnim *comandline* (traka za naredbe) uslugama za prijevod i obradu podataka. Unutar GDAL podatkovnog paketa nalazi se i OGR biblioteka koja sadrži slične sposobnosti za vektorske podatke. Više podataka i detaljniji opis same biblioteke možete potražiti na : http://www.gdal.org/index.html

## *2.2. GEOS*

GEOS *Geometry Engine – Open Source* je C++ sučelje *Java Topology Suite* i kao takvo sadrži potpunu funkcionalnost JTS-a izrađenu u C++ što uključuje sve OpenGIS jednostavne objekte za SQL, prostorne predikatne funkcije i prostorne operatore kao i JTS poboljšanje topološke funkcije. Više podataka i detaljniji opis same biblioteke možete potražiti na : http://trac.osgeo.org/geos/

## *2.3. OGR*

OGR *Simple Feature Library* je C++ *open source* biblioteka s *comandline* alatima koja omogućuje pristup, čitanje i pisanje datotekama različitih vektorskih formata među kojima su ESRI Shapefiles, PostGIS, Oracle Spatial, Mapinfo mid/mif i drugi formati. Više podataka i detaljniji opis same biblioteke možete potražiti na : http://www.gdal.org/ogr/

---

[1] Općenito,bilo kakav računalni softver čiji je izvorni programski kodili u javnoj domeni ili vlasnički zaštićen od strane jedne ili više osoba ili organizacija, a distribuira se u režimu licencije za otvoreni kod. (prof. dr. sc. Željko Panian, "Informatički enciklopedijski riječnik", Europapress holding d.o.o., Zagreb 2005.)

[2] Open Source Geospatial Foundation ili OSGeo neprofitna organizacija čiji je cilj podržavati i promicati zajednički razvoj prostornih podataka i tehnologija zasnovanih na otvorenom kodu. (http://www.osgeo.org/content/foundation/about.html 24. kolovoz. 2010.)

### 2.4. JTS

JTS *Java Topology Suite* je API za prostorne predikate i funkcije nad dvodimenzionalnim (2D) podacima. JTS je izrađen sukladno specifikacijama Open GIS Consortium[3], omogućava kompletnu i konzistentnu implementaciju temeljnih 2D prostornih algoritama. Više podataka i detaljniji opis same biblioteke možete potražiti na : http://www.vividsolutions.com/jts/jtshome.htm

### 2.5. Shapefile

ESRI Shapefile ili jednostavnije shapefile je popularni vektorski format prostornih podataka za GIS softver. ESRI ga je razvio kao otvorenu specifikaciju za interoperabilnost podataka između ESRI i softverskih proizvoda.

Shapefile prvenstveno opisuje geometriju koja se sastoji od osnovnih geometrijskih objekata kao što su točka, linija i poligon. Svaki od navedenih objekata sadržava neke od atributa koji ga opisuju bilo da se radilo o rednom broju, duljini, površini ili nekom drugom svojstvu koje je vezano za objekt.

Iako je shapefile uobičajen i jedinstven termin, shapefile se zapravo sastoji od seta datoteka sa različitim ekstenzijama od kojih su tri datoteke obavezne (Tabela 1), a postoji još cijeli niz opcionalnih datoteka (Tabela 2) koje pomažu u snalaženju s podacima, njihovim korištenjem i manipulacijom istih.

*Tabela 1 Tablični prikaz obveznih shapefile ekstenzija i pripadajućih podataka*

| Ekstenzija | Opis podataka |
|---|---|
| **.shp** | shape format, geometrija objekta |
| **.shx** | shape indeks format, indeks koji označuje geometriju i omogućava brzo pretraživanje u oba smjera, sprijeda ili straga |
| **.dbf** | atributni format, atributi poredani u stupce za svaki objekt |

[3] Open Geospatial Consortium je međunarodni industrijski konzorcij u kojem se nalaze 401 kompanija, vladine agencije i sveučilišta koja kensenzualno sudjeluju u razvoju standarda za javno dostupna sučelja… (http://www.opengeospatial.org/ogc 24. kolovoz. 2010.)

Potrebno je napomenuti kako su zapisi u tri, obvezne, gore navedene datoteke međusobno ovisni i to prema položaju zapisa. Prvi zapis u .shx ili .dbf datoteci odnosi se na prvi zapis zapisan u .shp datoteci.

Korisno je istaknuti kako shape file nema sposobnost pohranjivanja topoloških odnosa već se topološki odnosi među objektima dobivaju topološkim analizama a rezultati analiza se zapisuju u nove datoteke.

*Tabela 2 Tablični prikaz obveznih shapefile ekstenzija i pripadajućih podataka*

| ***Ekstenzija*** | ***Opis podataka*** |
|---|---|
| **.shp** | shape format, geometrija objekta |
| **.shx** | shape indeks format, indeks koji označuje geometriju i omogućava brzo pretraživanje u oba smjera, sprijeda ili straga |
| **.dbf** | atributni format, atributi poredani u stupce za svaki objekt |
| **.prj** | projekcijski format, koordinatni sustav i projekcijski parametri |
| **.sbn .sbx** | prostorni indeks objekata |
| **.fbn .fbx** | prostorni indeks objekata za shapefile koji je read-only (samo čitljiv) |
| **.ain .aih** | atributni indeks aktivnih polja u tablici ili tematskoj atributnoj tablici |
| **.ixs** | indeks geokodiranja za shape file koji se može čitat i pisat |
| **.mxs** | indeks geokodiranja za shape file koji se može čitat i pisat u ODB formatu |
| **.atx** | Atributni indeks za .dbf datoteku |
| **.shp.xml** | Metapodaci u XML formatu |
| **.cpg** | Definira korištenu kodnu stranicu za znakove korištene u .dbf datoteci |

# 3. Python

Python je programski jezik koji je 1990. godine prvi razvio Guido van Rossum. To je objektno orijentirani[4], interaktivni[5] i interpreterski[6] jezik kojeg su od 2000. godine prihvatile i ugledne institucije kao što su MIT, NASA, IBM, Google… Koristeći Python nećete naići na nove i revolucionarne ideje i rješenja u programiranju već ćete na jednome mjestu imati na optimalan način ujedinjene ideje i načela rada mnogih drugih programskih jezika. On je snažan i jednostavan istodobno jer je spoj tradicionalnih skriptnih jezika i sistematskih jezika pa na taj način omogućuje programeru lakšu orijentiranost na problem s obzirom da zahtjeva manje razmišljanja o samom jeziku. Potrebno je još napomenuti kako je Python besplatan (za akademske ustanove), open-source softver s izuzetno dobrom potporom, literaturom i dokumentacijom tako da nije problem nabaviti isti i naučiti ga koristiti.

Za veću funkcionalnost Pythona i lakše korištenje istoga potrebno je dodatno instalirati pojedine biblioteke (eng. library[7]). Za potrebe ovog diplomskog rada dodatno su instalirane GDAL i Shapely biblioteke koji omogućavaju rad i manipulaciju s datotekama prostornih podataka i samim prostornim podacima. Instalacijom istih povećana je funkcionalnost programskog jezika jer su se u programiranju mogle koristiti već unaprijed definirane funkcije i operacije pa je samim time program lakše izradit, a i program je pregledniji i kvalitetnije strukturiran. Također upotrebom navedenih biblioteka omogućeno je da se brzim dodavanjem svega nekoliko naredbi u samom programu pozovu dodatne funkcije iz biblioteka i provedu se dodatne analize nad prostornim podacima.

---

[4] Programski jezik čijom primjenom se mogu pisati objektima usmjereni programi (prof. dr. sc. Željko Panian, "Informatički enciklopedijski riječnik", Europapress holding d.o.o., Zagreb 2005.)

[5] Interaktivni računalni sustavi uključuju programe koji omogućuju korisnicima unos podataka ili naredba u dvosmjernoj, obostranoj komunikaciji (prof. dr. sc. Željko Panian, "Informatički enciklopedijski riječnik", Europapress holding d.o.o., Zagreb 2005.)

[6] Interpreter je program koji izvršava naredbe napisane u programskim jezicima visoke razine prevodeći liniju po liniju, odnosno naredbu po naredbuza vrijeme njihova izvršavanja. (prof. dr. sc. Željko Panian, "Informatički enciklopedijski riječnik", Europapress holding d.o.o., Zagreb 2005.)

[7] Biblioteka je skup već kompiliranih rutina (modula) koje koristi neki program, a pohranjene su u obliku, odnosno formatu objekta. Biblioteke su osobito korisne pri pohranjivanju često korištenih rutina jer ih tada nije potrebno eksplicitno povezivat sa svakim programom koji ih koristi već vezani objekt automatski traži željenu rutinu u bibliotekama. (prof. dr. sc. Željko Panian, "Informatički enciklopedijski riječnik", Europapress holding d.o.o., Zagreb 2005.)

## *3.1. Instalacija*

U ovome potpoglavlju opisan je detaljno postupak instalacije programskog jezika Python i ranije navedenih biblioteka koje ne dolaze unaprijed ugrađene u Python. Za instalaciju će biti naveden postupak u oba moguća slučaja: korištenje instalacijskih datoteka priloženih na digitalnom mediju koji čini sastavni dio ovog diplomskog rada ili preuzimanjem (eng. download[8]) istih s interneta.

### 3.1.1. Inastalacija Pythona

Kako je Python besplatan softver moguće ga je besplatno preuzeti s interneta na http://www.Python.org/download ili s medija koji se nalazi u prilogu diplomskog rada. Nakon preuzimanja istoga potrebno je pokrenuti čarobnjaka za instalaciju te dolazimo do izbornika (Slika 1) koji nam nudi mogućnost instaliranja za trenutnog korisnika ili za sve korisnike koji koriste računalo. Ova odluka ovisi o vlastitom odabiru korisnika, ali preporuča se instalaciju provesti tako da Python bude dostupan svim korisnicima.


*Slika 1: početni izbornik instalacijskog čarobnjaka za Python*

[8] Preuzeti, preuzimati. Kopirati datoteku s glavnog na periferni uređaj. Pojam se najčešće koristi za opis kopiranja datoteke s mrežnog poslužitelja na korisničko računalo. (prof. dr. sc. Željko Panian, "Informatički enciklopedijski riječnik", Europapress holding d.o.o., Zagreb 2005.)

Nakon izvedenog odabira, isti potvrdimo klikom na Next te dolazimo do sljedećeg izbornika (Slika 2) koji nam nudi mogućnost odabira lokacije na računalu i direktorija u koji ćemo instalirati Python. Preporuča se ostavljanje tvorničkih postavki kako u narednim koracima ne bi došlo do problema instaliranja biblioteka jer će postupak biti opisan za tvorničke postavke softvera.


*Slika 2 Izbornik instalacije s mogućnošću odabira mjesta i direktorija instalacije*

Nakon izvršenog odabira lokacije (particije i direktorija) za instalaciju programskog jezika Python, potvrdimo odabir klikom na Next i dolazimo do trećeg prozora izbornika (Slika 3) koji nam nudi mogućnost odabira koje dijelove Python instalacijskog paketa želimo instalirati. Preporučuje se ostavljanje tvornički podešenog izbora instalacije svih modula koji dolaze s Python instalacijom jer cjelokupna instalacija Pythona sa svim modulima zauzima veoma malo memorijskog prostora za računala današnjeg vremena. S obzirom na sve navedeno bilo bi pogrešno odabrati samo pojedine module i time si uskratiti određene mogućnosti programskog jezika osim ukoliko ste iskusni korisnik koji

poznaje sadržaj modula, njihovu funkcionalnost, moguće posljedice nedostatka pojedinog modula…

Potvrđivanjem odabira počinje instalacija programskog jezika Python koja je relativno kratka i nakon što je gotova instalacija pojavit će se prozor koji se zahvaljuje autorima i zahtjeva od korisnika da potvrdi završetak instalacije klikom na gumb Finish.

Završetkom instalacije programskog jezika stigli smo do sljedećeg koraka odnosno instaliranja biblioteka koje omogućuju lakši rad s prostornim podacima.

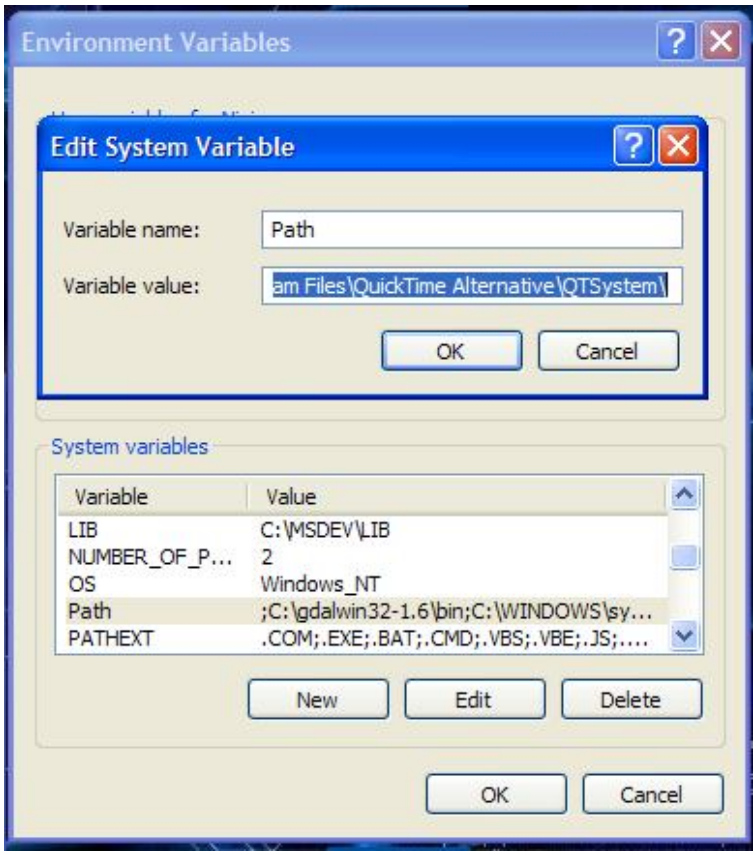

*Slika 3 Izbornik koji dozvoljava odabir pojedinih modula Pythona za instaliranje*

### 3.1.2. Instalacija GDAL biblioteke

GDAL je paket od dvije biblioteke koje služe za pristup i kreiranje GIS podataka (potrebno je samo instalirati ovaj paket). Cijeli postupak instalacije opisan je i na

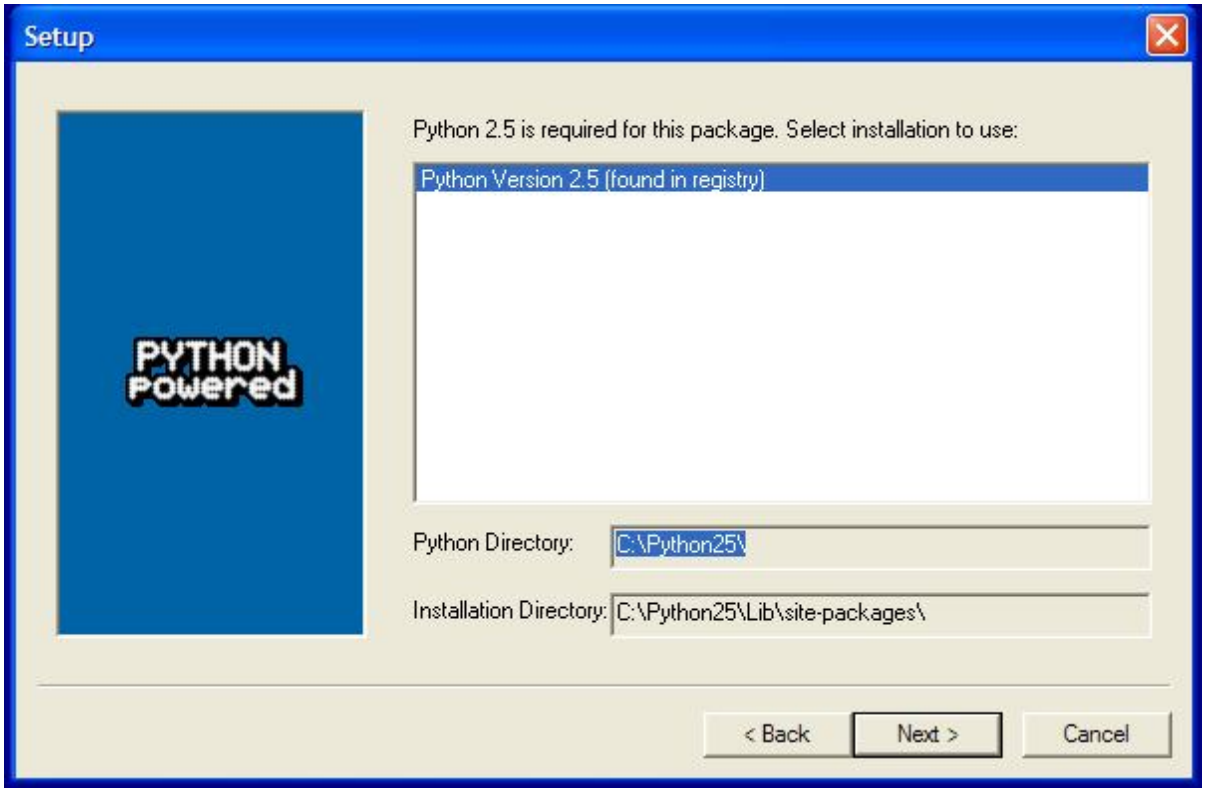
internetu na stranici http://pypi.Python.org/pypi/GDAL/ no u danjem tekstu dan je detaljan opis točnih postupaka potrebnih za potpunu i uspješnu instalaciju.

Nakon što s priloženog digitalnog medija na računalo kopirate i raspakirate dokument imena "gdalwin32exe160.zip" ili isti dokument preuzmete s Interneta na http://download.osgeo.org/gdal/win32/1.6/gdalwin32exe160.zip i otpakirate trebali biste imati direktorij C: \gdalwin32-1.6\ . Sada je potrebno promijeniti sistemske staze i varijable sukladno navedenim uputama. S izbornika *Start*, odaberemo *Control Panel*, od ponuđenog odaberemo *System* i otvori nam se prozor *System Properties*. U tome prozoru odaberemo karticu *Advanced* i s navedene kartice odaberemo *Environment Variables (skup dinamički imenovanih vrijednosti koje utječu na način izvršavanja procesa u računalu)*. Tada nam se otvara novi prozor Environment Variables u kojem moramo promijeniti staze (Slika 4). Pronađemo varijablu imena Path i pod vrijednost varijable dodamo bin direktorij GDAL biblioteka odvojen znakom " ; " (;C:\gdalwin32-1.6\bin;). Potom napravimo novu varijablu imena GDAL-DATA, a pod variable value upišemo stazu C:\gdalwin32-1.6\data.




*Slika 4 Dodavanje i uređivanje sistemskih varijabli i staza*

Nakon uspješno dodanih staza u sistemske varijable potrebno je instalirati priloženu izvršnu datoteku imena "GDAL-1.6.1.win32-py2.5.exe" te restartati računalo prije zadnjeg koraka instalacije. Pokretanjem GDAL-1.6.1.win32-py2.5.exe datoteke otvara se prozor s licencom koji samo potvrdimo klikom na gumb Next. U sljedećem prozoru potrebno je odabrati direktorij za instalaciju kao što je prikazano na slici (Slika 5).




*Slika 5 Izbor direktorija u koji će se instalirati GDAL biblioteke*

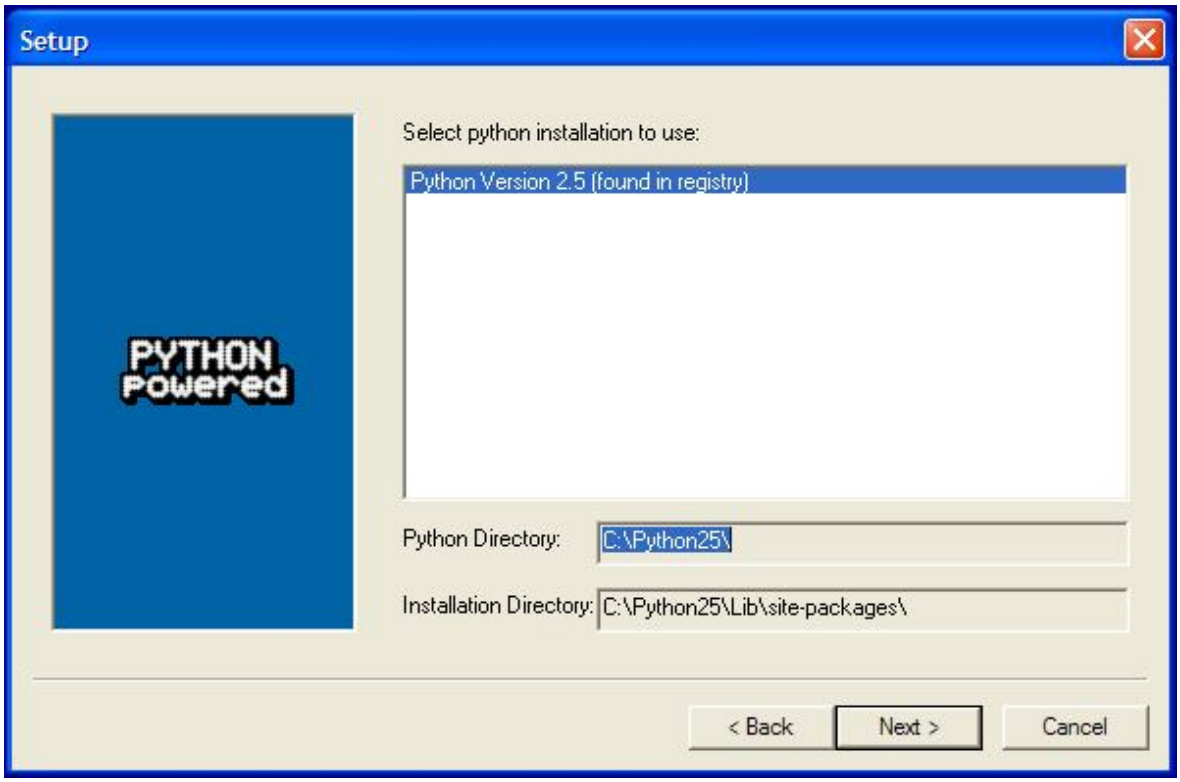
### 3.1.3. Instaliranje Shapely biblioteke

Posljednja u nizu instalacija za uspješnu izradu zadatka obrađenog u ovom diplomskom radu je instalacija Shapely biblioteke koja omogućava manipulaciju prostornim podacima preko programskog jezika Python. Mogućnosti manipulacije podacima i analize podataka pomoću metoda i modula Shapely biblioteka biti će detaljnije navedene i obrađene kasnije.

Shapely službena stranica na internetu je http://trac.gisPython.org/lab/wiki/Shapely te za korisnike Windows operativnog sustava za instalaciju Shaply i GEOS biblioteke moguće je preuzeti izvršnu datoteku sa internet stranice

http://pypi.Python.org/packages/2.5/S/Shapely/Shapely-1.0.12.win32.exe ili istu preuzeti sa priloženog digitalnog medija.

Za instalaciju potrebno je pokrenuti izvršnu datoteku nakon čijeg se pokretanja pojavljuje čarobnjak za instalaciju. U prvom prozoru navedena je licenca instalacije i uvjeti korištenja te se odabirom na gumb Next dolazi do sljedećeg prozora u kojem se odabire lokacija na računalu gdje će se Shapely biblioteka instalirati. Odabir lokacije potvrđuje se klikom na gumb Next i završetak instalacije potvrdi se klikom na gumb Finish u zadnjem prozoru čarobnjaka koji vodi korisnika kroz instalaciju.




*Slika 6 Izbor direktorija u koji će se instalirati Shapely biblioteka*

Instalacijom Shapely biblioteke završena je instalacija programskog jezika Python i svih u ovom radu korištenih biblioteka. Python je sada osposobljen za čitanje i kreiranje GIS podataka, te manipulaciju istima.

## *3.2. Pregled mogućnosti Shapely biblioteke*

Shapely je Pythonov paket za programiranje s prostornim podacima baziran na GEOS biblioteci kao dijelu Java Toplogy Suite. Kako se standardni GIS podaci sastoje od vektorskih podataka kao što su točka, linija, polilinija i poligona pomoću ovog paketa moguće je koristiti iste tipove podataka te na njima i s njima vršiti manipulacije, operacije i analize.

U sljedećim poglavljima sukladno priručniku o korištenju Shapely biblioteke i njenih modula biti će pojedinačno navedeno nekoliko od mnoštva definiranih operacija nad podacima koje se mogu pozvati iz programskog jezika i na taj način olakšati manipulaciju podacima i stvaranje novih podataka.

### 3.2.1. Stvaranje geometrije

Pomoću nekoliko Shapely modula moguće je stvarati razne tipove geometrijskih podataka. U sljedećim primjerima koristiti će se pojedine oznake koje ćemo sada ovdje objasniti. Neka je **a** klasičan koordinatni par kartezijevih koordinata (x,y), vrijednosti koordinata moraju biti numeričke i odgovarati jednom od postojećih tipova numeričkih podataka u Pythonu.

Pomoću modula Point moguće je kreirati točku. Navedeni modul od koordinatnog para stvara geometriju točke koja se kasnije može topološki obrađivat.

```
>>> from shapely.geometry import Point

>>> point=Point(a)
```

Pomoću modula Linestring moguće je kreirati liniju ili poliliniju. Navedeni modul od koordinatnih parova stvara geometriju linije i polilinije, a koordinatni parovi čine točke koje omeđuju liniju ili prikazuju mjesta loma linije koja se kasnije može topološki obrađivat.

```
>>> from shapely.geometry import LineString

>>> line=LineString( (a1, … , aM) )
```

Pomoću modula Polygon moguće je kreirati poligon tj. mnogokut. Navedeni modul od koordinatnih parova stvara geometriju mnogokuta koja se kasnije može topološki obrađivat. Ukoliko prva točka a1 nije identična posljednjoj točki aM modul će automatski zatvoriti poligon.

```
>>> from shapely.geometry import Polygon

>>> poligon=Polygon ( (a1, … , aM))
```

Navedenim primjerima prikazano je kreiranje najosnovijih topoloških relacija, no potrebno je napomenuti kako Shapely nudi mnogo veće mogućnosti.

Kod poligona moguće je kreirati poligon s prazninama (holes) ukoliko se navede da ista postoji i definiraju se granice praznine. Također je važno za određene topološke analize kao što su analize objekata lijevo i desno od granice poligona napomenuti da računanje strane ovisi o smjeru zadavanja koordinata pomoću koordinatnih parova pri kreiranju topologije pozivanjem modula Polygon (smjer kazaljke na satu ili obrnuti smjer).

Moguće je stvaranje takozvane višedijelne geometrije kao što su skupovi točaka, linija i poligona. Dakle koristeći već navedene module s prefiksom Multi moguće je definirati više točaka, kao jedan objekt geometrije odnosno jednu varijablu pri programiranju što uvelike može olakšat programiranje rješenja određenih problema jer omogućava lakše snalaženje početnicima u programiranju.

```
>>> from shapely.geometry import MultiPoint

>>> points=MultiPoint ( [a1, … , aM] )
```

Također korištenjem MultiLineString modula omogućeno je stvaranje izlomljenih i isprekidanih linija jer je moguće više linijskih segmenata spojiti u jedan iako se fizički ne dodiruju i čine skup zasebnih linija.

```
>>> from shapely.geometry import MultiLineString

>>> lines=MultiLineString ( [(a1, … , aM), (b1, … , bM), …])
```

Korištenje MultiPolygon modula omogućuje stvaranje poligona s više praznina ili jednostavno spajanje više poligona u jedan parametar od interesa.

```
>>> from shapely.geometry import MultiPolygon

>>> poligons=MultiPolygon ( [(shell), (holes), …])
```

I za kraj pregleda ovog djela shapely paketa kao zanimljivost potrebno je istaknuti i kreiranje null geometrije (eng. Null geometries) s kojom gotovo ništa nije moguće dalje raditi, odnosno nema mogućnosti analiziranja, obrade …

```
>>> line_null = LineString ()
```

Moguće je null geometriji dodijeliti koordinate te geometrija više nulta odnosno prazna i sa njom je moguće tada raditi određene obrade, analize i manipulacije.

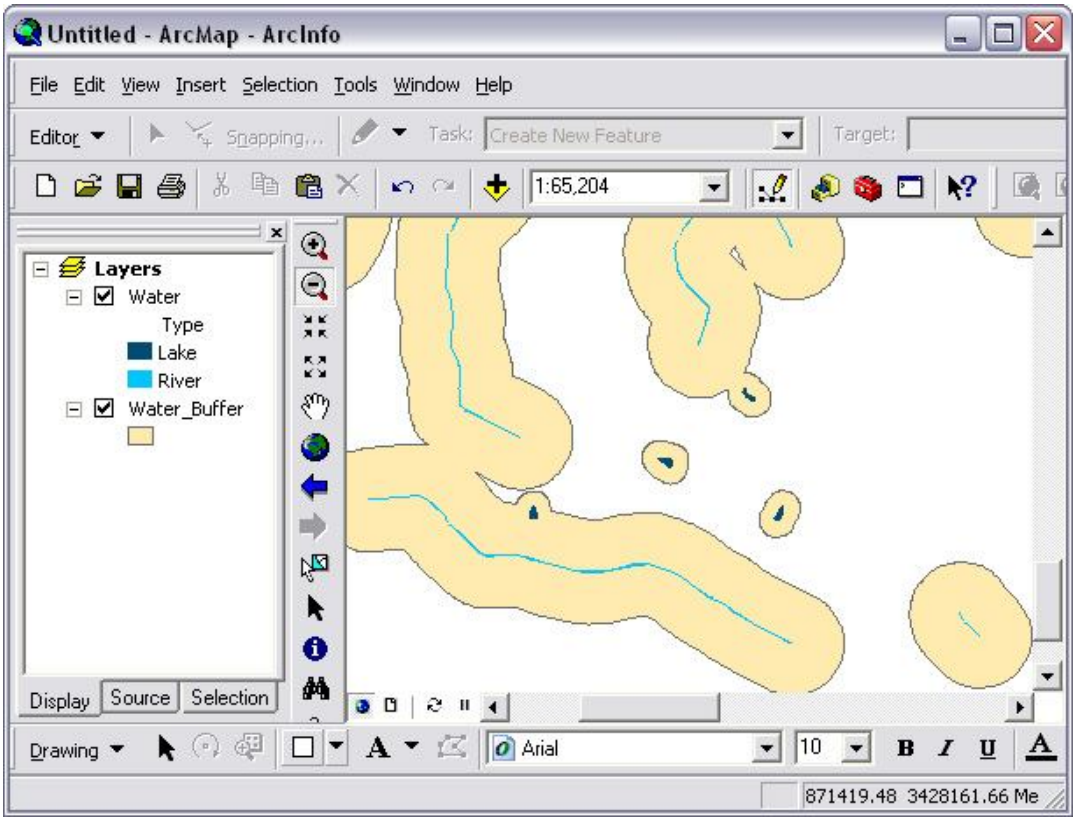
### 3.2.2. Metode prostornih analiza

U ovom poglavlju na sljedećim primjerima prikazani su načini pozivanja pojedinih operatora i njihova definicija. Potrebno je odmah napomenuti kako su pojedine metode ove klase geometrije tvorci novih geometrija te samim time dolazi do nastajanja nove topologije i moguće je da korisnici za rezultat dobiju ono što nije predvidivo očekivati od operatora na Python setovima podataka.

Buffer naredba stvara zonu određene širene oko zadanog objekta. U niže navedenom primjeru prikazano je kako se operator nad objektom poziva nakon imena objekta, a odvaja se " **.** ". Kako biste ispravno koristili mogućnosti buffer metode potrebno je znati da navedena metoda ima dva ulazna parametra osim objekta na kojem se izvršava. Ulazni parametri su širina zone i broj piksela koji se koriste kako bi se zaoblili uglovi poligona koji je rezultat obrade.

```
Objekt.buffer (širina, broj piksela koji se koriste za zaobljavanje linije)

>>>buffered = point.buffer (5, 16)

>>>buffered = line.buffer (5, 16)

>>>buffered = poligon.buffer (5, 16)
```

Potrebno je uzeti u obzir da operacija stvaranja zone odnosno buffer za rezultat oko bilo kojeg objekta vraća poligon, tj. buffer oko točke je aproksimacija kruga, oko linije to je lik sličan pravokutniku, a za poligon, to je poligonu sličan mnogokutu što se vidi na slici ispod (Slika 7).

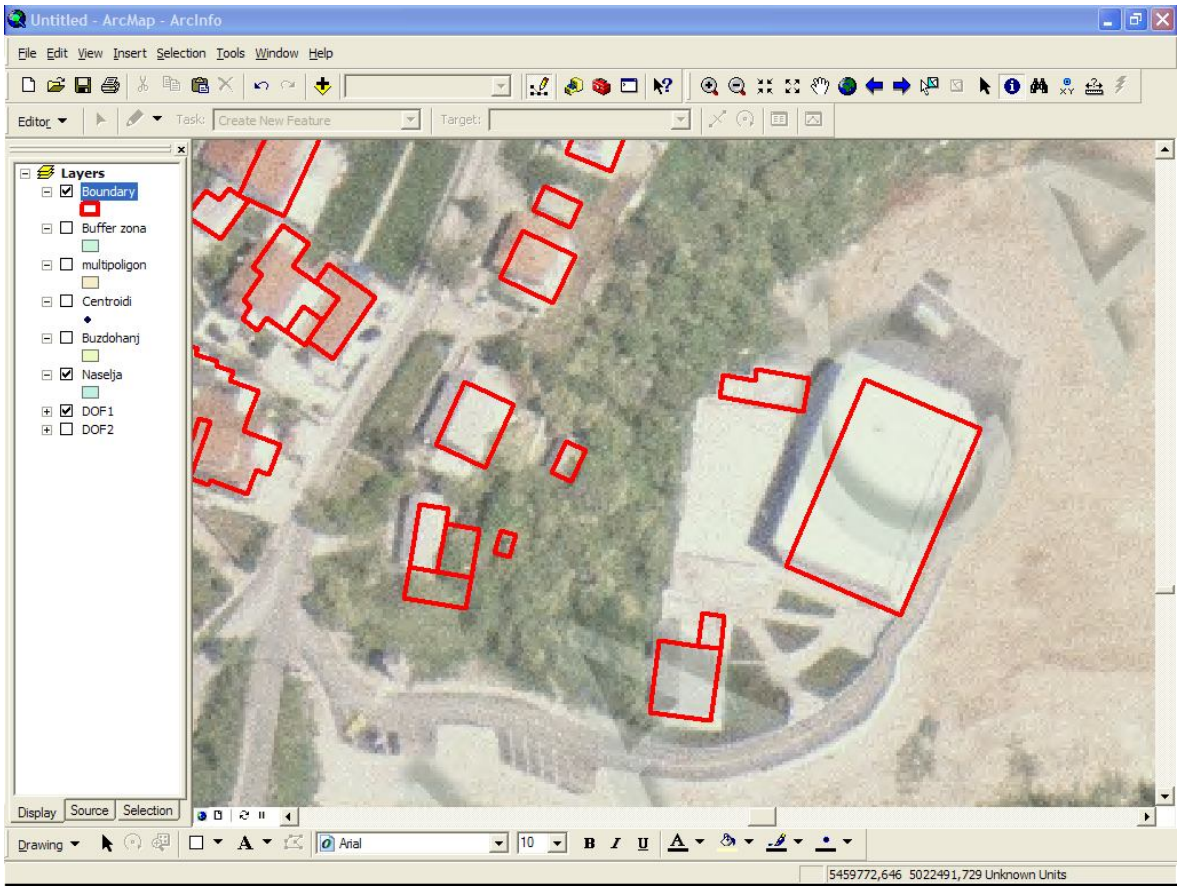



*Slika 7 Prikaz buffer zone oko linijskih i površinskih objekata*

Metoda boundary (granica) za svoj zadatak ima vratiti za dimenziju nižu geometriju nego li je to geometrija procesuiranog objekta. Proces pozivanja procedure jednak je kao i u svim drugim procedurama i prikazan je na primjeru. Sukladno zadatku i imenu metode povratni podatak je granica koja je nova geometrija. Granica poligona je linija, granica linije skup točaka, a granica točke je prazan skup odnosno null geometrija.

```
Objekt.boundary

>>> lines = poligon.boundary

>>>points = line.boundary

>>>null = point.boundary
```

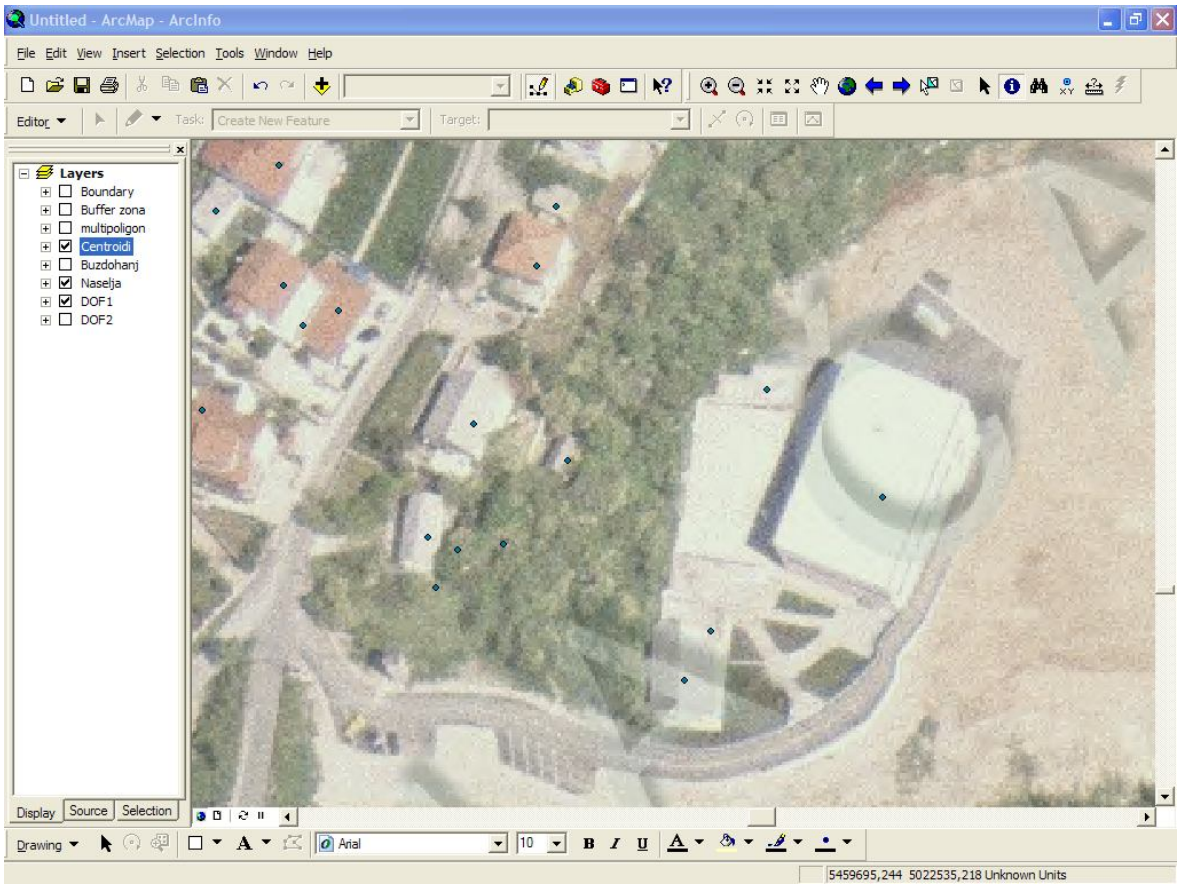
Na slici (Slika 8) ispod vidi se rezultat provedene operacije boundary nad izvornim podacima korištenima u radu. Izvorni poligoni zamijenjeni su zatvorenim polilinijama koje su granice tih poligona.




*Slika 8 Prikaz rezultata provođenja operacije boundary nad poligonima*

Centroid je veoma jednostavna metoda za objasniti. Njen zadatak je da vraća centroid ili geometrijsko središte poligona kako joj to i samo ime govori. Dakle objekt koji obrađuje je poligon (zatvoreni mnogokut), a rezultat je točka odnosno uređeni par kartezijevih koordinata. Nakon što u prozoru ispod vidite način pozivanja centroid metode na sljedećoj stranici na slici (Slika 9) možete vidjeti vizualni prikaz rezultata provedbe navedene metode na podacima korištenima pri izradi ovoga rada.

```
Objekt.centroid

>>> centar = poligon.centroid
```

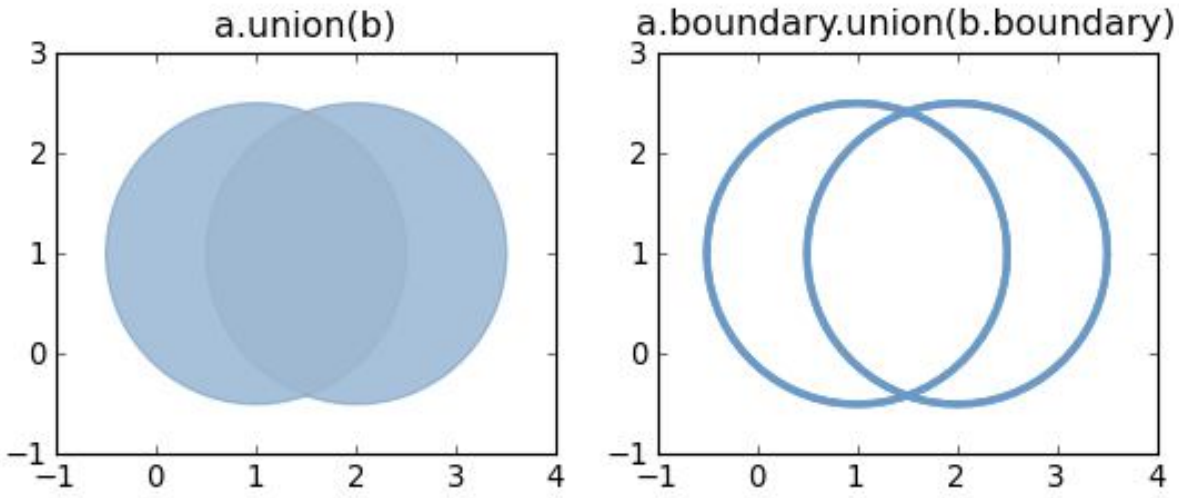



*Slika 9 Prikaz rezultata obrade podataka naredbom centroid*

Unija (eng. union) ima zadatak matematičkog operatora koji ovdje provodi operacije nad geometrijskim objektima i udružuje više istovrsnih objekata u jednu skupinu. U primjeru su navedeni jednostavne unije dviju točaka i dvaju poligona. Potrebno je napomenuti kako unija dvaju poligona može za rezultat dati jedan, dva ili više poligona ovisno o tome sijeku li se ulazni poligoni ili ne ( npr. poligon su identični, poligoni se dodiruju, poligoni se sijeku, jedan poligon je unutar drugoga…).

```
Objekt.union(drugi objekt)

>>>points = point1.union(point2)

>>>poligons = poligon1.union (poligon2)

>>>lines = line1.union (line2)

>>>poligons = poligon1.union (poligon2)
```

Na slici ispod (Slika 10) prikazana su dva slučaja obrade podataka naredbom kreiranja unije. U prvome slučaju radi se o dva poligona (plava kruga) koji nakon kreiranja unije postaju jedan objekt čije se granice sastoje od dva isječka kružnog luka. U drugome slučaju radi se o uniji dvaju linijskih objekata (plave kružnice) koji spajanjem postaju jedna izlomljenja krivljulja.

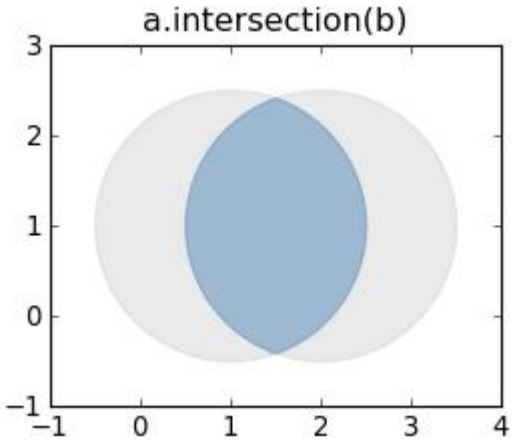



*Slika 10 Prikaz rada union naredbe na površinskim i linijskim objektima*

Presjek ( eng. Intersection ) kao i unija ima svojstvo matematičkog operatora prilagođenog funkcioniranju nad geometrijskim objektima. Rezultat presjeka dviju geometrija je nova geometrija. U primjeru je navedeno pozivanje modula intersection gdje je rezultat presjeka nazvan "intersected", a ulazni objekti su "poligon1" i "poligon2".

```
Objekt.intersection (drugi objekt)

>>> intersected= poligon1.intersection (poligon2)
```

Na slici (Slika 11) na sljedećoj stranici vidi se jasno prikazan rezultat naredbe intersection odnosno naredbe presjeka nad geometrijskim podacima. Početni podaci su dva blijedoplava kruga koji se djelomično preklapaju. Nakon što se provede operacija presjeka na navedenim podacima dobiva se tamnoplavo označeno područje. Dakle presjekom se dobiva novi poligon koji je identičan preklopu dvaju početnih poligona.

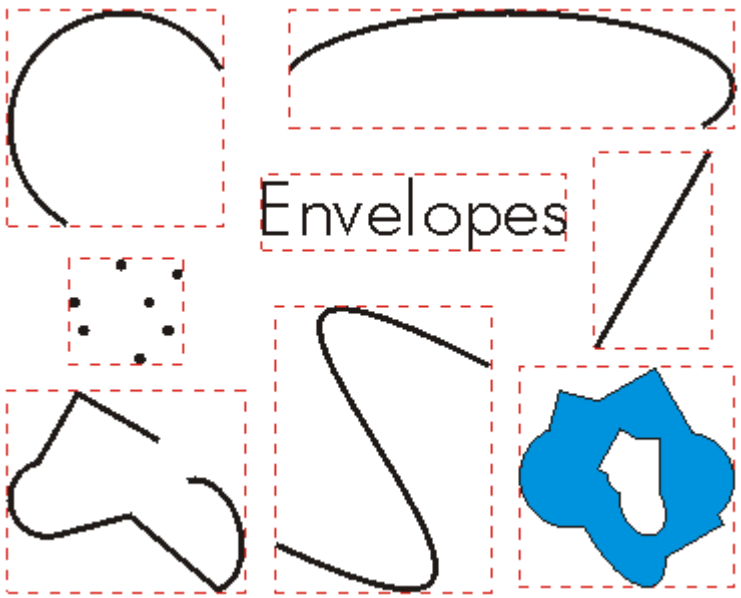



*Slika 11 Prikaz rada naredbe intersection tj. presjeka prostornih podataka*

Omotnica (eng. envelope) je operator koji nakon obrade poligona vraća kao rezultat obrade geometrijski pravokutan poligon omotnice.

```
Objekt.envelope

>>> enveloped = poligon.envelope
```

Na slici (Slika 12) ispod prikazani su rezultati koji bi se dobili provedbom metode omotnice na raznim podacima od kojih je sa Shapelyem sukladno korištenom priručniku moguće izvršiti metodu nad poligonom što odgovara primjeru u donjem desnom kutu slike.

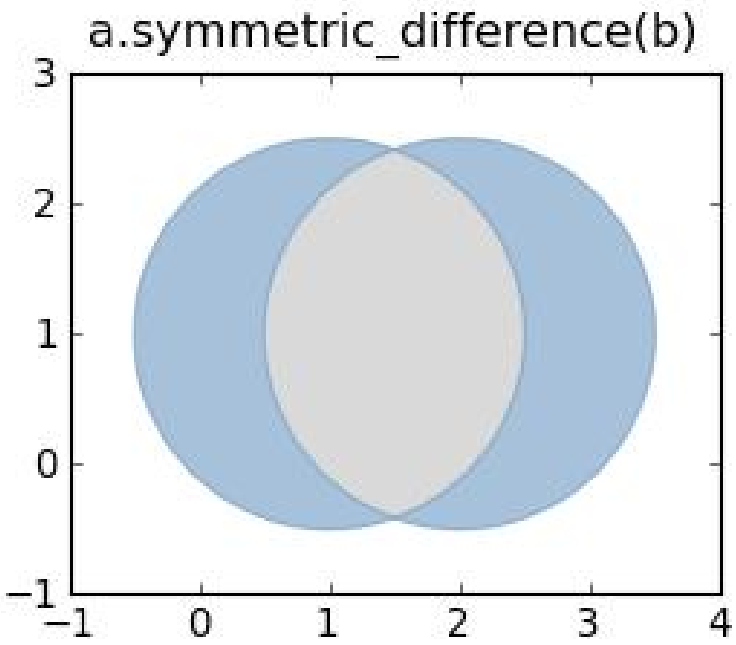



*Slika 12 Prikaz rezultata provedbe metode omotnice nad raznim tipovima geometrijskih podataka*

Za kraj ovog pregleda spomenut ćemo samo još jednu od nekoliko zanimljivih operacija koje su omogućene nad geometrijskim objektima. Simetrična razlika (eng. symmetric difference) je poseban oblik prostorne razlike koji za rezultat ima geometriju koja kombinira točke koje su u zadanoj geometriji, a nisu u drugoj i one koje nisu u zadanoj geometriji, a jesu u drugoj.

```
Objekt.symmetric_difference (objekt2)

>>> simdiff = poligon.symmetric_difference (poligon2)
```

Na slici (Slika 13) ispod vizualno je prikazan prije opisani princip rada metode. početni podaci su dva svijetlo plava kruga koji se međusobno djelomično preklapaju. Nakon provedbe metode rezultat su područja koja se ne preklapaju tj. dva tamnija mjesecolika isječka. Usporedbom slika priloženih uz intersection metodu (Slika 11) i slike priložene uz symmetric difference metodu (Slika 13) može se reći kako se radi o komplementarnim metodama. Dok jedna metoda izdvaja presjek dvaju ulaznih poligona tj. ono što im je zajedničko, druga metoda izdvaja samo razliku između ulaznih podataka tj. za rezultat daje svo područje koje im nije zajedničko.




*Slika 13 Prikaz rada metode simetrične razlike tj. "symmetric difference"*

### 3.2.3. Konverzija podataka

WKT Well-known text ili doslovno prevedeno "Dobro poznati tekst" je *markup language*[9] za prezentiranje vektorskih geometrijskih podataka na kartama, prostornim referentnim sustavima objekata i transformaciju prostornih referentnih sustava.

WKB well-known binary je binarni, računalni ekvivalent WKT-a i koristi se za prijenos i pohranu podataka i informacija na istim bazama podataka kao što su npr. PostGIS

U Shapely biblioteci moguće je koristiti oba zapisa WKT i WKB. U okviru ispod teksta naveden je način kreiranja nove geometrije korištenjem shapely.wkt.loads naredbe iz WKT zapisa podataka.

```
from shapely.wkt import loads
loads('POINT (0 0)')
```

WKB zapis bilo koje geometrije objekta može se ostvariti pomoću WKB atributa. Nove geometrije mogu biti kreirane iz WKB podataka koristeći shapely.wkb.loads naredbu za stvaranje geometrije. Ovaj se format koristi za interoperabilnost s ogr.py.

U prozoru ispod prikazane su naredbe koje pokazuju kreiranje nove geometrije čitanjem shapefila worl_borders.shp.

```
import ogr
from shapely.wkb import loads
source = ogr.Open("/tmp/world_borders.shp")
borders = source.GetLayerByName("world_borders")
feature = borders.GetNextFeature()
loads(feature.GetGeometryRef().ExportToWkb())
```

[9] Računalni jezik koji služi za oblikovanje tekstualnih datoteka i njihovo povezivanje. Naziv "jezik za označavanje" aludira na tradicionalnu nakladničku praksu, kada su urednici na marginama rukopisa bilježili oznake koje su za tiskare predstavljale upute o načinu slaganja i prijeloma teksta. (prof. dr. sc. Željko Panian, "Informatički enciklopedijski riječnik", Europapress holding d.o.o., Zagreb 2005.)

# 4. Razrada rješenja

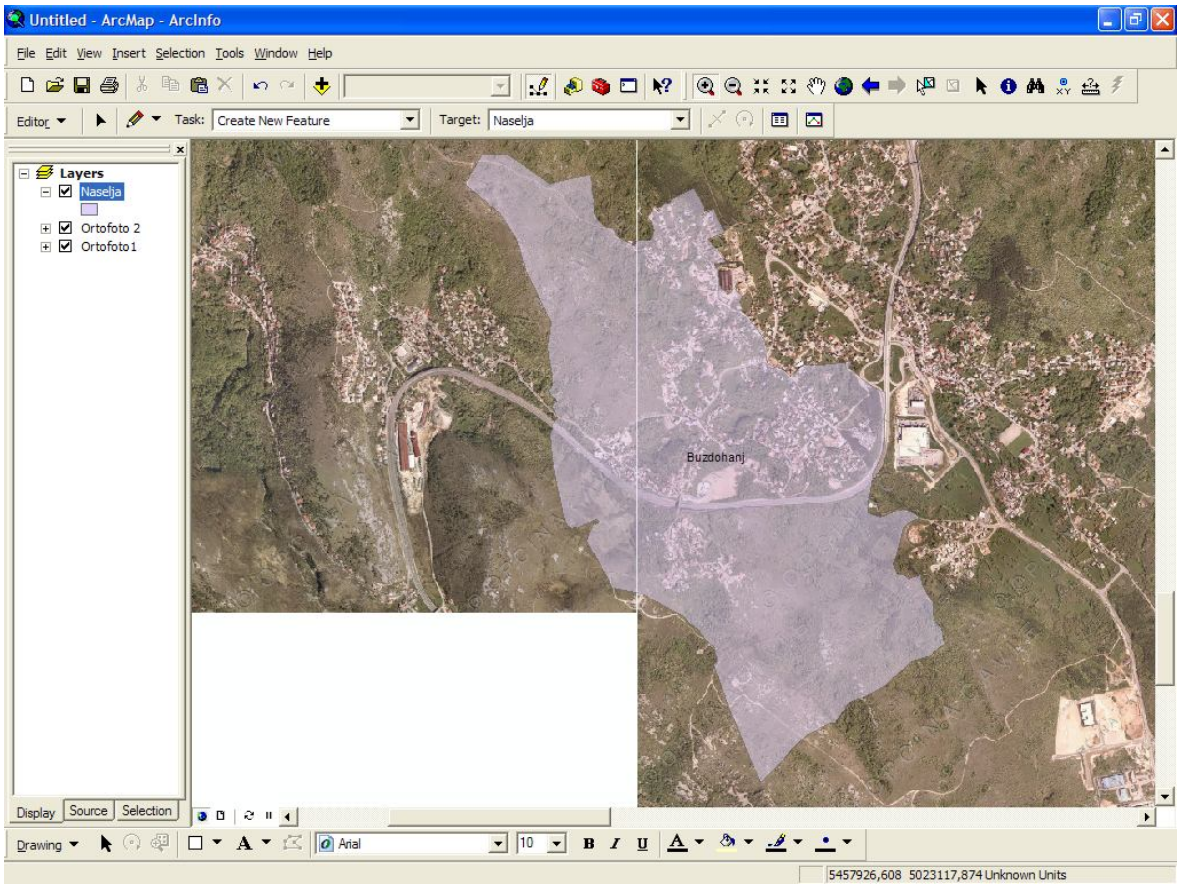
U narednim poglavljima detaljna će biti objašnjen problem koji je obrađen u ovome radu, prezentirani će biti prikupljeni podaci te o njima dostupne informacije. Nadalje detaljno će biti objašnjen koncept oba rješenja problema, definiranje postupaka kojima je problem riješen, te dobiveno rješenje. Kao zaključak ove cjeline rada biti će uspoređeni i prezentirani rezultati obaju rješenja.

## *4.1. Definiranje problema*

U ovom diplomskom radu zadatak je na primjeru prostorne analize podataka pomoću komercijalnog (ArcGIS) i slobodnog, besplatnog (Python) rješenja provjeriti mogućnosti slobodnog softverskog rješenja. Potrebno je usporediti brzinu izvršavanja dvaju rješenja, donijeti sud o upotrebi slobodnih softverskih rješenja baziranih na Python programskom jeziku te pokušati iz dobivenih rezultata prikazati pozitivne i negativne strane pojedinih rješenja koje se otkriju pri rješavanju postavljenog problema.

Sukladno navedenom zadatku potrebno je utvrditi koja će se prostorna analiza koristiti za usporedbu dvaju rješenja. Krećući od pretpostavke da bi jedno od rješenje trebalo biti sporije kao kriterij za odabir određene analize odabrano je vrijeme koje je potrebno komercijalnom rješenju da obradi prikupljene podatke. Zbog usporedbe brzine izvršavanja rješenja bilo je potrebno odabrati onu analizu čije izvršenje traje dovoljno dugo da razlika bude što uočljivija. Nakon analize nekolicine prostornih analiza kao testna odabrana je operacija kreiranja zone oko objekta (eng. buffer) za čije je izvršavanje na prikupljenim podacima potrebno 10 pa i više sekundi.

Kako ovaj problem ne bi ostao samo na teoretskom razmatranju mogućnosti pojedinih rješenja potrebno je bilo povezati problemski zadatak s realnim svijetom. Način povezivanja problema s realnom situacijom riješen je pomoću prikupljenih podataka. Pri izradi analiza korišteni su podaci o izmjerenim izgrađenim građevinama na području naselja Buzdohanj (Slika 14) u općini Čavle pokraj Rijeke koji su prikupljani za izradu komercijalnog projekta .




*Slika 14 Prikaz naselja Buzdohanj*

### 4.1.1. Prikupljeni podaci

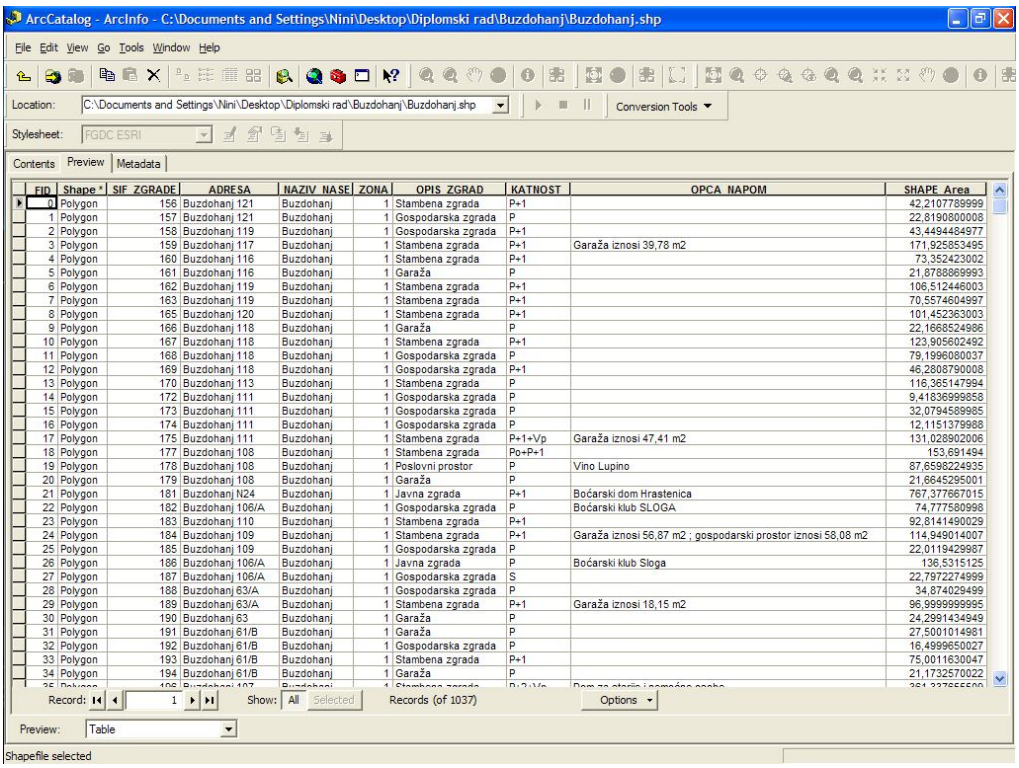
Podaci koji su analizirani u ovome radu ustupila je geodetska tvrtka Geodetski zavod Rijeka d.d. koja je pri izradi komercijalnog projekta, čija je svrha bila ustanoviti sve postojeće građevine i njihove tlocrtne površine, izvršila izmjeru svih izgrađenih objekata u naselju Buzdohanj. Podaci su prikupljani u proljeće 2009. godine neposrednim odlaskom na teren te mjerenjem frontova objekata pomoću laserskih daljinomjera i mjernih vrpca. Zahtijevana točnost izmjerenih duljina je ±10 cm. Navedeni podaci zbog zahtjeva projekta nisu apsolutno smješteni u prostor već je smještaj u prostor i orijentacija objekata vršena pomoću digitalnog ortofota. Ukupno je na terenu identificirano i izmjereno 1037 građevina. Navedeni podaci digitalno su ucrtani u Autodesk AutoCAD softveru te su kao takvi korišteni za izradu korištenog shapefila "Buzdohanj"

Svaki od navedenih 1037 objekata ima više atributa čija je definicija prikazana u tablici (Tabela 3). Svaki objekt ima svoj jedinstveni identifikator (FID) koji je automatski generiran pri kreiranju shapefila. Nadalje atributima su opisani dodatni podaci o objektima kao što su: "Shape" tj. oblik geometrije (point, line, polygon …), šifra zgrade (kućni broj zgrade ili zgrade kojoj pripada, npr. zasebna garaža u dvorištu obiteljske kuće), adresa, naziv naselja, zona (zona komunalne naknade u kojoj se objekt nalazi), opis zgrade (stambena, gospodarska ili javna zgrada, ruševina …), katnost (oznakama opisan broj katova i njihova uporaba, npr. P za prizemlje, VP za visoko potkrovlje, PO za podrum, TEM za temelje bez izgrađene građevine), opća napomena (opisani dodatni podaci, npr. ime poslovnog subjekta koji zauzima dio zgrade i površina koju koristi) te površina (automatski računana površina)

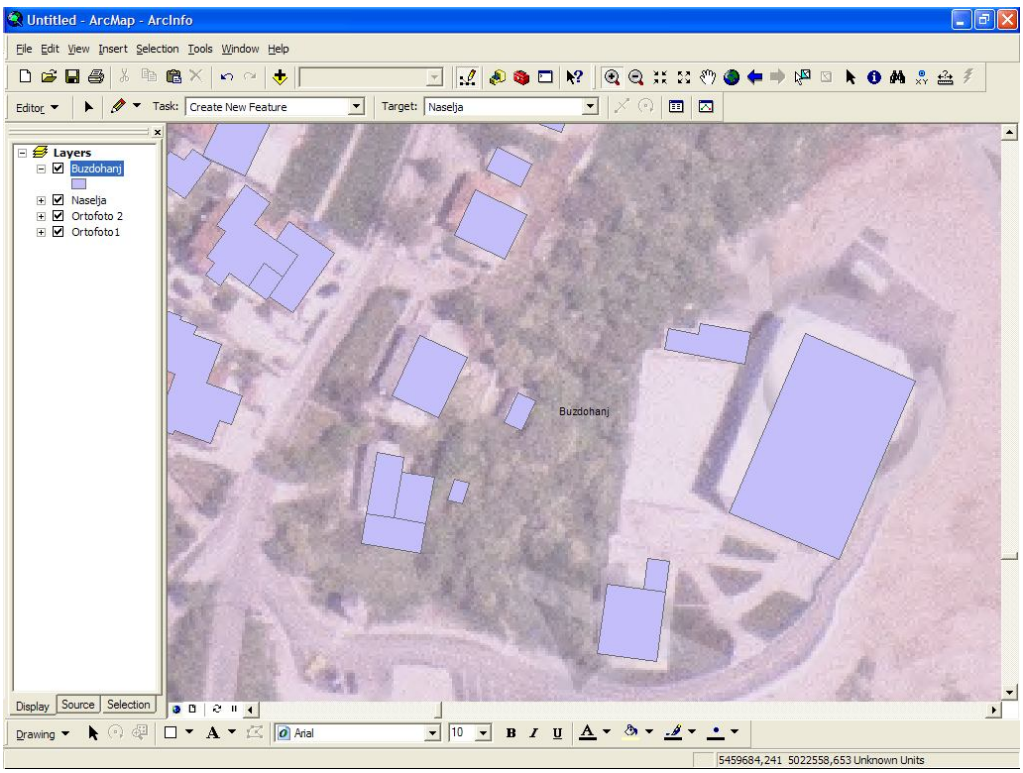
*Tabela 3 Prikaz atributa prikupljenih podataka*

| *Naziv* | *Opis* | *Vrijednost* | *Primjer* |
|---|---|---|---|
| **FID** | Jedinstveni identifikator | Integer | 23 |
| **Shape** | Opis geometrije | String | Polygon |
| **SIF_ ZGRADE** | Šifra zgrade, kućni broj | Integer | 183 |
| **ADRESA** | Poštanska adresa | String | Buzdohanj 183 |
| **NAZIV_NASE** | Ime naselja | String | Buzdohanj |
| **ZONA** | Zona komunalne naknade | Integer | 1 |
| **OPIS_ZGRAD** | Vrsta građevine | String | Stambena zgrada |
| **KATNOST1** | Opis katnosti zgrade | String | P+1 |
| **OPCA_NAPOM** | Dodatne napomene | String | |
| **SHAPE_Area** | Tlocrtna površina | Float | 92,81 |

Na sljedećoj strani prikazan je dio prikupljenih podataka u tabelarnom (Slika 15) i grafičkom (Slika 16) prikazu kako to omogućuje ArcGIS Catalog.



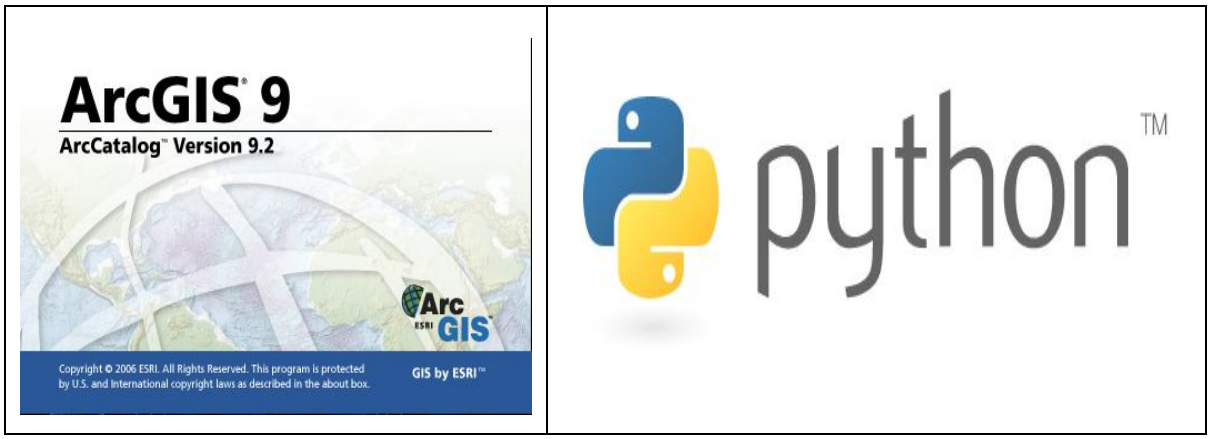

*Slika 15 Tabelarni prikaz dijela prikupljenih podataka*




*Slika 16 Grafički prikaz dijela prikupljenih podataka*

### 4.1.2. Korišteni softver

Pri izradi diplomskog rada korišteno je računalo pokretano Microsoftovim operativnim sustavom Windows XP Profesional na kojem je instaliran softverski paket ArcGIS 9.2. Također kao što je u ranijim poglavljima opisano za izradu alternativnog rješenja zadatka korišten je programski jezik Python sa instaliranim Shapely modulom i GDAL bibliotekom.




*Slika 17 Logo softverskog paketa ArcGIS i programskog jezika Python*

Softverski paket ArcGIS 9.2 čini više aplikacija od kojih su pri izradi korištene ArcCatalog i ArcMap – ArcInfo. Samo rješenje problema odrađeno je u aplikaciji ArcCatalog dok je ArcMap – ArcInfo aplikacija korištena za vizualizaciju prikupljenih podataka i prezentaciju rješenja.

Programski jezik Python korišten je za izradu skripte odnosno pisanje programa koji čini alternativno rješenje komercijalnoj obradi podataka koja je provedena pomoću ArcCatalog aplikacije. Ranije spomenuti Shapely modul i GDAL biblioteke korištene su u samome programu kako bi uopće bilo moguće obraditi podatke na zadani način pomoću Pythona. I u ovome slučaju rezultati su vizualizirani i prezentirani pomoću ArcMap – ArcInfo aplikacije iako je vizualizaciju shapefileova moguće provesti i u nekim alternativnim aplikacija (ArcView)

Kako bi se prikazala mogućnost rada s Python skriptama u ArcGISu rješenje problema izvedeno je i na taj način iako time nije ubrzan proces već je samo prikazana mogućnost automatizacije procesa u slučajevima kada je potrebno često vršiti određene nizove analiza ili manipulacija podacima. Automatizacija je izvršena izradom skripte u u arcgisscripting modulu.

## *4.2. Komercijalno rješenje*

Sam koncept komercijalnih rješenja problema predviđa kako korisnik ne treba ništa osim podataka koje želi obraditi. Plaćanjem rješenja briga oko načina rješavanja problema prelazi na onoga kome se plaća rješenje. Kada se koriste specijalizirani softverski paketi kao što je ArcGIS za očekivati je kako unutar aplikacija i njihovih modula većina korisnika može riješiti većinu zadataka koje trebaju obaviti.

Problem nastaje ukoliko korisnik treba izvršiti rijedak slučaj obrade podataka koji tvorci softverskog rješenja nisu predvidjeli jer legalno ta vrsta softvera ne omogućava korisniku da samostalno mijenja i dorađuje rješenje.

ArcGIS dolazi sa rješenjima za automatizaciju pojedinih procesa izradom skripti koje je moguće izraditi i u Pythonu pa će jedan od načina rješavanja definiranog problema biti kreiranje i pokretanje skripte koja izrađuje rješenje.

### 4.2.1. Koncept rješenja

Kako softverski paket ArcGIS ima u svoje aplikacije ugrađen velik broj rješenja za razno razne probleme koje korisnik može imati pri pregledavanju, analiziranju i obradi prostornih podataka potrebno je definirati problem i pronaći unaprijed pripremljeno rješenje.

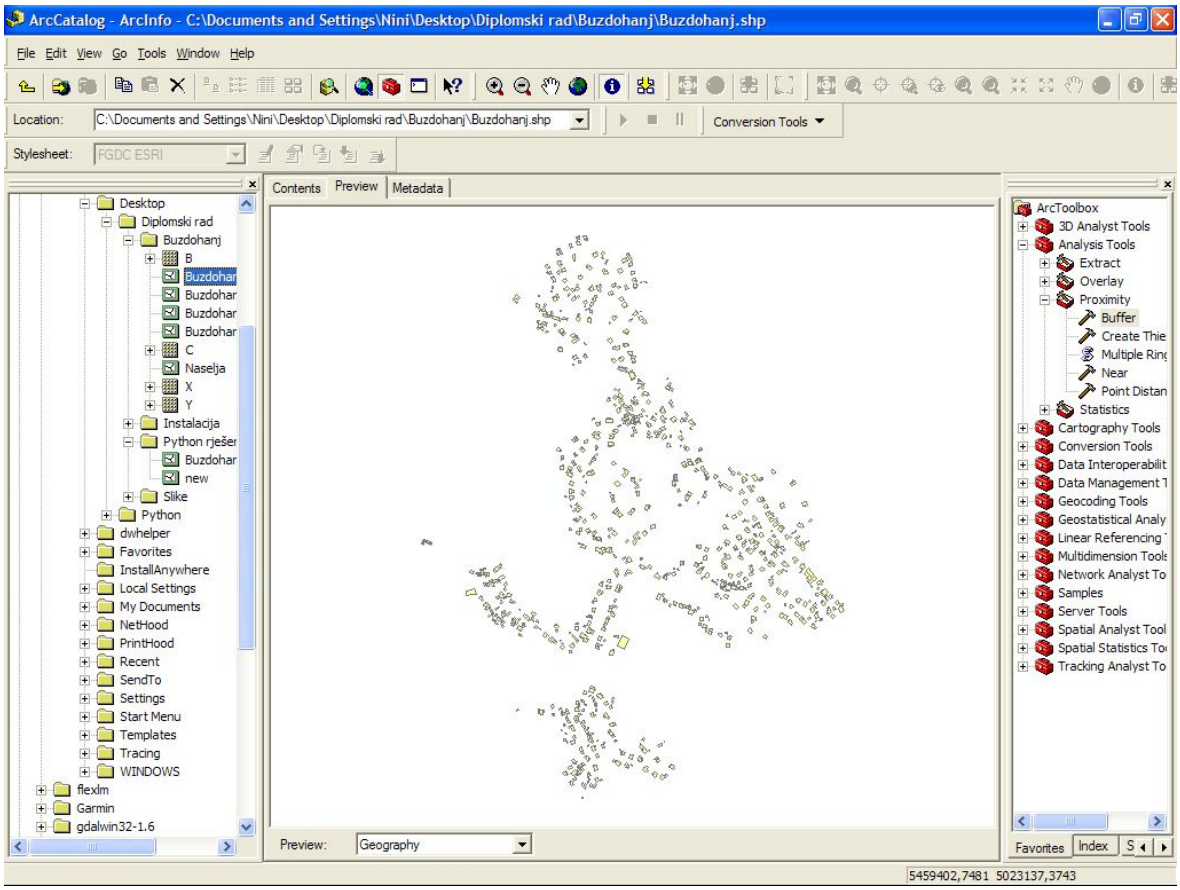
U ovome radu kao rješenje problema potrebno je izraditi buffer zonu oko svakog pojedinog objekta unutar naselja Buzdohanj s udaljenošću 5 metara od ruba zone do granice objekta. U ArcCatalog aplikaciji postoji cijeli paket unaprijed definiranih analiza i metoda obrade podataka koje su skupljene u modulu ArcToolbox. U tom modulu se pod Analysis Tools / Proximity nalazi i opcija buffer koja će se koristiti kao rješenje problema.

Kako bismo prikazali mogućnost automatizacije procesa čije se izvršavanje često ponavlja kao drugi dio rješenja biti će prikazana skripta u Pythonu koja će po pokretanju sama izraditi rješenje zadatka odnosno iz unaprijed definiranih podataka (shapefile) kreirati će pomoću ArcToolboxa buffer zonu širine 5metara i dati nam identično rješenje na automatiziran način.

### 4.2.2. Izrada rješenja

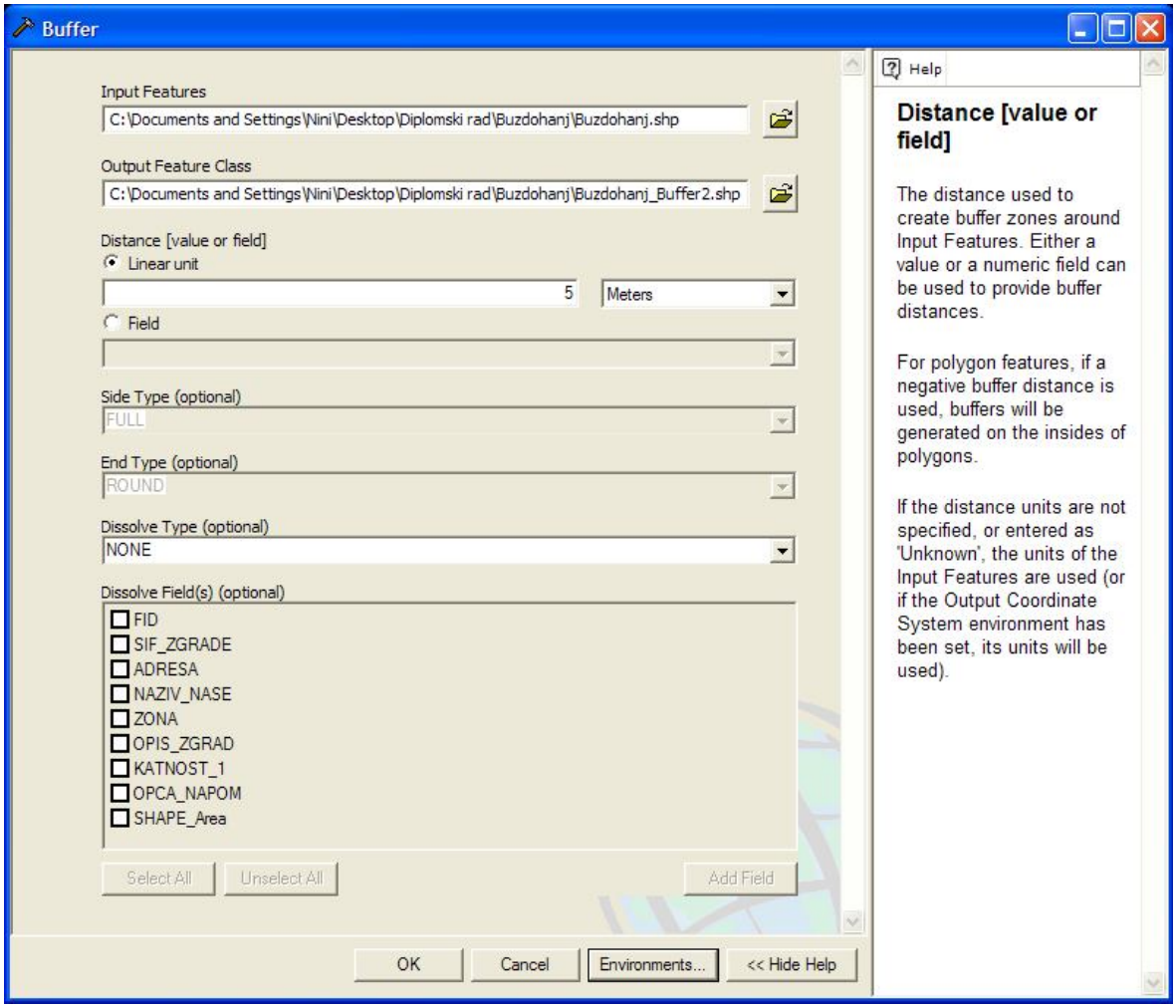
Sukladno prije navedenom konceptu rješavanja problema u nekoliko sljedećih poglavlja biti će detaljno opisan proces izrade buffer zone oko postojećih objekata pomoću ArcCatalog aplikacije softverskog paketa ArcGIS.

Pokretanjem aplikacije ArcCatalog otvara se početni prozor čiji se sadržaj vidi na slici (Slika 18). Gornji dio prozora zauzimaju alatne trake od kojih je ona osnovna odnosno standardna sa opcijama File, Edit, View, Go, Tools, Window i Help uvijek prisutna dok je ostale trake moguće uključiti (dodati) ili isključiti (sakriti). Donji dio prozora standardno dolazi podijeljen na dva djela od kojih prvi dio čini Catalog tree odnosno kaskadni prikaz dostupnih datoteka za lakšu navigaciju između mnoštva podataka. Drugi dio je Preview prozor u kojem je moguće pregledati sadržaj odabrane datoteke ili ukoliko se radi o shapefilu moguće je pregledati tabelarno ili grafički sadržane podatke.




*Slika 18 Početni prozor ArcCataloga s odabranim podacima*

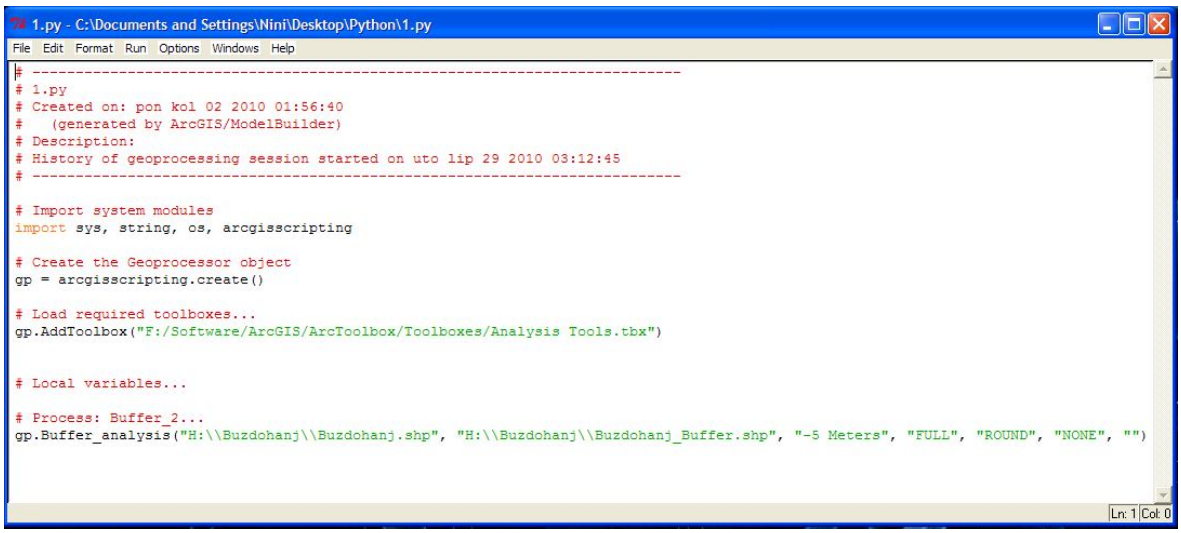
Na slici (Slika 18) vidi se i treći prozor ArcToolbox koji se može otvoriti odabirom gumba Window u standardnoj alatnoj traci te potom odabirom ArcToolbox gumba iz padajućeg izbornika. U navedenom prozoru odabirom Analysis Tools pa iz padajućeg izbornika odabirom Proximity dolazimo do mogućnosti odabira Buffer operacije. Odabirom navedene operacije otvara se novi prozor (Slika 19) sa izbornikom za podešavanje opcija izrade buffer zone oko odabranih objekata.




*Slika 19 Buffer izbornik*

U samom izborniku za izradu buffer zone potrebno je u prvome redu odrediti ulazne podatke i njihovu lokaciju. Potom se u drugom redu pojavljuje ime i mjesto gdje će biti pohranjeni obrađeni podaci. Nadalje u trećemu je redu potrebno definirati širinu zone i mjernu jedinicu u kojoj zadajemo širinu. U sljedećih nekoliko redova nalaze se dodatne opcije pri kreiranju zone oko objekata čijim se uključivanjem vrijeme potrebno za obradu podataka može povećati i više desetaka puta u odnosu na vrijeme potrebno za izradu zone bez uključenih dodatnih opcija.

Na slici (Slika 20) se vidi programski kod Python skripte čiji je zadatak automatizacija procesa stvaranja buffer zone odnosno automatizacija procesa obrade podataka. Iako se ovakve skripte koriste općenito za složenije obrade koje se češće ponavljaju i vremenski se dugo izvršavaju te je najveća korist na kraju radnoga dana pokrenuti skriptu i sljedeći dan nastaviti rad s obrađenim podacima kako je ovdje cilj samo pokazati mogućnosti automatizacije obrade napravljena je jednostavna Python skripta za stvaranje buffer zone. Ovakve je skripte moguće i generirati u arcgisscripting modulu te ih pohraniti kao Python skriptu.



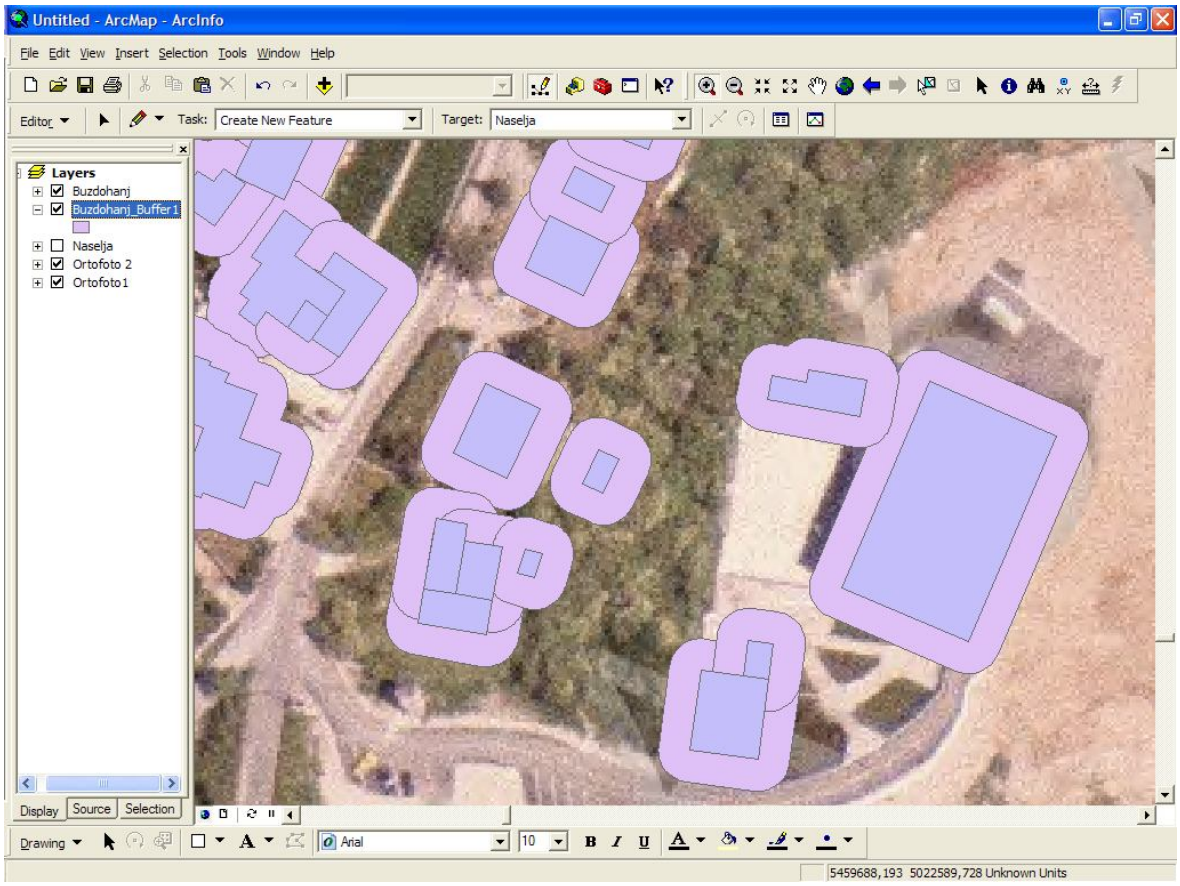

*Slika 20 Python buffer skripta za ArcGIS*

Početak skripte čini zaglavlje u kojem su navedeni osnovni podaci o skripti kao što su ime skripte, datum i način nastanka, eventualna povijest skripte ukoliko je naknadno mijenjana ili dorađivana i sl. Potom se pozivaju razne biblioteke kako bi Python mogao komunicirati s računalom i ArcGIS paketom te obrađivati shapefile podatke. Nadalje kreira se operator koji se povezuje s postojećim ArcToolbox rješenjima. I na kraju se definira operator pozivajući buffer naredbu uz koju se u zagradi navode parametri potrebni za izradu buffer zone sukladno parametrima unesenima osobno od strane korisnika u buffer izbornik (Slika 19).

### 4.2.3. Prezentacija rješenja

Kada se radi o prostornim podacima i rezultatima njihove obrade najkvalitetniji način je uvijek grafička prezentacija. Na slici (Slika 21) je prikazan isječak iz cjelokupne situacije kako bi se jasno vidjeli zadani objekti ( svijetlo plavi

pravokutnici) i zona oko njih (blijedo ljubičasti na uglovima zaobljeni poligoni). Interesantno je za napomenuti kako u situaciji kada su dva objekta dovoljno blizu dolazi do preklapanja zona što može biti ishodište daljnjih analiza.




*Slika 21 Rezultat buffer obrade podataka ArcGIS rješenjem*

### *4.3. Vlastito rješenje*

Za razliku od prije prikazanog rješenja i uopće načina prilazu problema kada se koristi vlastito rješenje problem je mnogo složeniji ali često je rješenje kvalitetnije i brže. Dakle za razliku od komercijalnog pristupa kada je dovoljno platiti rješenje koje je proizvođač osmislio za korisnika u slučaju samostalne izrade potrebno je mnogo više znanja i to ne samo o načinu izrade već i o samome problemu.

Za u ovome radu obrađeni problem potrebno je osim samog znanja programiranja u Pythonu što uključuje znanje sintakse programskog jezika, mogućnosti i ograničenja funkcija koje se mogu pozvati iz pojedinih biblioteka, općenitog znanja o strukturi pisanja programskog koda i znanja o načinu "razmišljanja" računala važno je poznavati i prirodu problema. Naime osim samog programiranja izuzetno

je važno poznavati problem koji se obrađuje do najsitnijih detalja kao što su količina, kvaliteta i detaljan opis ulaznih podataka obrade. Potrebno je poznavati i teoretsku pozadinu metoda kojima će se obrađivati podaci bilo da je metoda unaprijed definirana i moguće ju je pozvati iz neke od korištenih biblioteka ili je potrebno metodu isprogramirati koristeći se matematičkim operacijama i algoritmima koji čine njenu teorijsku pozadinu. I za kraj važno je znati što se očekuje kao rezultat obrade kako bi se i dobili željeni rezultati u zapisu koji se traži, a ne u zapisu koji sam programer korisnik osmisli.

Nakon što smo u ranijim poglavljima prikazali neke osnovne podatke o programskom jeziku Python i korištenim bibliotekama i na taj način stekli potrebno predznanje da možemo krenuti u izradu projekta potrebno je kreativno promisliti problem te stvoriti "tok misli" koji će pretočen u programski kod svojim pokretanjem dati rješenje.

### 4.3.1. Koncept rješenja

Kod kreativnog promišljanja o konceptu, koje kod samostalne izrade rješenja problema zahtjeva predviđanje svih problema i njihovih rješenja, koja će se pojaviti, kao posljedica ograničenih mogućnosti programskog jezika ili unaprijed definiranih metoda koje su korisniku na raspolaganju, uviđa se velika razlika od kreativnog pristupa pri korištenju komercijalnog rješenja gdje je sva kreativnost sadržana u pronalaženju adekvatnog modula koji nudi optimalno i najkvalitetnije rješenje.

Upoznavši Python i njemu priključene biblioteke funkcija i metoda kao što je to navedeno u ranijim poglavljima potrebno je uočiti kako nije moguće obrađivati direktno shapefile podatke već ih je potrebno pretvoriti u Shapely objekte. Kao posljedica toga javlja se problem izrade dvije konverzije podataka jer rješenje treba biti u shapefile zapisu. Potrebno je Python skriptom pristupiti ulaznim podacima, obaviti konverziju podataka, obraditi ih i potom obrađene podatke konverzijom u suprotnom smjeru vratiti u shapefile zapis.

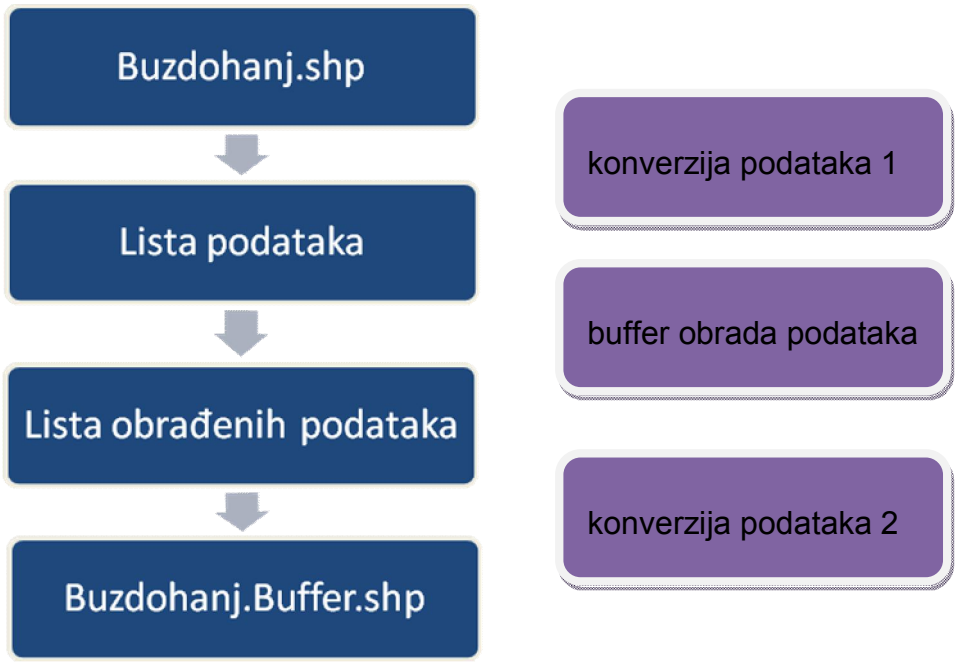
Također nakon konverzije podataka potrebno je uočiti problem koji nastaje kao posljedica manipulacije podacima. Veliki broj objekata, njih 1037 potrebno je na određeni način pohraniti u smislenu varijablu pri izradi programa. Za takve složene

podatke koji se sastoje od više vrijednosti, npr. geometrija koju čini niz točaka i svi navedeni atributi) nemoguće je koristiti jednodimenzionalni niz već se koriste liste (u nekim programskim jezicima nazivaju se stog) čiji je sadržaj i oblik najlakše zamisliti kao tablicu u kojoj su u istom retku zapisani podaci o jednom objektu, a u stupcima odnosno ćelijama u tom retku se nalaze podaci različitog tipa. (npr. U jednome retku u prvom polju nalazi se zapisana geometrija objekta čiji zapis čini niz točaka, u drugom polju može se nalaziti tekstualni podatak o nazivu naselja…). Dakle, potrebno je još napomenuti kako svaki stupac u svim redcima ima identičan tip podataka, nije moguće podatke o geometriji koji su zapisani na prvome mjestu, u sljedećem retku preseliti na drugu poziciju jer bi to narušilo strukturu podataka.

Nakon što je određen način na koji se pohranjuju podaci potrebno je krenuti u njihovu obradu. Za kvalitetnu obradu potrebno je stvoriti petlju koja će se kretati od prvog do posljednjeg podatka i osigurati da se ne pojave skokovi (da se ne preskoči koji podatak odnosno red) u izvršavanju naredbi. Taj problem jednostavno je riješiti petljom koja kreće od prvog (potrebno znati da se prvi objekt u strukturi podataka nalazi na nultom mjestu, drugi podatak na 1. itd.) objekta i red po red izvršava sve naredbe unutar petlje sve dok ne dođe do kraja liste.

Kako nakon obrade podataka nastaje nova geometrija potrebno je pohraniti novonastale podatke negdje. Moguća su dva rješenja ove situacije. Kao prvo rješenje moguće je nakon što se buffer naredbom obradi objekt i dobije se nova geometrija na mjesto geometrije objekta upisati nove vrijednosti. Dakle zamijenimo stare neobrađene vrijednosti novima. Druga mogućnost je kreiranje nove liste podataka i zapisivanje nove geometrije u praznu listu dodavanjem podataka. U objašnjenom rješenju korištena je druga mogućnost zbog preglednosti programa.

Kao što je na slici prikazano (Slika 22) shema izrade programskog koda kreće od ulaznih podataka, objašnjava redoslijed izvršavanja operacija i završava izradom izlaznog shapefila. Detaljniji opis operacija nad podacima sa svim navedenim naredbama biti će objašnjen kasnije pri prezentaciji i objašnjenju programskog koda.




*Slika 22 Shematski prikaz rješenja problema*

### 4.3.2. Izrada rješenja

Sama izrada rješenja u slučaju samostalne izrade skripte za obradu podataka sastoji se od već opisanog kreativnog promišljanja kojim se predviđaju postupci kojima će se podaci obraditi te pisanja programskog koda pomoću kojega se konceptualno rješenje pretvara u skriptu spremnu za izvršavanje.

Kako pri pisanju programskog koda veoma lako može doći do sitnih grešaka kao što su tipfeleri ili jednostavno propuštanje pojedine naredbe sami kod pisan je u Python APIu i prevođen je interpreterom u strojni jezik koje računalo razumije. Na taj način uočeni su određeni propusti koji su jednostavno ispravljeni.

#### 4.3.2.1 Pravila lijepog programiranja

Pod pravilima lijepog programiranja smatraju se pravila za pregledno i strukturirano programiranje koje olakšava snalaženje u programskom kodu kako autoru tako i budućem korisniku ukoliko se javi potreba za izmjenom ili doradom nekog dijela programskog koda. Takvih pravila ovisno o mišljenju pojedinog programera može biti od svega nekoliko pa gotovo do bezbroj. Na temelju osobnog iskustva autor ovog rada navesti će svega nekoliko pravila koja bi se trebalo pridržavati pri pisanju programskog koda.

Osnovno pravilo svakog dobrog programera je želja da se i nakon mnogo vremena vrlo lako prisjeti svog rada i olakša si eventualne dorade ili izmjene. Kako bi imao tu mogućnost svaki programer trebao bi koristiti mogućnosti komentara unutar samog programskog koda. Potrebno je definirati zaglavlje programa i navesti osnovne podatke o programu. Također preporuča se pojedine korake rješenja odnosno pojedine specifične linije programskog koda popratiti komentarom naročito ukoliko se radi o specifičnim rješenjima koja se ne koriste često te se lako zaborave.

Osim same dokumentacije programa većina programera koja ima iskustva rada na većim programima naučena je rastaviti problem na više manjih cjelina, potprograma. Takav pristup programiranju višestruko je isplativ jer je velika mogućnost da će određeni potprogrami biti potrebni i u nekim drugim slučajevima te ih nije potrebno ponovo stvarat već ih je potrebno samo pozvat ili kopirat u novi program čime se može uštedjeti puno vremena i rapidno povećati produktivnost.

I za kraj jedno od težih pravila za korisnike je strukturno pisanje programskog koda no Python taj je uvjet rješava na način da je obavezan i ukoliko ga se prekrši program se neće pravilno izvoditi. Pri strukturnom pisanju programskog jezika misli se na korištenje uvlaka kojima se odvajaju pojedini dijelovi programa, npr. naredbe koje se izvode unutar petlje uvuku se više od same petlje, (Slika 23).

```
def write_shapefile(lsFeat, outfile):
    """
    Stvara novi shapefile i zapisuje ga na disk.
    Input:
        lsFeat:    lista Featurea (a dico with geom/name/pop)
        outfile:   path za novokreirani fila
    Output:
        - (shapefile je zapisan na disk)

    """
    driver = ogr.GetDriverByName('ESRI Shapefile') #-- kreiramo novi SHP file
    if os.path.exists(outfile):
            driver.DeleteDataSource(outfile) #-- ako postoji prepisujemo ga
    ds = driver.CreateDataSource(outfile)
    layer = ds.CreateLayer(outfile, geom_type=ogr.wkbPolygon) #-- kreiramo SHP s poligonima

    for i in lsFeat:
        f = ogr.Feature(feature_def=layer.GetLayerDefn())
        p = ogr.CreateGeometryFromWkb(i['geom'].wkb)
        f.SetGeometry(p)
        layer.CreateFeature(f)
        f.Destroy()
    ds.Destroy()
    print "\nShapefile saved as:", outfile
```

*Slika 23 Prikaz pravilno napisanog programskog koda*

#### 4.3.2.2 Objašnjenje programskog koda

U ovome poglavlju detaljno će biti pojašnjen programski kod Python skripte izrađene u svrhu rješenja problemskog zadatka odnosno stvaranja buffer zone oko objekata. Kako smo već ranije naveli svaki se program sastoji do više manjih cjelina te će programski kod pri prezentaciji biti podijeljen na iste kako bi se lakše uočile same cjeline i olakšalo se shvaćanje rada programa. Samo spajanjem svih dijelova u jedan programski kod dolazimo do funkcionalne skripte koja pruža rješenje zadatka.

##### 4.3.2.2.1 Zaglavlje skripte

Kako smo već ranije naveli svaka skripta posjeduje zaglavlje. U zaglavlju su navedeni ime skripte, podaci o autoru, datum izrade i eventualno opis skripte koji služi kao dokumentacija skripte unutar samog programskog koda. Za ostvarenje zapisa svih navedenih podataka potrebno je koristiti određenu formu koja je u programskim jezicima definirana znakovima koji govore interpreteru kako pojedine retke treba preskočiti jer služe kao zaglavlje, dokumentacija, komentar, podsjetnik budućim korisnicama … U Python programskom jeziku znakom ljestvi daje se do znanja interpreteru da preskoči naredbe zapisane u tome radu, a znakom višestrukih navodnika označava se da je tekst unutar istih samo komentar i ne čini programske naredbe koje je potrebno prevesti u strojni jezik i izvesti ih.

```
#-- diplomski1050.py

#-- Marino Culjat <mculjat@geof.hr>

#-- 2010-07-15

""" Skripta napravljena kako bi citala shape file, obradila ga i stvorila novi izlazni file s rezultatima obrade. """
```

##### 4.3.2.2.2 Uvođenje biblioteka

Kako navedena skripta mora pristupiti na računalu pohranjenim podacima potrebno je u sam kod uvesti pojedine biblioteke. Prvo je uvedena os biblioteka čija je funkcija omogućiti pozivanje naredbi za pristupanje i pohranu podataka. Ova biblioteka ustvari omogućuje rad sa stazama (Path), a samim time osigurava

mogućnost pristupa podacima. Koristeći modul open() pristupa se fileu i koristi se pri čitanju i zapisivanju podataka.

```
#-- general import

import os
```

U narednom primjeru na sljedećoj stranici prikazane su naredbe za uvođenjem OGR biblioteka u Python skriptu. Prvi red čini naziv cjeline koji je nam govori kako se radi o OGR biblioteci. Tu se vidi kako se već koristi prije uvedena os biblioteka koja sada služi za pristup OGR biblioteci i njenim modulima. Izuzećem moguće greške pri uvođenju biblioteke osigurava se daljnji nastavak izvršenja programa i ukoliko uvođenje biblioteke ne uspije. Naravno u potonjem slučaju program bi se prestao izvršavati u trenutku prvog pozivanja neke naredbe iz OGR biblioteke. Prekid programa bio bi uzrokovan ne mogućnošću pristupa programa OGR biblioteci.

```
#-- OGR

try:

    from osgeo import ogr

except ImportError:

    import ogr
```

Kako je dosad navedeno pri uvođenu os i ogr biblioteka na isti način uvodi se i shapely. Navedenom naredbom iz shapely biblioteke uvode se potrebne naredbe i operacije korištene u izradi skripte kao štosu kreiranje Shapely objekata i operacija koje se nad njima mogu izvršavati

```
#-- Shapely

from shapely.wkb import loads as wkbloads
```

#### 4.3.2.2.3 Definiranje varijabli

Svaka skripta ili program podatke koji se obrađuje i čije se vrijednosti mogu mijenjati smatra varijablama (promjenjivim veličinama). Postoje dvije vrste varijabli, globalne i lokalne. Globalne varijable su podaci promjenjive vrijednosti koje se koriste u radu cijele skripte dok lokalne varijable čine trenutno potrebne vrijednosti bilo da se radi o brojaču pojedinih petlji ili derivatu globalnih varijabli kako bi se smanjila količina podataka koji se obrađuju a samim time i ubrzalo izvršavanje programa.

U programskom jeziku Python varijable se definiraju i zadaju veoma lako, nije ih potrebno unaprijed navodit, definirati tip podataka već se to automatski generira pri nastanku varijable kojoj se dodaje pojedina vrijednost. U slučaju naše globalne varijable imena INFILE znakom jednakosti "=" dodjeljuj joj se vrijednost i ona predstavlja shapefile "Buzdohanj .shp"

```
#-- Global variables

INFILE = "Buzdohanj.shp"
```

#### 4.3.2.2.4 Glavni program

Kako je već navedeno korisno je pri izradi programa koristiti strukturu koja se sastoji od komponenti (funkcija i procedura) tj. potprograma koji se u glavnom programu pozovu uz navođenje parametara koji su potrebni za njihovo izvršavanje.

U navedenom promjeru definiran je glavni dio programa koji se sastoji od pozivanja dvije funkcije. Prva funkcija "process_shapefile" čita podatke, kreira listu, obrađuje podatke u listi i stvara novu listu s obrađenim podacima koju kao rezultat funkcije vraća u glavni dio programa. Druga funkcija "write_shapefile" od vraćene liste stvara novi shapefile i u njega prepisuje podatke iz liste u shape formatu zapisa. Ulazni parametri navedeni su u zagradama nakon imena pozvanih funkcija. Kod prve funkcije to je globalna varijabla "INFILE" što je u stvari ulazni shapefile, a kod druge funkcije to su dva parametra, lista "lsFeat" koja je dobivena kao rezultat provedene obrade nad podacima i ima novoga shapefila "new.shp".

```
def main():

    lsFeat = process_shapefile(INFILE)

    write_shapefile(lsFeat, "new.shp")
```

4.3.2.2.5 Funkcija "process_shapefile"

Funkcija "process_shapefile" čiji je kod naveden u primjeru ispod kao ulazne podatke koje obrađuje koristi globalnu varijablu "infile". Ranije je navedeno kako je globalna varijabla u ovoj skripti ulazni shapefile Buzdohanj.shp. Rezultat funkcije je lista geometrija buffer zone. Dakle lista je ustvari niz koji u svakom svom polju ima za vrijednost niz koordinata rubnih točaka buffer zone.

Nakon samog definiranja funkcije nalazi se naredba " print "\nProcessing shapefile", infile" kako bi se pri izvršenju programa znalo kad počinje proces obrade i što skripta trenutno radi

Slijedeći programske naredbe koje su navedenu u primjeru ispod redom se putem ogr.Open naredbe pristupa shapefilu Buzdohanj.shp. Potom se pristupa layeru podataka (razini, nivou podataka) pri čemu je važno istaknuti kako se jednostavni shapefilovi poput obrađivanog Buzdohanj.shp filea sastoje od samo jednog layera. U sljedećem koraku definira se sadržaj i struktura layera naredbom "GetLayerDefn", a odmah potom naredbom "GetFeatureCount" (feature u ArcGIS paketu predstavlja objekt) utvrđuje se broj objekata sadržanih u fileu.

Nakon ostvarenog pristupa podacima te prikupljenih saznanja o broju objekata i osobinama podataka koji se obrađuju potrebno je kreirati listu. Naredbom "lsFeat= [] " kreira se prazna lista objekata (eng. featura) u Shapely formatu

Nakon što su ostvareni svi preduvjeti za pristup i obradu podataka potrebno je kreirati petlju koja će proći svim podacima i obraditi ih. To je učinjeno "for" petljom s brojačem "i" koji se kreće od nula do definiranog broja objekata "noFeat" s porastom brojača nakon izvršene naredbe za jedan što je u programu zapisano kao " for i in xrange(noFeat):".

```
def process_shapefile(infile):
    print "\nProcessing shapefile", infile
    ge = ogr.Open(infile)
    layer = ge.GetLayer(0)
    ld = layer.GetLayerDefn()
    noFeat = layer.GetFeatureCount()
    lsFeat = []
    for i in xrange(noFeat):
        f = layer.GetFeature(i)
        geom = wkbloads(f.GetGeometryRef().ExportToWkb())
        geom = geom.buffer(5,16)
        lsFeat.append({'geom': geom, 'name': f.GetFieldAsString(0)})
    return lsFeat
```

Unutar same petlje koristeći naredbu "GetFeature" pristupa se objektu. Potom se čita geometrija objekta u WKB (Well Known Binary) formatu koji je brži od WKT (Well Known Text ) zapisa i koristeći Shapely funkciju wkbloads provodi se konverzija podataka kojom se iz WKB zapisa kreira Shapely objekt. Kreirani Shapely objekt moguće je obraditi unaprijed definiranim procedurama te se pozivamo naredbe "buffer" sa zadanim parametrima (širina zone: 5, broj piksela koji se koriste pri zaglađivanju rubova: 16) kreira zahtijevano rješenje. Kako navedena funkcija kao rezultat obrade vraća listu objekata sa kreiranom buffer zonom, a lista koja je kreirana prije petlje je prazna potrebno je korištenjem naredbe "append" dodati na kraj liste novi objekt.

Cijeli, u prijašnjem poglavlju naveden i objašnjen, proces provodi se unutar for petlje. Dakle kako se u Buzdohanj.shp filu nalazi 1037 objekata ovaj će se proces

provesti isto toliko puta. Potrebno je samo naglasiti kako će zadnja vrijednost brojača "i" biti 1036 jer brojač počinje brojati od nule.

Konačno, kraj jednog potprograma kao što je funkcija "process:shapefile" završava naredbom koja usmjerava rezultat natrag u glavni program što se ovdje postiže naredbom "return lsFeat" ("vrati listu objekata").ž

#### 4.3.2.2.6 Funkcija "write_shapefile"

"Write_shapefile" funkcija kako joj samo ime govori piše shapefile, odnosno kreira novi shapefile i u njega zapisuje podatke o geometriji objekata buffer zone kako bi dobiveni rezultati bili pohranjeni u shape zapisu.

```
def write_shapefile(lsFeat, outfile):
    driver = ogr.GetDriverByName('ESRI Shapefile')
    if os.path.exists(outfile):
            driver.DeleteDataSource(outfile)
    ds = driver.CreateDataSource(outfile)
    layer = ds.CreateLayer(outfile, geom_type=ogr.wkbPolygon)
    for i in lsFeat:
        f = ogr.Feature(feature_def=layer.GetLayerDefn())
        p = ogr.CreateGeometryFromWkb(i['geom'].wkb)
        f.SetGeometry(p)
        layer.CreateFeature(f)
        f.Destroy()
    ds.Destroy()
    print "\nShapefile saved as:", outfile
```

Ulazni parametri funkcije su lista objekata (featura) i staza (path) za novokreirani shapefile. Koristeći naredbu "GetDriverByName" OGR biblioteke kreira se novi shape file. If upitom provjerava se postoji li već shapefile takvog imena na zadanoj lokaciji i ukoliko postoji briše ga se kako bi se nakon zapisivanja podataka iz liste featura u izlaznom shapefilu nalazili samo podaci koji su rezultat obrade. Nakon provjere sadržaja izlazne datoteke naredbom poveznice sa podacima i definiramo sadržaj razine podataka (layera). Za povezivanje s podacima koristi se naredba "CreateDataSource", a za kreiranje layera odnosno shapefila s poligonima naredba "CreateLayer"

Nakon kreiranog shapefila i zadanog opisa objekta koji će se u njemu nalaziti (poligoni) potrebno je pročitati cijelu listu objekt po objekt i prepisati podatke iz liste u shapefile kreirajući za svaki objekt u listi novi feature u shapefilu. Taj proces odrađen je "for" petljom s brojačem "i" koji se kreće od nule do kraja liste što je Python sintaksom zapisano kao " for i in lsFeat:"

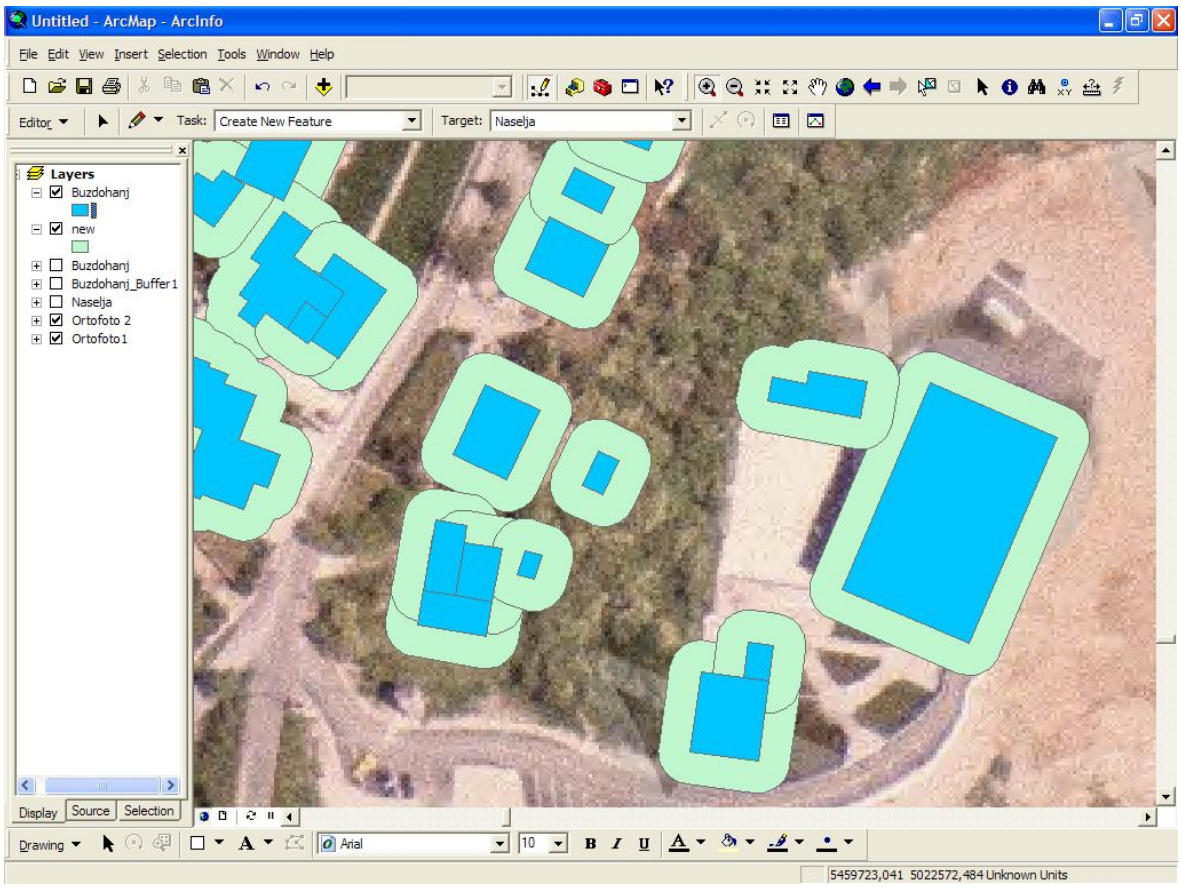
Unutar navedene for naredbe za svaki se objekt koji se nalazi upisan u listi prvo definira feature "f" pomoću naredbe "ogr.Feature", potom se od geometrijskih podataka upisanih u listu kreira poligon "p" pomoću naredbe iz OGR biblioteke, "ogr.CereateGeometryFromWkb". Naredbom "f.SetGeometry(p)" pridružuje se featuru "f" geomerija poligona "p" te se naredbom "layer.CreateFeature(f)" u shapefileu stvara novi feature sa geometrijom koja je pohranjena u featuru "f". Prije završetka jednog prolaza petlje potrebno je obrisati definirani feature "f" kako bi pri idućem prolazu petlje zadani feature bio prazan (bez upisanih podataka) i mogao se iskoristiti za manipulaciju sljedećim objektom.

Nakon završenog prepisivanja podataka iz liste u shapefile briše se veza između podataka i kreiranog shapefila i ispisuje se na ekranu računa mjesto na kojem korisnik može pronaći pohranjen novi shapefile sa rezultatima obrade podataka. Za ispis lokacije u memoriji računala na kojoj se izlazna datoteka nalazi koristi se naredba " print "\nShapefile saved as:", outfile "

### 4.3.3. Prezentacija rješenja

Kako je već navedeno kod prezentacije komercijalnog rješenja problema, kada se radi o prostornim podacima i rezultatima njihove obrade najkvalitetniji način je

uvijek grafička prezentacija. Na slici (Slika 24) je prikazan isječak iz cjelokupne situacije kako bi se jasno vidjeli zadani objekti ( plavi pravokutnici) i zona oko njih (svijetlo plavi na uglovima zaobljeni poligoni). Interesantno je za napomenuti kako u situaciji kada su dva objekta dovoljno blizu dolazi do preklapanja zona što može biti ishodište daljnjih analiza.

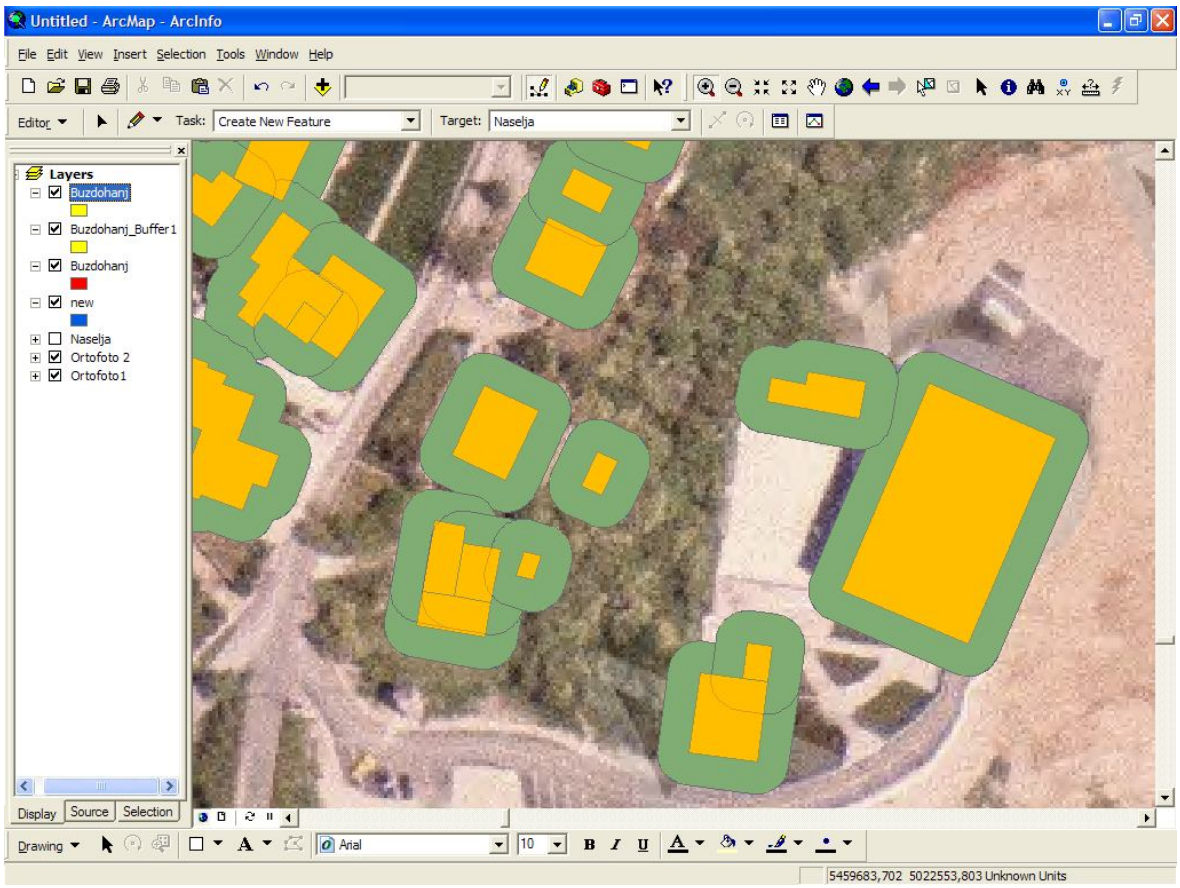



*Slika 24 Rezultat buffer obrade podataka Python rješenjem*

### *4.4. Usporedba rješenja*

Iako Python ne može direktno vršiti obradu shapefile podataka već ih mora konverzijom prevesti u WKB ili WKT zapis iz kojega pomoću Shapely biblioteke stvara geometrijske objekte nad kojima može vršiti analizu, nakon čega je potrebno opet konverzijom podataka stvoriti shapefile sa obrađenim podacima u slučaju buffer analize vrijeme potrebno za realizaciju svih navedenih i u ranijim poglavljima opisanih operacija za 1037 objekata iznosi manje od dvije sekunde i mnoge je kraće od vremena potrebnog ArcGIS aplikacijama za izradu identične analize.

Usporedni prikaz obaju rješenja te njihovo potpuno preklapanje prikazano je na slici (Slika 25). Ulazni podaci za analizu prikazani su u dva layera s različitim bojama, žutom i crvenom, pri čemu je za nadređeni layer primjenjena transparentnost ispune kako bi došlo do miješanja boje koje bi dokazalo preklapanje objekata. Na isti je način prikazana buffer zona samo su ovoga puta korištene žuta i plava boja koje su svojim miješanjem stvorile zelenu boju zone oko objekata. Također promatranjem svih objekata vidljivo je kako nije došlo do pomicanja pošto nije moguće uočiti dvostruke rubne linije objekata ili sl. anomalije.




*Slika 25 Istovremeni prikaz potpunog preklapanja oba rješenja*

Sukladno svemu dosad navedenome nameće se zaključak kako se radi o dva identična rješenja do kojih se došlo korištenjem dvaju različitih koncepata. Jedno je komercijalno unaprijed definirano rješenje koje ukoliko se koristi sve njegove opcije može podatke obrađivati i više od 10 sekundi. Drugo je rješenje nekomercijalno, rad je autora ovog rada, namijenjeno je isključivo izradi buffer zone i za izvođenje takvog programskog rješenja trebalo je značajno manje vremena odnosno manje od dvije sekunde.

### *4.5. Centroid*

*Centroid* je praktična naredba koja od cijelog poligona vraća samo jednu točku, centar poligona. Naime, u slučajevima kada se obrađuju određeni statistički podaci često nije bitan geometrijski oblik poligona već atributni podaci koji ga opisuju pa je za smještaj informacija u prostor dovoljno imati koordinate jedne točke uz koju se vezuju atributi. Na primjeru podataka korištenih u ovome radu za analize nad stanovništvom koje se nalazi u pojedinom stambenom objektu nije bitan sam geometrijski oblik objekta već su bitni prostorni podaci poput površine i u kojem naselju, odnosno jedinici lokalne samouprave se nalazi objekt. Kako bi se olakšalo izvođenje takvih statističkih analiza naredbom *centroid* moguće je poligon prezentirati točkom u prostoru.

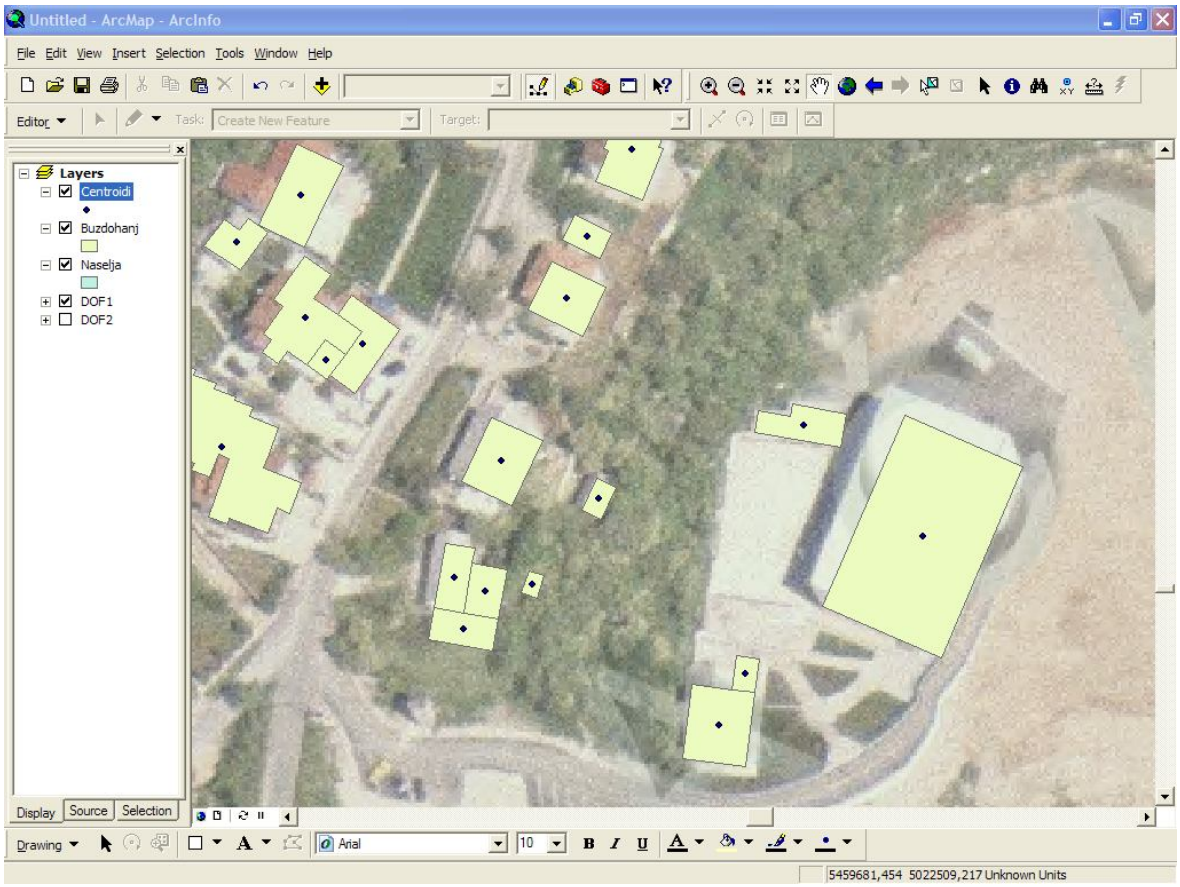
```
def process_shapefile(infile):
    print "\nProcessing shapefile", infile
    ge = ogr.Open(infile)
    layer = ge.GetLayer(0)
    ld = layer.GetLayerDefn()
    noFeat = layer.GetFeatureCount()
    lsFeat = []
    for i in xrange(noFeat):
        f = layer.GetFeature(i)
        geom = wkbloads(f.GetGeometryRef().ExportToWkb())
        geom = geom.centroid
        lsFeat.append({'geom': geom})
    return lsFeat
```

U prozoru iznad nalazi se kod potprograma koji vrši obradu prostornih podataka. Kako je već ranije objašnjen način čitanja podataka za ovaj potprogram jedino potrebno istaknuti kako se u odnosu na *buffer* operaciju sada koristi naredba *centroid*. Naredba *centroid* poziva se odvojena točkom od poligona nad kojim se izvršava (npr. geom = geom.centroid). Kako u navedenom primjeru koristimo dva puta varijablu "geom" potrebno je naglasiti kako plavo označena varijabla geom predstavlja točku koja kao rezultat operacije *centroid* reprezentira poligon geom koji je zapisan crnim tekstom.

Važan detalj koji je potrebno uočiti kao razliku između shapefila nastalog kao rezultat obrade podataka *buffer* naredbom i *centroid* naredbom je u tipu geometrijskih podataka koji su zapisani u izlaznoj datoteci. Provedbom *buffer* naredbe uvijek su izlazni podaci poligoni dok se provedbom *centroid* naredbe za izlazni tip geometrijskih podataka uvijek dobiju točke. Zbog navedene razlike bilo je potrebno promijeniti tip podataka koji će se zapisivati u izlaznu shapefile datoteku što se vidi u prozoru ispod. Pri kreiranju *layera* definirano je kako je tip geometrijskih podataka koji će se zapisivati točka i to naredbom "geom_type=ogr.wkbPoint"

```
def write_shapefile(lsFeat, outfile):
    driver = ogr.GetDriverByName('ESRI Shapefile')
    if os.path.exists(outfile):
            driver.DeleteDataSource(outfile)
    ds = driver.CreateDataSource(outfile)
    layer = ds.CreateLayer(outfile, geom_type=ogr.wkbPoint)
    for i in lsFeat:
        f = ogr.Feature(feature_def=layer.GetLayerDefn())
        p = ogr.CreateGeometryFromWkb(i['geom'].wkb)
        f.SetGeometry(p)
        layer.CreateFeature(f)
        f.Destroy()
    ds.Destroy()
```

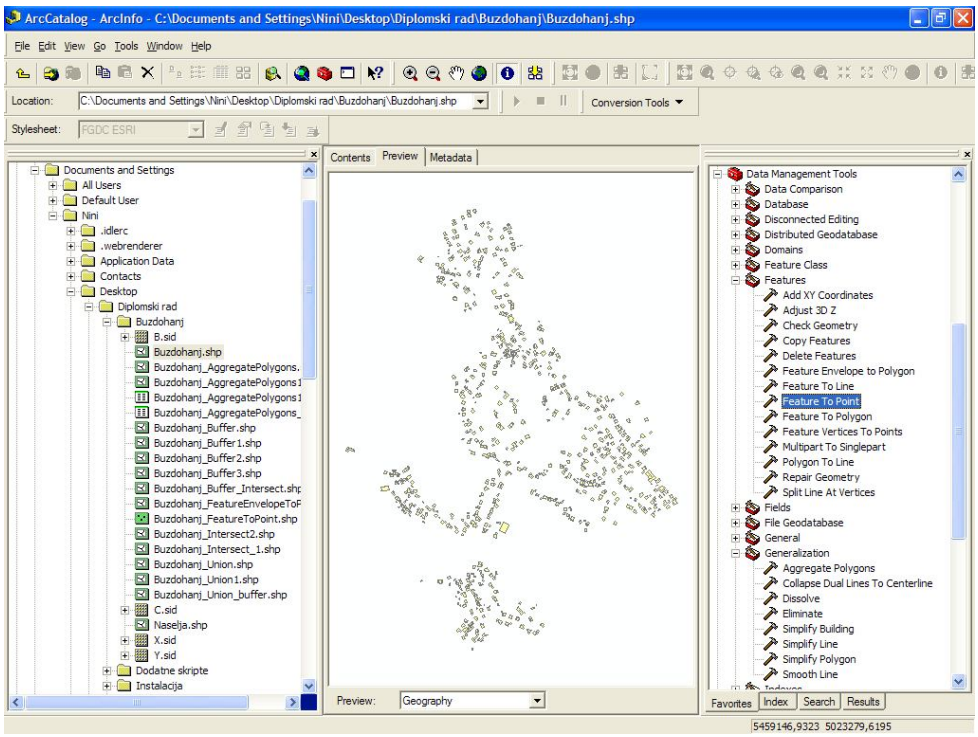
Kao vizualna prezentacija dobivenog rješenja služi nam slika (Slika 26) na idućoj stranici na kojoj se u podlozi nalazi digitalni ortofoto i žutozeleni poligoni (originalni podaci). Na podlozi se nalazi više plavih točaka koje su točkasta prezentacija poligona, tj. centroidi istih. Kako je već ranije objašnjeno na taj način je zadržana prostorna komponenta atributnih podataka koji se mogu statistički analizirati i obrađivati.




*Slika 26 Prikaz prezentacije poligona centroidima (plava točka unutar poligona)*

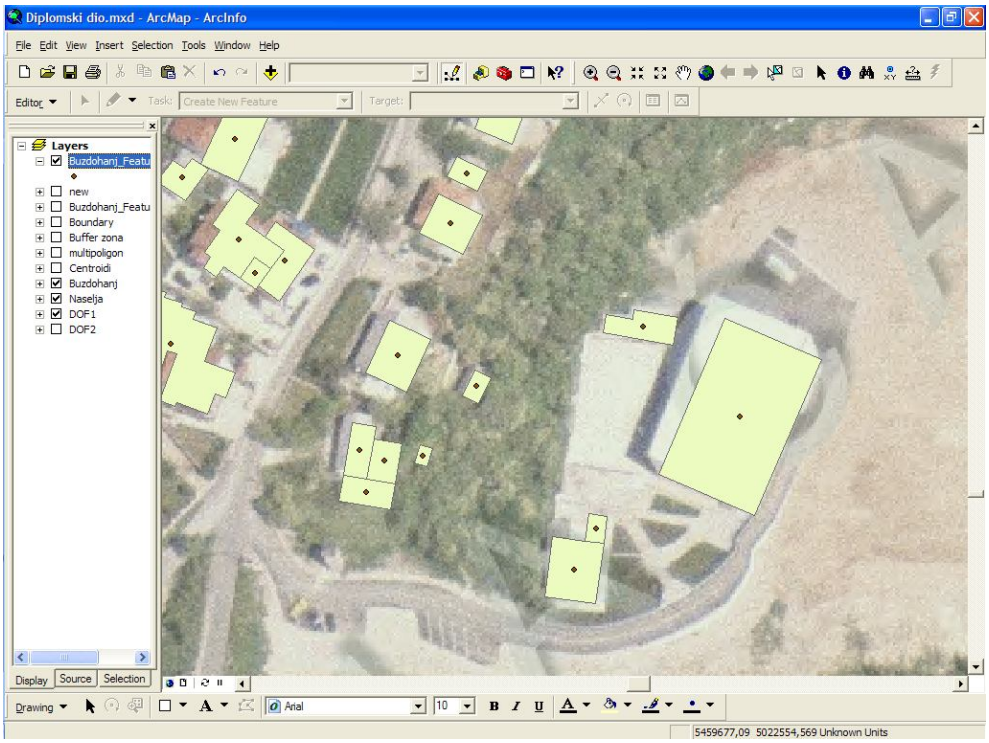
Isti se rezultat može dobiti i izradom centroida u ArcGIS aplikaciji korištenjem za to postojećih predefiniranih rješenja. Pri izradi prezentacije poligona točkom odnosno centroidom potrebno je koristiti alat *Feature to Point* koji nam daje identičan rezultat kao i naredba centroid u Shapely biblioteci programskog jezika Python.

Kako se vidi na slici (Slika 27) u prozoru Arccataloga potrebno je trećem djelu prozora koji sadrži prikaz u obliku stabla svih dostupnih alata odabrati skupinu *Data Management Tools* te s otvorenog padajućeg izbornika treba odabrati grupaciju alata namijenjenih obradi objekata pod nazivom *Features*. Kako bismo pokrenuli alat tj. naredbu koja poligone pretvara u točke tj. zamjenjuje poligone njihovim centroidima potrebno je pokrenuti alat *Feature to point*. Nakon pokretanja navedene naredbe otvoriti će se prozor koji ima samo dva polja. Prvo polje je *Input Features* i u tom je polju potrebno definirati ulaznu datoteku. U ovome slučaju to je Buzdohanj.shp. ArcGIS sam kreira ime i mjesto pohrane izlazne datoteke koje je moguće ručno promijeniti, a sve navedeno radi se u drugome polju *Output Feature Class*.




*Slika 27Prikaz prozora ArcCataloga i pristupa Feature To Point alatu*

Nakon izvršene obrade dobiveni rezultat istovjetan je onome dobivenome pomoću Python skripte gdje je obrada izvršena korištenjem naredbi Shapely biblioteke. Rezultat obrade podataka u ArcGISu vidi se na slici ispod (Slika 28).




*Slika 28 Prikaz rješenja dobivenog korištenjem ArcGIS alata*

### *4.6. Envelope*

Envelope odnosno prevedeno na hrvatski omotnica je naredba koja za rezultat svoje obrade vraća pravilni četverokut (pravokutnik) opisan poligonu na kojem je izvršena naredba. Četverokut koji je rezultat provedbe opisane naredbe definiran je na način da se najistaknutije rubne točke originalnog poligona za svaki smjer (sjever, jug, istok i zapad) nalaze na liniji koja omeđuje omotnicu. Također međne linije omotnice paralelne su sa X i Y osima kartezijevog pravokutnog koordinatnog sustava.

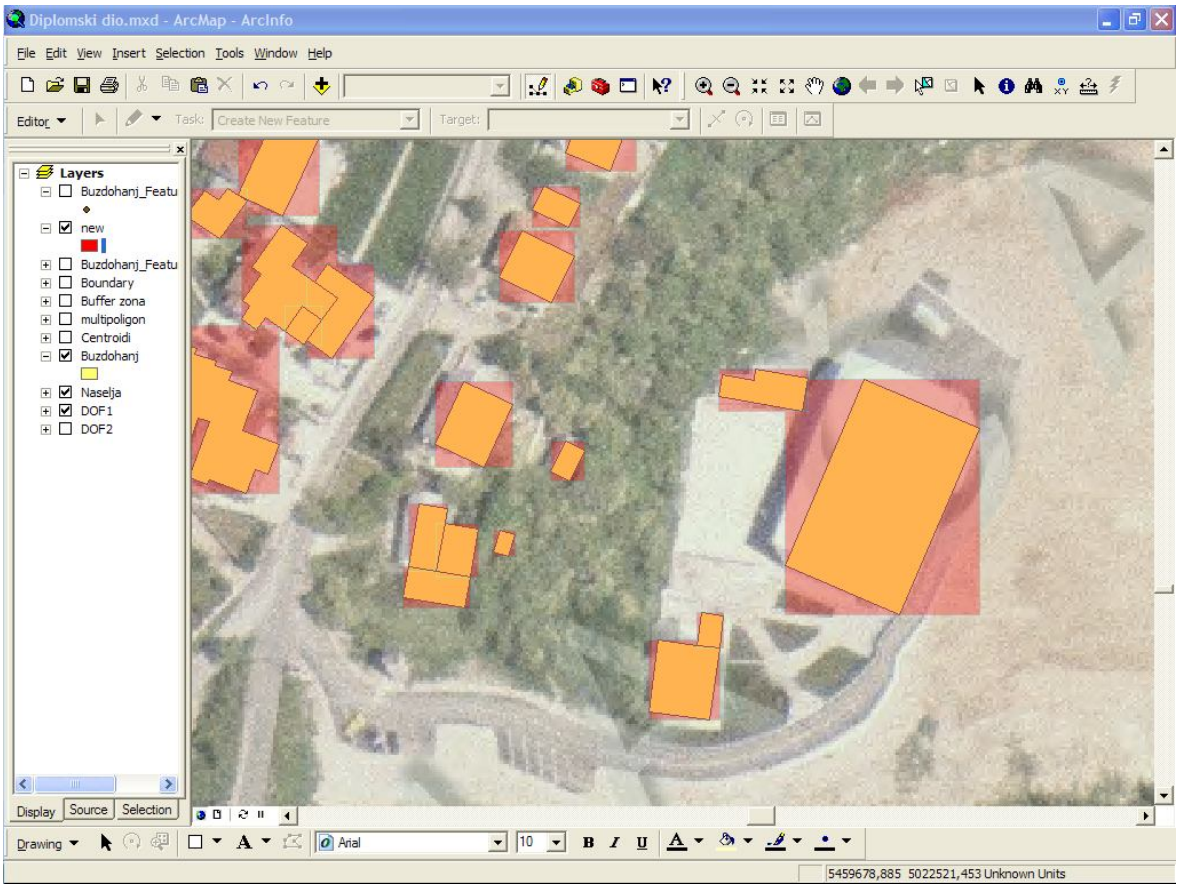
Pogledom na programski kod koji se nalazi u prozoru ispod vidi se kako je metoda čitanja zapisa ulazne datoteke shapefila identična metodama korištenim u dosadašnjoj izradi npr. *buffer* zone. Kako se radi o jednostavnim analizama jedina zbilja prava razlika je u naredbi čije se izvršavanje poziva. U ovome slučaju pozivamo nakon navedene varijable ulaznog poligona "geom" naredbu *envelope* koju znakom "." odvajamo od imena varijable koja će biti obrađena.

```
def process_shapefile(infile):
    print "\nProcessing shapefile", infile
    ge = ogr.Open(infile)
    layer = ge.GetLayer(0)
    ld = layer.GetLayerDefn()
    noFeat = layer.GetFeatureCount()
    lsFeat = []
    for i in xrange(noFeat):
        f = layer.GetFeature(i)
        geom = wkbloads(f.GetGeometryRef().ExportToWkb())
        geom = geom.envelope
        lsFeat.append({'geom': geom})
    return lsFeat
```

Također kako su rezultat ove analize opet poligoni, a takav je kod potprograma za stvaranje izlazne datoteke već objašnjen ranije u poglavlju 4.2.2 ovdje nećemo ponovno navoditi ista objašnjenja već samo prikazujemo kod potprograma u prozoru ispod.

```
driver = ogr.GetDriverByName('ESRI Shapefile')
    if os.path.exists(outfile):
            driver.DeleteDataSource(outfile)
    ds = driver.CreateDataSource(outfile)
    layer = ds.CreateLayer(outfile, geom_type=ogr.wkbPolygon)
    for i in lsFeat:
        f = ogr.Feature(feature_def=layer.GetLayerDefn())
        p = ogr.CreateGeometryFromWkb(i['geom'].wkb)
        f.SetGeometry(p)
        layer.CreateFeature(f)
        f.Destroy()
    ds.Destroy()
```

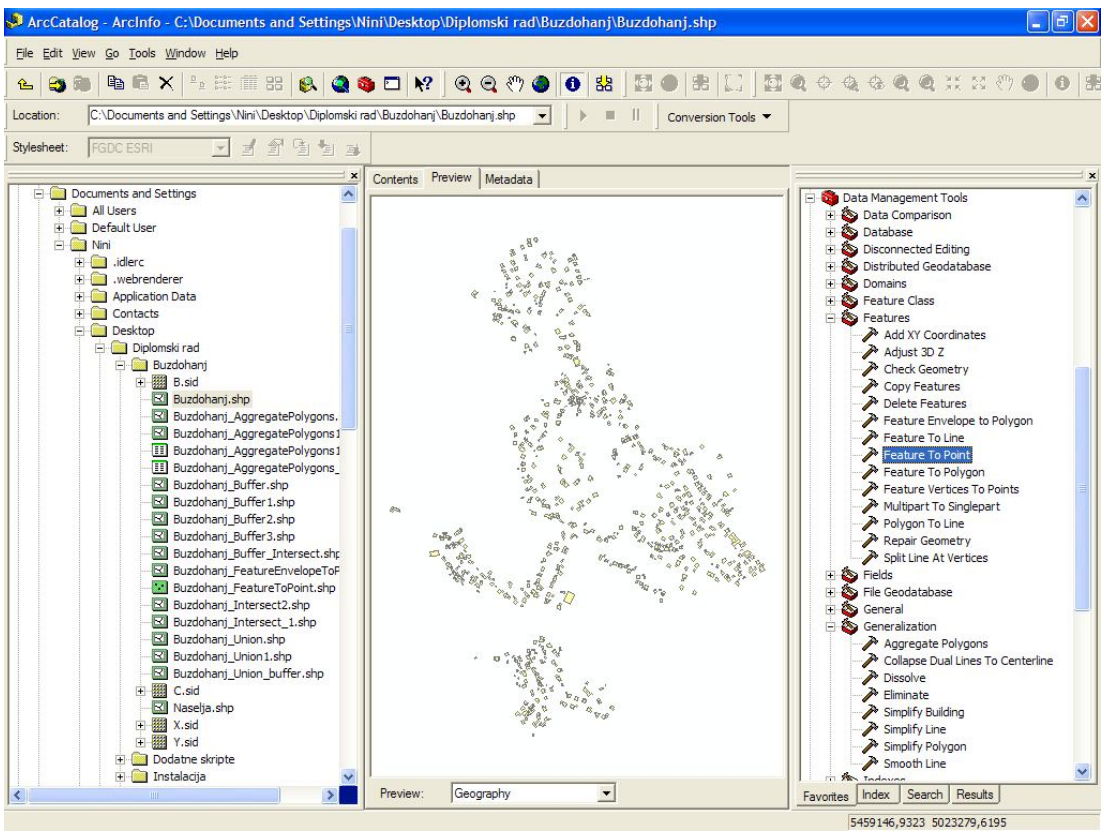
Rezultat obrade podataka metodom kreiranja omotnice vidi se na slici ispod (Slika 29). Crvenom bojom prikazane su omotnice oko objekata koji su zbog preklapanja prikazani u narančastoj boji.




*Slika 29 Prikaz rezultat obrade podataka metodom omotnice*

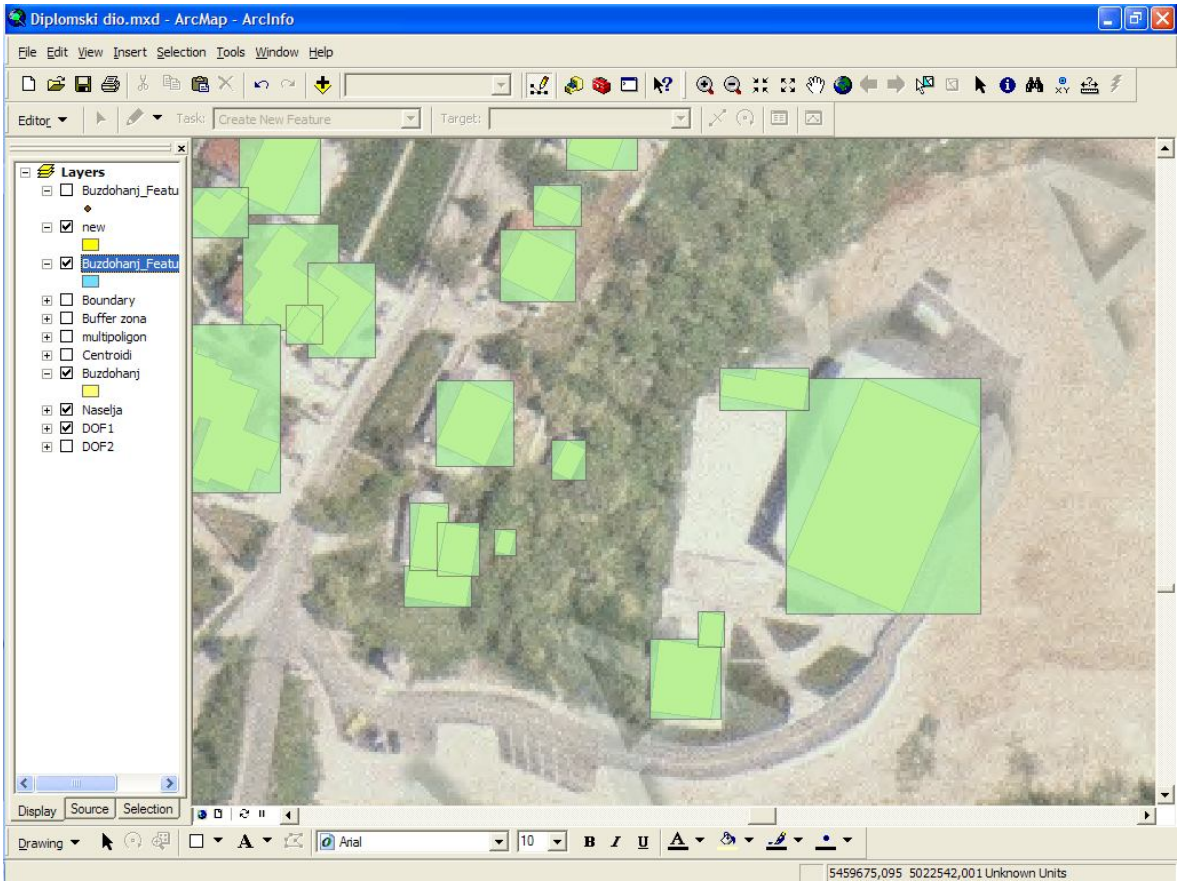
Isti se rezultat može dobiti i izradom omotnice u ArcGIS aplikaciji korištenjem za to postojećih predefiniranih rješenja. Pri izradi omotnice oko postojećeg objekta (poligona) potrebno je koristiti alat *Feature Envelope to Polygon* koji nam daje identičan rezultat kao i naredba *envelope* u Shapely biblioteci programskog jezika Python.

Kako se vidi na slici (Slika 30) u prozoru Arccataloga potrebno je trećem djelu prozora koji sadrži prikaz u obliku stabla svih dostupnih alata odabrati skupinu *Data Management Tools* te s otvorenog padajućeg izbornika treba odabrati grupaciju alata namijenjenih obradi objekata pod nazivom *Features*. Kako bismo pokrenuli alat tj. naredbu koja opisuje pravilni četverokut svakom poligonu tj. objektu potrebno je pokrenuti alat *Feature Envelope to Polygon*. Nakon pokretanja navedene naredbe otvoriti će se prozor koji ima samo dva polja. Prvo polje je *Input Features* i u tom je polju potrebno definirati ulaznu datoteku. U ovome slučaj uto je Buzdohanj.shp. ArcGIS sam kreira ime i mjesto pohrane izlazne datoteke koje je moguće ručno promijeniti, a sve navedeno radi se u drugome polju *Output Feature Class*.




*Slika 30 Prikaz prozora ArcCataloga i pristupa Feature Envelope to Polygon alatu*

Nakon izvršene obrade dobiveni rezultat istovjetan je onome dobivenome pomoću Python skripte gdje je obrada izvršena korištenjem naredbi Shapely biblioteke. Rezultat obrade podataka u ArcGISu vidi se na slici ispod (Slika 31). Zbog potpunog preklapanja rješenja dobivenih Python skriptom (prikazana žutom bojom) i rješenja dobivenih korištenjem ArcGIS alata (prikazana plavom bojom ) na slici su vidljivi poligoni zelene boje predstavljaju potpuno preklopljena rješenja dobivena na dva različita načina.




*Slika 31 Prikaz rješenja dobivenog korištenjem ArcGIS alata*

# 5. Pregled dodatnih Python rješenja

U sljedećim ćemo poglavljima prikazati još neke od mogućnosti obrada prostornih podataka pomoću skripti izrađenih u programskom jeziku Python. Za svaku od tih operacija nad podacima navedeni su i neki primjeri uporabe.

## *5.1. Unija*

Iz sličnih razloga onima navedenima kod operacije kreiranja centroida moguće je da se pojavi potreba i za ujedinjavanjem pojedinih poligona ili mnoštva poligona u jedan poligon. Veoma korisna stvar kod Pythona sa instaliranom Shapely bibliotekom je postojanje *MultiPolygon* tipa geometrijskih podataka. Taj tip podataka ustvari omogućava spajanje više poligona u jedan neovisno o topološkom odnosu između njih. Dakle za kreiranje *Multipolygona* nije potrebno da se poligoni dodiruju ili preklapaju već mogu biti bez dodirnih točaka i udaljeni jedan od drugoga.

Upotreba takvog tipa poligona biti će prikazana u sljedećoj Python skripti koja za cilj ima prikazati kreiranje unije između više poligona. U navedenoj skripti kreirati će se poligon koji će sadržavati sve poligone iz ulaznog shapefila odnosno iz naselja Buzdohanj. Tako dobiveni rezultat može poslužit za obrađivanje podataka naselja. Ukoliko se još matematičkim metodama i raznim drugim algoritmima zbroje svi podaci koji su od interesa jedinicama lokalne samouprave dobiva se ukupni zbir tih podataka i olakšava se računanje prosječnih vrijednosti (npr. broj stambenih kvadrata po stanovniku naselja, ukupna iskorištenost površina građevinskog zemljišta, …) Nadalje, kada smo naveli da se na taj način formira jedan poligon koji prikazuje svo izgrađeno zemljište olakšava se i daljnje projektiranje razvoja naselja, bilo da se radi o komunalnoj infrastrukturi (plinovod, vodovod, kanalizacija..) ili o kreiranju novog urbanističkog plana. Na taj se način omogućuje i lakše kreiranje razmještaja uslužnih objekata za stanovnike jer se zna koje su površine slobodne.

Naravno, moguće je stvarati i manje unije poligona koje ovise o nekom atributnom podatku. Na navedenom setu podataka postoje više vrste objekata (stambeni, gospodarski, javni) pa je po kriteriju vrste objekta moguće načiniti uniju objekata.

Također moguće je i načiniti uniju objekata koji pripadaju istom vlasniku ili pripadaju pod isti kućni broj, npr. stambeni i gospodarski objekt (garaža ili podrum). Takvim spajanjem više poligona pojedinog vlasnika moguće je odrediti davanja koja je vlasnik dužan podmirit na ime nekretnina koje posjeduje. Valja istaknuti kako se u slučaju više nekretnina istog vlasnika tada može isporučivati jedan račun i jedno rješenje za sve nekretnine dok bez takvog grupiranja objekata za svaki bi objekt bilo potrebno zasebno rješenje i zasebni račun.

Nakon što smo istaknuli mnoge koristi i upotrebe kreiranja unije podataka u prozoru ispod nalazi se kod potprograma koji čita ulazni shapefile te kroz petlju prolazi od prvog do posljednjeg zapisa.

```
def process_shapefile(infile):
    print "\nProcessing shapefile", infile
    ge = ogr.Open(infile)
    layer = ge.GetLayer(0)
    ld = layer.GetLayerDefn()
    noFeat = layer.GetFeatureCount()
    lsFeat = []
    z = layer.GetFeature(0)
    geom = wkbloads(z.GetGeometryRef().ExportToWkb())
    for i in xrange(noFeat):
        f = layer.GetFeature(i)
        pol = wkbloads(f.GetGeometryRef().ExportToWkb())
        geom = geom.union(pol)
    lsFeat.append({'geom': geom})
    return lsFeat
```

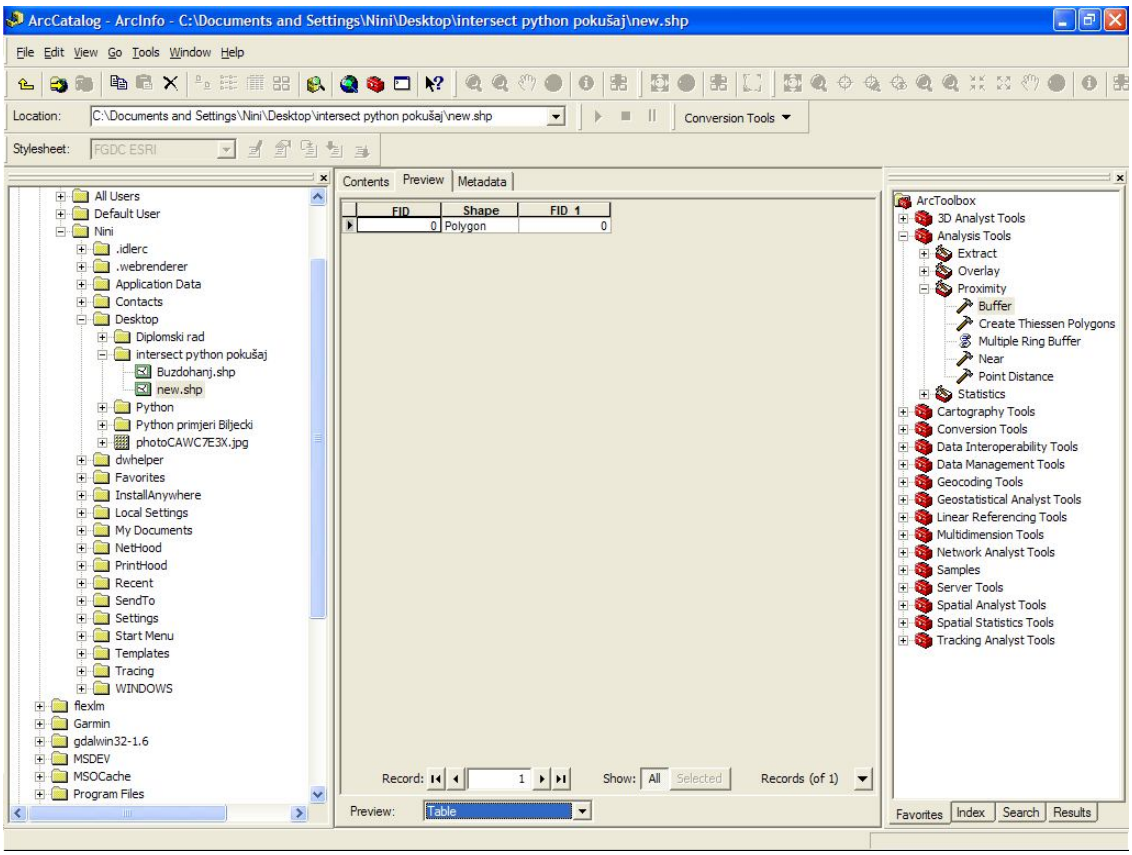
Svaki puta kada se pročita novi zapis pomoću naredbe *union* kreira se unija prijašnjih poligona i novog tek pročitanog. Kako bi se mogla izvršiti unija objekata potrebno je zadati početnu vrijednost što je napravljeno na način da je prije same petlje pročitan samo prvi zapis i postavljen kao početna vrijednost. Zbog navedene potrebe formiranja početne vrijednosti valjalo je koristiti više varijabli pa se u ovome potprogramu koriste varijabla "geom" kao varijabla kojoj se zadaje početna vrijednost i koja je na kraju rezultat svih obrada. Svaki novi zapis privremeno se

sprema u varijablu "pol" koja je pomoćna varijabla i koristi se isključivo tijekom obrade podataka. Nakon što su pročitani svi zapisi i izvršeno kreiranje svih unija poligona rezultat se zapisuje u listu koja sadrži samo jedan zapis sa *Multipolygonom*.

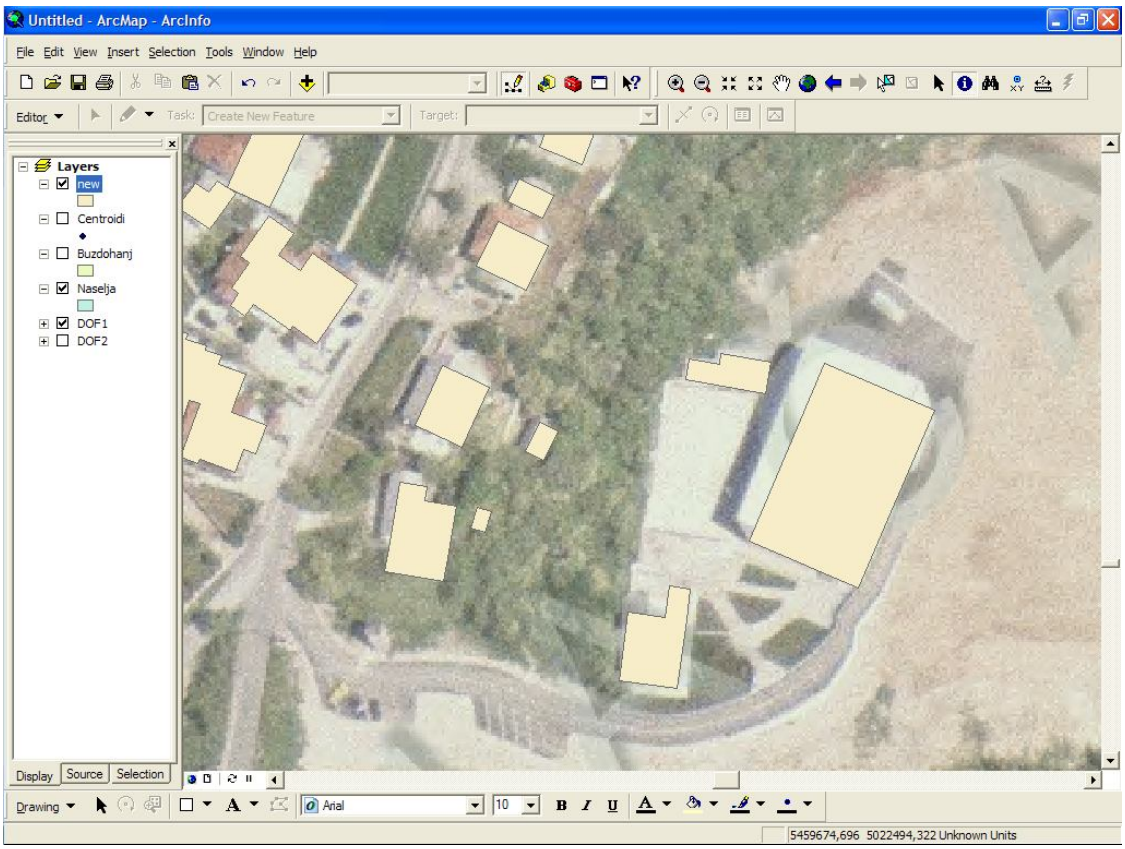
Zbog upotrebe *MultiPolygon* tipa podataka potrebno je izvršiti manje promjene u potprogramu koji zapisuje obrađene podatke u izlaznu datoteku odnosno novi shapefile. Potrebno je definirati pri kreiranju *layera* geometrijski tip podataka koji će biti zapisivani što se radi pomoću naredbe " geom_type=ogr.wkbMultiPolygon "

```
def write_shapefile(lsFeat, outfile):
    driver = ogr.GetDriverByName('ESRI Shapefile')
    if os.path.exists(outfile):
            driver.DeleteDataSource(outfile)
    ds = driver.CreateDataSource(outfile)
    layer = ds.CreateLayer(outfile, geom_type=ogr.wkbMultiPolygon)
    for i in lsFeat:
        f = ogr.Feature(feature_def=layer.GetLayerDefn())
        p = ogr.CreateGeometryFromWkb(i['geom'].wkb)
        f.SetGeometry(p)
        layer.CreateFeature(f)
        f.Destroy()
    ds.Destroy()
```

Kako je problem vizualno prikazati razliku između mnoštva poligona i jednog višestrukog poligona koji ih sadrži sve prikaz ovog rješenja vidi se u tabličnom prikazu na slici (Slika 32). Iz prikaza je vidljivo kako izlazna shapefile datoteka ima samo jedan zapis, a na slici (Slika 33) se vidi kako su prikazani svi poligoni koji postoje u izvornim podacima.




*Slika 32 Tabelarni prikaz zapisa u izlaznoj datoteci (1 poligon)*




*Slika 33 Prikaz podataka izlazne datokeke koja sadrži 1 zapis*

### *5.2. Razlika (eng. Diference)*

Nakon navedenih pojedinih operacija na prostornim podacima koje mogu imati direktnu primjenu u smanjenju troškova ili izradi mnogih analiza sljedeća operacija koju ćemo provest nad podacima pokazat će nam kako se pomoću Python skripti u jednom koraku, odnosno u jednoj skripti, može izvršiti više analiza koje bismo u ArcGIS paketu morali provoditi jednu po jednu da bismo dobili traženo rješenje.

U ovome poglavlju biti će objašnjena operacija razlike. Strogo gledano ovo je matematička operacija i njeno ponašanje identično je matematičkom oduzimanju jedino što u ovome slučaju radimo s prostornim podacima. Umjesto oduzimanja dvaju brojeva oduzimamo dva geometrijska entiteta tj. dva poligona.

Kao što je poznato iz matematike **a-b=c**, ali **b-a≠c** isto vrijedi i kod prostorne razlike ili oduzimanja. Nije svejedno koji poligon tj. objekt ćemo oduzeti od kojega. Ukoliko vršimo obradu dva poligona od kojih veći u potpunosti prekriva manji, a matematički odnos uspostavimo tako da od manjeg poligona oduzimamo veći dobiti ćemo prazan skup. U obrnutom slučaju rezultat će biti poligon s otokom, odnosno rupom unutar sebe.

Kombinirajući dvije naredbe kao što su *buffer* i *diference* moguće je kreirati nove poligone prstenastog oblika, odnosno dobiti poligone koji su isključivo *buffer* zona i ne sadržavaju u sebi i originalne objekte nad kojima je provedena *buffer* operacija. Rezultat takve obrade podataka odnosno dobivena zona ukoliko se njena širina zada sukladno urbanističkim planovima i zakonskim aktima predstavlja prostor unutar kojega se ne bi smio pojaviti niti jedan drugi objekt (građevina) jer postoji ograničenje o minimalnoj udaljenosti između objekata. Daljnjom obradom podataka moguće je izvršiti upit preklapali se dobivena zona s nekim od objekata te se na taj način automatski detektiraju potencijalno bespravno sagrađeni objekti tj. objekti sagrađeni u neskladu s dobivenim dozvolama.

Pošto smo dali uvodne napomene vezane uz operaciju oduzimanja i ukratko naveli jednu od mogućih upotreba dobivenih rezultata u prozoru ispod nalazi se kod potprograma koji *buffer* naredbom kreira poligon sa *buffer* zonom te od njega oduzima originalni objekt kako bismo dobili samo *buffer* zonu. Zbog lakšeg snalaženja u samom kodu potprograma u ovome je slučaju korišteno više varijabli.

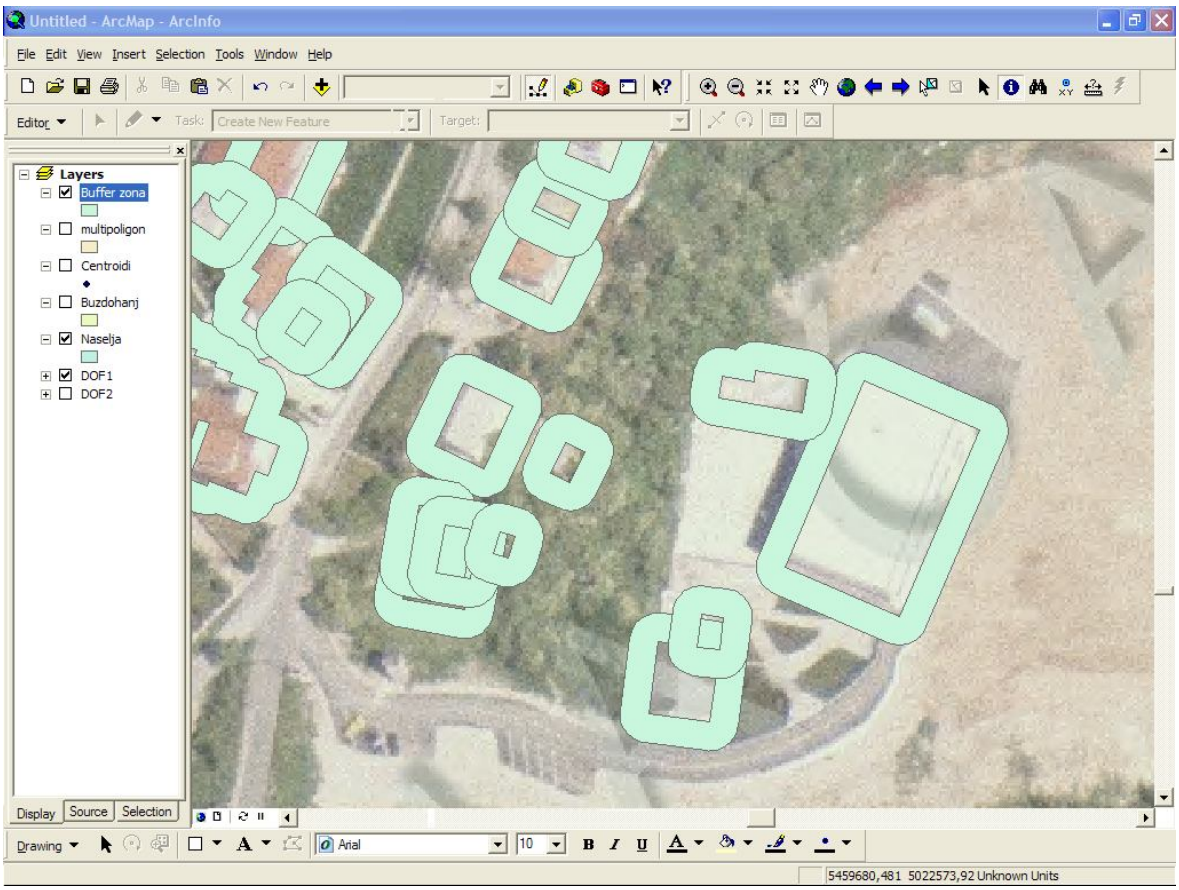
Varijabla "geom" predstavlja pročitani zapis ulazne datoteke odnosno izvorni objekt, "pol" predstavlja *buffer* naredbom obrađen izvorni objekt, a "presjek" predstavlja rezultat oduzimanja izvornog objekta od obrađenog objekta naredbom "pol.difference(geom)".

```
ge = ogr.Open(infile)
    layer = ge.GetLayer(0) #-- a simple shapefile always has 1 layer only
    ld = layer.GetLayerDefn()
    noFeat = layer.GetFeatureCount()
    lsFeat = [] #-- create an empty list of Features, in Shapely format
    lsFeat1 = []
    for i in xrange(noFeat):
        f = layer.GetFeature(i)
        geom = wkbloads(f.GetGeometryRef().ExportToWkb())
        pol = geom.buffer(5,16)
        presjek = pol.difference(geom)
        lsFeat1.append({'geom': presjek})
```

Također kako su rezultat ove analize opet poligoni, a takav je kod potprograma za stvaranje izlazne datoteke već objašnjen ranije u poglavlju 4.2.2 ovdje nećemo ponovno navoditi ista objašnjenja već samo prikazujemo kod potprograma u prozoru ispod.

```
driver = ogr.GetDriverByName('ESRI Shapefile')
    if os.path.exists(outfile):
            driver.DeleteDataSource(outfile)
    ds = driver.CreateDataSource(outfile)
    layer = ds.CreateLayer(outfile, geom_type=ogr.wkbPolygon)
    for i in lsFeat:
        f = ogr.Feature(feature_def=layer.GetLayerDefn())
        p = ogr.CreateGeometryFromWkb(i['geom'].wkb)
        f.SetGeometry(p)
        layer.CreateFeature(f)
        f.Destroy()
    ds.Destroy()
```

Na slici (Slika 34) je prikazan vizualni prikaz *buffer* zone oko objekata. Moguće je uočiti preklapanja pojedinih zona koja upućuju da postoje objekti koji se dodiruju ili su veoma blizu pa se zone oko njih međusobno djelom preklapaju. Uočljiva je razlika između rješenja dobivenog korištenjem samo *buffer* naredbe i ovoga rješenja dobivenog korištenjem dviju naredbi. *Buffer* naredbom dobiva se poligon koji sadržava izvorni poligon i proširuje ga za zonu zadane vrijednosti dok se na ovaj način dobiva samo zona oko izvornog objekta.




*Slika 34 Prikaz prstena buffer zone dobivenog upotrebom dviju naredbi*

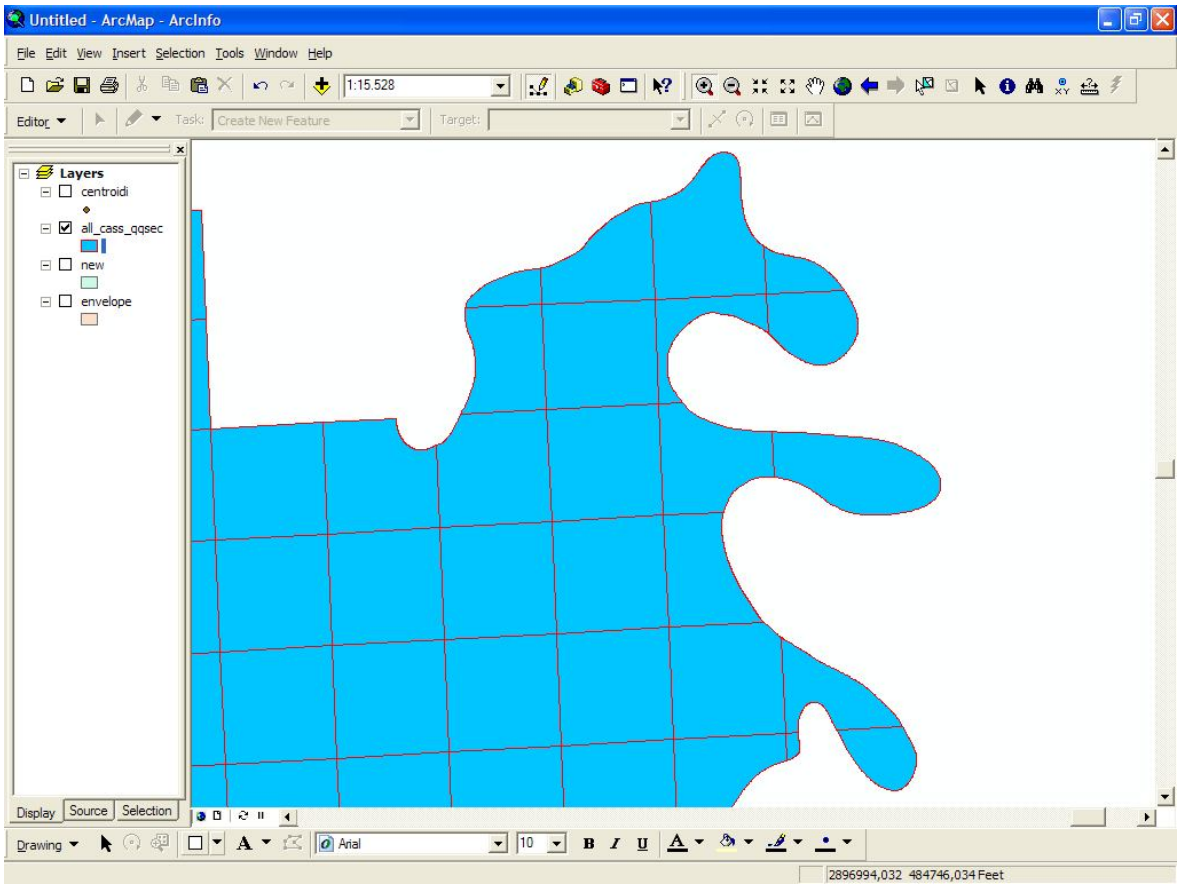
# 6. Usporedba brzine Python i ArcGIS rješenja

U narednom poglavlju sistematski će biti prikazana usporedba utrošenog vremena za provedbu dosad objašnjenih naredbi. Prvo će biti opisani korišteni podaci jer se zbog lakše detekcije razlike u utrošku vremena koristio veći set podataka. Potom će biti prikazana i objašnjena dobivena rješenja obrade podataka Python skriptama čiji je kod u ranijim poglavljima detaljno opisan. Nakon navedenih Python rješenja biti će prezentirana i ArcGIS rješenja te pokretanje ArcGIS alata pomoću Python skripti koje koriste modul arcgisscripting. Kao zaključak ovog poglavlja u tabličnom obliku će se prezentirati dobivene vrijednosti te će biti ponuđeno moguće objašnjenje rezultata.

## *6.1. Korišteni podaci*

Kako bi se što lakše uočile razlike u vremenu potrebnom za izvršavanje pojedinih metoda obrade podataka bilo je potrebno prikupiti podatke koji sadrže mnogo veći broj objekata. Jasno je naravno da male vremenske razlike koje se dešavaju na svakom objektu s porastom broja objekata kumulativnim učinkom čine velike razlike ukoliko se koristi dovoljno veliki set podataka.

U ovome slučaju korišteni su besplatni, javno dostupni podaci koji su preuzeti sa strane Internet domene, radi se o službenim stranicama vlade Sjeverne Dakote na kojima je moguće preuzeti mnoštvo različitih prostornih podataka www.casscountynd.gov/county/depts/GIS/download/Pages/shapefiles.aspx.
Preuzeta datoteka u ovome slučaju sadrži podatke o podjeli države na zone. Tom podjelom nastao je shapefile koji sadrži 28369 poligona čiji prikaz u ArcMap aplikaciji možete vidjeti na slici (Slika 35) na sljedećoj stranici. Korisno je za istaknuti kako sadržani podaci nisu sastavljeni od poligona čije se sve strane sijeku pod pravim kutovima već postoje i poligoni koji se očigledno nalaze uz obalu jezera ili mora te njihovu granicu čine krivulje. Kako krivulje nisu obične linije već se sastoje od mnogo malih segmenata moguće je da takav sastav podataka olakša uočavanje razlika između dvaju rješenja.




*Slika 35 Prikaz podataka korištenih pri izradi analiza*

## *6.2. Python rješenja*

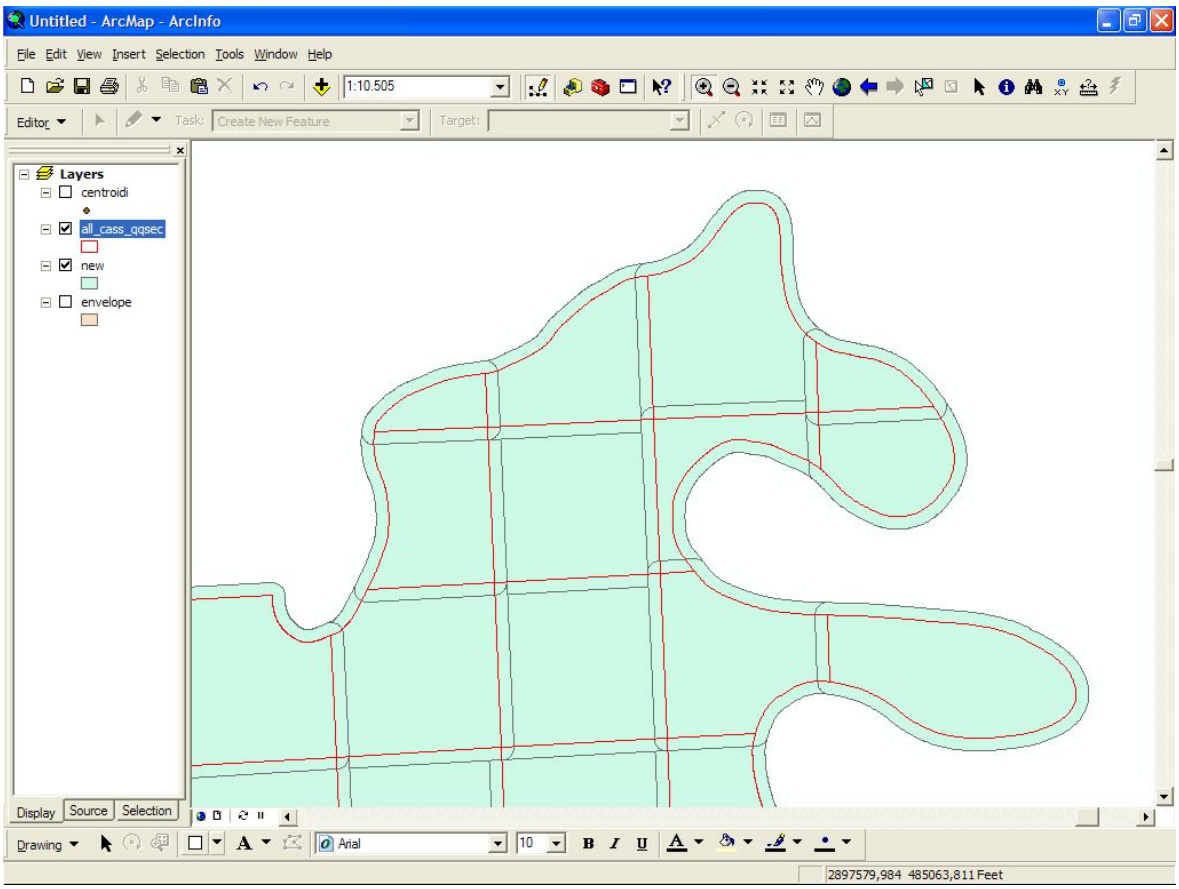
U narednim poglavljima sistematski će redom biti navedene metode koje su provedene nad operacijama. Prikazat ćemo dobivena rješenja i navesti vrijeme potrebno za provedbu obrade.

### 6.2.1. Buffer

Prva od korištenih metoda za usporedbu je metoda izrade *buffer* zone koja je već ranije opisana u poglavlju 3.2.2, a čiji je kod detaljno objašnjen u poglavlju 4.2.2 pa se smatra kako nema potrebe ponovno navodit iste činjenice.

Samo ćemo se podsjetiti što je cilj provedbe *buffer* obrade. Nakon provedene obrade nad podacima trebali bismo dobiti poligon kojem je opisana zona proizvoljne širine koju definira korisnik i jer su korištene uobičajene postavke, *buffer* zona imati će zaobljene uglove. Programski kod skripte može se vidjeti u prozoru na idućoj stranici.

```
def process_shapefile(infile):
    print "\nProcessing shapefile", infile
    ge = ogr.Open(infile)
    layer = ge.GetLayer(0)
    ld = layer.GetLayerDefn()
    noFeat = layer.GetFeatureCount()
    lsFeat = []
    for i in xrange(noFeat):
        f = layer.GetFeature(i)
        geom = wkbloads(f.GetGeometryRef().ExportToWkb())
        geom = geom.buffer(100,16)
        lsFeat.append({'geom': geom})
    return lsFeat
```




*Slika 36 Rezultat obrade podataka buffer naredbom*

Na slici iznad (Slika 36) prikazan je dobiveni rezultat. Crvene linije koje se vide unutar poligona su rubne linije (granice) poligona koji su bili ulazna vrijednost.

Može se primijetiti kako su rubovi *buffer* zone zaobljeni te možemo reći kako je postignut cilj obrade podataka. Vrijeme potrebno Python skripti koja koristi Shapely module za izradu buffer zone oko navedenih objekata iznosilo je 16,2 sekunde

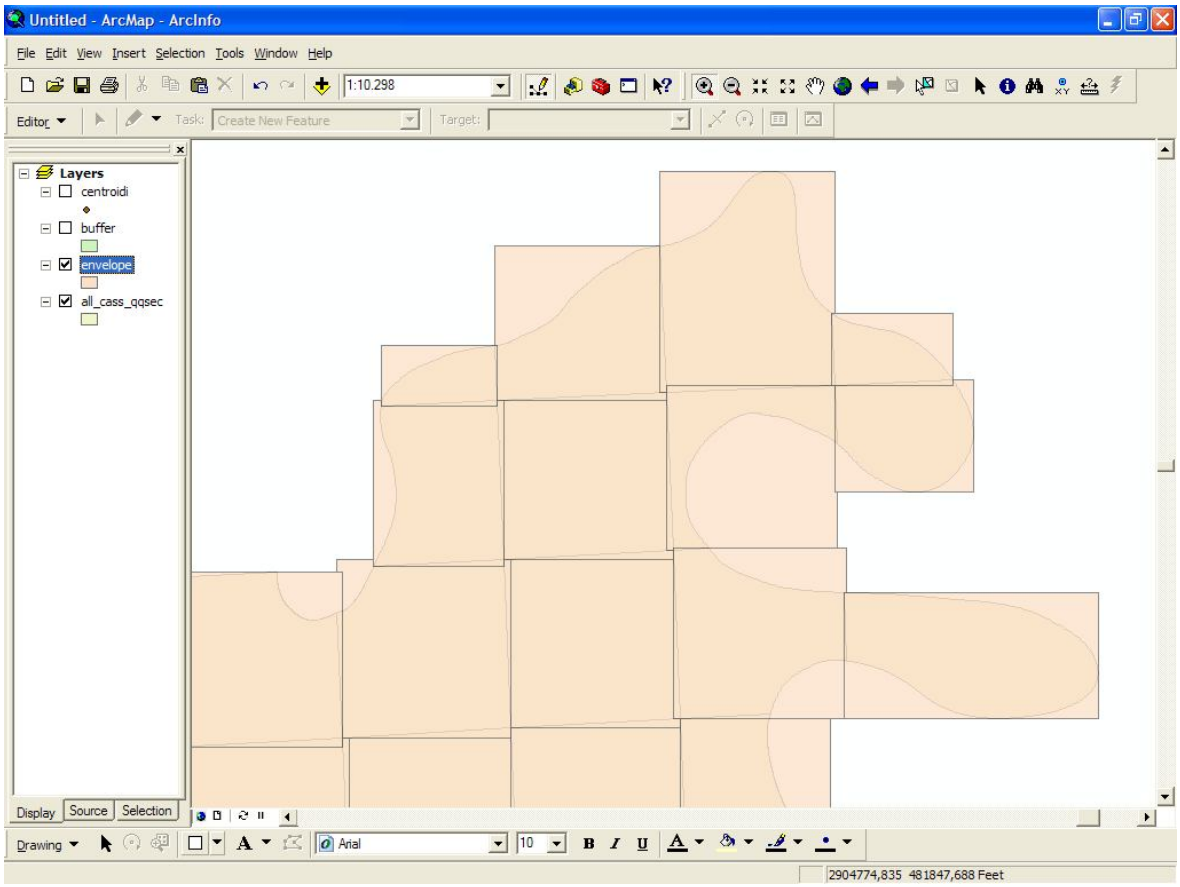
### 6.2.2. Omotnica

Druga od korištenih metoda za usporedbu je metoda izrade omotnice koja je već ranije opisana u poglavlju 3.2.2, a čiji je kod detaljno objašnjen u poglavlju 4.6 pa se smatra kako nema potrebe ponovno navodit iste činjenice.

Podsjetiti ćemo se samo kako se provedbom kreiranja omotnice oko poligona dobiva četverokut sa četiri prava kuta a stranice su paralelne s X ili Y osi pravokutnog kartezijevog koordinatnog sustava. Programski kod skripte može se vidjeti u prozoru ispod.

```
def process_shapefile(infile):
    print "\nProcessing shapefile", infile
    ge = ogr.Open(infile)
    layer = ge.GetLayer(0)
    ld = layer.GetLayerDefn()
    noFeat = layer.GetFeatureCount()
    lsFeat = []
    for i in xrange(noFeat):
        f = layer.GetFeature(i)
        geom = wkbloads(f.GetGeometryRef().ExportToWkb())
        geom = geom.envelope
        lsFeat.append({'geom': geom})
    return lsFeat
```

Na slici na sljedećoj stranici (Slika 37) prikazan je dobiveni rezultat. U pozadini, ispod rozih pravokutnika naziru se obrisi ulaznih poligona. Vrijeme potrebno Python skripti koja koristi Shapely module za izradu omotnice oko navedenih objekata iznosilo je 7,8 sekunde.




*Slika 37 Rezultat obrade podataka envelop naredbom*

### 6.2.3. Centroid

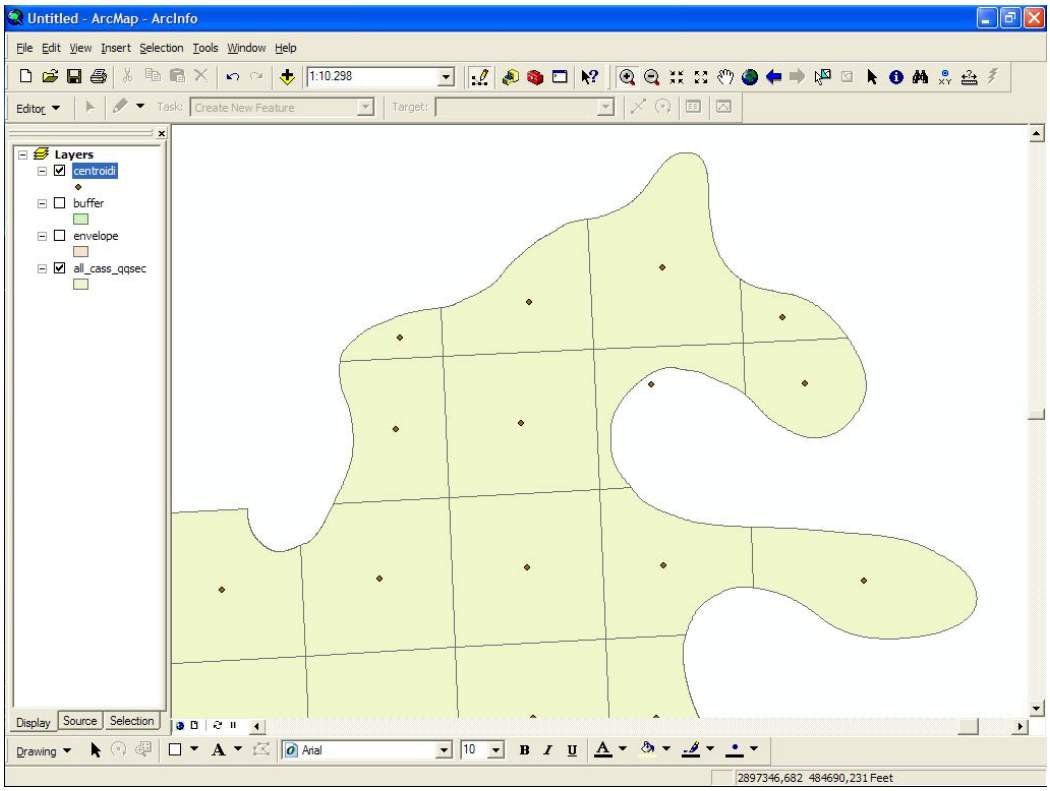
Treća od korištenih metoda za usporedbu je metoda izrade *centroida* koja je već ranije opisana u poglavlju 3.2.2, a čiji je kod detaljno objašnjen u poglavlju 4.5 pa se smatra kako nema potrebe ponovno navodit iste činjenice.

Dakle, samo da ponovimo, cilj koji postižemo provođenjem naredbe centroid je aproksimacija poligona točkom. U onim slučajevima kada nam nije važan geometrijski oblik poligona već nam samo treba prostorna komponenta informacije možemo poligon zamijeniti točkom. Naredbom centroid, poligon zamjenjujemo točkom koja predstavlja geometrijski centar poligona. Programski kod skripte možemo vidjeti u prozoru koji se nalazi na sljedećoj stranici.

Kao što smo već i ranije navodili, u potprogramu zaduženom za stvaranje nove izlazne shapefile datoteke potrebno je odrediti da su izlazni podaci točke a ne poligoni kao u prijašnja dva primjera što je objašnjeno ranije.

```
def process_shapefile(infile):
    print "\nProcessing shapefile", infile
    ge = ogr.Open(infile)
    layer = ge.GetLayer(0)
    ld = layer.GetLayerDefn()
    noFeat = layer.GetFeatureCount()
    lsFeat = []
    for i in xrange(noFeat):
        f = layer.GetFeature(i)
        geom = wkbloads(f.GetGeometryRef().ExportToWkb())
        geom = geom.centroid
        lsFeat.append({'geom': geom})
    return lsFeat
```

Na slici ispod (Slika 38) prikazan je dobiveni rezultat. U prvome planu, iznad rozih ulaznih poligona vide se točkasti objekti. Vrijeme potrebno Python skripti koja koristi Shapely module za izradu izlazne shapefile datoteke u kojoj su poligoni zamijenjeni centroidima objekata iznosilo je 7,2 sekunde.



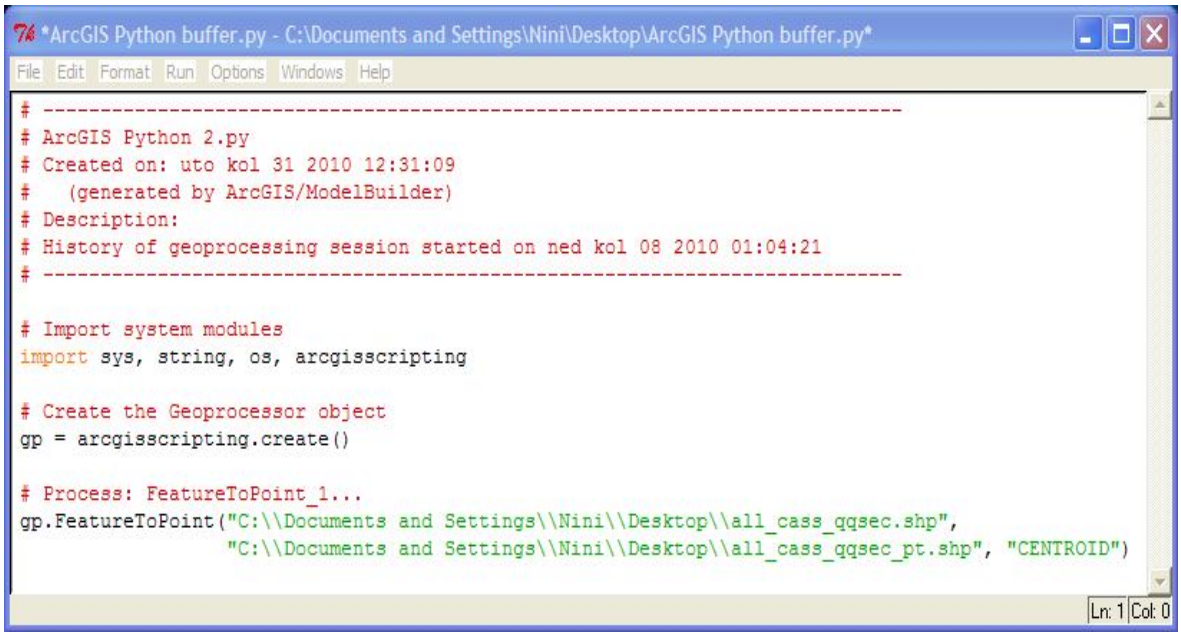

*Slika 38 Rezultat obrade podataka centroid naredbom*

## 6.3. ArcGis rješenja

U narednim poglavljima sistematski će redom biti navedene metode koje su provedene nad operacijama. Prikazat ćemo način pozivanja alata za provedbu obrada, dobivena rješenja i navesti vrijeme potrebno za provedbu obrade. Sve navedeno potkrijepit ćemo slikama.

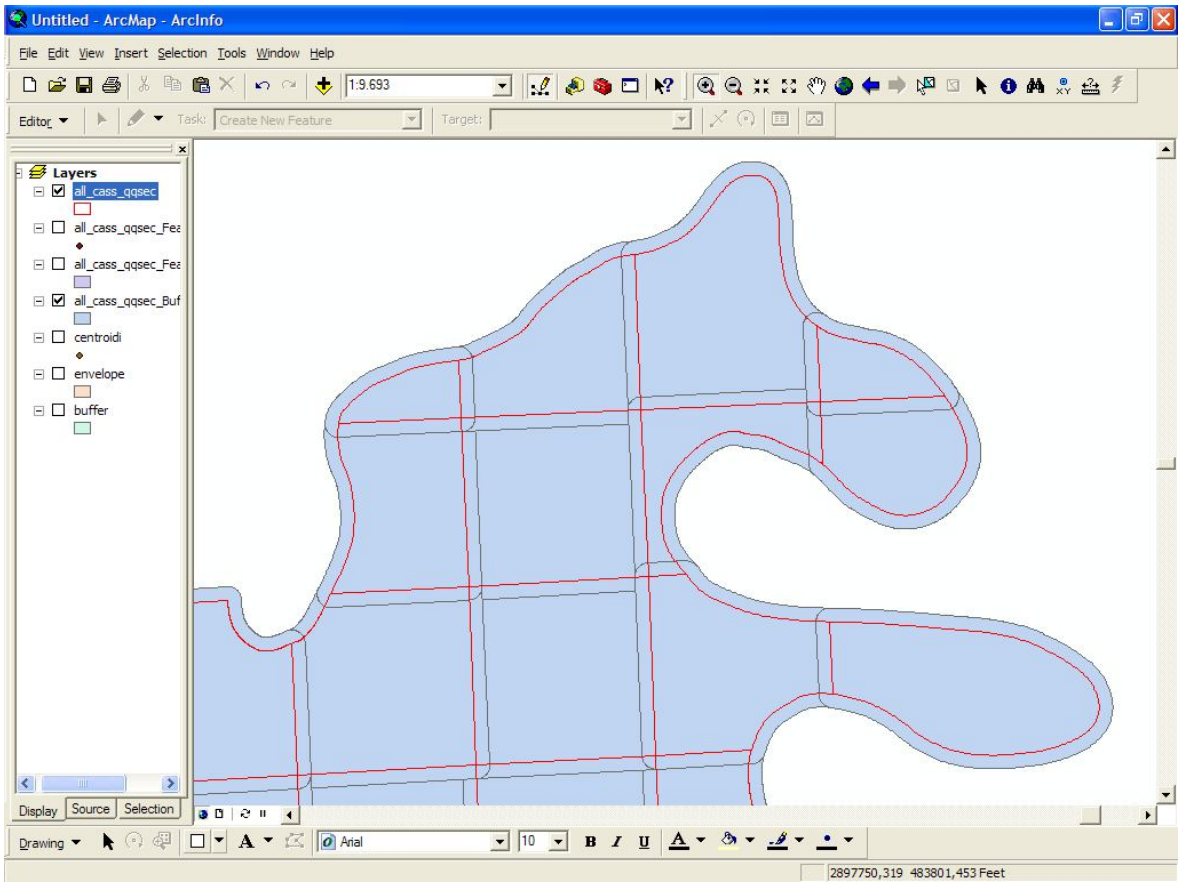
### 6.3.1. Buffer

Sukladno mišljenju kako je potrebno što je više moguće automatizirati proces u ovom poglavlju naredba *buffer* nije pozvana manualno već pomoću Python skripte koja koristi arcgisscripting modul čiji kod vidimo na slici ispod (Slika 39).



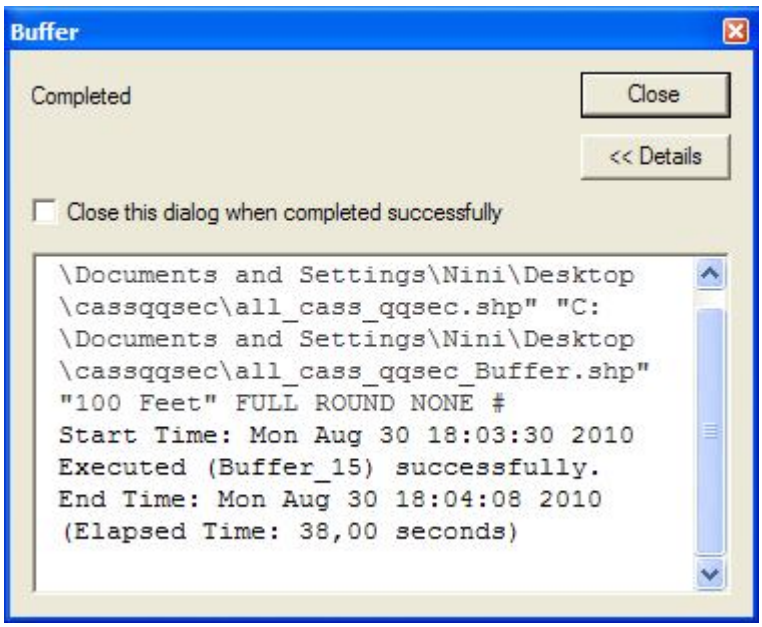

*Slika 39 Python skripta sa arcgisscripting modulom i naredbom buffer*

Na slici na sljedećoj stranici (Slika 40) prikazan je dobiveni rezultat. Crvene linije koje se vide unutar poligona su rubne linije (granice) poligona koji su bili ulazna vrijednost. Može se primijetiti kako su rubovi *buffer* zone zaobljeni te možemo reći kako je postignut cilj obrade podataka. Rezultat je istovjetan rezultatu dobivenom s Python skriptom koja koristi Shapely module.




*Slika 40 Rezultat obrade podataka buffer naredbom u ArcGISu*

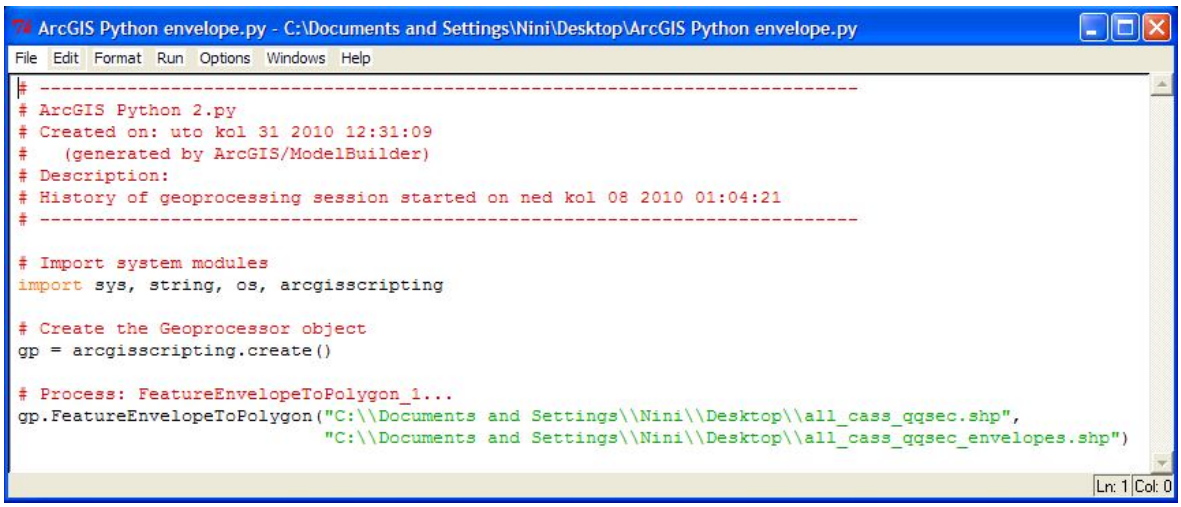
Vrijeme potrebno Python skripti koja koristi arcgisscripting module za izradu buffer zone oko navedenih objekata iznosilo je 38 sekundi što se vidi na slici (Slika 41).




*Slika 41 Prikaz utrošenog vremena za provedbu buffer naredbe u ArcGISu*

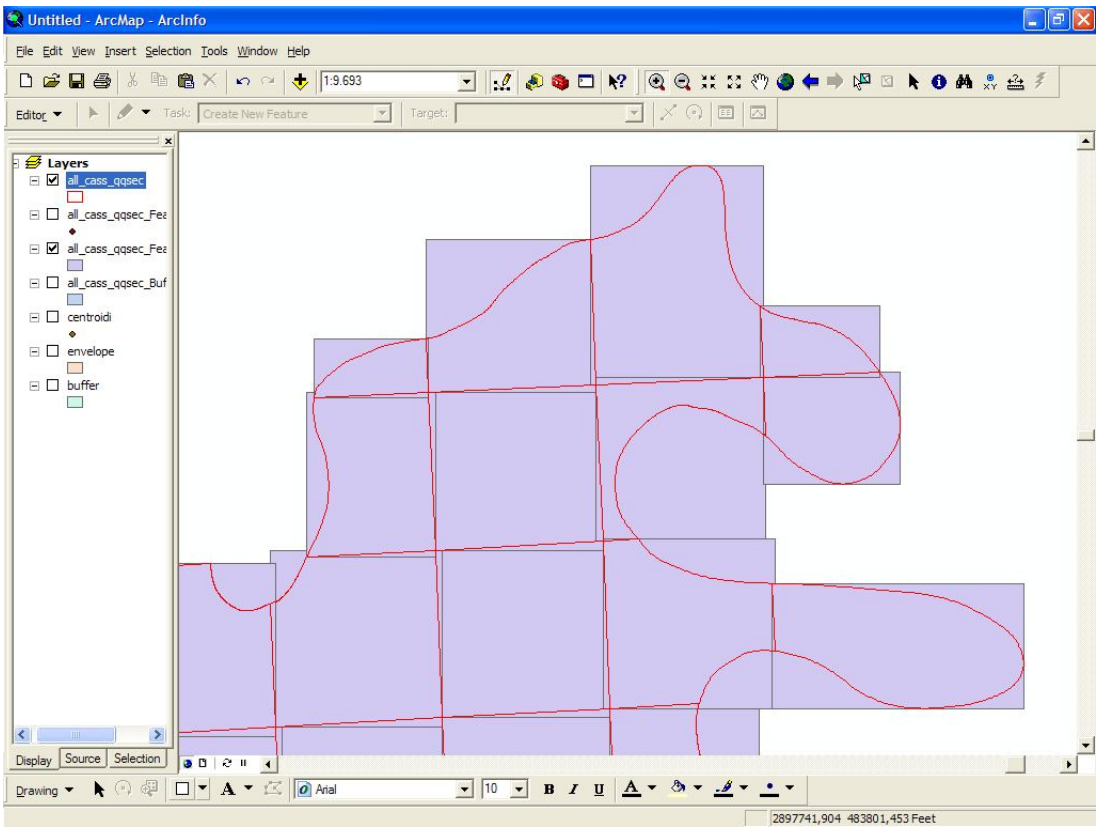
### 6.3.2. Omotnica

Kako bismo osigurali jednake uvjete testiranja za sve procedure i metodu omotnice pozvali smo pomoću Python skripte čiji se kod vidi na slici (Slika 42).



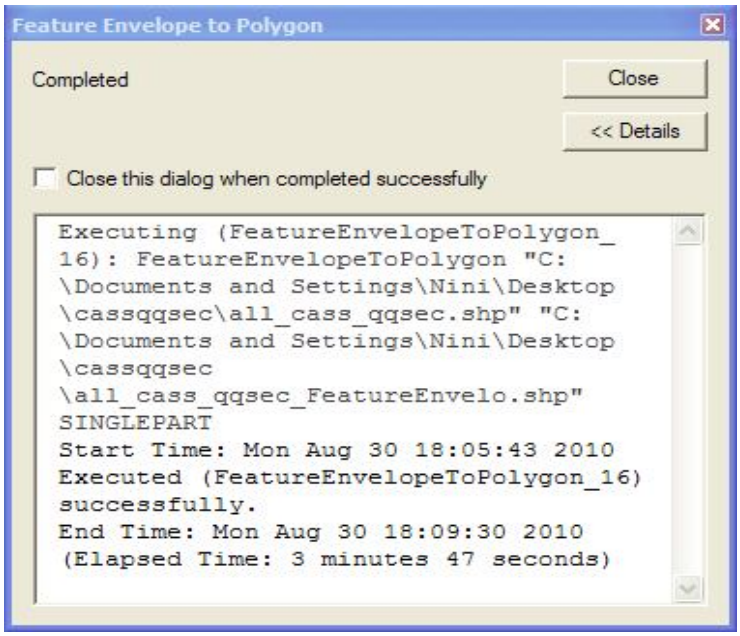

*Slika 42 Python skripta sa arcgisscripting modulom i naredbom envelope*

Na slici ispod (Slika 43) prikazan je dobiveni rezultat. U pozadini, ispod sivih pravokutnika naziru se obrisi ulaznih poligona (crvene linije).




*Slika 43 Rezultat obrade podataka envelope naredbom u ArcGISu*

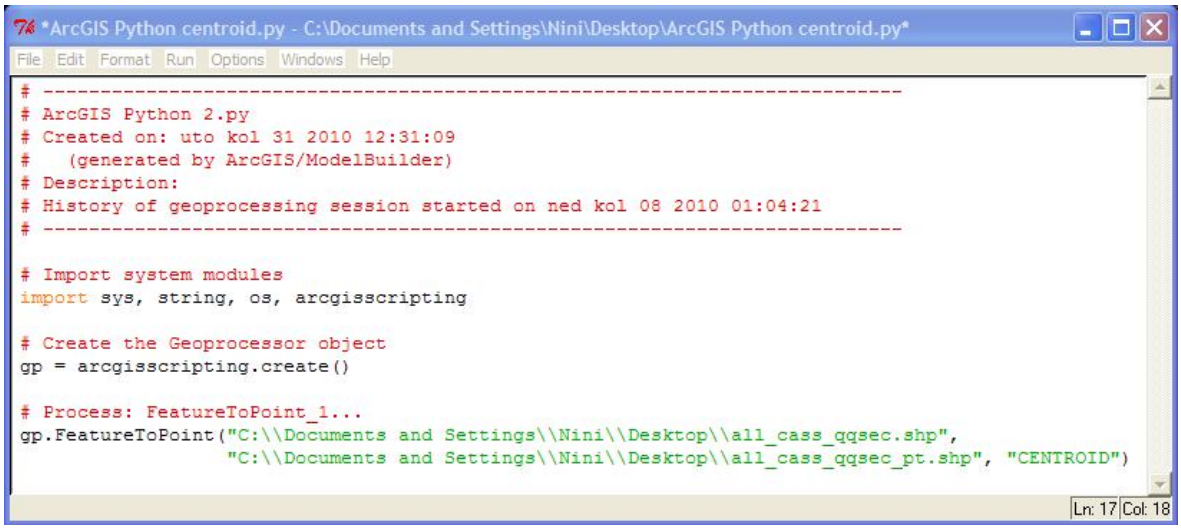
Vrijeme potrebno Python skripti koja koristi arcgisscripting module za izradu omotnice oko navedenih objekata iznosilo je 3 minute i 47 sekunde što se vidi na slici ispod (Slika 44).




*Slika 44 Prikaz utrošenog vremena za provedbu envelope naredbe u ArcGISu*

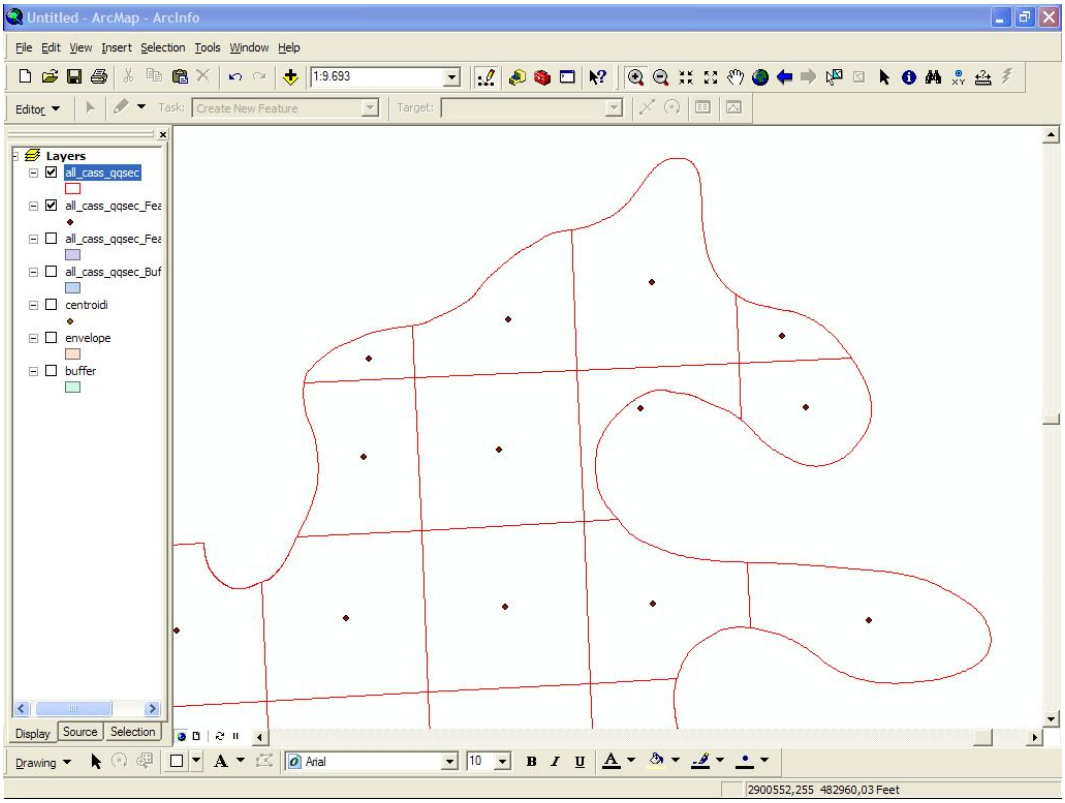
### 6.3.3. Centroid

Kako bismo osigurali jednake uvjete testiranja za sve procedure i metodu centroida pozvali smo pomoću Python skripte čiji se kod vidi na slici (Slika 45).



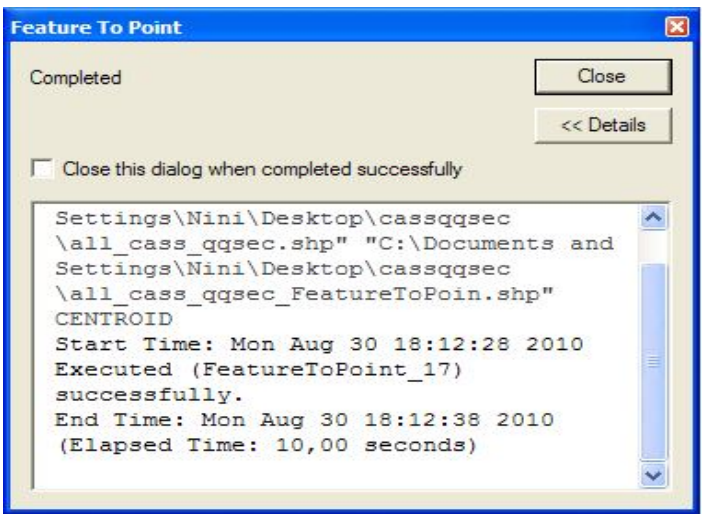

*Slika 45 Python skripta sa arcgisscripting modulom i naredbom centroid*

Na slici ispod (Slika 46) prikazan je dobiveni rezultat. Prikazane su međne linije ulaznih poligona i rezultata obrade, točkasti objekti u sredini.




*Slika 46 Rezultat obrade podataka centroid naredbom u ArcGISu*

Vrijeme potrebno Python skripti koja koristi arcgisscripting module za izradu izlazne shapefile datoteke u kojoj su poligoni zamijenjeni centroidima objekata iznosilo je 10 sekundi što se vidi na slici ispod (Slika 47).




*Slika 47 Prikaz utrošenog vremena za provedbu centroid naredbe u ArcGISu*

### *6.4. Vremenska usporedba*

Nakon prikazanih rezultata obrade podataka trima različitim metodama čije je pozivanje izvedeno na dva različita načina moguće je postaviti određene zaključke u vezi s brzinom izvršavanja pojedinih metoda ovisno o načinu njihova pozivanja.

Iz prije prikazanih obrada podataka može se zaključiti kako se Shapely metode izvršavaju brže od ArcGIS *toolbox* metoda, neovisno pozivaju li se ArcGIS metode manualno ili automatski uz pomoć Python skripti koje koriste *arcgisscripting* modul.

Također potrebno je istaknuti kako nisu uočene razlike u brzini izvršavanja metoda obrade podataka koje bi ovisile o indeksnim datotekama kao što su ranije spomenute .shx, .sbn, .sbx …

U tablici ispod prikazani su podaci o vremenu potrebnom za izvršavanje svake od navedenih obrada. Vidljivo je da su Python rješenja u sva tri slučaja bila brža, no zanimljivošću se ističe metoda omotnice. Iako su i u ostalim metodama *arcgisscripting* rješenja sporija od 50% pa čak i do 100% kod metode omotnice dolazi do puno većeg skoka.

*Tabela 4 Tablični prikaz vremena potrebnog za izvršavanje pojedine operacije korištenjem Python\Shapely skripte ili Python\arcgisscripting aplikacije*

| Metoda\ Korišteni softver | Python\Shapely | ArcGIS |
|---|---|---|
| *Buffer* | 16,2 sec | 38 sec |
| *Envelope* | 7,8 sec | 3 min 47 sec |
| *Centroid* | 7,2 sec | 10 sec |

Kako se vidi kod metode omotnice *arcgisscripting* rješenje sporije je za oko 30 puta iako su podaci identični onima upotrebljavanima pri testiranju i preostalih metoda. Vjerojatni razlog za ovako izraženu razliku u vremenu potrebnom za izvršenje analize leži u činjenici kako međne linije ulaznih poligona u određenome broju čine krivulje koje su specifičan oblik linijske geometrije.

# 7. Prilog (digitalni oblik)

Sukladno temi i opisu rada primjereno je diplomski rad iz područja geoinformatike opremiti i digitalnim medijem čiji će opis, sadržaj i eventualne upute za korištenje biti navedeni u sljedećim odlomcima.

## *7.1. Sadržaj priloženog medija (CD-a, DVD-a)*

Na priloženom mediju pohranjeni su podaci korišteni pri izradi diplomskog rada i svi postignuti rezultati. Logički su organizirani prema smislu (Tabela 5).

*Tabela 5. Sadržaj priloženog medija*

| RB. | Mapa/ Datoteka | Sadržaj |
|---|---|---|
| 1. | Diplomski.docx | Pisani dio rada |
| 2. | Python rješenje | Skripta i shapefile s obrađenim podacima |
| 3. | ArcGIS rješenje | Python skripta za automatizaciju i shapefile s obrađenim podacima |
| 4. | Dodatne skripte | Skripte za obradu shapefile-ova |
| 5. | Podaci | Shapefile "Buzdohanj.shp", shapefile "Naselja.shp" te DOF 5 i DOF2 kao podloga |
| 6. | Usporedba vremena | Korišteni podaci, Python skripte sa Shapely modulima i arcgisscripting modulom |
| 7. | Instalacijske datoteke | Datoteke za instalaciju Pythona i svih navedenih biblioteka (Shapely, OGR) |
| 8. | Literatura | Korištena literatura iz popisa literature u digitalnom formatu (pdf) |

# 8. Zaključak

Provedenim istraživanjem o temi diplomskog rada došlo se do saznanja kako je sama tema diplomskog rada veoma opširna i njena je primjena i razvoj u današnje vrijeme u svijetu veoma aktivan. Mnoge znanstvene ustanove povezane s obradom prostornih podataka, kao i mnoge komercijalne kompanije koriste skripte i programske jezike koji podržavaju izradu skripti kako bi automatizirale svoje radne procese. Također mnogo komercijalnih kompanija i tvrtki, kao i znanstvenih ustanova u cilju rezanja svojih proizvodnih troškova okreću se besplatnim ili vlastitim softverskim rješenjima baziranim na besplatnim platformama kao što je programski jezik Python. Na bazičnom primjeru izrade buffer zone oko skupine poligona prikazano je komercijalno rješenje zatvorenog koda koji nije moguće uređivati i slobodno softversko rješenje koje je moguće mijenjati ukoliko naredni zadatak iziskuje određene adaptacije trenutnog rješenja. Ideja je bila i da se izradom takva dva rješenja od kojih je jedno korporativno, komercijalno i sklop velikog i kvalitetnog paketa alata kao što je ArcGIS, a drugo je rješenje mali softverski uradak studenta, budućeg inženjera geodezije pokaže razlika u programerskom pristupu rješavanja identičnog problema. Iako nije moguće na legalan način vidjeti programski kod komercijalnog rješenja usporedbom brzine izvođenja sa slobodnim softverskim rješenjem koje je od pet pa do čak dvadeset puta brže uviđa se kako komercijalna rješenja pokušavaju unaprijed predvidjeti što je moguće veći broj potencijalnih zadataka dok slobodna rješenja nude program za pojedinu situaciju. Izradom više malih programa za pojedini problem postiže se brži i optimalan rad sustava, a sam time je i zahtjevanost za snagom računala smanjena. Sukladno svemu navedenom nameće se zaključak da komercijalna rješenja u svome programskom kodu imaju veliku količinu takozvanog smeća odnosno opcija koje u većini slučaja nisu potrebne, a usporavaju rad računala, opterećuju sustav i zahtijevaju više vremena za svoju obradu. Nakon navedenog zaključka stav je autora kako u danom trenutku vremena, gledajući ekonomsku, političku i društvenu situaciju potrebno se je okrenuti modernim tehnologijama, njihovom razvoju i unapređenju. Postepeno treba uvesti vlastite nove tehnologije i na taj način odgovorit trenutnoj gospodarsko političkoj situaciji na tržištu prikupljanja, obrade i prodaje prostornih podataka.

## Literatura:

Catherine Jones, Jill McCoj: ArcGIS 9, Geoprocessing in ArcGIS Tutorial, ESRI, Redlands

Jill McCoj: ArcGIS 9, Geoprocessing in ArcGIS, ESRI, Redlands

Željko Panian: "Informatički enciklopedijski rječnik A-L", Europapress holding d.o.o., Zagreb 2005.

Željko Panian: "Informatički enciklopedijski rječnik M-Z", Europapress holding d.o.o., Zagreb 2005.

Guido van Rossum, Fred L. Drake : The Python Library Reference Release 2.6.4 Python Software Fondation, 04. 01. 2010.

Guido van Rossum, Fred L. Drake : The Python Tutorial Release 2.6.4, Python Software Fondation, 04. 01. 2010.

## Popis slika:

## Popis URL-ova:

URL 1: Python, http://www.Python.org/download (15. 07. 2010.)

URL 2: GDAL, http://www.gdal.org/ (18. 08. 2010.)

URL 3: GEOS, http://trac.osgeo.org/geos/ (18. 08. 2010.)

URL 4: OGR, http://www.gdal.org/ogr/ (14. 08. 2010.)

URL 5: JTS, http://www.vividsolutions.com/jts/jtshome.htm (14. 08. 2010.)

URL 6: Shapefile, http://en.wikipedia.org/wiki/Shapefile (19. 06. 2010.)

URL 7: Environment variable,
http://en.wikipedia.org/wiki/Environment_variable (15. 08. 2010.)

URL 8: WKT, http://en.wikipedia.org/wiki/Well-known_text (22. 07. 2010.)

URL 9: Sean Gillies, The Shapely 1.2 User Manual (Preview) 19. 08. 2010.
http://gispython.org/shapely/docs/1.2/manual.html (23. 08. 2010.)

URL 10: Sean Gillies, The Shapely 1.0 Manual (20. 05. 2008.)
http://gispython.org/shapely/docs/1.0/manual.html (24. 06. 2010.)