SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 2380

VIZUALIZACIJA GEOPROSTORNIH LOKACIJA TEMELJENA NA STVARNIM PODACIMA

Adrian Komadina

Zagreb, lipanj 2021.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 2380

VIZUALIZACIJA GEOPROSTORNIH LOKACIJA TEMELJENA NA STVARNIM PODACIMA

Adrian Komadina

Zagreb, lipanj 2021.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 12. ožujka 2021.

DIPLOMSKI ZADATAK br. 2380

Pristupnik:	**Adrian Komadina (0036499396)**
Studij:	Računarstvo
Profil:	Računarska znanost
Mentor:	prof. dr. sc. Željka Mihajlović

Zadatak: **Vizualizacija geoprostornih lokacija temeljena na stvarnim podacima**

Opis zadatka:

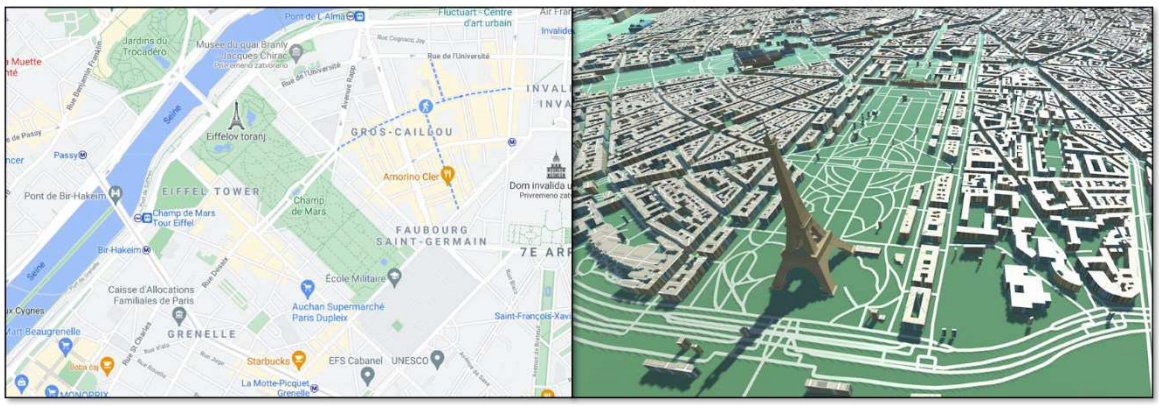
Proučiti izvore javno dostupnih kartografskih podataka te pripadna programska sučelja koja omogućuju korištenje ovih podataka. Razraditi mogućnosti korištenja javno dostupnih izvora za ostvarivanje 3D vizualizacije zadanog dijela geoprostora. Posebice obratiti pažnju na izvore podataka kao što su OSM (engl. Open Street Map) i GIS Zrinjevac. Ostvariti vizualizaciju zadanog geoprostornog područja. Načiniti testiranje na nizu primjera. Analizirati i ocijeniti ostvarene rezultate. Diskutirati upotrebljivost ostvarenih rezultata kao i moguća proširenja. Izraditi odgovarajući programski proizvod. Koristiti programski alat Unity. Rezultate rada načiniti dostupne putem Interneta. Radu priložiti algoritme, izvorne kodove i rezultate uz potrebna objašnjenja i dokumentaciju. Citirati korištenu literaturu i navesti dobivenu pomoć.

Rok za predaju rada: 28. lipnja 2021.

Sadržaj

Uvod

Kako u svim područjima tako i u kartografiji današnje tehnologije omogućavaju nam dohvat podataka iz brojnih izvora s preciznošću većom nego ikad. Takve je podatke najlakše obraditi i prikazati u 2D prostoru, dok je prikaz u 3D prostoru već nešto zahtjevniji zadatak jer su brojni atributi nedostatni ili ih je teško vizualizirati. Prikaz u dvije dimenzije je ponekad praktičan ali nam ipak ne daje cjelokupnu sliku promatranog područja koja je nekad potrebna u različite svrhe. Na slici 1.1. vidljiva je usporedba prikaza karte za isto područje u dvije i u tri dimenzije. Kao primjeri aplikacija za prikaz geografskog prostora u tri dimenzije možemo izdvojiti Maps SDK za Unity od tvrtke Google i istoimeni alat od tvrtke MapBox koji koriste svoje zasebne skupove podataka za prikaz karte. U ovom ćemo se radu fokusirati na one javno dostupne izvore kartografskih podataka kao što su OpenStreetMap i GIS Zrinjevac, te ćemo pomoću dostupnih podataka stvoriti što vjerniji prikaz nekog zadanog geografskog područja u tri dimenzije. Na početku ćemo se upoznati s ključnim tehnologijama potrebnim za stvaranje ovakvog prikaza, zatim ćemo pogledati u kojoj formi dolaze i koje sve atribute posjeduju podaci iz izvora koje smo odabrali u ovome radu. Nakon toga ćemo razvrstati podatke u određene skupine, te ćemo za svaku od njih opisati kako smo reprezentirali pojedini podatak na našoj karti. Na kraju ćemo razmotriti koliki utjecaj imaju različite skupine podataka i veličina skupa podataka na performanse naše aplikacije.



Slika 1.1. Usporedba 2D (lijevo) i 3D (desno) prikaza karte

1. Korištene tehnologije

1.1. Pogon za izradu igara Unity

Kako bismo ostvarili prikaz naše karte u virtualnom okruženju najvažniji alat kojim ćemo se služiti bit će Unity3D pogon za izradu igara (engl. Game engine). Osim samog prikaza 3D modela na karti, Unity nam još pruža mogućnosti uređivanja cijele scene i njezine kamere u kojoj se nalazi prikaz karte, te korisničkog sučelja preko kojega unosimo početne postavke za rad aplikacije. Unity nam također pruža mogućnost kreiranja skripti preko kojih možemo kontrolirati tijek rada aplikacije. Unity3D pogon za izradu igara je softversko-razvojno okruženje koje posjeduje brojne funkcionalnosti koje se koriste za izradu igara, a neke od najvažnijih su: 2D i 3D grafički pogon, skriptiranje, animacije, otkrivanje sudara, upravljanje zvukom, itd. Unity je prvi put predstavljen javnosti 2005. godine isključivo za Mac operacijski sustav, no danas se i s razlogom naziva višeplatformski pogon jer je podržan na različitim platformama kao što su Windows, Android, IOS, Linux i mnogo druge. Unity alat je besplatan za uporabu i kao takav je jedan od najpopularnijih takvih alata za izradu aplikacija s uporabom računalne grafike. Također postoje brojne dodatne biblioteke i dodaci, izrađeni od brojnih korisnika, koji omogućavaju njegovu lakšu uporabu i izradu projekata. Danas se Unity najčešće koristi za izradu igara bilo u dvije bilo u tri dimenzije, te interaktivnih simulacija i drugih iskustava [2].

Osim statičkog dijela aplikacije o kojem se brine Unity-ev grafički pogon razlikujemo i onaj dinamički dio aplikacije kojim omogućavamo interakcije među objektima, upravljanje tijekom izvođenja i sveukupnu dinamiku aplikacije. Taj je dio aplikacije u ovome alatu omogućen preko stvaranja skripti odnosno skriptiranja. Pri pisanju skripti, Unity kao programsko aplikacijsko sučelje koristi programski jezik C# kojim ćemo se koristiti u ovome radu, te će pripadni isječci izvornog koda biti prikazani upravo u tom programskom jeziku. Kao razvojno okruženje za pisanje i uređivanje C# skripti korišten je Visual Studio 2019, dok su isječci skripti prikazanih u radu preuzeti iz uređivača koda Visual Studio Code radi bolje preglednosti koda.

Za potrebe ovoga rada korištena je stabilna verzija Unity-a 2019.3.13 koja je postala dostupna 6. svibnja 2020. godine.

1.2. Blender

Kako bismo ostvarili što realističniji prikaz 3D karte u aplikaciji moramo se koristiti što većim brojem 3D modela objekata koji će nam kasnije predstavljati neki podatak iz stvarnog svijeta. S time na umu nerijetko će biti potrebno modele kreirati ili ih pak prilagoditi prema našoj potrebi, u tu svrhu korišten je programski alat Blender. Blender je besplatan i javno dostupan programski alat korišten kod trodimenzionalnih scene računalne grafike za izradu 3D modela, animiranih filmova, pokretne grafike, vizualnih efekata, interaktivnih aplikacija, računalnih igara, itd. Upravo sljedeći skup funkcionalnosti omogućuje tako široku navedenu primjenu ovoga alata: 3D modeliranje, teksturiranje, simulaciju čestica, skulpturiranje, animiranje, izrada prikaza (engl. rendering), simulacija fluida, komponiranje, uređivanje rasterske grafike i mnoge druge. Ovaj programski alat javno je i besplatno objavljen 1998. godine, te je do danas podržan na brojnim platformama [3].

Za potrebe ovog rada bit će korištena verzija 2.92.0 prethodno opisanog programskog alata.

2. Izvori podataka

U ovom ćemo poglavlju razmotriti koje smo izvore podataka koristili u ovome radu i zašto, kako bismo dobili što detaljniji 3D prikaz karte. Pokazat ćemo kako možemo dohvatiti svaki od skupova podataka, što oni sadrže, zašto su nam važni i koja im je struktura. Također navest ćemo s kojim problemima smo se susreli sa svakim od izvora podataka, te koji su im nedostatci. Isto tako treba napomenuti da svaki izvor podataka ne mora imati isti koordinatni referentni sustav, međutim svi izvori podataka koje ćemo opisati se temelje na istom referentnom elipsoidu. Referentni elipsoid je matematički definirana površina koja aproksimira geoid, odnosno oblik Zemlje kao tijela, te je u ovom slučaju riječ o referentnom elipsoidu WGS84.

2.1. Google Maps Elevation API

Kao osnova svake trodimenzionalne karte stoji podloga odnosno teren na kojoj će se karta prikazivati. Taj teren najčešće nije samo ravna podloga već ovisi o visinskoj karti određenog područja kojeg prikazujemo, odnosno posjeduje udoline i uzvisine. Informacije o visinskoj karti ćemo upravo dobiti preko Google Maps Elevation API-ja. Ovo jednostavno aplikacijsko sučelje služi za traženje informacije o nadmorskoj visini na nekom mjestu na Zemlji. Podaci o nadmorskim visinama dostupni su gotovo za sve lokacije na Zemlji pa tako i za položaje u morima i oceanima gdje je visina negativna. Međutim postoji mogućnost da Google nema točne podatke o nadmorskoj visini za neku lokaciju, u tom se slučaju traže četiri najbliže lokacije za koje postoji pouzdano mjerenje o nadmorskoj visini, te se vraća prosječna vrijednost nadmorskih visina kako bi se dobila procjena nadmorska visine tražene lokacije. Sve vrijednosti nadmorskih visina dohvaćene ovim servisom izražene su u metrima i to u odnosu na lokalnu srednju razinu mora [4].

Rad s ovim aplikacijskim sučeljem očituje se slanjem HTTP zahtjeva preko HTTP protokola u čijem URL-u se nalazi informacija o lokaciji za koju tražimo vrijednost nadmorske visine. Kada poslužitelj primi naš HTTP zahtjev on u svojoj bazi nalazi tražene vrijednosti te nam šalje HTTP odgovor u čijem tijelu se nalaze podaci koje smo tražili.

Predložak HTTP zahtjeva koji šaljemo poslužitelju ima sljedeći oblik: *https://maps.googleapis.com/maps/api/elevation/**{outputFormat}**?**{parameters}*** , gdje su podebljani dijelovi varijabilni te su oni dani na izbor korisniku. Varijabilni dio *outputFormat* predstavlja oblik u kojem će biti dostavljen odgovor od poslužitelja, odnosno jezik kojim će biti označeni dostavljeni podaci, pa tako postoje dvije opcije: *json* ili *xml*. Odabirom opcije *json* odgovor će biti dostavljen u formatu JavaScript Object Notation, dok ćemo odabirom opcije *xml* dobiti odgovor u obliku EXtensible Markup Language u kojemu će podaci biti ograđeni oznakom <*ElevationResponse*>. Drugi varijabilni dio *parameters* označava skup parametara koji prenosimo poslužitelju na temelju čega će on napraviti upit nad bazom i vratiti nam tražene rezultate. Spomenuti parametri odvajaju se znakom *&* unutar URL-a, a njihove vrste su sljedeće [4]:

- *key* – označava ključ potreban za pozivanje aplikacijskog sučelja kako bi se identificirao korisnik
- *locations* – predstavlja lokacije za koje se postavlja upit, odnosno traže vrijednosti nadmorskih visina. Lokacija se navodi kao par geografske širine i geografske dužine odvojeni zarezom, dok se više lokacije odvaja znakom |
- *path* – definira put na Zemlji za koji se traži nadmorska visina, a predstavlja se skupom od dva ili više parova geografske širine i geografske dužine
- *samples* – predstavlja broj uzoraka koji će biti uzeti duž postavljenog puta za koje se dohvaća nadmorska visina

Uporaba ovog aplikacijskog sučelja obilježena je odabirom jednog od dva načina postavljanja upita i obaveznim navođenjem parametra *key*. Prvi način rada je korištenje isključivo parametra *locations* gdje se navodi diskretni skup lokacija za koje se traže vrijednosti nadmorske visine. Drugi način dohvaćanja vrijednosti nadmorskih visina je navođenjem parametara *path* i *samples*. U parametru *path* se definiraju lokacije koje predstavljaju neki put, a u parametru *samples* se navodi broj točaka na tom putu, koje se raspoređuju na njemu jednoliko, za koje će se vratiti vrijednost nadmorskih visina [4].

Rezultat slanja zahtjeva na poslužitelj rezultira dobivanjem odgovara čije tijelo sadrži dva elementa, a to su *status* koji govori o uspješnosti izvođenja zahtjeva i *results* koji predstavlja rezultate izvođenja upita. Element *results* je zapravo polje koje je sastavljeno od sljedećih dijelova [4]:

- *location* – lokacija predstavljena parom geografske širine i geografske dužine čija je nadmorska visina dostavljena
- *elevation* – nadmorska visina lokacije izražena u metrima
- *resolution* – maksimalna udaljenost između točaka za koje je nadmorska visina interpolirana

Na slici 2.1. prikazan je jedan primjer slanja upita na opisano programsko sučelje u svrhu dohvaćanja vrijednosti visine jedne lokacije i dobiveni odgovor.

Zahtjev:

Request URL: https://maps.googleapis.com/maps/api/elevation/json?locations=45.81,15.98&key={API_KEY}

Request Method: GET

Status Code: 200

Odgovor:


```
{
   "results" : [
      {
         "elevation" : 127.8884735107422,
         "location" : {
            "lat" : 45.81,
            "lng" : 15.98
         },
         "resolution" : 152.7032318115234
      }
   ],
   "status" : "OK"
}
```


Slika 2.1. Prikaz slanja zahtjeva i dobivanja odgovora putem Google Maps Elevation API-ja

Treba napomenuti kako Google-ovi servisi koriste referentni koordinatni sustav naziva EPSG:4326, što znači da se radi o geografskom koordinatnom sustavu koji je definiran pripadnom geografskom širinom i geografskom dužinom u decimalnim stupnjevima temeljen na referentnom elipsoidu WGS84.

2.2. Mapbox Static Images API

Osim samog visinskog izgleda terena poželjno bi bilo prije iscrtavanje 3D dijelova karte na sam teren zalijepiti određenu sliku odnosno teksturu koja će predstavljati satelitsku snimku upravo tog dijela prostora kojeg prikazujemo. Stavljanje spomenute slike dat će nam dodatno na realizmu te će pokriti možda neke dijelove terena s detaljima koji su dostupni preko satelita, a mi ih nismo uspjeli prikazati u trodimenzionalnoj reprezentaciji.

Mapbox Static Images API je aplikacijskog programsko sučelje koje stvara statičku sliku karte stvorene od strane Mapbox GL poslužitelja [5]. Dohvat statičke slike karte moguće je dobiti upućivanjem HTTP zahtjeva sljedećeg oblika: *https://api.mapbox.com/styles/v1/* ***{username}/{style_id}****/static/****{overlay}/{lon},{lat},{zoom},{bearing},{pitch}|{auto}|{bbox}/{width}x{height}****?****{token}***, gdje su podebljano označeni dijelovi varijabilni i dani korisniku na odabir.

Osim navedenih glavnih parametara koje ćemo ukratko opisati razlikujemo i opcionalne parametre koje nećemo koristiti pa ih nećemo niti navoditi, no njihov opis je moguće pronaći u dokumentaciji Mapbox Static Images API-ja.

Parametri *username* i *style_id* nam omogućuju odabir stila karte iz koje generiramo statičku sliku. Ako želimo dodavati oznake, vlastite slike ili različite oblike povrh već generirane statičke slike karte koristimo parametar *overlay*. Parametrima *lon* i *lat* odabiremo lokaciju koja će biti u centru karte iz koje generiramo sliku, dok parametrom *zoom* reguliramo faktor uvećanja slike. Parametrom *bbox* moguće je odrediti, putem koordinata vrhova, pravokutnik unutar kojeg će se nalaziti dio karte čiju sliku želimo dobiti. Ako pak želimo koristiti parametar *auto* tada će se pozicija karte automatski odrediti na temelju parametra *overlay* ili predefiniranih središnjih koordinata za odabrani stil karte. Veličinu slike u pikselima određujemo parametrima *width* i *height*. Kako bismo mogli ispravno pozvati aplikacijsko sučelje dužni smo poslati i parametar *token* koji identificira pozivatelja sučelja [5].

Kako bismo odredili poziciju za koju želimo stvoriti statičku sliku karte na raspolaganju nam je jedna od sljedeće tri opcije. Možemo se koristiti parametrom *auto*, parametrom *bbox* ili skupom od 3 tri parametra: *lon, lat* i *zoom*. Za zadavanje lokacije koristi se referentni koordinatni sustav EPSG:4326. Na slici 2.2. prikazana je jedna satelitska snimka generirana od strane opisnog poslužitelja.

Slika 2.2. Primjer satelitske snimke dobivene od MapBox Static Images API-ja

2.3. OpenStreetMap

Kao glavni izvor podataka za vizualizaciju područja u obliku 3D karte upotrijebit ćemo OpenStreetMap za koji ćemo u nastavku koristiti skraćenicu OSM. Velika ograničenja korištenja i nedostupnost velikog dijela kartografskih podataka diljem svijeta bila su motivacija Steve Coastu za stvaranje OSM-a. OSM projekt je nastao 2004. godine inspiriran popularnosti Wikipedije s primarnim ciljem stvaranja javne i besplatne karte svijeta s mogućnosti njezinog uređivanja. Autori OSM-a navode kako se on temelji na lokalnom znanju, zajedničkom radu, javnim podacima, legalnosti i suradnji s partnerima. Zajednica koja doprinosi poboljšanju OSM-a uključuje kartografske entuzijaste, stručnjake za GIS, inženjere koje održavaju OSM poslužitelje, humanitarce, volontere i mnoge druge koji su spremni surađivati. Danas OSM broji više od 7 milijuna korisnika koji surađuju u svrhu poboljšanja cjelokupnog sustava. Oni doprinose poboljšanju OSM-a koristeći zračne slike, GPS uređaje i jednostavne zemljovide kako bi održali i potvrdili točnu i ažurnu bazu kartografskih podataka. Prikupljeni podaci se zatim unose u OSM bazu podataka koristeći neki od dostupnih programskih alata. Kartografski podaci su zatim vidljivi na kartama generiranim od strane raznih specijaliziranih aplikacija za tu namjenu. Podaci u sklopu OSM-a su javni i slobodno dostupni na korištenje svima, te je njihovo uređivanje i distribuiranje dostupno pod licencom Open Database Licence [6].

OSM koristi topološku strukturu podataka koja je građena od četiri sastavna primitiva [6]:

- Čvor (engl. Node) – točka s geografskim koordinatama, namijenjena za označavanje značajke bez veličine
- Staza (engl. Way) – uređena lista čvorova koja može predstavljati liniju kao oznaku linearne značajke ili poligon ako tvori zatvorenu petlju kao oznaka površine
- Relacija (engl. Relation) – uređena lista čvorova, staza ili relacija, koji se nazivaju član (engl. Member), čija je uloga reprezentacija odnosa postojećih čvorova i staza
- Oznaka (engl. Tag) – par ključ-vrijednost koja služi za čuvanje meta podataka o objektima na karti, te ona nikad ne stoji samostalno već uz čvor, stazu ili relaciju

Imena oznaka su uglavnom standardizirana i preporučuje se korištenje koje je navedeno na idućoj web stranici: *https://wiki.openstreetmap.org/wiki/Map_features*.

Kartografske podatke moguće je preuzeti za neko područje u obliku .osm datoteke koja je zapravo XML datoteka s ograničenim skupom oznaka. Datoteku je moguće preuzeti preko

HTTP zahtjeva oblika: *https://www.openstreetmap.org/api/0.6/map?bbox=**{minLon}**%2C**{minLat}**%2C**{maxLon}**%2C**{maxLat}***. Parametri *minLon, minLat, maxLon* i *maxLat* predstavljaju vrijednosti minimalne i maksimalne geografske širine i geografske dužine pravokutnika na karti za čiji dio želimo dohvatiti kartografske podatke.

Na slici 2.3. prikazana su tri isječka datoteke koja sadrži OSM podatke.

OSM se temelji na referentnom koordinatnom sustavu EPSG:4326, pa su tako lokacije podataka izražene putem geografske širine i dužine.

```
<?xml version="1.0" encoding="UTF-8"?>
<osm version="0.6" generator="CGImap 0.8.3 (3283378 spike-08.openstreetmap.org)" copyright="OpenStreetMap and contributors" attribution="http://www.openstreetmap.org/copyr
 <bounds minlat="45.8000000" minlon="15.9658000" maxlat="45.8100000" maxlon="15.9913000"/>
 <node id="18018884" visible="true" version="8" changeset="49838672" timestamp="2017-06-26T15:03:00Z" user="starwars425" uid="5531147" lat="45.8062980" lon="15.9923061">
  <tag k="converted_by" v="Track2osm"/>
  <tag k="highway" v="traffic_signals"/>
 </node>
 <node id="18019055" visible="true" version="12" changeset="94591748" timestamp="2020-11-22T17:26:48Z" user="Janjko" uid="244754" lat="45.8054579" lon="15.9840639">
  <tag k="highway" v="traffic_signals"/>
 </node>
 <node id="20840179" visible="true" version="11" changeset="82132378" timestamp="2020-03-12T22:29:58Z" user="Rostaman521" uid="3349405" lat="45.8074059" lon="15.9722472">
  <tag k="highway" v="traffic_signals"/>
 </node>
 <way id="265911034" visible="true" version="4" changeset="91888315" timestamp="2020-10-03T01:38:13Z" user="Rostaman521" uid="3349405">
  <nd ref="2715189202"/>
  <nd ref="7967758173"/>
  <nd ref="2715190400"/>
  <tag k="highway" v="service"/>
  <tag k="service" v="driveway"/>
 </way>
 <way id="267626685" visible="true" version="2" changeset="70163122" timestamp="2019-05-12T15:16:10Z" user="simulator1" uid="7595534">
  <nd ref="2730695568"/>
  <nd ref="2730695588"/>
  <tag k="highway" v="footway"/>
  <tag k="surface" v="gravel"/>
 </way>
 <relation id="123072" visible="true" version="17" changeset="91138114" timestamp="2020-09-19T02:37:21Z" user="Rostaman521" uid="3349405">
  <member type="node" ref="25323157" role="via"/>
  <member type="way" ref="327862982" role="to"/>
  <member type="way" ref="327862981" role="from"/>
  <tag k="restriction" v="only_straight_on"/>
  <tag k="type" v="restriction"/>
 </relation>
 <relation id="123074" visible="true" version="7" changeset="14226038" timestamp="2012-12-10T13:31:23Z" user="Janjko" uid="244754">
  <member type="way" ref="141582254" role="to"/>
  <member type="way" ref="195557694" role="from"/>
  <member type="node" ref="1549698055" role="via"/>
  <tag k="restriction" v="only_right_turn"/>
  <tag k="type" v="restriction"/>
 </relation>
```

Slika 2.3. Isječci OSM datoteke s kartografskim podacima

Naposljetku valja spomenuti kako se kao alternativa ovom način dohvata OSM podataka nameće korištenje već statičkih datoteka s nekih web stranica poput *www.geofabrik.de* koju smo koristili u samom začetku ovog rada. Međutim kao veliki nedostatak ovakvog pristupa je to što moramo unaprijed znati o kojoj se državi radi za odabrano područje da bismo znali koju statičku datoteku preuzeti. Kasnije je tu datoteku moguće jednostavno procesirati pomoću nekih od dostupnih komandnih alata poput Osmosis-a kako bismo izvukli točno određeni podskup informacija koji nam je potreban, no u konačnici smo se ipak odlučili za prethodno opisani način dohvata OSM podataka, a u 3. poglavlju ćemo opisati kako ih procesiramo.

2.4. GIS Zrinjevac

Iako nam opisani izvor podataka OpenStreetMap pruža širok spektar informacija iz kojeg možemo crpiti podatke za našu 3D kartu, on je prilično generaliziran pa je kao takav dosta ne detaljan u određenim aspektima. Stoga za područje grada Zagreba postoji izvor podataka GIS Zrinjevac koji predstavlja katastar zelenila glavnoga grada.

GIS Zrinjevac sadrži informacije o tri tipa podataka [7]:

- Grmlje – površine predstavljene poligonima na kojima se nalaze grmolike biljke
- Stabla – lokacije na kojima se nalaze stabla
- Urbana oprema – lokacije na kojima se nalazi javna urbana oprema kao što su klupe, stolovi, kante za smeće, fontane, oprema za dječje igralište, itd.

GIS Zrinjevac svojim korisnicima omogućava pregled navedenih podataka na interaktivnoj 2D karti koja je prikazana na slici 2.4., međutim kako bismo mi iskoristili te podatke trebamo naći neki drugi način dohvata podataka koji se prikazuju na karti [7]. Ako se pomnije pogleda u internetski promet moguće je vidjeti da se prilikom svakog novog pogleda na kartu šalje određeni HTTP zahtjev poslužitelju, te se dobiva struktura u kojoj su zapisane informacije koje smo tražili.

Slika 2.4. Interaktivna karta GIS Zrinjevac

Oblik HTTP zahtjeva koji je potrebno poslati kako bi dobili informacije od poslužitelja izgleda ovako: *https://gis.zrinjevac.hr/**{request_type}**?bbox=**{minLon}**,**{minLat}**,**{maxLon}**,**{maxLat}**&srid=3857.* Parametar *request_type* određuje koji tip podataka želimo

dohvatiti, pa tako ako se radi o grmlju postavljamo parametar na vrijednost *grmlje_geom.php*, a za stabla *stabla_geom.php*, ako pak želimo dohvatiti urbanu opremu šaljemo vrijednost *oprema_geom.php*. Parametri *minLon, minLat, maxLon, maxLat* su ekstremne vrijednosti geografske širine i dužine koje predstavljaju rubne koordinate područja za koje tražimo podatke. Na slici 2.5. prikazani su prvih desetak redaka odgovora od poslužitelja za sva tri tipa parametra *request_type* na istom proizvoljno odabranom području. Ovako opisanim slanjem zahtjeva poslužitelju moguće je dobiti tek popis traženih tipova podataka na nekom području sa svojom jedinstvenom šifrom u bazi i opisom geometrije koja uključuje tip opisne geometrije i skup koordinata koje ju opisuju.

Slika 2.5. HTTP odgovor od GIS Zrinjevac poslužitelja za podatke o grmlju (lijevo), stablima (sredina) i urbanoj opremi (desno)

Za detaljnije informacije o pojedinom podatku potrebno je poslati dodatni upit sljedećeg oblika: *https://gis.zrinjevac.hr/**{request_type}**?sifra=**{id}***, gdje je parametar *id* jedinstvena šifra podatka u bazi prema kojoj se dohvaćaju detalji za traženi podatak. Ako se radi o tipu podatka grmlje onda detaljne informacije sadrže podatak o nazivu grma, odnosno njegovoj vrsti, te površini grma u m^2. Detalji o stablima sadrže naziv stabla, prostorni raspored, zaštitne elemente, promjer debla, promjer krošnje i visinu stabla, dok detalji o urbanoj opremi posjeduju informacije o vrsti i stanju opreme, te datumu pregleda.

Ovaj izvor podataka se temelji na referentnom koordinatnom sustavu EPSG:3857, a takav koordinatni sustav je sustav koji je projiciran WGS84 koordinatni sustav na ravnu kvadratnu plohu, te se još naziva Pseudo-Mercatorov koordinatni sustav. U ovakvom sustavu lokacije nije određena preko geografske širine i dužine, već je određen u metrima tako da je lokacije u središtu karte označena kao par $(0\,m, 0\,m)$, te se onda koordinate kreću dodavanjem metara prema istoku i sjeveru odnosno oduzimanjem metara u smjeru zapada i juga.

3. Dohvat i obrada podataka

Nakon opisa svakog od pojedinog izvora podataka opisat ćemo kako smo konkretno dohvatili potreban podskup potrebnih podataka u programskom alatu Unity, te kako smo ga obradili i prilagodili kako bi bio prikladan za daljnju uporabu u izradi 3D karte. U nastavku se pretpostavlja kako su na početku rada aplikacije unijete rubne koordinate dijela karte koju izrađujemo, odnosno maksimalne i minimalne vrijednosti geografske širine i dužine. Koordinate koje se unose preko grafičkog korisničkog sučelja pretpostavljene su u referentnom koordinatnom sustavu EPSG:4326. Osim samih koordinata na početku rada aplikacije unosi se i faktor skaliranja koji govori koliko će jedan metar u stvarnosti odgovarati jedinica u virtualnom prostoru. Više o korisničkom sučelju, samom pokretanju i radu aplikacije moguće je naći pod predviđenom temom u privitku ovog rada.

Prije nego što krenemo s opisom kako smo stvorili teren i postavili ga ukratko ćemo objasniti predloške (engl. Prefab) u programskom alatu Unity. Ključna stvar za Unity je ta da se pokretanje aplikacije temelji na prikazu trenutno postavljene scene i objekata u njoj. Objekte je moguće dodati ručno u scenu i namjestiti ih preko grafičkog sučelja Unity-a, no takav način stvaranja objekta ne omogućava nam dinamičko stvaranje i promjenu svojstava objekata putem skripti što je ključno za rad aplikacije. U tu nam svrhu služe već spomenuti predlošci. Predložak bilo kojeg objekta moguće je spremiti u određeni direktorij te ga je moguće povezati preko reference s nekom skriptom. Tako je moguće unutar skripte, odnosno tijekom rada aplikacije, pozivom metode *Instantiate()* s referencom na predložak stvoriti objekt u našoj sceni. Tada ćemo imati iscrtan objekt u sceni i referencu na njega tipa *GameObject* preko koje ćemo moći mijenjati postavke čiji raspon ovisi o konkretnom tipu objekta, odnosno komponentama koje posjeduje [9].

3.1. Postavljanje terena

Kao podloga svake karte pa tako i naše bit će teren. Stvaranje terena u Unity-u je prilično jednostavno zato što Unity posjeduje objekt tipa teren (engl. Terrain), pa ga je tako potrebno samo pretvoriti u predložak i povezati ga sa skriptom, te ga onda instancirati u skripti [9]. Zatim je na temelju unesenih rubnih koordinata u glavnom izborniku aplikacije potrebno izračunati dimenziju našeg terena. Dimenziju terena ćemo dobiti koristeći metodu

SphericalCosinus za izračunavanje udaljenosti u metrima između dvije točke na temelju njezinih koordinata u sustavu EPSG:4326 koja je prikaza na slici 3.1.. Navedenu metodu ćemo provesti dva puta kako bi dobili duljinu i širinu terena. Dobivene vrijednosti pomnožene s koeficijentom skaliranja ćemo pridružiti varijabli *size* koja je tipa *Vector3*, tako da njezina x komponenta odgovara širini (engl. width) terena, a z komponenta dužini (engl. length). Opis vrijednosti y komponente dati ćemo nešto kasnije. Dužina terena odgovarat će razlici između maksimalne i minimalne geografske širine, a širina terena razlici maksimalne i minimalne geografske dužine.

```
//Udaljenost dvije točke  (u metrima) u prosotoru na temelju geografske dužine i širine
public static double SphericalCosinus(double lat1, double lon1, double lat2, double lon2)
{
    double R = 6371.0;
    double dLon = toRad(lon2 - lon1);
    lat1 = toRad(lat1);
    lat2 = toRad(lat2);
    double distance = Math.Acos(Math.Sin(lat1) * Math.Sin(lat2) + Math.Cos(lat1) * Math.Cos(lat2) * Math.Cos(dLon)) * R;
    return distance * 1000;
}
```

Slika 3.1. Metoda SphericalCosinus

Nakon stvaranja terena i postavljanja dimenzija po x i z osi ostaje pitanje kako ćemo urediti visinu terena odnosno dimenziju po y osi. Tu informaciju ćemo dobiti preko izvora podataka Google Elevation API. Treba naglasiti kako kod objekta tipa terena u Unity-u nije moguće za proizvoljnu njegovu točku postaviti visinu već je unaprijed potrebno postaviti visinsku mrežu predefinirane rezolucije te se tek onda za svaku točku te mreže može postaviti visina, odnosno odmak po y osi. Rezoluciju visinske mreže moguće je promijeniti preko parametra *Heightmap Resolution* pod komponentom *Terrain* terena, kao jednu od nekoliko predefiniranih opcija između $33x33$ i $4097x4097$, gdje je taj umnožak jednak broju ravnomjerno raspoređenih točaka na terenu za koje možemo definirati visinu. Zaključujemo kako je potrebno poslati upit Elevation API poslužitelju za svaku točku iz mreže. Kako aplikacijsko sučelje ima postavljeno ograničenje o broju lokacija za koje šalje odgovor o visini nije moguće poslati upit za sve točke odjednom. Stoga ćemo podijeliti razliku unesene maksimalne i minimalne geografske širine na toliko dijelova koliko posjeduje jedna dimenzija postavljenje mreže točaka što će nam biti inkrement pojedine iteracije za raspon geografske širine, dok ćemo geografsku dužinu držati na punom zadanom rasponom. Tako ćemo pozivati HTTP zahtjev za svaki dio geografske širine s brojem točaka jednakim jednoj dimenziji visinske mreže.

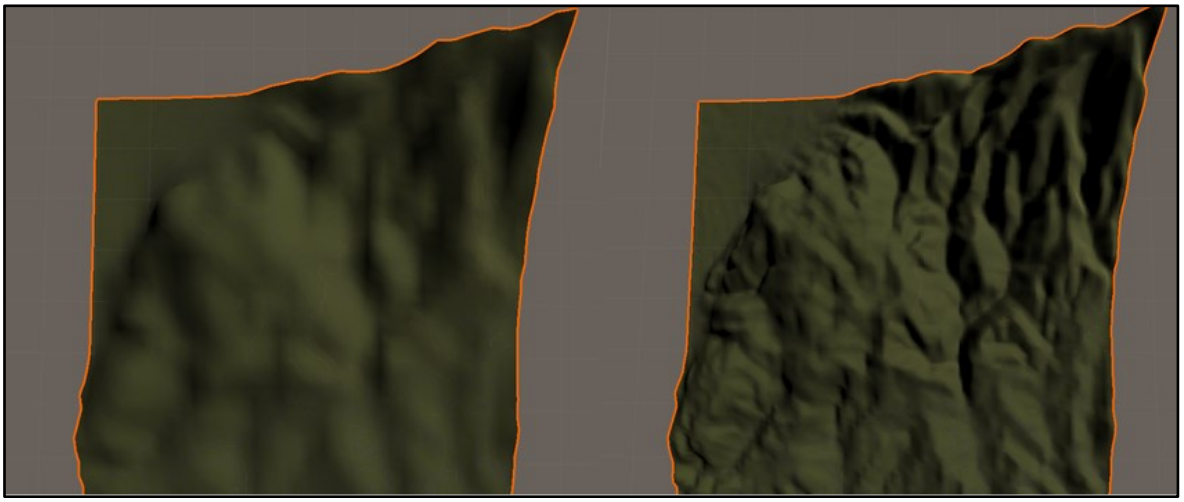
Opisane informacije ćemo dobiti od poslužitelja slanjem sljedećeg zahtjeva: *https://maps.googleapis.com/maps/api/elevation/json?path=**{latitude},{minLongitude}|{lat***

itude}*,*{maxLongitude}*&*samples=terrainRes*&*key=*{API_KEY}***, gdje parametar *latitude* ima vrijednost minimalne geografske širine uvećane za umnožak inkrementa geografske širine i broja iteracije. Slanjem ovakvih zahtjeva dobit ćemo visinsku vrijednost za svaku točku naše visinske mreže. S takvim skupom informacija pozvat ćemo metodu *ApplyGoogleElevationAPI* koja će na temelju informacija o terenu i informacija o visinama postaviti vrijednosti točaka visinske mreže. U Unity-u se to radi tako da su vrijednosti tih točaka decimalni brojevi u rasponu od 0 do 1, te se na temelju tih koeficijenata i maksimalne visine terena postavljaju stvarne vrijednosti visina koje smo dohvatili iz izvora podataka. Spomenutu metodu moguće je vidjeti na slici 3.2., dok ćemo maksimalnu visinu terena predati kao y komponentu već spomenutoj varijabli *size* stvorenog terena.

```
//Metoda za postavljanje visinske mape na teren temeljem podataka iz Google Elevation API-ja
void ApplyGoogleElevationAPI(TerrainData terrain, int terrainRes, List<List<ElevationObject>> elevations, double maxElev)
{
    float[,] heights = terrain.GetHeights(0, 0, terrainRes, terrainRes);
    double[,] array = new double[terrainRes, terrainRes];
    for (int i = 0; i < terrainRes; ++i)
    {
        List<ElevationObject> elevationRow = elevations[i];
        for (int j = 0; j < terrainRes; ++j)
        {
            double elev = elevationRow[j].elevation;
            double normElev = elev/maxElev;
            array[i, j] = normElev;
        }
    }
    for (int x = 0; x < terrainRes; x++)
    {

        for (int y = 0; y < terrainRes; y++)
        {
            heights[terrainRes-y-1, x] = (float)array[x,y];

        }
    }
    terrain.SetHeights(0, 0, heights);
}
```

Slika 3.2. Metoda ApplyGoogleElevationAPI

Treba napomenuti kako Google Elevation API nije u potpunosti besplatan već je njegovo korištenje određeno brojem poslanih HTTP zahtjeva. Na slici 3.3. prikazana je usporedba stvorenog terena uz odabir dviju različito odabranih rezolucija visinske mreže. Na lijevom dijelu te slike moguće je vidjeti teren generiran uz rezoluciju visinske mreže od $65x65$ što uzrokuje trošak od 65 HTTP zahtjeva, dok je na desnom dijelu vidljiv teren s rezolucijom od $257x257$ s troškom od 257 zahtjeva. Također što je odabrana veća rezolucija visinske mreže dohvat i obrada tih podataka traje duže, no o tome ćemo nešto više reći u poglavlju 7.



Slika 3.3. Usporedba stvorenog terena uz odabir dviju rezolucija visinske mreže

Nakon stvorenog terena, postavljenih njegovih dimenzija i uzvisina odnosno udubina zadnji korak kod postavljanja terena u naše virtualno okruženje bit će dohvat satelitske snimke koja predstavlja satelitsku fotografiju za područje koje prikazujemo. Opisanu statičku sliku dohvatit ćemo iz Mapbox Static Images API-ja, gdje ćemo tom poslužitelju poslati sljedeći HTTP zahtjev: *https://api.mapbox.com/styles/v1/mapbox/satellite-v9/static/[**{minLon}**, **{minLat}**,**{maxLon}**,**{maxLat}**]/**{x}**x**{y}**@2x?access_token={**API_KEY**}*.

Parametre *minLon, minLat, maxLon* i *maxLat* dobit ćemo unosom korisnika preko korisničkog sučelje. Parametar *@2x* govori kako želimo mapu visoke gustoće radi boljeg prikaza. Parametri *x* i *y* koji određuju širinu odnosno visinu slike te su u rasponu od 1 do 1280. Njih ćemo odrediti tako da će *x* biti povezan s komponentom x veličine terena, *y* s komponentom z veličine terena čime smo osigurali ispravan omjer slike. Međutim veličina terena često neće biti u rasponu koji je dozvoljen za parametre *x* i *y* stoga ćemo veći parametar postaviti na 1280, a manji na vrijednost tamo da očuvamo ispravan omjer koji smo dobili iz dimenzija terena.

3.2. OSM podaci

Dohvat OSM podataka je jako jednostavan i temelji se samo na slanju jednog HTTP zahtjeva poslužitelju koji izgleda ovako:
*https://www.openstreetmap.org/api/0.6/map?bbox=**{minLon}**%2C**{minLat}**%2C**{maxLon}**%2C**{maxLat}***, gdje parametri *minLon, minLat, maxLon* i *maxLat* određuju rub područja za koje preuzimamo podatke.

Obrada preuzetih podataka odnosno parsiranje se izvodi tako što čitamo liniju po liniju preuzetog sadržaja te razlikujemo nekoliko slučajeva:

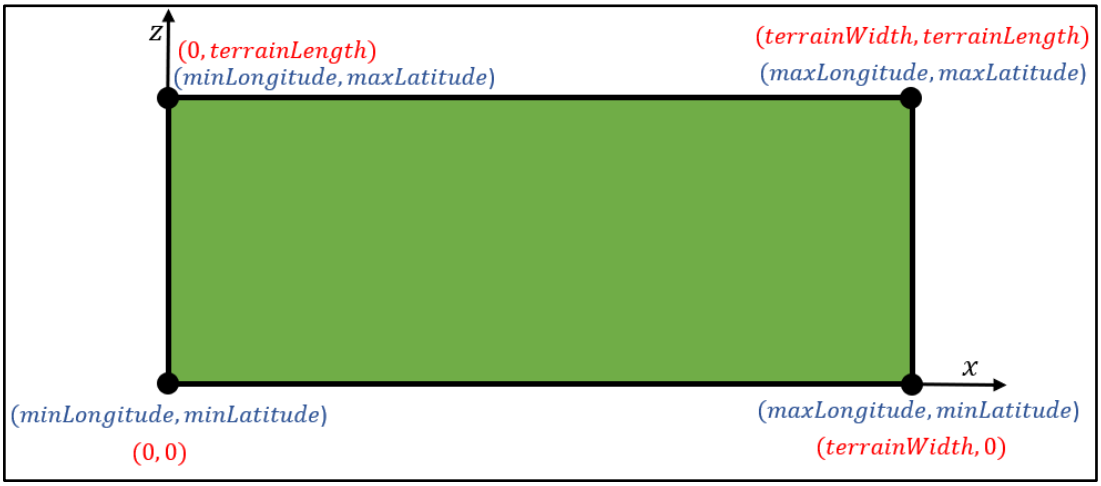
1. Ako linija sadrži skup znakova „*<node“* tada znamo da se radi o početku opisa nekog čvora. Takva linija uvijek sadrži podatke o identifikatoru čvora (*id*) te geografsku širinu (*lat*) i dužinu (*lon*) koje izvlačimo i spremamo u varijable. Ako ta linija također sadrži i „*/>“* skup znakova to znači da je cijeli čvor opisan u toj liniji i da nema više podataka o njemu, tada stvaramo objekt tipa *Node* s prethodno spremljenim varijablama, spremamo ga u listu čvorova i čistimo sve varijable za dolazak idućeg elementa. U suprotnom iza ove linije dolazi popis oznaka koje opisuju detaljnije čvor.
2. Ako linija sadrži skup znakova „*</node>“* to označava kako je završeno opisivanje čvora te tada sve informacije o njemu možemo spremiti u stvoreni objekt tipa *Node*, a njega spremiti u listu čvorova. Na kraju čistimo sve upotrjebljene varijable.
3. Ako linija sadrži skup znakova „*<way“* tada je riječ o liniji koja predstavlja početak opisa neke staze. Unutar te linije nalazimo identifikator staze te ga spremamo u pripadnu varijablu. Iza ove linije dolazi popis referentnih čvorova i popis oznaka koje opisuju stazu.
4. Ako linija sadrži skup znakova „*<nd“* tada znamo da ta linija sadrži referencu na neki čvor koji gradi neku stazu sa informacijom o njegovom identifikatoru (*id*). Taj identifikator spremamo u listu sa identifikatorima referentnih čvorova.
5. Ako linija sadrži skup znakova „*<tag k=“**{key}**“ v=“**{value}**““,* gdje su *key* i *value* parametri koje tražimo predefinirani u skladu s time koji su nam podaci relevantni za rad naše aplikacije, tada znamo da se radi o nekoj informaciji odnosno detalju (oznaci) koja opisuje neki čvor ili stazu.
6. Ako linija sadrži skup znakova „*</way>“* to označava kako je gotov opis staze te tada stvaramo objekt tipa *Way* ili *MultiPolygon* u koji spremamo sve informacije o stazi koje smo su nam korisne i spremamo ju u pripadnu listu staza i nakon toga čistimo sve korištene varijable kao pripremu za dolazak novog elementa. Tip objekta koji stvaramo i u koju ju listu spremamo ovisi o oznakama koju su opisivale stazu.

U konačnici sve ćemo čvorove koje smo dohvatili iz našeg skupa podataka spremiti u jednu listu objekata tipa *Node* koja će sadržavati geografsku lokacija čvora, njegov identifikator i tip koji govori što taj čvor predstavlja.

Staze dijelimo u dvije kategorije. Prva kategorija predstavlja one staze za koje kreiramo objekt tipa *Way*, a one obuhvaćaju ceste, željeznice i građevine. U drugu kategoriju spadaju ograde, prirodne površine, sportske površine, rekreativne površine, povijesne površine, zemljišta, društveni sadržaji i objekti stvoreni od čovjeka, a njih spremamo u objekt tipa *MultiPolygon*. Objekt tipa *MultiPolygon* sadrži samo identifikator, listu referentnih čvorova i tip koji govori što ta staza predstavlja, dok objekt tipa *Way* sadrži sve ono što i *MultiPolygon* i još posebne varijable koje sadrže posebne informacije o željeznicama, cestama i građevinama kao što su npr. broja katova, adresa, naziv ulice i širina pruge. Ovakva raspodjela u kategorije i potkategorije nije bila nužna već je rezultat stvaranja aplikacije kroz više faza i lakšeg dohvata ciljanog dijela podataka u kasnijim fazama rada aplikacije.

Potpuni popis oznaka koje su uzete u obzir pri radu aplikacije odnosno obradi OSM skupa podataka, te opis njihove reprezentacije i tip podatka za koji su iskorištene nalazi se u privitku ovog rada pod naslovom „Popis oznaka za čvorove i staze“.

Sljedeći korak koji moramo obaviti je preslikati dohvaćene geografske koordinate, za svaki od čvorova koji smo spremili u listu, u poziciju u virtualnoj sceni. Kako je jedan kut terena poravnat s ishodištem koordinatnog sustava scene, a teren se proteže duž x i z koordinatnih osi u pozitivnom smjeru, porast x koordinate predstavlja veću geografsku dužinu, a porast z koordinate predstavlja veću geografsku širinu. Sukladno tome i opisu stvaranja terena iz poglavlja 3.1. svi će čvorovi biti raspoređeni u sceni unutar $[0, \check{s}irinaTerena]$ vrijednosti x koordinate i $[0, \ dužinaTerena]$ vrijednosti z koordinate. Ilustraciju koja opisuje odnos koordinatnih osi virtualnog prostora odnosno scene, geografskih koordinata i dimenzija terena moguće je vidjeti na slici 3.4. gdje su plavom bojom označene geografske koordinate, a crvenom pozicije na terenu. Isječak koda koji obavlja preslikavanje čvora iz geografskih koordinata u koordinate na sceni prikazan je na slici 3.5. Izračun pozicije u sceni se temelji na vrijednosti odstupanja geografskih koordinata od minimalnih koordinata, te računanju koliko to odstupanje iznosi u prostoru scene na temelju zadanog odnosa sa slike 3.3. Međutim opisani postupak se odnosi samo za x i z poziciju čvora u sceni, no kako bismo odredili pripadnu y komponentu potrebno je pozvati metodu *SampleHeight(position)* nad terenom, gdje je varijabla *position* tipa *Vector3* pozicija na terenu za koju želimo dobiti visinu.



Slika 3.4. Ilustracija odnosa geografskih koordinata, dimenzija terena i koordinatnih osi

```
double dx = Math.Abs(node.longitude - minLongitude);
double dz = Math.Abs(node.latitude - minLatitude);
double dxNorm = dx * (terrainSize.x / (maxLongitude - minLongitude));
double dzNorm = dz * (terrainSize.z / (maxLatitude - minLatitude));
Vector3 position = new Vector3((float)dxNorm, 0, (float)dzNorm );
position.y = Terrain.activeTerrain.SampleHeight(position);
node.setPosition(position);
```

Slika 3.5. Isječak koda koji obavlja preslikavanje geografskih koordinata u koordinate u sceni

Na ovaj način učinili smo obradu datoteke koja sadrži OSM podatke, a kao rezultat obrade imamo različite liste objekata koje kasnije predajemo pripadnim skriptama čiji je zadatak reprezentacija tih objekata u virtualnom prostoru.

3.3. GIS Zrinjevac podaci

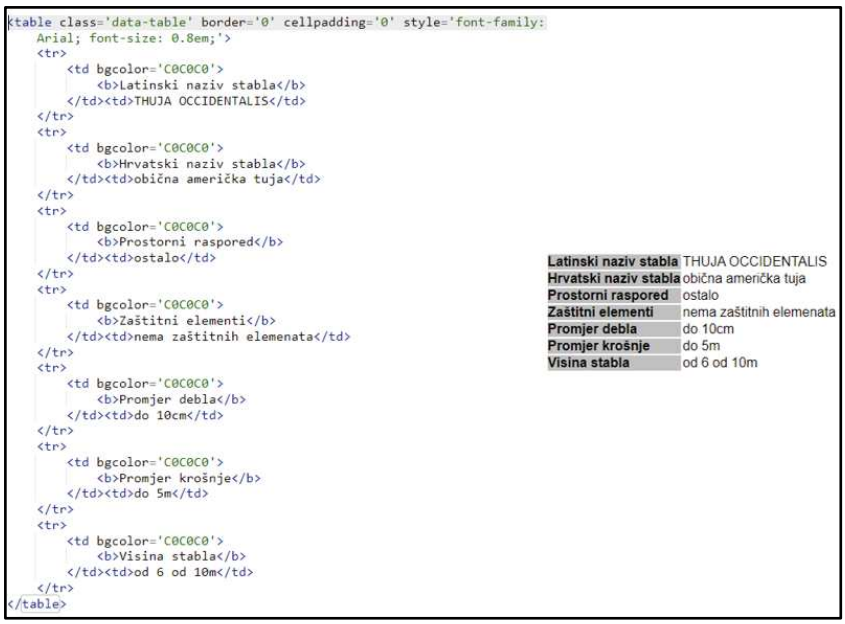
Iz izvora GIS Zrinjevac preuzet ćemo tri vrste podataka: stabla, grmlje i javnu opremu. Kako se ovaj izvor podataka temelji na referentnog koordinatnom sustavu EPSG:3857, potrebno je naše granične koordinate pretvoriti u taj sustav. Metoda koja pretvara zadanu lokaciju izraženu preko geografske širine i geografske dužine u sustavu EPSG:4326 u lokaciju izraženu metrima u sustavu EPSG:3857 prikazana je na slici 3.6.

```
//Metoda za pretvorbu koordinata iz sustava EPSG4326 u sustav EPSG3857
public static double[] transformCoordinatesEPSG4326To3857(double longitude, double latitude)
{
    double x = longitude * 20037508.34 / 180;
    double y = Math.Log(Math.Tan((90 + latitude) * Math.PI / 360)) / (Math.PI / 180);
    y = y * 20037508.34 / 180;
    return new double[] { x, y };
}
```

Slika 3.6. Metoda za pretvorbu koordinata iz sustava EPSG:4326 u sustav EPSG:3857

Pozivima te metode moći ćemo dobiti parametre *minBoundX*, *minBoundY*, *maxBoundX* i *maxBoundY* koje predajemo u HTTP zahtjev koji šaljemo poslužitelju GIS Zrinjevac. Za dohvaćanje stabala nekog područja šaljemo zahtjev sljedećeg oblika: *https://gis.zrinjevac.hr/stabla_geom.php?bbox=**{minBoundX}**,**{minBoundY}**,**{maxBoundX}**,**{maxBoundY}**&srid=3857.*

Nakon uspješno primljenog odgovora od poslužitelja potrebno ga je parsirati tako da ćemo za svako stablo iz odgovora izvući njegov jedinstveni identifikator označen u odgovoru kao *sifra:{id}* i njegovu lokaciju u EPSG:3857 sustavu koju pronalazimo u odgovoru pod nazivom *coordinates:[{x},{y}]*. Osim ovih podataka kako bi saznali visinu svakog stabla potrebno je za pojedino stablo poslati dodatni zahtjev na poslužitelj oblika: *https://gis.zrinjevac.hr/stabla.php?sifra=**{id}***. Odgovor ovakvog zahtjeva je zapravo HTML kod za razliku od ostalih odgovora koji su bili u JSON formatu. Primjer jednog takvog odgovora moguće je vidjeti na slici 3.7. gdje je na lijevom dijelu slike prikazan „čisti“ izvorni kod, a na desnom njegova HTML reprezentacija. Iz takvog ćemo odgovora dobiti visinu pod oznakom *Visina stabla* i tip stabla pod oznakom *Hrvatski naziv stabla* koje će nam kasnije biti potrebne za realističniju reprezentaciju. Za svako stablo stvorit ćemo objekt tipa *Node* u koji će sadržavati osim njegovog identifikatora i lokacije, vrstu te visinu stabla. Lokacija će početno biti spremljena kao koordinate u EPSG:3857 sustavu, a po stvaranju objekta pokrenut ćemo člansku metodu objekta *Node* koja pretvara lokaciju iz sustava EPSG:3857 u sustav EPSG:4326 kako bi bio u skladu s OSM podacima. Navedenu metodu moguće je vidjeti na slici 3.8. Na kraju stvorene objekte spremit ćemo u listu svih stabala.



Slika 3.7. Odgovor GIS Zrinjevac poslužitelja za detalja o stablu

```
public void transformCoordinatesEPSG3857To4326()
{
    float e = 2.7182818284f;
    float x = 20037508.34f;
    double longitude = (coordinateX * 180) / x;
    double latitude = (coordinateY / (x / 180));
    latitude = ((Math.Atan(Math.Pow(e, ((Math.PI / 180) * latitude)))) / (Math.PI / 360)) - 90;
    this.latitude = latitude;
    this.longitude = longitude;
}
```

Slika 3.8. Članska metoda za pretvorbu koordinata iz sustava EPSG:3857 u sustav EPSG:4326

Postupak identično izgleda kada se dohvaćaju podaci o javnoj urbanoj opremi, a jedina je razlika što se kod tih podataka izdvajaju samo informacije o identifikatoru, lokaciji i vrsti opreme. Pripadni HTTP zahtjev za dohvat urbane opreme nekog područja izgleda *https://gis.zrinjevac.hr/oprema_geom.php?bbox=**{minBoundX}**,**{minBoundY}**,**{maxBoundX}**,**{maxBoundY}**&srid=3857,* a HTTP zahtjev za dohvat detalja o nekom komadu opremu izgleda ovako *https://gis.zrinjevac.hr/ oprema_ geom.php?sifra=**{id}***. Stvorene objekte ćemo na posljetku ćemo spremiti u listu javne urbane opreme.

Dohvat i obrada grmlja je malo drugačija s obzirom na to da je ono reprezentirano poligonalnim tipom geometrije, a ne točkastim kao kod prethodno opisanih podataka. S tim na umu popis poligona koji predstavljaju grmlje koje dobivamo kao odgovor na zahtjev *https://gis.zrinjevac.hr/grmlje_geom.php?bbox=**{minBoundX}**,**{minBoundY}**,**{maxBoundX}**,**{maxBoundY}**&srid=3857,* umjesto para koordinata sadrži listu parova koordinata. Upravo iz tog razloga ćemo za svaki grm stvoriti objekt tipa *MultiPolygon* kojem ćemo predati identifikator i listu parova koordinata. Za ovaj tip podataka nećemo slati pojedine zahtjeve za detalje za svaki grm, već su nam dovoljni dobiveni opći podaci. Sve koordinate ćemo opet pretvoriti u EPSG:4326 sustav, te stvorene objekte spremiti u listu koja sadrži objekte grmlja.

Na kraju je potrebno još proći kroz sve opisane liste objekata i za svaki objekt, kao i kod OSM podataka, izračunati i spremiti poziciju koji objekt ima u sceni na temelju spremljenih koordinata.

4. Teksturiranje terena

Nakon dohvaćenih i obrađenih potrebnih podataka, krenut ćemo sa stvaranjem karte. U ovom ćemo se poglavlju još uvijek baviti samo prikazom 2D detalja koji će našu kartu učiniti uvjerljivijom, a čije smo podatke dobili obradom OSM skupa podataka. Konkretno upoznat ćemo postupak kako smo obojili odnosno teksturirali pojedine dijelove terena kako bi nam predstavljali površine raznih namjena i vrsta, ceste i putove i satelitsku kartografsku podlogu.

Prije no što krenemo detaljnije opisivati kako smo prikazali pojedine podatke kao teksture, osvrnut ćemo se na sam način izvođenja teksturiranja terena u Unity-u. Kako bismo omogućili dinamično teksturiranje terena kroz skripte, temeljem dobivenih podataka, moramo prvo stvoriti slojeve terena (engl. Terrain Layer). U Unity-u sloj terena je način na koji se definira površinska karakteristika terena. Svaki sloj sadrži teksture i druga svojstva koja materijal terena koristi za prikazivanje površine terena [9]. Sve teksture koje smo koristili za stvaranje slojeva preuzete su s web stranice textures.com [10]. Dodavanje preuzetih tekstura, stvaranje slojeva i dodavanje slojeva terenu ostvarivo je jednostavno i intuitivno kroz grafičko sučelje Unity-a. Popis korištenih slojeva i njihov opis moguće je naći u privitku ovog rada pod naslovom „Popis slojeva terena“.

Kada smo dodali slojeve terenu, teksturiranje terena putem skripte se izvodi tako što prolazimo kroz sve točke teksturalne mreže te postavljamo koliki udio u toj točki odnosno kvadratiću 1x1, ima određeni sloj. Taj se udio predaje kao polje vrijednosti u intervalu $[0, 1]$. Kako mi nećemo miješati niti jednu od tekstura polje koje predaje bit će uvijek ispunjeno s $k - 1$ članova vrijednosti 0 i jednim članom vrijednosti 1, gdje je k broj različitih slojeva. Teksturalna mreža će biti rezolucije 4096x4096 što je maksimalno dopušteno u Unity-u, a što je veća njezina vrijednost to je teksturiranje preciznije shodno tome i iscrtani prikaz točniji.

Sad kada smo upoznali pojam teksturalne mreže valja još napomenuti kako smo prilikom obrade OSM podataka kada smo računali poziciju čvorova u sceni također izračunali i poziciju čvorova na mreži, što smo obavili jednostavno tako da smo izračunali omjer dimenzija teksturalne mreže i dimenzija terena, te smo vrijednosti pozicija u sceni pomnožili s tim omjerom.

4.1. Satelitska snimka

Dohvaćenu satelitsku snimku opisanu pred kraj poglavlja 3.1. postavili smo putem skripte kao teksturu za sloj naziva *MapLayer*, te mu postavili veličinu temeljem veličine stvorenog terena. Tako stvoren sloj ćemo pridružiti svakoj točki terena, te će nam on biti podloga za ostalo teksturiranje terena. Isječak koda koji prikazuje pridruživanje opisanog sloja terenu nalazi se na slici 4.1. Ovim smo načinom u varijabli *splatmapData* spremili koji će sloj biti primijenjen u svakoj točki teksturalne mreže. U tu ćemo varijablu i u idućim potpoglavljima spremati odabir sloja u pojedinim točkama koji će zapravo preslikati satelitsku snimku točnijim podacima. Kada smo prošli kroz sve podatke koje ćemo prikazivati slojevima pozivom ugrađene metode *SetAlphamaps(0, 0, splatmapData)* nad terenom primijenit ćemo odabrane slojeve na stvoreni teren. Na slici 4.2. prikazan je jedan primjer prikaza satelitskog sloja na terenu dobivenog pomoću preuzete satelitske slike.

```
//Prekrij cijeli teren teksturom satelitskog sloja
for (int y = 0; y < terrainData.alphamapHeight; y++)
{
    for (int x = 0; x < terrainData.alphamapWidth; x++)
    {
        float[] splatWeights = Constants.roadSurfaceOptions["map"];

        for (int i = 0; i < terrainData.alphamapLayers; i++)
        {
            splatmapData[x, y, i] = splatWeights[i];
        }
    }
}
```

Slika 4.1. Pridruživanje satelitskog sloja terenu

Slika 4.2. Prikaz satelitskog sloja na terenu

4.2. Površine

Iz OSM izvora podataka dobili smo poligonalne podatke koji predstavljaju neku vrstu podloge na terenu, kao što su građevinska zemljišta, travnate površine, asfaltirane plohe, dječja igrališta, zemljane površine, parkirne površine i ostale čiji je popis dostupan u privitku. Sve su te površine spremljene u pojedine liste objekata *MultiPolygon*.

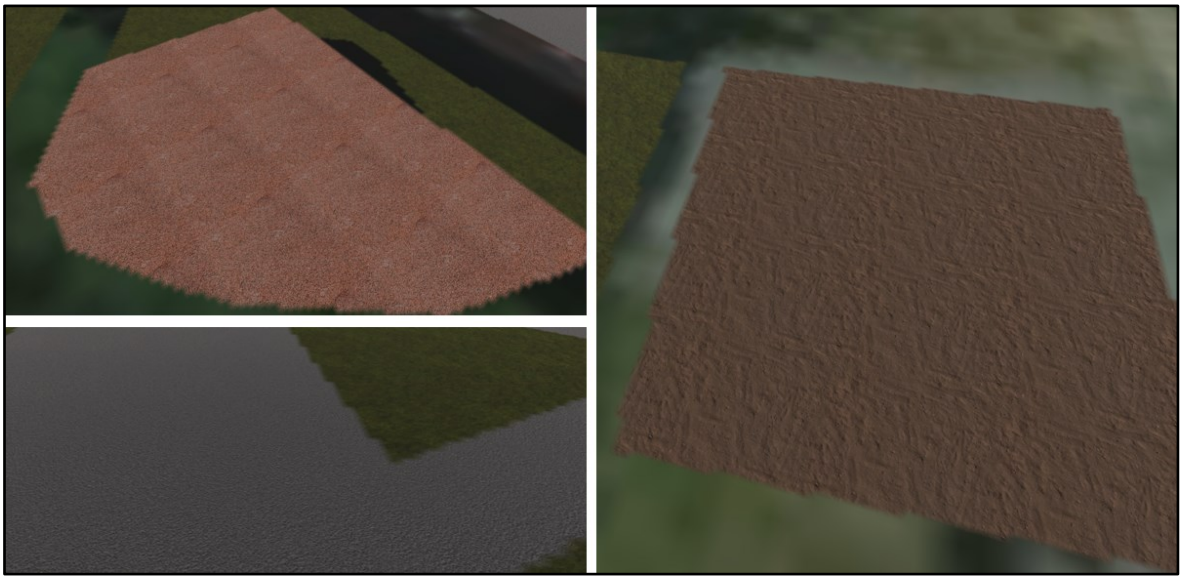
Vizualizacija navedenih površina na terenu ostvaruje se tako da se područje koje svaka od njih obuhvaća, teksturira pripadnim slojem koji je određen za prikaz te površine. Kako bismo to ostvarili prvo je potrebno proći kroz sve identifikatore koje imamo spremljene kao referentne čvorove u objektu *MultiPolygon*, te iz liste svih čvorova naći te čvorove i iz njih uzeti sve njihove lokacije na teksturalnoj mreži. Sad kada imamo sve lokacije na teksturalnoj mreži koje čine vrhove poligona koji predstavlja površinu načinit ćemo granični okvir (engl. Bounding Box). Granični okvir je najmanji pravokutnik unutar kojeg se nalaze sve točke iz nekog skupa, a to su u našem slučaju vrhovi poligona. Metoda koja prikazuje nalaženje graničnog okvira za 2D poligon nalazi se na slici 4.3.

```
//Metoda za traženje bounding boxa od niza točaka poligona 2D
public static List<float> getBoundingBox2D(List<Vector2> points)
{
    float maxX = float.MinValue;
    float minX = float.MaxValue;
    float maxY = float.MinValue;
    float minY = float.MaxValue;
    for (int i = 0; i < points.Count; ++i)
    {
        float currX = points[i].x;
        float currY = points[i].y;

        if (currX <= minX)
        {
            minX = currX;
        }
        if (currX >= maxX)
        {
            maxX = currX;
        }
        if (currY <= minY)
        {
            minY = currY;
        }
        if (currY >= maxY)
        {
            maxY = currY;
        }
    }
    List<float> boundingBox = new List<float>();
    boundingBox.Add(minX);
    boundingBox.Add(maxX);
    boundingBox.Add(minY);
    boundingBox.Add(maxY);
    return boundingBox;
}
```

Slika 4.3. Metoda za pronalazak graničnog okvira iz skupa 2D točaka

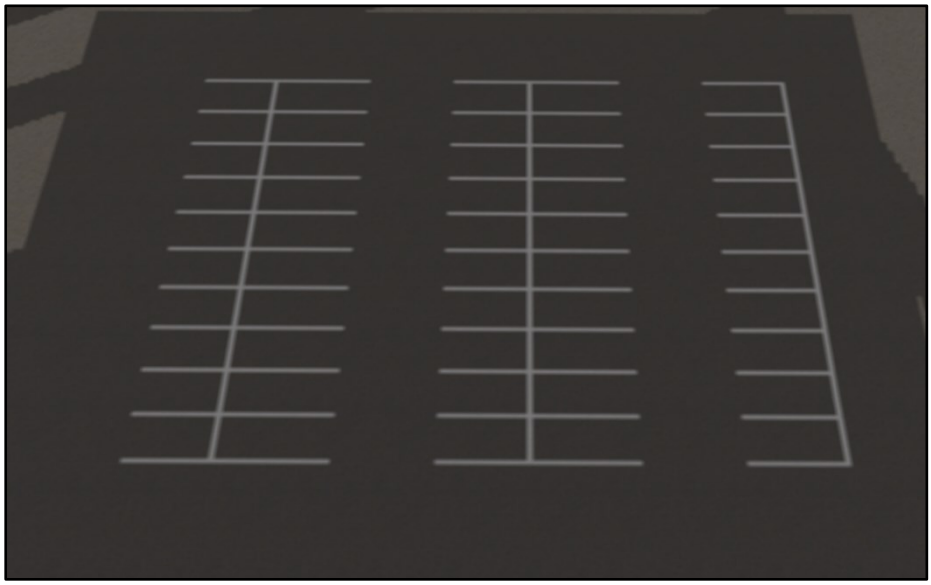
U idućem ćemo koraku proći kroz sve točke tog graničnog okvira i obavit ćemo provjeru nalazi li se točka unutar našeg poligona i unutar granica terena, te na temelju toga odrediti hoćemo li toj točki teksturalne mreže pridružiti zadani sloj površine. Na slici 4.4. nalazi se nekoliko isječaka prikaza površina na terenu teksturiranih na opisani način.

Metoda za ispitivanje nalazi li se točka unutar poligona preuzeta je s web stranice W. Randolpha Franklina, profesora na institutu Rensselaer Polytechnic Institute [11]. U tu je metodu još ugrađena na početku brza provjera nalazi li se točka unutar graničnog okvira kako bi se na temelju te informacije odmah došlo do odgovora nalazi li se točka unutar poligona.



Slika 4.4. Isječci prikaza površina na terenu: dječje igralište (gore lijevo), asfaltirana podloga (dolje lijevo), zemljana površina (desno)

Nešto kompliciraniji postupak teksturiranja odnosi se na vizualizaciju parkirnih površina, gdje je pored asfalta na površinu potrebno još postaviti bijele linije koje čine parkirna mjesta. Cijeli algoritam tog teksturiranja nećemo u detalje objašnjavati, jer se sastoji od već opisanih metoda. Ključno je samo izračunati koliko horizontalnih linija odnosno parkirnih odvojaka stane na tu površinu, obojiti ih te onda isto napraviti s vertikalnim linijama koje čine parkirna mjesta. Kako bismo mogli to napraviti zahtijevat ćemo pronalazak pravokutnika koji se cijeli nalazi unutar površine parkinga i poravnat je s koordinatnim osima s obzirom na to da teksturiramo po teksturalnoj mreži. Također morali smo proračunati i veličinu jedne pločice u teksturalnoj mreži kako bi dobili točan omjer parkirnog mjesta sa stvarnim svijetom, a to smo napravili tako da smo dimenzije terena podijelili s brojem 4096 koliko ima polja po svakoj osi mreže. Tim postupkom stvorili smo reprezentaciju parkirališta, a izgled jednog tako generiranog parkirališta vidljiv je na slici 4.5.



Slika 4.5. Primjer prikaza parkirališta

4.3. Ceste i putevi

Ceste i puteve koje smo kod obrade OSM podataka spremili u objekte tipa *Way* sada ćemo vizualizirati kao niz obojenih pločica na teksturalnoj mreži.

Na početku morat ćemo za svaki objekt odrediti širinu pripadne ceste odnosno puta. To ćemo ostvariti tako da prvo odredimo koliko traka ima određena cesta, ako smo taj podatak preuzeli iz OSM skupa podataka tada ćemo ga koristiti, u suprotnom ćemo uzeti pretpostavljeni broj traka ovisno o važnosti odnosno vrsti ceste čiji ćemo podatak sigurno dobiti iz OSM-a. Osim toga potreban nam je i podatak o širini jedne trake, no taj podatak ne postoji u OSM-u, pa ćemo opet uzeti pretpostavljenu vrijednost ovisno o važnosti ceste. Konačni podatak o širini ceste će biti jednak umnošku broja traka, širini jedne trake i faktoru skaliranja terena.

Kako objekt koji sadrži podatke o cestama ima spremljen skup referentnih čvorova koje čine stazu, ti su čvorovi neravnomjerno razmješteni duž toga puta, stoga je potrebno interpolirati vrijednosti između njih. Kako moramo interpolirati cjelobrojne vrijednosti između polja na teksturalnoj mreži koristit ćemo Bresenhamov algoritam [17]. Taj algoritam zapravo služi za iscrtavanje ravne linije u rasterskoj grafici, a skup točaka koji nam on daje za iscrtavanje linije između dvije točke bit će upravo točke koje nama nedostaju između pozicije dva referentna čvora na teksturalnoj mreži. Nakon toga odredit ćemo u kojem smjeru se proteže

dio ceste između dva čvora koji iscrtavamo, jeli riječ o smjeru bliže x ili bliže z osi. Kod koji prikazuje metodu koja određuje opisani smjer moguće je vidjeti na slici 4.6.

Tako dobiven smjer pomoći će nam pri određivanju koliko ćemo susjednih ploča mreže želimo teksturirati pri iscrtavanju nekog segmenta ceste. Konkretno to ćemo odrediti tako da dobivenu širinu ceste podijelimo s dužinom ili širinom jednog polja ovisno o osi za koju je određen smjer. Nakon toga ćemo za te točke postaviti sloj koji pripada određenoj vrsti ceste odnosno puta i po potrebi rubove ceste obojiti u bijelo.

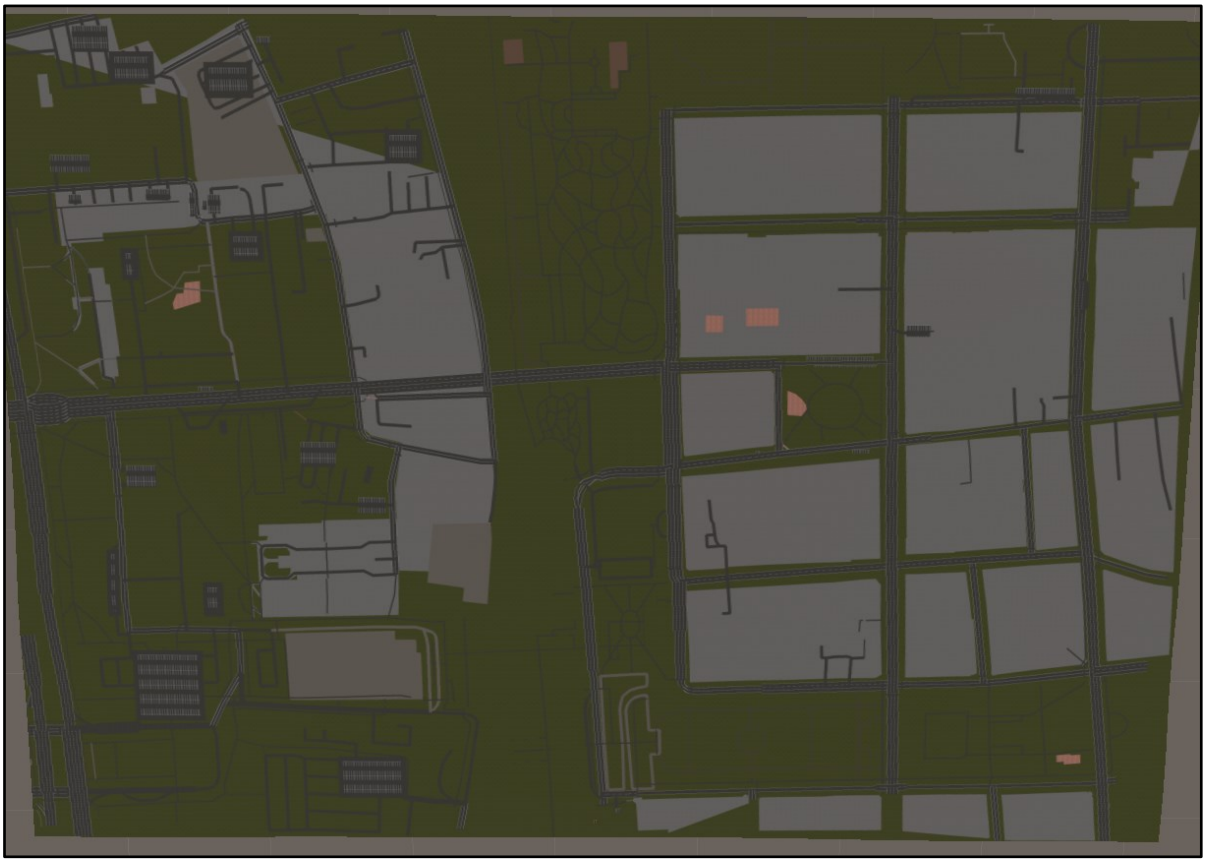
```
//Pronalazak smjera između dva čvora
public static string findDirection(Node current, Node prev)
{
    Vector3 currPos = current.position;
    Vector3 prevPos = prev.position;
    Vector3 dir = currPos - prevPos;

    float angle = Vector3.Angle(dir, new Vector3(1f, 0f, 0f));
    if (angle >= 45f && angle <= 135f)
    {
        return "z";
    }
    else
    {
        return "x";
    }
}
```

Slika 4.6. Metoda za određivanje smjera između dva čvora

Na kraju nam još preostaje iscrtati linije na cesti. Iscrtavanje crta se provodi tako da se prvo na temelju broja traka odredi broj isprekidanih uzdužnih linija, tako da je njihov broj za jedan manji od broja traka. One se onda iscrtavanju naizmjeničnom uporabom slojeva asfalta i bijele boje na jednolikom udaljenostima jedna od druge. Ako je cesta jednosmjerna tu postupak staje, a ako pak nije jedna od središnjih isprekidanih linija se iscrta potpuno u bijelo kako bi predstavljala punu liniju.

Rezultat ovako opisanog postupka prikazivanja cesta i puteva na terenu prikazan je na slici 4.7., dok je na slici 4.8. prikazan primjer cijelog terena pobojanog svim dosad navedenim teksturama iz četvrtog poglavlja bez teksture satelitske snimke.

Slika 4.7. Prikaz cesta i puteva na terenu



Slika 4.8. Primjer pobojanog terena teksturama za površine, ceste i puteve

5. Vizualizacija 3D OSM podataka

U prošlom smo poglavlju opisali koje smo skupove podataka iz OSM-a prikazali na terenu odnosno kao teksture na njemu u dvije dimenzije. U ovom ćemo pak poglavlju vidjeti kako smo reprezentirali ostale OSM podatke koji su prikladni za prikaz u tri dimenzije kao što su građevine, te objekti koji su točkaste, linijske i poligonalne građe.

5.1. Modeli objekata

Objekte u tri dimenzije smo dosad prikazivali samo stvaranjem jednostavnih poligonalnih 2D objekata, no to nam neće biti dovoljno da prikažemo realistično ostale objekte kao na primjer kante za smeće, klupe i hidrante. Stvaranje takvih modela nekim od dostupnih alata iziskuje mnogo vremena, stoga ćemo većinu modela koje smo upotrijebili za reprezentaciju objekata u aplikaciji preuzeti s nekih od web stranica koje sadrže veliki skup besplatnih 3D modela za korištenje. Web stranice koje smo koristili u tu svrhu su CGTrader [14], Free3D [15] i TurboSquid [16]. Preuzete modele ponekad je potrebno prilagoditi našem konkretnom slučaju uporabe, a u tu smo svrhu koristili neke od značajki programskog alata Blender. Gotovo uvijek smo koristili Blenderov ugrađen modifikator *Decimate.* Ovaj modifikator nam omogućuje smanjenje broja vrhova mreže objekta s minimalnim promjenama njegova oblika [18]. Ta funkcija nam je značajna jer su preuzeti modeli često prekompleksni i sadrže previše detalja koji nam nisu presudni u našoj prezentaciji, a narušavaju brzinu i ugodnost izvođenja aplikacije. Na kraju još treba spomenuti kako smo modele preuzimali u formatu .fbx ili .obj zbog kompatibilnosti s Blenderom i Unity-em. Poneke objekte koji su sastavljeni od jednostavnih dijelova smo i sami izradili, a to su: znak s imenom ulice i ograničenjem brzine, boćalište, sprave za vježbanje, kupola, prometni otok, nadvožnjak, nogostup, pješčanik, sat, itd.

Kasnije svi instancirani modeli u radu aplikacije su skalirani pripadnim koeficijentima kako bi odgovarali stvarnim dimenzijama i bili u ispravnom omjeru s drugim dijelovima prikaza u aplikaciji.

5.2. Stvaranje građevina

Prva stvar koja nam pada na pamet kada želimo napraviti vizualni prikaz karte u tri dimenzije je prikazati građevine koje se nalaze na nekom području kao trodimenzionalne objekte. Ideja je da ćemo bočne zidove građevina reprezentirati preko mreže (engl. Mesh) u Unity-u. To je najjednostavniji objekt koji se u Unity-u može stvoriti, te je za njegovo stvaranje potrebno predati samo listu vrhova i listu rasporeda vrhova po trokutima te po potrebi listu uv koordinata za teksture. Na bočne ćemo zidove zalijepiti odgovarajuću teksturu i dodati vrata te po potrebi stvoriti prozore. Nakon toga ćemo stvoriti krov te naposljetku prikazati detalje kao što su imena ustanova, kućanski brojevi i oznake ustanova. Sve informacije koje su nam potrebne za opisani postupak spremljene su u objekt tipa *Way* u kojem su sadržane pozicije točaka koje opisuju tlocrt zgrade i varijable s ostalim informacijama o zgradi.

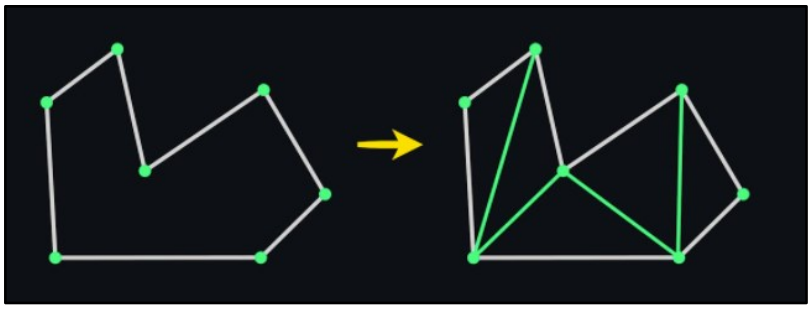
Za početak potrebno će biti odrediti visinu pojedine zgrade koju želimo prikazati. Tu informaciju ponekad imamo eksplicitno dobivenu iz OSM skupa podataka pa ju koristimo, a u drugom slučaju imamo tek broj katova, tada ćemo visinu aproksimirati prema idućoj formuli $visina = brojKatova * 3 + 1.5$. Često nemamo niti jednu niti drugu informaciju dostupnu pa ćemo pretpostaviti kako se radi o zgradi s tri kata te ćemo na temelju prethodne formule odrediti visinu.

5.2.1. Zidovi

Vizualni prikaz zidova bit će izabran kao jedan od sljedeće dvije opcije. Prva je da ćemo cijeli zid prekriti jednostavnom teksturom fasade i nasumično pridijeliti boju toj teksturi te ćemo nakon toga dodati objekte prozora, dok je druga ta da ćemo zalijepiti jednu od nekoliko predefiniranih tekstura koje već sadrži prikaz prozora pa nam oni onda nisu potrebni. Ovu drugu opciju ćemo odabrati u idućim situacijama: ako je zagrada viša od 20 metara, ako zgrada ima više od 6 katova ili ako je zgrada ima određenu ulogu kao što je hotel, škola, sveučilište, policija, ambasada, pošta, apartmanski blok, itd. Ako zgrada ne ispunjava jedan od navedenih kriterija prikaz bočnih zidova se odvija prema opisu prve opcije gdje se boja određuje nasumično prema preddefiniranoj raspodjeli tako da se neke boje češće pojavljuju, a neke rjeđe.

Kako bismo prikazali bočne zidove potrebno je na početku izračunati pozicije vrhova svakog zida. To ćemo ostvariti tako da ćemo iterirati kroz sve točke tlocrta te ćemo uzeti svake dvije susjedne točke, duplicirati ih i dodati im odmak po y osi za izračunatu visinu zgrade. Time

ćemo dobiti sve četiri točke koje su nam potrebne za opis jednog zida. Osim točaka kako bi stvorili točan *Mesh* objekt potrebno je odrediti raspored vrhova po trokutima. Naime svaki je *Mesh* objekt sadržan od sastavnica koje se nazivaju trokuti, pa ako se radi o poligonu s puno točaka njega je potrebno razdvojiti na trokute, te je Unity-u kod stvaranja potrebno predati polje koje sadrži indekse vrhova koji određuju na koji način će trokuti biti iscrtani. Postupak razdvajanja poligona na trokute se naziva triangulacija, a ilustracija jednog primjera je prikazana na slici 5.1.



Slika 5.1. Primjer triangulacije poligona

Kako su zidovi uvijek sastavljeni od četiri točke jednostavno je napraviti njihovu triangulaciju. Kako triangulacija ovisi o tome jesu li vrhovi zapisani u smjeru kazaljke na satu ili obrnuto od tog smjera, potrebno ju je odrediti. To ćemo ostvariti tako što ćemo najprije izračunati predznačnu površinu poligona, a metoda koja to računa je prikazana na slici 5.2. Ako je dobivena predznačna površina manja od nule tada su vrhovi u poligonu zadani u smjeru kazaljke na satu, a u suprotnom obrnuto od smjera kazaljke na satu.

```
//Metoda za izračun površine poligona
public static double signedAreaOfPolygon(List<Vector3> vertices)
{
    double sum = 0;
    if (vertices.Count < 3)
    {
        return 0;
    }
    for (int k = 0; k < vertices.Count; ++k)
    {
        Vector2 v1 = new Vector2(vertices[k].x, vertices[k].z);
        Vector2 v2 = new Vector2(vertices[(k + 1) % vertices.Count].x, vertices[(k + 1) % vertices.Count].z);
        double crossProd = CrossProduct2D(v1, v2);
        sum += crossProd;
    }
    return sum / 2;
}
```

Slika 5.2. Metoda koja izračunava predznačnu površinu poligona

Ovisno o dobivenom smjeru uzet ćemo jedan od dva rasporeda indeksa vrhova pri zadavanju trokuta objekta *Mesh* za zidove. Na slici 5.3. prikazani su rasporedi vrhova za bočni zid čije

su točke u smjeru kazaljke na satu i na dnu za one koju su suprotno od smjera kazaljke na satu.

```
public static int[] quadTrianglesClockwise = new int[]
    {
        0,1,2,
        2,3,0
    };
public static int[] quadTrianglesCounterClockwise = new int[]
    {
        0,3,2,
        2,1,0
    };
```

Slika 5.3. Raspored vrhova za bočne zidove

Sada kada imamo sve potrebne podatke za stvaranje *Mesh* objekta koji predstavlja jedan zid građevine možemo ga stvoriti, a način na koji je to ostvareno je prikazan na slici 5.4. gdje je *MeshObject* referenca na predložak praznog objekta, a *vertices* i *triangles* strukture koje čuvaju vrhove odnosno njihov raspored.

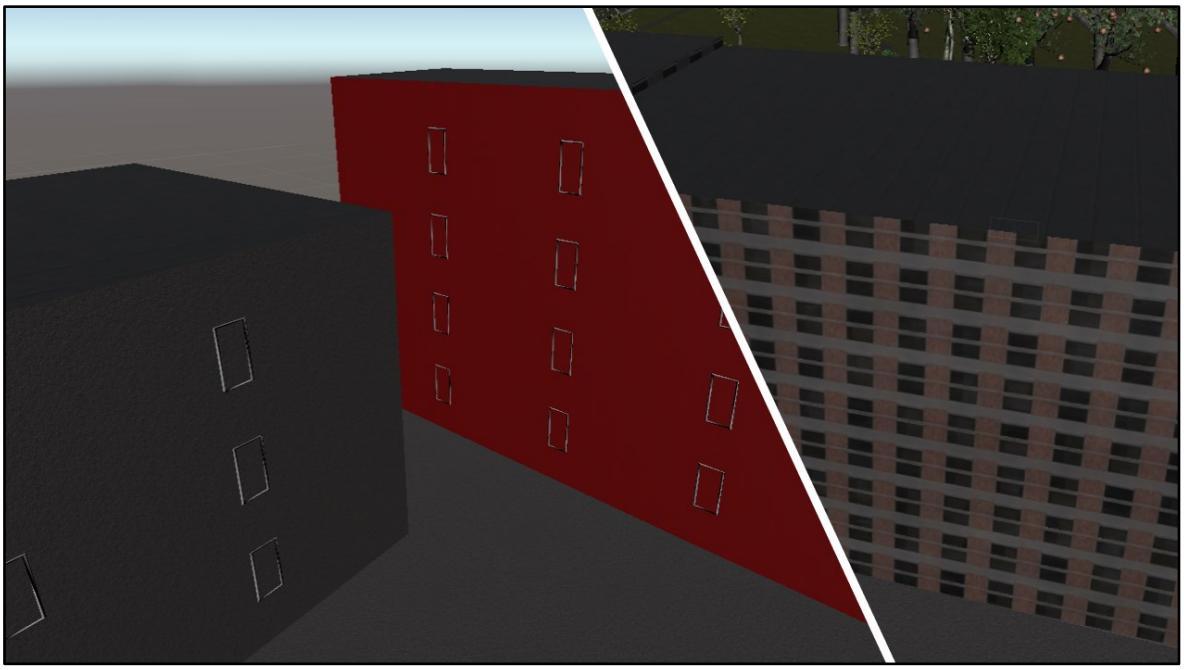
```
//Stvaranje mesh objekta
GameObject meshObj = Instantiate(MeshObject);
Mesh mesh = new Mesh();
meshObj.GetComponent<MeshFilter>().mesh = mesh;
mesh.Clear();
mesh.vertices = vertices;
mesh.triangles = triangles;
mesh.RecalculateBounds();
mesh.RecalculateNormals();
```

Slika 5.4. Stvaranje Mesh objekta

Tako stvorenom *Mesh* objektu potrebno je još samo dodati teksturu na jedan od prethodno opisanih načina. Tekstura se *Mesh* objektu dodaje preko materijala sljedećom naredbom *mesh.GetComponent<MeshRenderer>().material=mat*, gdje je varijabla *mat* tipa *Material* i dohvaća se iz direktorija resources metodom *Resources.Load('NazivMaterijala')*. Nakon toga moguće je još promijeniti boju materijala naredbom *mesh.GetComponent<Renderer>().material.SetColor("_Color", color)*, gdje je varijabla *color* tipa *Color* i predstavlja odabranu boju.

Osim samog zida i teksture na njemu potrebno je još dodati i prozore, ako ih pridodana tekstura već ne sadrži. Procedura koja to obavlja na temelju dužine zida računa koliko će prozora biti po dužini i koliko će biti takvih redova prozora na temelju visine zida. S tim se informacijama instanciraju objekti prozora po jednolikom rasporedu te im se postavlja rotacija na temelju vektora normale zida. Vektor normale zida dobiven je po stvaranju *Mesh* objekta, a rotacija prozora se postavlja metodom *Quaternion.LookRotation(normala)*. Na

slici 5.5. nalazi se prikaz dviju građevina iz aplikacije stvorene dvama prethodno opisanim postupcima. Na lijevom dijelu slike nalazi se građevina stvorena postupkom primjene teksture fasade i dodavanjem prozora, a na desnom dijelu slike se nalazi građevina s teksturom koja već sadrži prozore.



Slika 5.5. Prikaz građevina u aplikaciji

Pored prozora potrebno je još dodati i ulazna vrata na zgradama. Potrebno je odabrati na koji ćemo zid staviti vrata, a to ćemo odrediti na temelju podatka o veličini zida i blizini ceste. Pa tako iz skupa svih zidova uzet ćemo dva najveća zida i dva zida najbliža cesti, te ćemo na temelju algoritma prikazanog na slici 5.6. odabrati željeni zid na koji ćemo postaviti vrata.

```
//Metoda za određivanje glavnog zida (koja sadrži vrata)
public static int getDoorIndex(int largestInd1, int largestInd2, int closestInd1, int closestInd2)
{
    if (largestInd1 == closestInd1)
    {
        return largestInd1;
    }
    else if (closestInd1 == largestInd2)
    {
        return closestInd1;
    }
    else
    {
        return largestInd1;
    }

}
```

Slika 5.6. Metoda za određivanje glavnog zida zgrade

Kada smo odredili na koji ćemo zid staviti vrata, objekt koji predstavlja vrata ćemo jednostavno instancirati na mjestu centroida tog zida i pomaknuti po y osi na visinu terena u

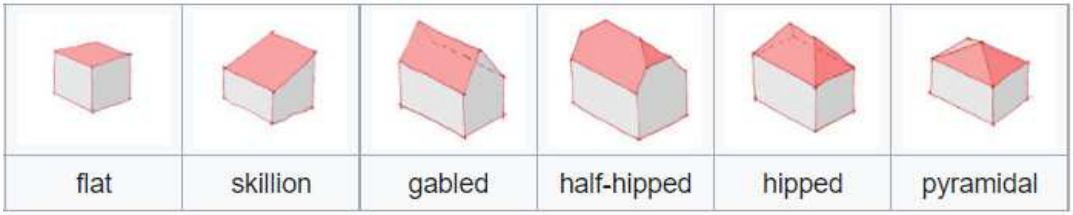
toj točci, a rotirat ćemo ih u smjeru normale kao i kod prozora. U aplikaciju su uneseni nekoliko reprezentacija vrata, a njihov odabir ovisi o tipu građevine.

Prilikom stvaranja vrata stvaraju se i reprezentacije ostalih informacija o zgradi ako ih poznajemo. Te informacije se odnose na kućanski broj zgrade, naziv građevine i logotip odnosno oznaku određenih ustanova. Oznake s logotipima ustanova su unaprijed unijete u aplikaciju i odabiru se na temelju tipa zgrade dobivenog iz OSM izvora podataka. Pa su tako neke od oznaka u sustavu one za hotel, sveučilište, školu, poštu, ambasadu, kazalište, javni wc, crkvu, policiju, itd. Pripadnu oznaku instanciramo pored ulaznih vrata i orijentiramo pomoću normale zida. Što se tiče kućnih brojeva i naziva zgrada tu na isti način instanciramo pločicu s imenom odnosno brojem kao i oznaku ustanove, te na isto mjesto, malo odmaknuto od pločice, stvaramo još objekt koji sadrži komponentu tipa *TextMeshPro* čijoj ćemo varijabli *text* dati vrijednost broja ili naziva kako bi informacija bila vidljiva na zgradi. Na slici 5.7. je vidljiv rezultat ovakvog stvaranja vrata i dodatnih informacija o zgradama na jednom primjeru iz aplikacije.

Slika 5.7. Primjer prikaza ulaznih vrata i dodatnih informacija na građevini

5.2.2. Krovovi

Stvaranje krovova podijelit ćemo na dva slučaja. U slučaju da je broj točaka tlocrta zgrade jednak četiri stvorit ćemo krov prema informaciji o njegovom obliku iz OSM izvora podataka. Na slici 5.8. prikazani su oblici krovova za koje smo napravili prikladnu reprezentaciju u aplikaciji [12].



Slika 5.8. Oblici krovova uzeti u obzir iz OSM izvora podataka

Ako je izgled krova nepoznat ili nije jedan od navedenih uzet ćemo nasumično jedan od oblika sa slike 5.8., dok će se oblik „half-hipped“ prikazati kao oblik „hipped“. Sve navedene oblike krovova ćemo stvoriti tako da ćemo odrediti sve potrebne točke krova (gdje se bridovi sijeku prema slici 5.8.), te ćemo instancirati četverokutne i trokutne *Mesh* objekte s tim točkama. Raspored vrhova za stvaranje *Mesh* objekta s tri točke iznosi $[0, 1, 2]$ ili $[2, 0, 1]$ ovisno o orijentaciji vrhova. Što se tiče teksture krovova ravan krov ima svoju teksturu dok ostali oblici krovova imaju teksturu crijepa.

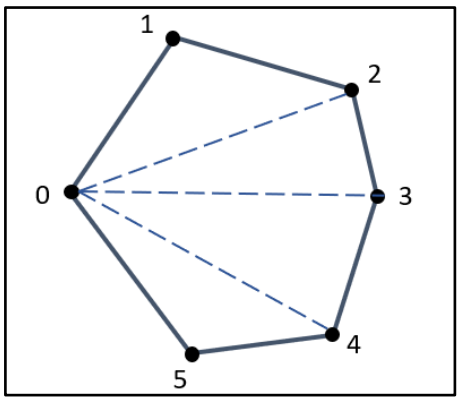
Krov čiji tlocrt ima više od četiri točke ćemo prikazati ili kao ravan ili kao piramidalan. Piramidalan krov ćemo odabrati u slučaju da je opisani tlocrt konveksan i ako iz izvora podataka nije eksplicitno zapisano kako je krov ravan. U ostalim slučajevima krov će biti ravan. Piramidalni krov ćemo ostvariti tako da uzmemo svaki susjedni par vrhova tlocrta i njima dodamo centroid svih točaka tlocrta i na temelju njih napravimo *Mesh* objekt.

Ravan krov napravit ćemo na način kako smo kreirali i zidove, ali ćemo za postupak triangulacije koristiti posebnu metodu s obzirom na to da je teško statički stvoriti raspored vrhova za proizvoljno velik broj točaka. Razlikovat ćemo dva slučaja, prvi kada je poligon koji želimo triangulirati konveksan, a drugi kada je konkavan. Metoda koja ispituje konveksnost poligona prikazana je na slici 5.9. Prikaza metoda sadrži varijablu *semi* koja označava hoćemo li neke poligone ipak proglasiti konveksnim iako to striktno gledajući matematički nisu. Najčešće su to poligoni koji sadrže točke koje čine ravnu liniju odnosno kut između njih je 180 stupnjeva, no u procesu obrade podataka mogu nastati sitnija odstupanja od tog kuta te ćemo tada dozvoliti malu grešku kod ispitivanja konveksnosti odnosno preskočiti takve točke pri ispitivanju.

```
//Metoda za određivanje konveksnosti poligona
public static bool isPolygonConvex(List<Vector3> vertices, bool semi)
{
    bool negative = false;
    bool positive = false;
    for (int k = 0; k < vertices.Count; ++k)
    {
        Vector2 v1 = new Vector2(vertices[k].x, vertices[k].z);
        Vector2 v2 = new Vector2(vertices[(k + 1) % vertices.Count].x, vertices[(k + 1) % vertices.Count].z);
        Vector2 v3 = new Vector2(vertices[(k + 2) % vertices.Count].x, vertices[(k + 2) % vertices.Count].z);
        Vector2 dir1 = v1 - v2;
        Vector2 dir2 = v3 - v2:

        double crossProd = CrossProduct2D(dir1.normalized, dir2.normalized);
        if (semi)
        {
            if (Math.Abs(crossProd) < 0.015) {
                continue;
            }
        }
        if (crossProd < 0)
        {
            negative = true;
        }
        else if (crossProd > 0)
        {
            positive = true;
        }
        if (negative && positive)
        {
            return false;
        }
    }
    return true;
}
```

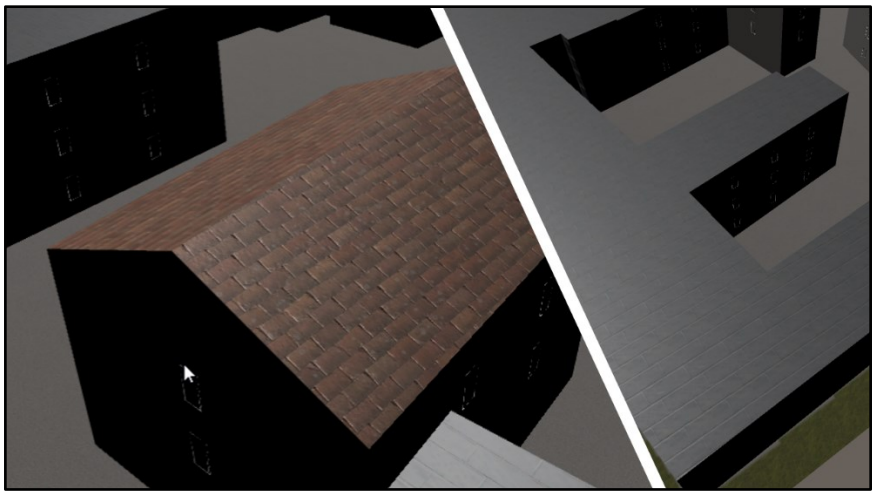
Slika 5.9. Metoda za ispitivanje konveksnosti poligona

Kada je poligon koji predstavlja krov konveksan koristit ćemo metodu triangulacije koju nazivamo triangulacija oblika ventilatora (engl. Fan triangulation). Takva triangulacija se temelji na tome da iteriramo kroz indekse vrhova poligona počevši od trećeg vrha, te u listu rasporeda vrhova po trokutima u k-toj iteraciji dodajemo indekse 0, k i $k-1$ čiji redoslijed ovisi o smjeru vrhova u poligonu. Na taj način za svaki poligon opisan s k točaka dobivamo raspored vrhova za $k-2$ trokuta. Ilustracija ovog načina triangulacije vidljiva je na slici 5.10.



Slika 5.10. Ilustracija triangulacije oblika ventilatora poligona

Ako je pak poligon koji predstavlja krov konkavan, triangulaciju ćemo provesti pozivom metode *Triangulate()* unutar skripte *Triangulator.cs* koja je preuzeta s web stranice Unify Community Wiki [13]. Na slici 5.11. nalaze se prikazi iz aplikacije za stvorene krovove prethodno opisanim metodama.



Slika 5.11. Prikaz krovova građevina iz aplikacije

5.3. Prikaz točkastih podataka

Podaci koje smo izvukli iz OSM skupa podataka, a bili su reprezentirani elementom *Node* sada ćemo prikazati na našoj karti. To su većinom različiti ulični objekti kao npr. kante za smeće, bankomati i hidranti. Za ovakav točkasti tip podataka posjedujemo jedino njegovu lokaciju na karti i njegov tip odnosno ono što reprezentira.

S time na umu, zadatak stvaranja ovakvih objekata bit će prilično lak. Ono što moramo učiniti je proći kroz točkaste podatke i naći tipove koje naša aplikacija podržava, odnosno za koje posjedujemo vizualnu reprezentaciju. Tada ćemo stvoriti pripadni objekt u virtualnoj sceni na za to predviđenu poziciju i skalirati ga po koordinatnim osima kako bi dimenzije bile u mjerilu. Osim toga za objekte koji stoje pored ceste kao što su telefon, parkirni automat, rasvjeta, spomenici, itd. bit će potrebno odrediti njihovu orijentaciju tako da gledaju prema najbližoj cesti kako bismo dobili što realističniju reprezentaciju. To ćemo odrediti tako što ćemo naći točku na cesti koja je najbliža lokaciji objekta, a te su točke spremane tijekom teksturiranja terena cestama. Nakon toga rotaciju objekta postavimo putem sljedeće naredbe *Quaternion.LookRotation(closestRoad-object.localPosition, Vector3.forward)*. Međutim postoje i neki objekti kao što su prometni znakovi koji stoje uz cestu, ali su

orijentirani duž ceste. Takve ćemo objekte staviti na poziciju najbliže rubne točke ceste, te ćemo naći dvije najbliže rubne točke ceste i pomoću njih postaviti samo y komponentu rotacije koju ćemo dobiti pomoću naredbe *Quaternion.LookRotation(closestPoint2–closestPoint1, Vector3.forward)*.

Drugi dio prikaza točkastih objekata je stvaranje znakova na cesti za jednosmjernu ulicu, ograničenje brzine i naziv ulice. Ovi podaci nemaju točnu lokaciju već su ugrađeni u podatke o cestama. Stoga ćemo za referentnu lokaciju uzeti čvor koji se nalazi u sredini skupa čvorova koje čine neku cestu. Od tog ćemo čvora naći najbližu rubnu točku u čijoj uskoj okolini se već ne nalazi neki točkasti objekt i na njenu lokaciju stvoriti pripadni znak. Taj ćemo znak rotirati na isti način kao i prethodne znakove i dodati mu natpis ako je potrebno (broj za ograničenje brzine ili naziv ulice) kao i za natpise kod građevina.

Na kraju ćemo još dodati uličnu rasvjetu čije podatke ne posjedujemo, ali ćemo ju dodati po svim cestama tako da ćemo ju stvoriti u razmaku od svakih nekoliko rubnih točaka ceste. Kod važnijih i većih cesta to će biti reprezentirano velikom uličnom lampom, a kod manjih malom.

Na slici 5.12. prikazani su isječci iz aplikacije koji prikazuju neke od opisanih točkastih objekata u ovome potpoglavlju.

Slika 5.12. Isječci prikaza točkastih objekata iz aplikacije

5.4. Prikaz linijskih podataka

Nakon što smo prikazali kako reprezentiramo podatke tipa *Node* iz OSM izvora podataka, sada ćemo pogledati kako ćemo prikazati podatke koji su dobiveni iz elemenata staza. Prikaz ovih podataka ćemo podijeliti u dvije kategorije. U prvoj se nalaze oni podaci koji predstavljaju ograde, pruge i nogostupe o čime ćemo sada nešto više reći, dok se u onoj drugoj nalaze razni objekti čija je lokacija predstavljena poligonom, a o njima je moguće nešto više pročitati u poglavlju 5.5.

Kada je riječ o ogradnim objektima na raspolaganju ćemo imati listu pozicija duž kojih se proteže ograda te njezin tip. Postupak stvaranja ograde kreće tako da između svake pozicije ograde računamo udaljenost, te na temelju nje i dužine jednog segmenta ograde određujemo broj međutočaka koje ćemo naći između dvije pozicije. S tim informacijama i pozivom metode *Lerp* koja je prikazana na slici 5.13. dobit ćemo listu međutočaka od kojih svaka predstavlja poziciju gdje treba stvoriti jedan segment ograde. Sve te međutočke spremit ćemo u jednu listu kroz koju ćemo kasnije iterirati. U svakoj iteraciji uzet ćemo trenutnu i sljedeću točku te ćemo segment ograde stvoriti na poziciju centroida te dvije točke, dok ćemo x i y komponentu rotacije postaviti pomoću naredbe *Quaternion.LookRotation(currentPosition – nextPosition, Vector3.forward)* kako bi usmjerili segment duž puta. Treba naglasiti kako posjedujemo podatke o pozicijama na kojima postoje ogradna vrata. Pa tako ako prepoznamo kako se u neposrednoj blizini točke nalazi pozicija ogradnih vrata, preskočit ćemo dvije iteracije stvaranja ogradnog objekta i na to mjesto stvoriti ogradna vrata.

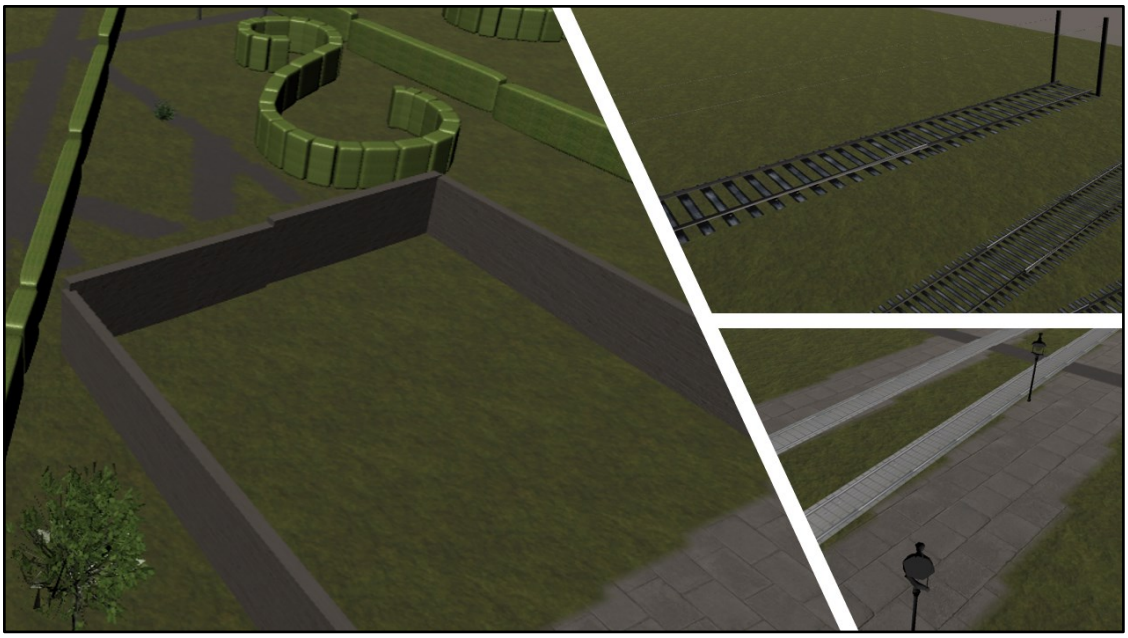
```
//Metoda linearne interpolacije između dvije točke
public static List<Vector2> Lerp(Vector3 start, Vector3 end, int count)
{
    List<Vector2> pointsArray = new List<Vector2>();
    for (float t = 0; t <= count; t++)
    {
        float T = (float)(t / count);

        Vector3 interpolated = Vector3.Lerp(start, end, T);

        pointsArray.Add(new Vector2(interpolated.x, interpolated.z));

    }
    return pointsArray;
}
```

Slika 5.13. Metoda linearne interpolacije između dvije točke

Ovim postupkom još stvaramo tramvajsku prugu, nogostupe i platforme za stajališta u našoj sceni, no taj je postupak stvaranja identičan onom opisanom za ogradne objekte.

Male razlike u postupku vidimo kod stvaranja željezničke pruge. Za nju posjedujemo još informacije o njezinoj širini, broju traka i elektrifikaciji. Razlike u postupku očitujemo u tome što na temelju podatka o broju traka stvaramo toliki broj paralelnih pruga duž staze, a podatak o elektrifikaciji određuje hoćemo li koristiti model pruge s električnim vodovima iznad nje ili nećemo.

Na slici 5.14. prikazani su isječci iz aplikacije koji prikazuju neke od opisanih linijskih objekata u ovome potpoglavlju.



Slika 5.14. Isječci prikaza linijskih objekata iz aplikacije

5.5. Prikaz poligonalnih podataka

Osim linijskih podataka u podatke koje smo dobili iz elemenata staza OSM skupa podataka spadaju i poligonalni podaci. Razlika između ova dva tipa podatka je ta da poligonalni podaci tvore neku zatvorenu krivulju odnosno služe za predstavljanje površine nekog oblika. Međutim ovdje nije riječ o površinama kao podlozi koje smo opisali u poglavlju 4.2., već je riječ samo o poziciji koja je opisana poligonom. Što upućuje na to kako ovi podaci svejedno najčešće služe za spremanje informacije o postojanju nekog objekta samo njegova pozicija nije određena jednom geografskom točkom nego skupom njih.

Za početak ćemo opisati najjednostavniju grupu podataka, a to su oni kod kojih je potrebno stvoriti objekt koji je reprezentacija podatka na mjesto poligona. Među takvim se podacima

nalaze košarkaški tereni, boćališta, kućice s nadstrešnicom i benzinske postaje. Generalni postupak kako ćemo te podatke predstaviti počinje tako što ćemo iz skupa pozicija poligona izračunati njihov centroid, a metoda koja ga računa je prikazana na slici 5.15. Nakon izračunatog centroida, aproksimirat ćemo u kojem smjeru gleda poligon. To ćemo učiniti tako da ćemo pronaći najdužu stranicu zadanog poligona odnosno dvije točke koje ju čine te odrediti smjer metodom *Quaternion.LookRotation(secondPoint-firstPoint)*. Ovim postupkom uz dodatak od 90 stupnjeva y komponenti rotacije dobit ćemo orijentaciju objekta kojeg trebamo stvoriti u smjeru najduže stranice poligona. Na mjesto centroida i s uređenom rotacijom ćemo stvoriti objekt koji predstavlja pojedini podatak.

```
//Metoda za traženje centroida iz niza točaka
public static Vector3 getCentroid(Vector3[] points)
{
    int len = points.Length;
    float sumX = 0;
    float sumY = 0;
    float sumZ = 0;
    for (int i = 0; i < len; ++i)
    {
        sumX += points[i].x;
        sumY += points[i].y;
        sumZ += points[i].z;
    }
    Vector3 centroid = new Vector3(sumX / len, sumY / len, sumZ / len);
    return centroid;
}
```

Slika 5.15. Metoda za računanje centroida točaka

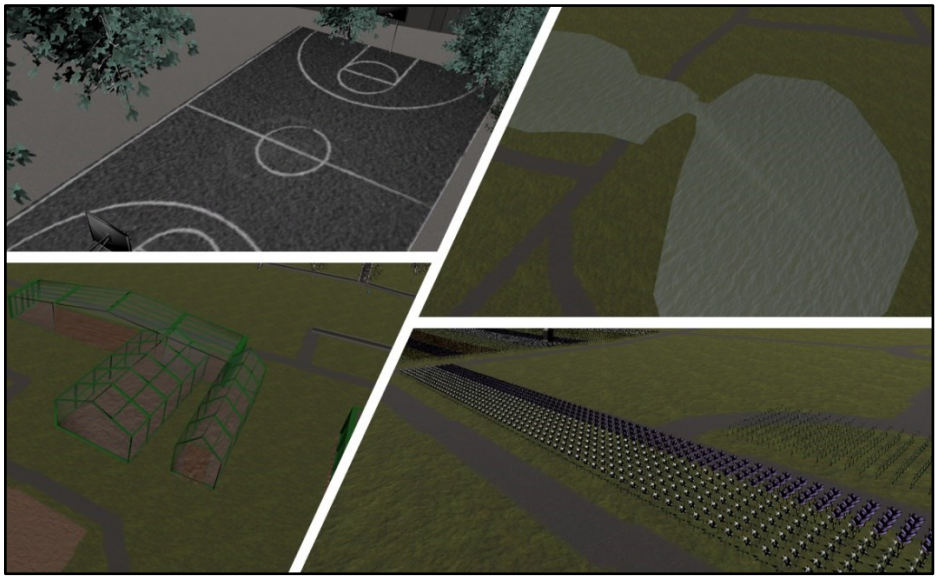
Malu razliku u postupku primjećujemo kod stvaranja staklenika, autopraonice, nadstrešnice i nasipa. Kod takvih objekata prvo tražimo granični okvir točaka koje predstavljaju poligon te onda računamo udaljenosti oba brida tog okvira i na temelju tih udaljenosti skaliramo stvoreni objekt po x i z osima. Kod stvaranja mostova odnosno nadvožnjaka je sličan ideja postupaka, ali je izvedba nešto kompliciranija jer zahtjeva poravnavanje s obližnjom cestom kako bi se dobio dojam nadvožnjaka kao produžetka ceste. Također za reprezentaciju mosta razlikujemo dva objekta, jedan koji predstavlja nadvožnjak ceste, a drugi most za pješake.

Kada je riječ o podacima koji opisuju parking za bicikle, cvjetno polje ili polje sadnica tada ćemo na površinu opisanu točkama poligona stvoriti više odgovarajućih objekata, odnosno skup stalaka za bicikle, skup cvjetova ili skup sadnica po nekom prostornom rasporedu. Prvo ćemo odrediti granični okvir za skup točaka koji opisuju poligon, te ćemo u ravnini x i z osi stvarati pojedine objekte uz određeni razmak i provjeru nalazi li se pozicija na kojoj želimo stvoriti objekt unutar poligona.

Posebnu proceduru reprezentacije imaju podaci koji predstavljaju neku vodenu površinu. Takve ćemo podatke prikazati pomoću teksturiranog *Mesh* objekta odnosno ravne plohe. Taj

Mesh objekt ćemo stvoriti na isti način na koji smo stvarali ravne krovove opisane s više od četiri točke iz poglavlja 5.2.2.

Naposljetku nam je još ostao zadatak napraviti reprezentaciju sportske dvorane. To ćemo učiniti na isti način kako smo radili prikaz građevina. Visinu dvorane ćemo postaviti na 6 metara kao pretpostavljenu, dok će krov biti ravan s posebnom teksturom. Osim toga na centroid krova ćemo postaviti objekt koji predstavlja natpis odnosno oznaku da se radi o sportskom objektu. Na slici 5.16. prikazani su isječci iz aplikacije koji prikazuju neke od opisanih poligonalnih objekata u ovome potpoglavlju.



Slika 5.16. Isječci prikaza poligonalnih objekata iz aplikacije

6. Vizualizacija GIS Zrinjevac podataka

Nakon što smo kroz prethodna poglavlja opisali kako smo ostvarili reprezentaciju svih podataka dobivenih iz OSM izvora podataka, preostaje nam još napraviti reprezentaciju podataka koje smo dobili iz GIS Zrinjevac izvora podataka. Postupci reprezentacije bit će slični kao oni opisani za OSM podatke, a u idućim ćemo potpoglavljima redom opisati kako smo prikazivali stabla, grmlje i javnu urbanu opremu.

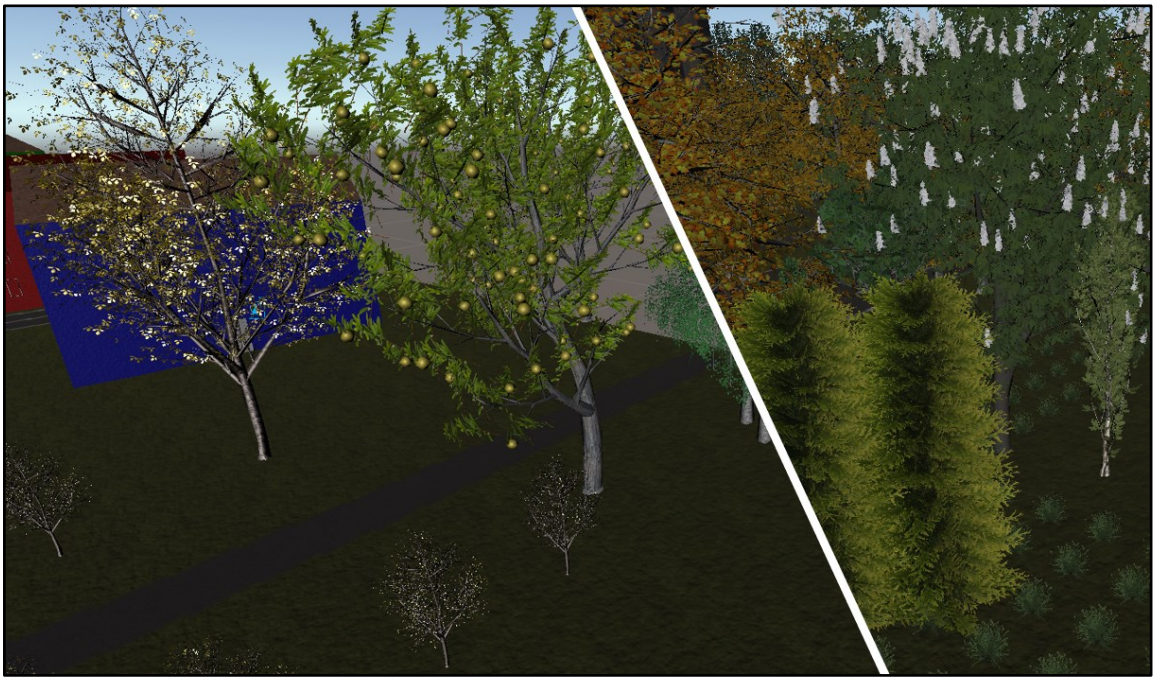
6.1. Prikaz stabala

Strukture koje čuvaju informacije o stablima raspolažu s podacima o visini stabla, vrsti stabla i poziciji stabla u sceni gdje se ono nalazi. U aplikaciju su uneseni modeli za 20-ak vrsta različitih stabala, a to su: breza, platana, čempres, kesten, hrast, bor, cedar, grab, javor, klen, smreka, omorika, jasen, jablan, topola, brijest, lijeska, tuja, breskva, jabuka i kruška. Kada se pak dogodi da za neku vrstu stabla ne posjedujemo model kojim ju možemo reprezentirati tada ćemo slučajnim odabirom uzeti neki model iz skupa modela koje posjedujemo.

Postupak stvaranja pojedinog stabla je jako jednostavan te je isječak koda koji ga prikazuje vidljiv na slici 6.1. Sve što je potrebno učiniti je postaviti poziciju objekta na lokaciju na karti i temeljem dobivene visine skalirati objekt. Prije toga ćemo svaki objekt skalirati još jednom zasebnom konstantom iz razloga kako bismo svaki od modela normalizirali na dimenziju od jednog metra. Na slici 6.2. može se vidjeti primjer prikaza stvorenih stabala u aplikaciji.

```
GameObject nodeObj = Instantiate(breza);
Transform nodeObjT = nodeObj.transform;
float scaleFactor = height * 10f * (float)terrainScaleFactor;
nodeObjT.localScale = new Vector3(scaleFactor, scaleFactor, scaleFactor);
nodeObjT.localPosition = node.position;
```

Slika 6.1. Isječak koda koji prikazuje stvaranje objekta stabla



Slika 6.2. Prikaz stabala u aplikaciji

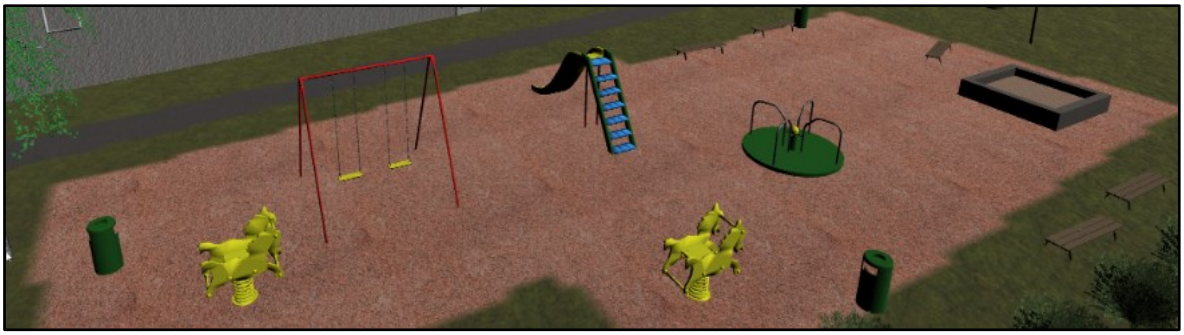
6.2. Prikaz grmlja

Kako smo već naveli u opisu obrade GIS Zrinjevac izvora podataka podaci o grmljima su zadani isključivo sa skupom pozicija u sceni koje čine poligon. Kako su takvi podaci prilično šturi morat ćemo sami osmisliti kako na najbolji način reprezentirati grmlje. Za reprezentaciju tih podataka odlučili smo se za stvaranje pojedinih objekata koji predstavljaju zaseban grm po nekom prostornom rasporedu unutar zadanog poligona. To smo učinili tako da smo prvo pomoću zadanih točaka poligona načinili granični okvir. Onda smo iterirali kroz točke tog graničnog okvira te s razmacima od dva metra te provjeravali nalazi li se trenutna pozicija unutar poligona. Ukoliko se zadana pozicija nalazi unutar poligona na to mjesto stvorit ćemo objekt grma na isti način kako je pokazano kod stvaranja stabala u poglavlju 6.1. Ako se dogodi da je granični okvir premali i onda ne dođe do stvaranja niti jednog grma na području poligona, tada je taj slučaj zabilježen i na poziciju centroida poligona se stvori jedan objekt grma. Na slici 6.3. prikazan je primjer reprezentacije grmlja u virtualnoj sceni.

Slika 6.3. Reprezentacija grmlja u sceni

6.3. Prikaz javne urbane opreme

Kako podaci o javnoj urbanoj opremi sadrže jedino lokaciju pojedinog komada opreme u sceni i tip pojedinog komada opreme način na koji ćemo ih prikazivati bit će skoro identičan onome kod prikaza stabala. Tipovi javne urbane opreme čija reprezentacija postoji u aplikaciji su: klupa s naslonom, klupa bez naslona, košarica za smeće, stol, fontana, tobogan, vrtuljak, ljuljačka, igračka na opruzi, penjalica, pješčanik, klackalica i sprava za vježbanje. Razlika u odnosu na stvaranje stabala je ta što ovdje nemamo nikakav podatak o dimenziji pojedinog objekta pa će ga trebati skalirati prema dimenzijama iz stvarnog svijeta. Osim toga klupe ćemo još orijentirati prema najbližoj cesti kako smo to već radili i s OSM točkastim podacima radi većeg realizma. Na slici 6.4. prikazan je vizualni prikaz opisanih objekata unutar aplikacije.



Slika 6.4. Prikaz javne urbane opreme u sceni

7. Analiza i rezultati izvođenja

Kako bismo analizirani izvođenje i rad stvorene aplikacije u sklopu ovoga rada provest ćemo više različitih testiranja koji ciljaju različite aspekte izvođenja aplikacije. Kako rad same aplikacije možemo podijeliti u dvije faze, tako ćemo i analizu raspodijeliti u dvije skupine. U prvu fazu spadaju svi dijelovi izvođenja aplikacije koji su zaduženi za dohvat podataka, obradu podataka, stvaranje terena i objekata odnosno sav posao što aplikacija treba izvesti kroz skripte prije samog prikaza naše trodimenzionalne karte, dok u drugu fazu rada spada sam prikaz karte i korisničko pomicanje po iscrtanoj karti. Analizu prve faze rada aplikacije prikazat ćemo preko vremena potrebnog za izvođenje u odnosu na postavljene parametre, dok ćemo drugu fazu ispitivati pomoću broja sličica u sekundi (engl. FPS) opet u ovisnosti o zadanim parametrima. Sva mjerenja izvršena su na grafičkom procesoru Nvidia GTX 1070 uz Intelov procesor i7-4770k s četiri jezgre i 16 gigabajta radne memorije. Korištena platforma bit će Windows 10.

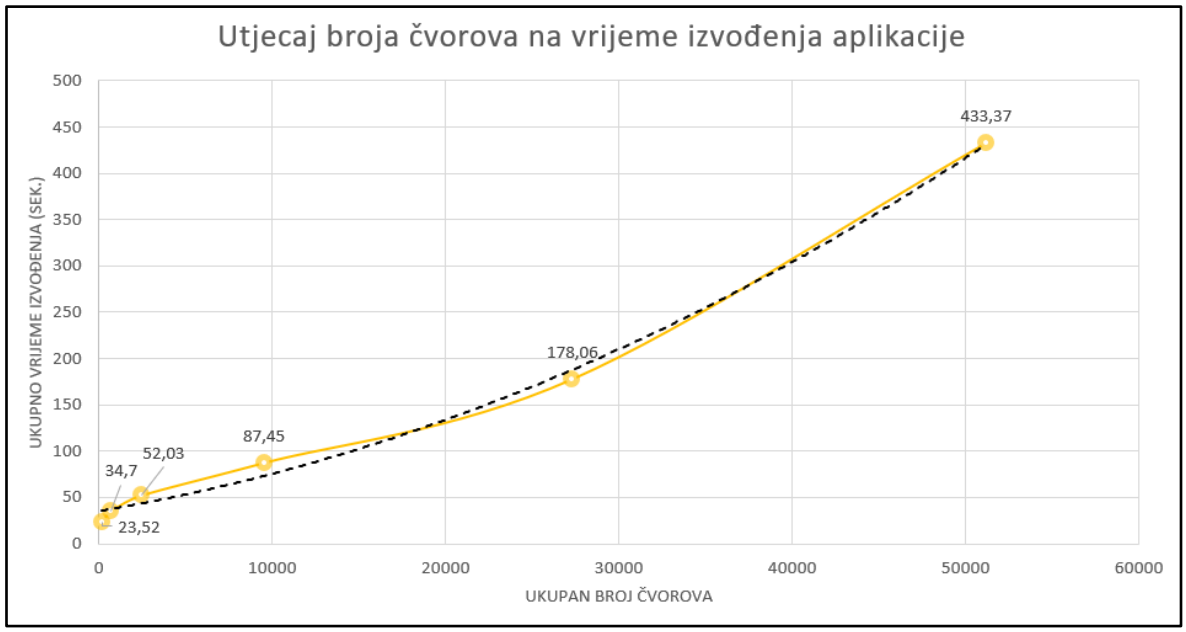
7.1. Vrijeme izvođenja aplikacije

U ovome ćemo potpoglavlju razmotriti kako se odnosi vrijeme potrebno za izvršavanje svih skripta od pokretanja aplikacije do prikaza karte u virtualnom prostoru u ovisnosti o odabranim koordinatama za koje iscrtavamo kartu. Odabrane koordinate zadaju primarno veličinu samog terena, no ona zapravo nema utjecaj na vrijeme izvođenja već primarni utjecaj ima količina podataka dostupna za to odabrano područje. U tu količinu podataka ubrajamo podatke dobivene iz OSM izvora, GIS Zrinjevac izvora i Google Maps Elevation API-ja. Zaključujemo kako što više podataka moramo preuzeti i obraditi to ćemo imati veći vremenski utrošak, a sada ćemo istražiti kako se on mijenja ovisno o količini podataka.

Kako bismo izvršili ovo testiranje uzet ćemo kao centralnu točku koordinate $(45.81, 15.98)$, te u odnosu na tu točku uzimati sve veću i veću površinu (sve veći raspon koordinata) i bilježiti vrijeme potrebno od početka rada aplikacije odnosno unesenih parametra i poslanog zahtjeva za stvaranje karte do prikaza karte na ekranu. Za svako mjerenje ćemo također voditi zapis o tome koliko je spremljeno čvorova koji uključuju sve čvorove dohvaćene iz OSM izvora podataka i sve čvorove odnosno svaki podatak koji predstavlja grm, stablo ili urbanu opremu iz GIS Zrinjevac izvora podataka. Broj točaka visinske mreže držat ćemo

konstantnim, konkretno visinska mreže bit će veličine $33x33$, a kasnije ćemo razmotriti utjecaj podataka iz Google Elevation API-ja na vrijeme izvođenja.

Na slici 7.1. prikazan je graf koji pokazuje ovisnost ukupnog vremena izvođenja u sekundama o ukupnom broju spremljenih čvorova tijekom dohvata i obrade podataka. Na grafu su prikazana vremena izvođenja za ukupno 6 mjerenja, a iz grafa je vidljivo kako se vrijeme izvođenja aplikacije odnosi polinomijalno i to drugog stupnja (kvadratno) u odnosu na broj čvorova. Kvadratna funkcija koja bi opisivala ovaj odnos prikazana je na grafu crnom iscrtkanom crtom.

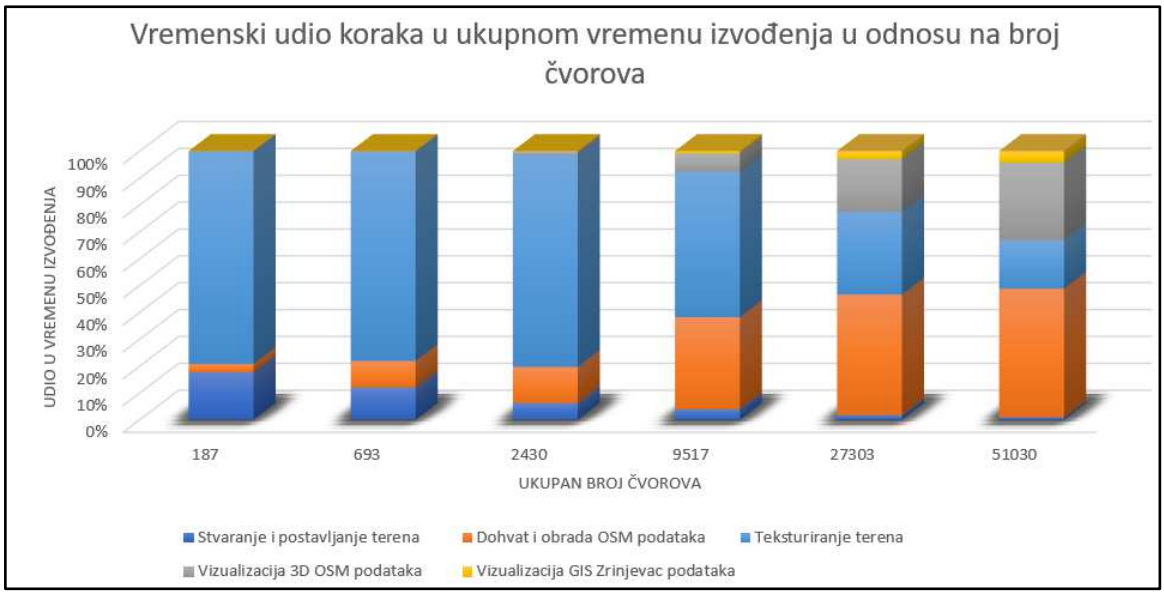


Slika 7.1. Graf utjecaja ukupnog broja čvorova na vrijeme izvođenja aplikacije

Drugo testiranje u sklopu ovog potpoglavlja koje ćemo provesti bit će koliko udio u vremenu izvođenja zauzima pojedini korak u radu aplikacije. Za potrebe ovog testiranja uzet ćemo iste postavke mjerenja kao kod prošlog jedino što ćemo nakon svakog koraka u radu aplikacije koji se izvrši spremiti proteklo vrijeme te na temelju tih podataka izračunati postotak svakog koraka u ukupnom vremenu izvođenja. Rad aplikacije podijelit ćemo u šest koraka na sličan način kako su i opisani kroz poglavlja ovog rada, a oni su: stvaranje i postavljanje terena, dohvat i obrada OSM podataka, teksturiranje terena, vizualizacija 3D OSM podataka i vizualizacija GIS Zrinjevac podataka.

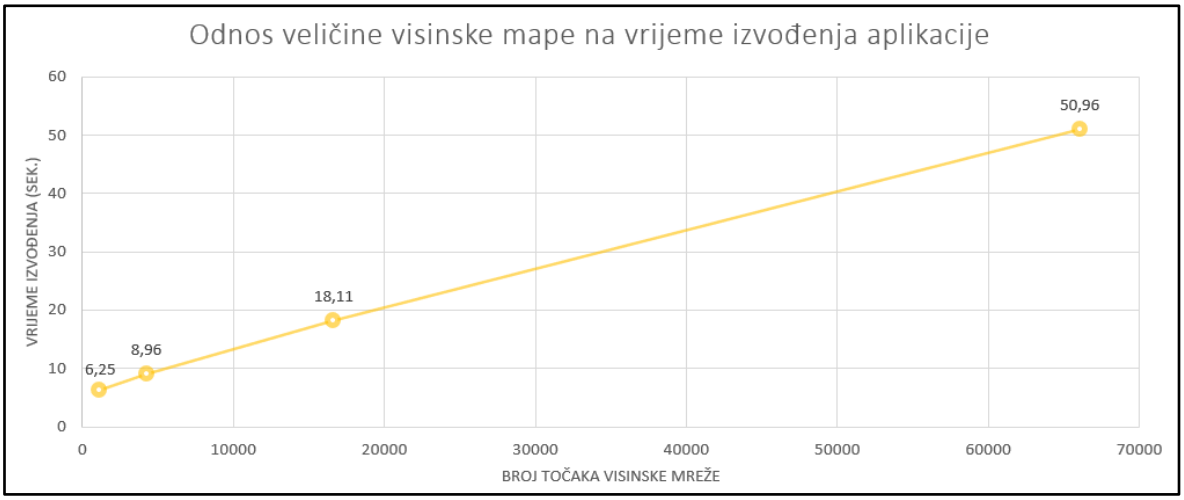
Na slici 7.2. prikazan je graf koji pokazuje udio u vremenu izvođenja za svaki od navedenih koraka rada aplikacije kroz 6 mjerenja različitih broja čvorova. Iz prikazanog grafa moguće je zaključiti kako s malim brojem čvorova najveći utjecaj na vrijeme izvođenja imaju stvaranja, postavljanje i teksturiranje terena, no kako se vrijeme potrebno

za te korake ne povećava suviše s povećanjem broja čvorova njihov utjecaj pada kod velikog broja čvorova. S velikim brojem čvorova najveći utjecaj preuzima dohvat i obrada OSM podataka zato što je potrebno proći kroz puno linija OSM datoteke i spremiti sve potrebne podatke u strukture. Osim toga s velikim brojem čvorova utjecaj vizualizacije 3D OSM podataka značajno raste, što je posljedica toga što se pri stvaranju točkastih podataka često traži orijentacija pojedinog objekta prema cesti što uključuje prolazak kroz sve rubne čvorove cesta kojih u ovom slučaju ima mnogo. Utjecaj obrade GIS Zrinjevac podataka je prilično malen kako s malim tako i s velikim brojem čvorova iako i on raste opet zbog računanja orijentacije nekih objekata.



Slika 7.2. Graf udjela koraka izvođenja aplikacije u ukupnom vremenu izvođenja u odnosu na broj čvorova

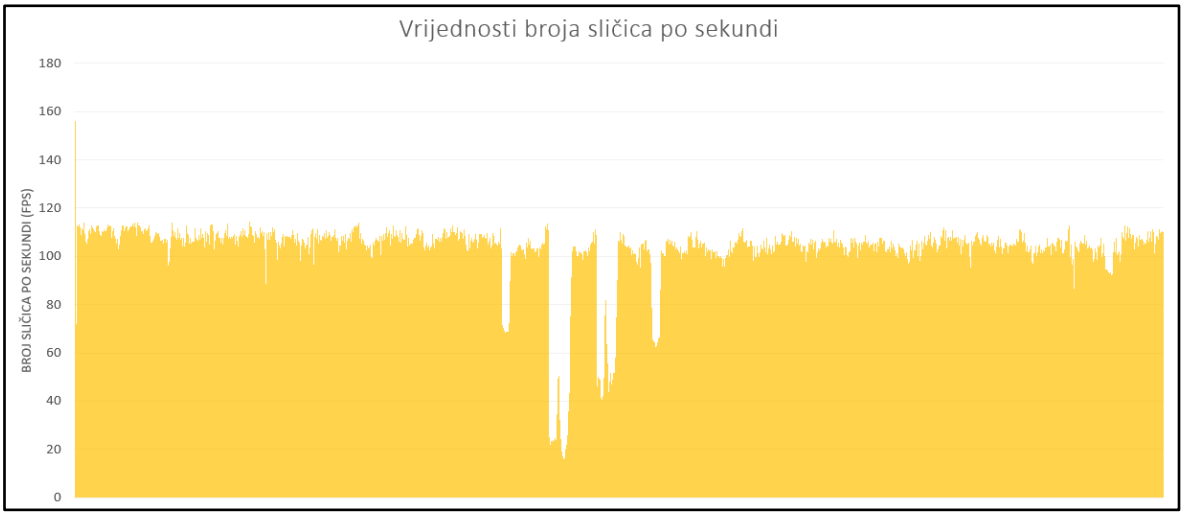
Promotrit ćemo još kako se odnosi vrijeme izvođenja o veličini postavljene visinske mreže. U svrhu toga proveli smo mjerenja na visinskim mrežama veličina $33x33$, $65x65$, $129x129$ i $257x257$. Na slici 7.3. stoji graf koji prikazuje traženi odnos gdje se na x osi nalazi ukupan broj točaka visinske mape, a njegov je korijen jednak broju HTTP zahtjeva koji moramo poslati Google Maps Elevation API-ju kako bismo dohvatili informacije za sve točke zadane mreže. Iz grafa na slici 7.3. možemo primijetiti kako se vrijeme izvođenja potrebno za stvaranje i postavljanje terena linearno odnosi s ukupnim brojem točaka visinske mreže.



Slika 7.3. Graf ovisnosti vremena izvođenja aplikacije o veličini visinske mape terena

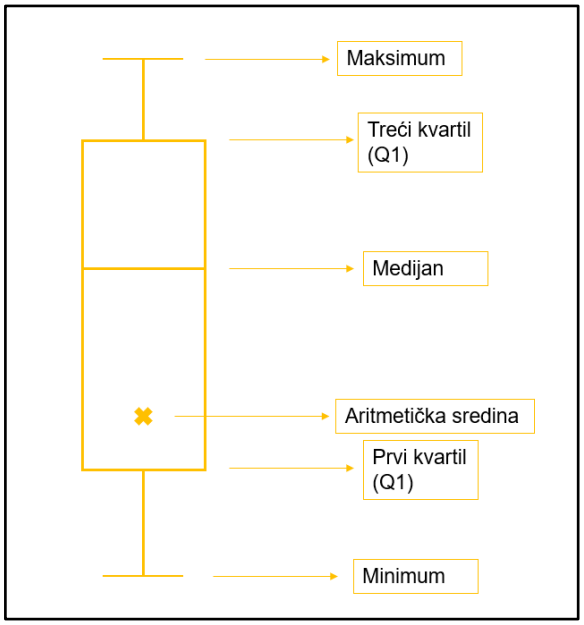
7.2. Frekvencija osvježavanja aplikacije

Nakon uspješno dohvaćenih, obrađenih i stvorenih podataka u ovome ćemo se potpoglavlju baviti time kako pojedini podaci ima utjecaj na frekvenciju osvježavanja naše aplikacije kada se korisnik odluči micati po prikazu naše karte. U svrhu ovoga mjerenja koristit ćemo metriku broja sličica po sekundi, a metoda ispitivanja bit će ta da ćemo postaviti kameru tako da gleda stvorenu kartu kao tlocrt te ćemo ju micati duž x i duž z osi za određene inkremente nakon svakog perioda vremena. U konkretnom slučaju ovog testiranja micat ćemo kameru 5 jedinica po x osi, te nakon prijeđenog cijelog terena po toj osi pomaknut ćemo kameru 50 jedinica po z osi i tako sve dok ne prijeđemo cijeli teren i po z osi. Period uzet za dodavanje inkrementa pozicije kamera bit će 0.0005 sekundi. Prilikom svakog pomicanja kamere uzet ćemo trenutan broj sličica po sekundi izračunat naredbom $\frac{1}{Time.deltaTime}$, gdje vrijednost *Time.deltaTime* predstavlja koliko je prošlo sekundi između prethodne sličice i trenutne sličice. Ovakvim postupkom po svakom mjerenju dobit ćemo skup od 3589 vrijednosti sličica po sekundi. Mjerenja ćemo provesti za stvaranje karte koja se nalazi unutar raspona od $[45.8, 45.81]$ geografske širine i $[15.97, 15.98]$ geografske dužine. Na slici 7.4. ćemo prikazati graf na kojem se nalaze sve vrijednosti jednog mjerenja broja sličica u sekundi dobivene opisanim postupkom za navedeno područje.



Slika 7.4. Graf skupa vrijednosti broja sličica u sekundi

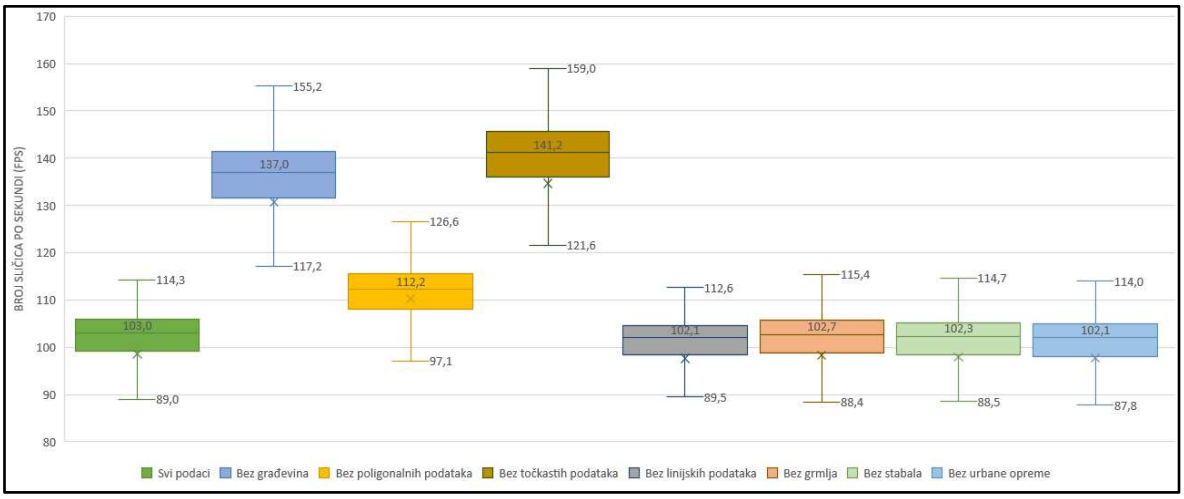
U svrhu testiranja provesti ćemo ukupno 8 mjerenja gdje ćemo u svakom od njih izbacivati jednu skupinu podataka koja ima 3D reprezentaciju u aplikaciji te ćemo tako promotriti utjecaj svakog skupa reprezentacija na rad aplikacije. Dobivene rezultate prikazat ćemo kutijastim dijagramom (engl. Box plot) koji sadrži informacije o minimumu, maksimumu, medijanu, aritmetičkoj sredini te prvom i trećem kvartilu podataka, a na slici 7.5. moguće je vidjeti ilustraciju dijela kutijastog dijagrama i uz njega povezane informacije.



Slika 7.5. Dijelovi kutijastog dijagrama

Na slici 7.6. prikazani su kutijasti dijagrami koji prikazuju u predviđenom obliku podatke dobivene mjerenjima na temelju skupa svih podataka, podataka bez građevina, podataka bez poligonalnih podataka, podataka bez točkastih podataka, podataka bez linijskih podataka, podataka bez grmlja, podataka bez stabala i podataka bez urbane opreme. Na temelju tih

dijagrama možemo zaključiti kako linijski podaci, podaci o grmlju, stablima i urbanoj opremi nemaju značajnog utjecaja na rad aplikacije u pogledu broja sličica po sekundi. Međutim oni podaci koji imaju najznačajniji utjecaj su podaci o građevinama i točkasti podaci. Uklanjanjem podataka o građevinama dobivamo otprilike 32% povećanja broja sličica po sekundi u prosjeku dok izostankom točkastih podataka vidimo povećanje od 36% u prosjeku. Umjeren rast broja sličica po sekundi zamjećujemo još neprikazivanjem poligonalnih podataka čime dobivamo 11% povećanja broja sličica po sekundi u prosjeku.



Slika 7.6. Kutijasti dijagrami rezultata mjerenja broja sličica po sekundi određenih skupina podataka

Na kraju možemo još napomenuti kako se prikazivanjem karte zadanih parametara uzimajući u obzir sve podatke stvara 110.6 milijuna trokuta odnosno 169.3 milijuna vrhova u virtualnoj sceni. Direktan pogled kamere na cijelu tako prikazanu kartu rezultira drastičnim padom frekvencije osvježavanja na samo 5 sličica po sekundi.

Zaključak

Na kraju ovog rada osvrnut ćemo se na informacije koje smo dali u uvodu, a koje su nas motivirale na proučavanje ove teme te ćemo ocijeniti dobivene rezultate i donijeti valjan zaključak. Kako smo naveli u uvodu javno su dostupni mnogi izvori kartografskih podataka, a mi smo odabrali njih četiri i opisali na koji način ih je moguće iskoristiti za dobivanje podataka. Osim dohvata samih podataka iz izvora Google Maps Elevation API, MapBox Static Images API, OpenStreetMap i GIS Zrinjevac opisali smo i postupke njihove obrade. Pomoću ta četiri izvora dobili smo vizualno impresivan prikaz nekog geoprostornog područja u obliku trodimenzionalne karte u virtualnoj sceni, a pri tome je najveći utjecaj imao izvor podataka OpenStreetMap koji je raširen diljem svijeta i sadrži pregršt kartografskih podataka koji se ažuriraju i validiraju iz dana u dan.

U odnosu na klasične karte ovaj prikaz je pogodan i za prikazivanje određenih struktura s velikim visinskim razlikama poput planinskih lanaca gdje dolaze do izražaja podaci korišteni iz izvora Google Maps Eelevation API-ja. Najčešće ćemo pak programsko rješenje koristiti za prikaz dijelova grada gdje pak dominiraju OpenStreetMap podaci. Takvim je prikazom moguće uočiti razne manje trodimenzionalne objekte koje ne posjeduje dvodimenzionalna karta i na kojoj ih nije mogući prikazati poput klupa, koševa za smeće, građevina, ograda, itd. Sam prikaz dobiven unutar pokretača Unity 3D može biti korišten za statički prikaz cijelog ili dijela područja kao makete područja ili kao interaktivna trodimenzionalna karta za virtualnu šetnju kroz reprezentirano područje.

Kada promotrimo analizu izvođenja programa možemo vidjeti kako se vrijeme izvođenja odnosi u najgorem slučaju kvadratno s količinom dohvaćenih podataka, a tu ovisnost mogli bi još smanjiti kada ne bismo računali orijentaciju objekata prema čvorovima cesta. Što se tiče frekvencije osvježavanja aplikacije vidljivo je iz rezultata kako pogledom na manji dio stvorenog područja nemamo nikakvih problema u vidu broja sličica po sekundi dok pogledom na veći dio ili cijelo područje frekvencija osvježavanja značajno pada, no kako nam je taj pogled zanimljiv jedino iz statičkom aspekta, odnosno bez značajnog pomicanja po karti, to nam ne predstavlja velik problem. U analizi smo ustvrdili kako najveći utjecaj imaju građevine te pojedini modeli korišteni za poligonalne i točkaste podatke zbog velikog broja vrhova. U konačnici brzina rada aplikacije ovisit će nam o količini i tipu podataka koji su dostupni za područje koje želimo prikazati, a to nije moguće unaprijed odrediti.

U aspektu nadogradnje stvorenog programskog rješenja moguće je još detaljnije proučiti OpenStreetMap izvor podataka koji sadrži ogroman skup informacija. Pogotovo je moguće proučiti elemente relacija koji nisu korišteni u ovome radu, a sadrže dodatne informacije o odnosima između korištenih elemenata staza i čvorova čime bi se dobila realističnija integracija objekata u virtualnu scenu i odnos među njima. Informacije iz GIS Zrinjevac izvora podataka su gotovo sve iskorištene u ovome radu, od podataka neiskorištene su dodatne dimenzije stabala, a da bi se one iskoristile potrebno bi bilo proceduralno generirati stabla. Također način reprezentacije cesta i površina bi se mogao promijeniti iz teksturiranja u 3D objekte što bi značilo precizniji prikaz, no iziskivalo bi veću računalnu snagu. Pored boljeg vizualnog prikaza ovom programskom rješenju mogla bi se dodati komponenta provođenja prostornih analiza na temelju stvorene karte. Također je moguće na podlozi stvorene karte napraviti simulaciju prometa ili navigacije između geoprostornih točaka.

Literatura

[1] Joling, A. *Open Data Sources for 3D Data Visualisation - Generating 3D Worlds based on OpenStreetMaps Data*. International Conference on Information Visualization Theory and Applications, Porto, (2017), str. 251-258.

[2] Povijest pokretača Unity. Poveznica: https://en.wikipedia.org/wiki/Unity_(game_engine); pristupljeno 18. svibnja 2021.

[3] Povijest alata Blender. Poveznica: https://en.wikipedia.org/wiki/Blender_(software); pristupljeno 18. svibnja 2021.

[4] Google Maps Elevation API dokumentacija. Poveznica: https://developers.google.com/maps/documentation/elevation/overview; pristupljeno 19. svibnja 2021.

[5] Mapbox Static Images API dokumentacija. Poveznica: https://docs.mapbox.com/api/maps/static-images/; pristupljeno 19. svibnja 2021.

[6] OpenStreetMap povijest. Poveznica: https://en.wikipedia.org/wiki/OpenStreetMap; pristupljeno 20. svibnja 2021.

[7] Katastar zelenila grada Zagreba. Poveznica: https://gis.zrinjevac.hr/; pristupljeno 20. svibnja 2021.

[8] OSM izvor podataka. Poveznica: https://www.openstreetmap.org/; pristupljeno 20. svibnja 2021.

[9] Unity 3D dokumentacija. Poveznica: https://docs.unity3d.com/Manual/index.html; pristupljeno 10. lipnja 2021.

[10] Izvor tekstura textures.com. Poveznica: https://www.textures.com/; pristupljeno 27. svibnja 2021.

[11] Algoritam određivanja točke unutar poligona. Poveznica: https://wrf.ecse.rpi.edu/Research/Short_Notes/pnpoly.html; pristupljeno 17. lipnja 2021.

[12] Oblik krovova u OSM-u. Poveznica: https://wiki.openstreetmap.org/wiki/Key:roof:shape; pristupljeno 1. lipnja 2021.

[13] Algoritam triangulacije poligona. Poveznica: https://wiki.unity3d.com/index.php/Triangulator; pristupljeno 1. lipnja 2021.

[14] CGTrader izvor 3D modela. Poveznica: https://www.cgtrader.com/; pristupljeno 1. lipnja 2021.

[15] Free3D izvor 3D modela. Poveznica: https://free3d.com/; pristupljeno 1. lipnja 2021.

[16] TurboSquid izvor 3D modela. Poveznica: https://www.turbosquid.com/; pristupljeno 1. lipnja 2021.

[17] Bresenhamov algoritam. Poveznica: http://members.chello.at/easyfilter/bresenham.html; pristupljeno 8. lipnja 2021.

[18] Blender dokumentacija. Poveznica: https://docs.blender.org/manual/en/latest/; pristupljeno 10. lipnja 2021.

Sažetak

Naslov: Vizualizacija geoprostornih lokacija temeljena na stvarnim podacima

U ovome je diplomskom radu opisana vizualizacija geoprostornog područja na temelju javno dostupnih kartografskih podataka. Programsko rješenje koje prikazuje rad opisanih postupaka ostvareno je korištenjem pokretača Unity 3D gdje je geoprostorno područje prikazano je u obliku trodimenzionalne karte. Prikazan je postupak dohvaćanja i obrade informacija iz četiri javno dostupna izvora podataka. Glavni izvor podataka koji je korišten je OpenStreetMap te su iz njega raznim opisanim postupcima prikazane ceste, površine, građevine i ostali 3D objekti u virtualnoj sceni. U svrhu oblikovanja terena dohvaćeni su podaci o visinama iz izvora Google Maps Elevation API. Radi realističnijeg prikaza grada Zagreba korišteni su podaci o grmlju, stablima i urbanoj opremi dobiveni iz izvora GIS Zrinjevac. Na kraju je razmotren utjecaj količine dohvaćenih podataka na vrijeme izvođenja i frekvenciju osvježavanja izrađenog programskog rješenja.

Ključne riječi: vizualizacija podataka, kartografski podaci, virtualno okruženje, Unity 3D, OpenStreetMap, GIS, GIS Zrinjevac, Google Maps Elevation API

Summary

Title: Visualization of Geospatial Locations Based on Real Data

This thesis describes the visualization of the geospatial area based on publicly available cartographic data. A software solution that shows the described procedures in the work was achieved using the Unity 3D engine where the geospatial area is shown in the form of 3D map. The process of retrieving and processing information from four publicly available data sources is presented. The main data source used is the OpenStreetMap from which roads, surfaces, buildings and other 3D objects are shown by various described procedures in a virtual scene. For terrain shape purposes, elevation data was retrieved from the Google Maps Elevation API source. For a more realistic view of the city of Zagreb, data on shrubs, trees and urban equipment obtained from the GIS Zrinjevac source were used. Finally, the influence of the amount of retrieved data on the execution time and the refresh rate of the developed software solution is discussed.

Keywords: data visualization, cartographic data, virtual environment, Unity 3D, OpenStreetMap, GIS, GIS Zrinjevac, Google Maps Elevation API

Skraćenice

API	*Application Programming Interface*	aplikacijsko programsko sučelje
OSM	*Open Street Map*	javno dostupna ulična karta
HTTP	*Hypertext Transfer Protocol*	protokol za prijenos hiperteksta
URL	*Uniform Resource Locator*	usklađeni lokator sadržaja
WGS84	*World Geodetic System 1984*	svjetski geodetski sustav 1984
GIS	*Geographic Information System*	geografski informacijski sustav
XML	*EXtensible Markup Language*	proširivi jezik za označavanje
HTML	*HyperText Markup Language*	hipertekstni jezik za označavanje
JSON	*JavaScript Object Notation*	notacija JavaScript objekata
FPS	*Frames per second*	sličice po sekundi

Privitak

Popis oznaka za čvorove i staze

Ovo poglavlje privitka sadrži listu svih oznaka koje su uzete u obzir pri obrada OSM podataka. Ovdje će u dvije tablice biti izlistane oznake i način na koji one utječu na prezentaciju naše trodimenzionalne karte.

U nastavku u tablici 1.1. se nalazi lista svih oznaka koje su uzete u obzir pri opisu OSM elementa čvora (Node). U prvom stupcu nalazi se skup znakova kojim se raspoznaje pojedini podatak, dok se u drugom stupcu nalazi naziv i po potrebi objašnjenje tog podatka, dok se u trećem stupcu nalazi tip podatka. Tip podatka nam govori u kojem djelu aplikacije je taj podatak iskorišten kako bi se mogli pripadni podaci povezati s načinom njihovog stvaranja opisanog u radu. Ako oznaka ne sadrži dio u koje je opisana vrijednost *v="value"* tada je uzeto više vrijednosti ili sve vrijednosti u obzir.

Tablica 1.1. Popis oznaka za opis čvorova

Oznaka	Opis	Tip podatka
<tag k="barrier" v="bollard"	Prometni stupić	Točkasti podatak
<tag k="barrier" v="lift_gate"	Ulazna rampa	Točkasti podatak
<tag k="barrier" v="gate"	Ogradna vrata	Linijski podatak
<tag k="traffic_calming"	Vrsta smirivanja prometa (npr. ležeći policajac)	Točkasti podatak
<tag k="highway" v="crossing"	Križanje cesta	Nema reprezentaciju, samo kao informacija
<tag k="highway" v="bus_stop"	Autobusna stanica	Točkasti podatak
<tag k="shelter" v="yes"	Čvor ima nadstrešnicu, dolazi uz oznaku <tag k="highway" v="bus_stop"	Točkasti podatak
<tag k="shelter" v="no"	Čvor nema nadstrešnicu, dolazi uz oznaku <tag k="highway" v="bus_stop"	Točkasti podatak
<tag k="highway" v="street_lamp"	Ulična lampa	Točkasti podatak
<tag k="highway" v="traffic_signals"	Semafor	Točkasti podatak
<tag k="highway" v="stop"	Znak STOP	Točkasti podatak
<tag k="highway" v="give_way"	Znak za sporednu ulicu	Točkasti podatak

<tag k="man_made" v="flagpole"	Zastava	Točkasti podatak
<tag k="amenity" v="bicycle_parking"	Parking za bicikle	Točkasti podatak
<tag k="amenity" v="atm"	Bankomat	Točkasti podatak
<tag k="amenity" v="toilets"	Javni WC	Točkasti podatak
<tag k="amenity" v="clock"	Sat	Točkasti podatak
<tag k="amenity" v="post_box"	Poštanski sandučić	Točkasti podatak
<tag k="amenity" v="telephone"	Javni telefon	Točkasti podatak
<tag k="amenity" v="drinking_water"	Izvor pitke vode, najčešće kao fontana za piće	Točkasti podatak
<tag k="amenity" v="taxi"	Znak za taxi stajalište	Točkasti podatak
<tag k="historic" v="memorial"	Spomenik	Točkasti objekt
<tag k="emergency" v="fire_hydrant"	Hidrant	Točkasti podatak
<tag k="shop" v="kiosk"	Kiosk	Točkasti podatak
<tag k="recycling:plastic" v="yes"	Kanta za plastiku	Točkasti podatak
<tag k="recycling:paper" v="yes"	Kanta za papir	Točkasti podatak
<tag k="recycling:glass" v="yes"	Kanta za staklo	Točkasti podatak
<tag k="vending " v="parking_tickets"	Automat za parkirne karte	Točkasti podatak

U idućoj tablici 1.2. nalazi se lista svih oznaka koje su uzete u obzir pri opisu OSM elementa staze (Way). U prvom stupcu nalazi se skup znakova kojim se raspoznaje neki podatak, u drugom se stupcu nalazi naziv i po potrebi objašnjenje tog opisa, a u trećem tip podatka.

Tablica 1.2. Popis oznaka za opis staza

Oznaka	Opis	Tip podatka
<tag k="building"	Zgrada, a kao vrijednost oznake, ako je poznato, se pojavljuje tip zgrade	Građevine
<tag k="building:part"	Zgrada, a kao vrijednost oznake, ako je poznato, se pojavljuje tip zgrade	Građevine
<tag k="addr:street"	Naziv ulice, pojavljuje se uz opis zgrade	Građevine
<tag k="addr:housenumber"	Kućni broj, pojavljuje se uz opis zgrade	Građevine
<tag k="building:levels"	Broj katova, pojavljuje se uz opis zgrade	Građevine
<tag k="height"	Visina, pojavljuje se uz opis zgrade	Građevine

<tag k="roof:shape"	Oblik krova, pojavljuje se uz opis zgrade	Građevine
<tag k="highway"	Cesta, a kao vrijednost oznake se pojavljuje vrsta ceste po važnosti	Ceste
<tag k="name"	Naziv, pojavljuje se uz opis ceste ili željeznice ili zgrade	Ceste, građevine i linijski podatak
<tag k="footway" v="sidewalk"	Nogostup	Linijski podatak
<tag k="surface"	Površina ceste, pojavljuje se uz opis ceste	Ceste
<tag k="lanes"	Broj traka, pojavljuje se uz opis ceste	Ceste
<tag k="maxspeed"	Ograničenje maksimalne brzine, pojavljuje se uz opis ceste	Ceste
<tag k="maxspeed:type" v="HR:urban"	Ograničenje maksimalne brzine za urbana područja u RH (iznosi 50 km/h)	Ceste
<tag k="oneway"	Jeli je cesta jednosmjerna	Ceste
<tag k="lit"	Jeli je cesta osvijetljena	Ceste
<tag k="railway"	Željeznička pruga, a kao vrijednost oznake pojavljuje se njezin tip	Linijski podatak
<tag k="gauge"	Širina željezničke pruge	Linijski podatak
<tag k="electrified"	Jeli željeznica elektrificirana	Linijski podatak
<tag k="passenger_lines"	Broj traka željezničke pruge	Linijski podatak
<tag k="water"	Površina prekrivena vodom	Poligonalni podatak
<tag k="natural" v="water"	Površina prekrivena vodom	Poligonalni podatak
<tag k="memorial"	Površina s memorijalnim objektima	Poligonalni podatak
<tag k="historic" v="memorial"	Površina s memorijalnim objektima	Poligonalni podatak
<tag k="landuse"	Opis ljudske uporabe zemljišta	Poligonalni i teksturalni podatak
<tag k="leisure"	Površina rekreacijske namjene	Poligonalni i teksturalni podatak
<tag k="sport"	Površina sa sportskom namjenom, a kao vrijednost stoji naziv sporta	Poligonalni podatak
<tag k="barrier"	Ogradni objekt, a kao vrijednost stoji tip ograde	Linijski podatak
<tag k="public_transport" v="platform "	Platforma kao stajalište	Linijski podatak
<tag k="amenity" v="car_wash"	Autopraonica	Poligonalni i teksturalni podatak
<tag k="amenity" v="fuel"	Benzinska postaja	Poligonalni i teksturalni podatak

<tag k="amenity" v="parking"	Površina parkirališta za automobile	Teksturalni podatak
<tag k="amenity" v="shelter"	Natkrivena kućica kao skloništе	Poligonalni podatak
<tag k="amenity" v="bicycle_parking"	Površina parkirališta za bicikle	Poligonalni i teksturalni podatak
<tag k="amenity" v="taxi"	Površina parkirališta za taxi automobile	Teksturalni podatak
<tag k="building" v="greenhouse"	Staklenik	Poligonalni podatak
<tag k=" building " v="roof"	Nadstrešnica	Poligonalni podatak
<tag k="amenity"	Ako se ne radi o nekoj već spomenutoj vrijednosti ključa amenity onda se radi o vrsti građevine	Građevine
<tag k="man_made" v="bridge"	Most	Poligonalni podatak
<tag k="man_made" v="embankment"	Nasip	Poligonalni podatak

Popis slojeva terena

Na sljedećoj tablici 1.3. nalazi se popis svih slojeva koji su korišteni unutar aplikacije za teksturiranje terena. U prvom se stupcu nalazi naziv sloja kako on stoji u aplikaciji, dok se u drugog stupcu nalazi njegov opis, odnosno što on prikazuje te iz kojeg je izvora dobiven.

Tablica 1.3. Popis korištenih slojeva

Naziv sloja	Opis sloja
GrassLayer	Sloj korišten za prikaz travnate podloge, iz oznake *landuse*
AsphaltLayer	Sloj korišten za prikaz cesta, parkinga i površine za autopraonicu i benzinsku postaju
WhiteLayer	Sloj korišten za prikaz bijelih crta na cesti i parkingu
GravelLayer	Sloj korišten za prikaz šljunčanog puta
CompactedLayer	Sloj korišten za prikaz makadamskog puta
ConcreteLayer	Sloj korišten za prikaz betonskog puta i nekih površina iz oznake *landuse*
GroundLayer	Sloj korišten za prikaz zemljanog puta i površine sa sadnicama iz oznake *landuse*
SandLayer	Sloj korišten za prikaz pješčanog puta
RubberLayer	Sloj korišten za prikaz gumene površine dječjeg igrališta, iz oznake *leisure*
AsphaltResiLayer	Sloj korišten za prikaz površine biciklističkog parkinga i gradskog asfaltiranog područja

CycleLayer	Sloj korišten za prikaz biciklističke staze
PavingLayer	Sloj korišten za prikaz uređenog popločanog puta
SettLayer	Sloj korišten za prikaz neuređenog popločanog puta
MapLayer	Sloj korišten za prikaz satelitske karte
YellowLayer	Sloj korišten za prikaz žutih crta kod taxi parkinga

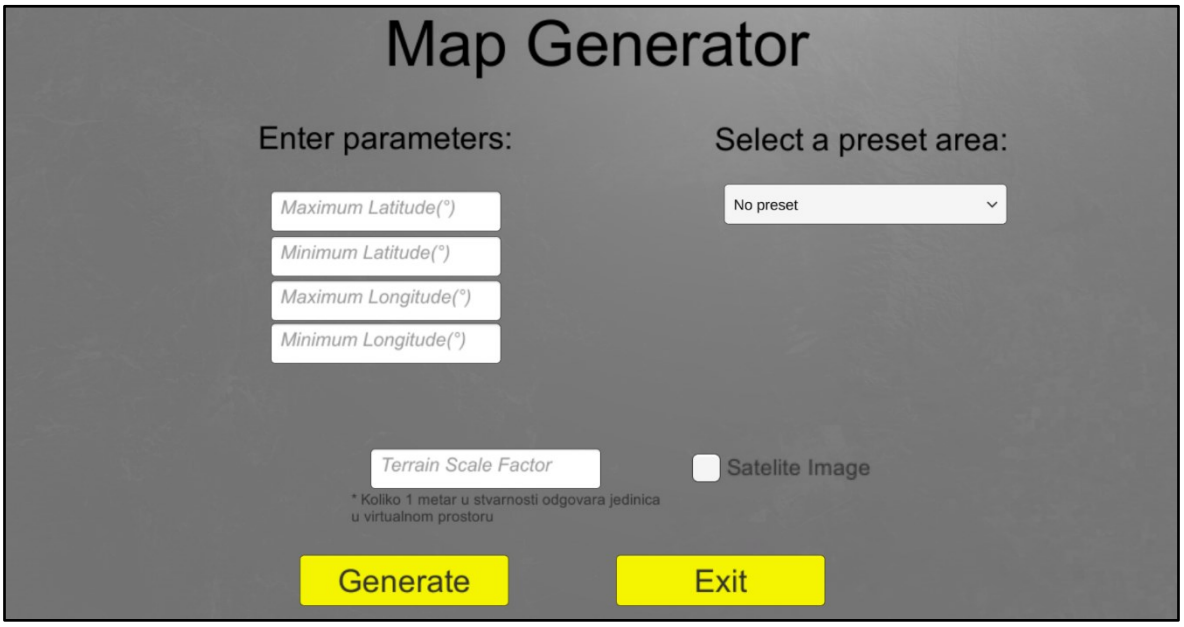
Upute za korištenje programske podrške

Rad opisane aplikacije moguće je samostalno isprobati preuzimanjem direktorija pod nazivom „Izvršni kod“ koji se nalazi u git repozitoriju na sljedećoj poveznici https://gitlab.com/AKomadina/diplomskirad.git. Na istoj je poveznici dostupan i izvorni kod aplikacije pod direktorijom „Izvorni kod“ koji se može uvesti u pokretač Unity te tako mijenjati postavke programa i također isprobati njegov rad. U direktoriju s izvršnim kodom nalazi se izvršna datoteka MapGenerator.exe koju je potrebno pokrenuti te će se tako pokrenuti aplikacije preko cijelog ekrana. Priložena izvršna datoteka prilagođena je za Windows operacijski sustav, te su uz nju dostupne sve ostale datoteke potrebne za njen uspješan rad. Važno je naglasiti kako aplikacija zahtijeva povezanost na internet jer se služi HTTP protokolom kako bi se uspješno preuzele informacije o nekom području za koje želimo ostvariti reprezentaciju preko trodimenzionalne karte.

Pri otvaranju aplikacije dočekat će vas početni izbornik u koji je moguće unijeti parametre potrebne za izvođenje aplikacije. Prva četiri parametra redom označavaju maksimalnu geografsku širinu, minimalnu geografsku širinu, maksimalnu geografsku dužinu te minimalnu geografsku dužinu. Navedene parametre je potrebno unijeti kao decimalne brojeve (npr. 45.6178). Kako ne postoji nikakva validacija njihove ispravnosti, potrebno je paziti kako su navedeni parametri ispravno zadani inače aplikacije neće raditi. Do problema može doći u slučaju kada je zadana minimalna koordinata veća ili jednaka maksimalnoj koordinati.

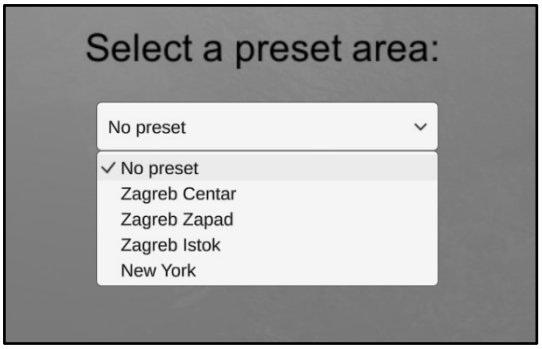
Posljednji parametar koji je potrebno unijeti je faktor skaliranja terena koji određuju veličinu stvorene karte, preciznije govori o tome koliko jedan metar u stvarnosti odgovara jedinica u virtualnom prostoru (npr. ako je faktor skaliranja 0.5 tada će 1 jedinica u virtualnom prostoru predstavljati 2 metra u stvarnosti). Taj je broj također potrebno unijeti kao decimalni te se ne vrši nikakva validacija njegove ispravnosti. Pored faktora skaliranja nalazi se potvrdni

okvir na čiji klik izabiremo opciju postavljanja satelitske snimke na teren koji ćemo stvoriti. Izgled početnog izbornika prikazan je na slici 8.1.



Slika 8.1. Izgled početnog izbornika

Osim unosa opisanih parametara aplikaciju je moguće pokrenuti i odabirom jednog od predefiniranih područja u padajućem izborniku na desnoj strani početnog izbornika, a prikaz njegovih opcija vidljiv je na slici 8.2. U tom je slučaju potrebno unijeti jedino parametar faktora skaliranja.



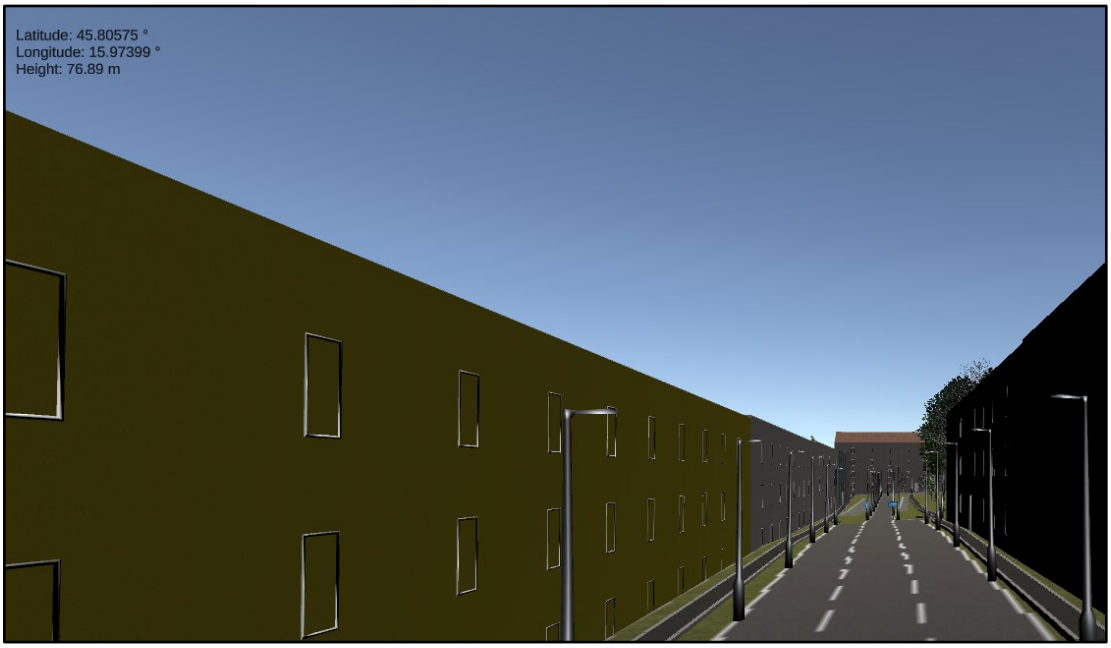
Slika 8.2. Opcije predefiniranih područja

Nakon što smo unijeli sve parametre pritiskom na gumb Generate puštamo aplikaciju neka stvori mapu u zadanim koordinatama i mjerilu. Klikom na gumb Exit izlazimo iz aplikacije.

Ovisno o veličini odabranog područja aplikaciji će biti potrebno manje odnosno više vremena da dohvati i obradi sve podatke pa je nekad potrebno malo pričekati dok se ne prikaže karta.

Nakon uspješno procesiranih podataka i stvaranja karte kamera će biti pozicionirana na njezino središte te će prikaz biti iz trećeg lica odnosno moći ćemo lebdjeti iznad mape. Po karti se moguće kretati korištenjem tipki W, A, S, D na tipkovnici za promjenu pozicije i mišem za promjenu smjera pogleda. Držanjem pritisnute tipke SHIFT tijekom kretanja imat ćemo mogućnost bržeg mijenjanja pozicije na karti. U svakom je trenutku moguće stisnuti tipku ESC na tipkovnici kako bi izašli iz aplikacije.

Tijekom kretanja po karti u gornjem lijevom kutu bit će vidljive informacije o geografskoj širini i dužini koje odgovaraju poziciji na karti gdje se nalazimo. Osim toga bit će vidljiva i nadmorska visina na kojoj se nalazimo. Na slici 8.3. prikazan je isječka iz rada aplikacije na kojem su vidljive navedene informacije.



Slika 8.3. Isječak s prikazom geografskih informacija i nadmorske visine