*Volume 1 (1982), Number 1*

# Vizualizacija oblaka točk Slovenije

Mitja Rizvič

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko, Slovenija

**Abstract**

*Ministrstvo za okolje in prostor je v zadnjih letih zagotovilo javen dostop do LIDAR podatkov Slovenije. Čeprav so podatki objavljeni na spletu, jih je mogoče le prenesti, saj spletna stran ne omogoča vizualizacije. Naša naloga je izdelati spletno aplikacijo, ki bo omogočala interaktivno upodobitev LIDAR podatkov neposredno v spletnem brskalniku. Glavni izziv je zagotoviti dobro uporabniško izkušnjo kljub zelo veliki količini podatkov.*

## 1. Uvod

Kratica LIDAR v angleščini pomeni *"Light Detection And Ranging"*, kar v slovenščini pomeni Svetlobno zaznavanje in merjenje. Gre za tehnologijo, ki omogoča merjenje razdalj do posameznih točk s pomočjo laserskih žarkov. Metoda je široko uporabljena za izdelavo reliefnih slik oziroma zemljevidov. Tako so pred nekaj leti tudi slovensko ozemlje skenirakli z uporabo omenjejne tehologije in rezultate objavili na spletnih straneh agencije za okolje - ARSO [ars].

Celotno slovensko ozemlje je razdeljeno na bloke velikosti enega kvadratnega kilometra. Le-ti so na voljo za prenos v LAZ obliki, ki je ena od standardnih oblik za shranjevanje oblakov točk (*angl. pointcloud*). Oblak točk je predstavitev podatkov, kjer poleg samih točk ne hranimo nobenih drugih informacij, kot na primer povezav med njimi, normal za izračun osvetlitve itd. Lahko pa posamezna točka vsebuje nekatere dodatne informacije kot so na primer intenziteta odbitega laserskega žarka, klasifikacija in še nekaj drugih [LASa].

Upodobitev posameznega bloka ne predstavlja posebnih težav. Datoteko, ki predstavlja blok enostavno prenesemo in jo upodobimo z eno od metod upodobitve oblakov točk. Problem se pojavi pri upodabljanju večjega območja, ki se razteza čez več sosednjih blokov. V tem primeru moramo prenesti vse potrebne bloke, kar pa lahko nanese zelo veliko podatkov. Če bi v skrajnem primeru želeli pogled na celotno območje Slovenije, bi to pomenilo, da moramo prenesti celotno podatkovno bazo, kar pa predstavlja več terabajtov podatkov. Poleg tega, da bi imeli težave pri prenosu tudi osebni računalnik ne bi zmogel upodobiti takšne količine točk v realnem času.

V naslednjem poglavju bomo predstavili pristop, s katerim rešimo zgornji problem in omogočimo upodabljanje oblaka točk v realnem času ne glede na velikost področja, ki ga želimo upodobiti.

## 2. Implementacija

V nadaljevanju bomo najprej opisali podatkovno strukturo na kateri bazira aplikacija, ter orodje za njeno izdelavo. Nato pa sledi opis implementacije same spletne aplikacije.

### 2.1. Podatkovna struktura

Za zagotavljanje dobre uporabniške izkušnje je potrebno podatke upodobiti čim hitreje ne glede na velikost področja, ki ga upodabljamo. Zato je potrebno implementirati mehanizem, ki bo omogočal, da pri ogledu večjega področja zmanjšamo gostoto točk, ki jih prikazujemo in s tem privarčujemo pri prenosu podatkov. Ker je pri ogledu večjega področja kamera bolj oddaljena, lahko to tudi storimo brez večjega vpliva na kvaliteto upodobitve.

Prilagajanje gostote točk glede na velikost področja zahteva, da se podatki hranijo v ustrezni obliki. Podatkovna struktura, ki je široko uporabljena za tovrstne probleme, je osmiško drevo. Le-tega uporablja tudi knjižnica Potree, ki je namenjena prav vizualizaciji oblakov točk v spletnem brskalniku [pot]. Ideja je v tem, da tri dimenzionalni prostor lahko razdelimo na osem manjših delov enake velikosti. Vozlišče drevesa predstavlja celotni prostor (na primer celotno področje Slovenije), katerega potem razdelimo na osem manjših delov. Vsakega od njih nato naprej razdelimo na osem manjših delov in postopek ponavljamo do željene globine. Ker pa so v primeru LIDAR podatkov točke porazdeljene predvsem vzdolž X in Y osi lahko pri delitvi prostora, Z os ignoriramo in tako prostor namesto na osem razdelimo

le na štiri dele enake velikosti. Tako namesto osmiškega drevesa lahko uporabimo preprostejše štiriško drevo.

Ko imamo prostor razdeljen, lahko posamezno vozlišče 'napolnimo' s točkami, ki se nahajajo v prostoru vozlišča. Poleg tega lahko za vozlišča, ki predstavljajo večje področje točke ustrezno podvzorčimo. Tako dobimo strukturo, kjer korensko vozlišče predstavlja celotni prostor in vsebuje vzorec točk celotnega področja pri majhni gostoti. Z vsakim naslednjim nivojem se gostota točk povečuje, področje pa zmanjšuje vse do zadnjega nivoja, ki vsebuje vse točke brez podvzorčenja, vendar predstavlja zelo majhno področje.

Program za izdelavo podatkovne baze je napisan v jeziku C++. Poleg standardne knjižnice uporablja še odprtokodni knjižnici LASlib [LASb] za delo z LAS datotekami ter ZLIB [zli] za stiskanje izhodnih datotek. Gradnja podatkovne strukture se prične z branjem vhodne LAS datoteke. Najprej preberemo najmanjše in največje koordinate točk in potem iz njih izračunamo, kje je potrebno razdeliti prostor na štiri dele enake velikosti.

V vsaki iteraciji zanke dodamo v koren drevesa točko z verjetnostjo, ki je odvisna od željene globine drevesa. Večja kot je željena globina, z manjšo verjetnostjo bo točka dodana v koren. V primeru, da točka ni dodana v koren drevesa, se glede na razdelitev prostora izbere ustrezni naslednik, kateremu dodamo točko z verjetnostjo, ki je za faktor štiri večja od trenutne. Zaradi štirikrat večje verjetnosti, bo gostota točk z vsakim nivojem štirikrat večja. Ker pa se območje z vsakim nivojem hkrati štirikrat zmanjša, bodo vozlišča ne glede na globino vsebovala približno enako število točk. Postopek ponavljamo toliko časa, dokler točke ne dodamo enemu od vozlišč. Ker se verjetnost z vsakim nivojem poveča za faktor štiri na neki globini pridemo do verjetnosti, ki je enaka ena. Ta nivo bo tudi zadnji nivo in bo vseboval vse točke, ki niso bile dodane v enega od višje ležečih vozlišč.

Vsako vozlišče se shrani v ločeno datoteko. Hierarhija drevesa je določena s poimenovanjem datotek in sicer po naslednjem pravilu. Koren drevesa je poimenovan s črko "r". Vsako vozlišče ima štiri naslednike, ki jih poimenujemo tako, da vozlišča oštevilčimo s številkami od 0 do 3, katere pripnemo imenu starševskega vozlišča. Tako na globini ena dobimo štiri vozlišča z imeni: r0, r1, r2, r3. Enak postopek nato ponovimo za vsakega od vozlišč. Vozlišče r0 ima tako štiri naslednike z imeni r00, r01, r02, r03. Tak pristop omogoča postavitev spletne aplikacije brez potrebe po programu, ki teče na strežniku in zagotavlja dostop do podatkov, saj lahko potrebne datoteke preberemo iz strežnika direktno iz spletnega brskalnika.

Originalni podatki predstavljajo sorazmerno veliko področje. Zaradi majhne verjetnosti, da bi uporabnik želel naenkrat pregled nad celotnim ozemljem, smo obdržali originalno porazdelitev na posamezne bloke enega kvadratnega kilometra. Le-te nato v realnem času nalagamo v odvisnosti od pogleda kamere. Tako se izognemo upodabljanju točk, ki trenutno niso vidne in na tak način pohitrimo program ter še dodatno privarčujemo pri prenosu podatkov.

### 2.2. Spletna aplikacija

Spletna aplikacija je implementirana v programskem jeziku Javascript in za delovanje uporablja dve zunanji knjižnici in sicer "Three.js"ter "Pako" [thr], [pak]. Prva je visokonivojska knjižnica za izris 3D geometrije, ki temelji na WebGL-u, druga pa omogoča enostavno dekompresijo podatkov. V nadaljevanju bomo najprej predstavili tehnično ozadje implementacije, nato pa aplikacijo predstavili še iz uporabniškega vidika.

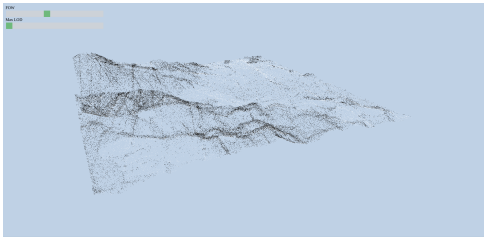

**Figure 1:** *Primer upodobitve devetih blokov pri LOD 1. Prikazujejo se torej le korenska vozlišča vsakega štiriškega drevesa.*

Ob zagonu aplikacija iz strežnika prebere datoteko "tiles.txt", ki vsebuje seznam blokov, ki se nahajajo na strežniku. Za vsak blok se nato prebere pripadajoča konfiguracijska datoteka, iz katere je razvidno kakšno območje le-ta pokriva. Glede na področje, ki ga pokriva blok lahko v vsakem trenutku določimo ali se le-ta nahaja znotraj vidnega polja kamere ali ne. V kolikor se posamezni blok nahaja znotraj vidnega področja kamere iz strežnika, prenesemo potrebne podatke za upodobitev.

Za upodobitev posameznega bloka, najprej zgradimo štiriško drevo s praznimi vozlišči. Globino drevesa in meje za razdelitev prostora preberemo iz konfiguracijske datoteke, ki se nahaja na strežniku poleg datotek, ki predstavljajo posamezna vozlišča drevesa. Ob vsaki spremembi pogleda oziroma uporabnikovi interakciji se pokliče metoda "updateTree", ki poskrbi za to, da so vozlišča 'napolnjena' z ustreznimi točkami. Na večji globini kot se nahaja vozlišče večja je gostota točk, ki jih vsebuje. Ker želimo, da gostota točk pada z razdaljo od kamere, metoda "updateTree"posamezno vozlišče napolni s točkami v odvisnosti od tega, koliko je trentno vozlišče oddaljeno od kamere. Torej vozlišča na manjši globini bodo vidna tudi na večji razdalji od kamere, medtem ko se bodo najglobja vozlišča prikazovala le blizu kamere. Razdaljo pri kateri se prikazujejo vozlišča določene globine, lahko uporabnik nastavi s pomočjo drsnika. Prav tako se lahko s pomočjo drsnika omeji nivo podrobnosti, kar pomeni, da ne prikazujemo vozlišč globje od določene meje ne glede na razdaljo od kamere.

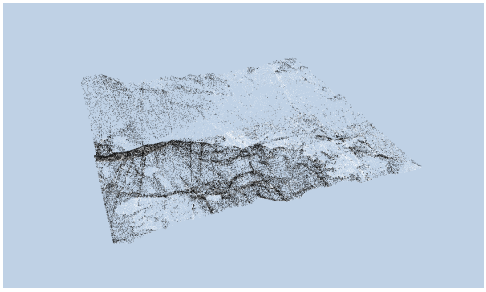

**Figure 2:** *Primer upodobitve devetih blokov, kjer je razvidna meja med prvim in drugim nivojem gostote točk.*

## 3. Zaključek

V okviru projekta smo razvili spletno aplikacijo, ki je na voljo za prenos preko GitHub strani [git]. Aplikacija omogoča upodabljanje oblakov točk znotraj spletnega brskalnika. Namesto osmiškega drevesa, ki je pogosto uporabljen pri podobnih rešitvah, smo za delitev prostora uporabili štiriško drevo, kar pomeni, da je aplikacija bolj prilagojena podatkom, ki so porazdeljeni predvsem vzdolž X in Y osi. Na tak način smo aplikacijo optimizirali za uporabo z LIDAR podatki, kar je bil tudi prvotni namen projekta.

Aplikacija točke prikazuje v črno-belih odtenkih, saj se za izračun barve uporablja informacija o intenziteti odbitega laserskega žarka pri skeniranju površja. Tako smo se odločili, ker originalni podatki ne vsebujejo podatkov o barvi posamezne točke. Nadalnje delo bi tako lahko vključevalo dodajanje barvne informacije posamezni točki ter izračun normal za osvetlitev, s čimer bi aplikacijo naredili privlačnejšo za uporabo.

## References

[ars] Arso. http://gis.arso.gov.si/evode/profile.aspx?id=atlas_voda_Lidar@Arso. Dostopano 12.1.2017. 1

[git] Github. https://github.com/mitjari/GrafikaSeminar.git. Dostopano 12.1.2017. 3

[LASa] Las specifications. http://www.asprs.org/wp-content/uploads/2010/12/LAS_1_3_r11.pdf. Dostopano 12.1.2017. 1

[LASb] Laslib. https://github.com/LAStools/LAStools/tree/master/LASlib. Dostopano 12.1.2017. 2

[pak] Pako. https://nodeca.github.io/pako/. Dostopano 12.1.2017. 2

[pot] Potree. http://www.potree.org/. Dostopano 12.1.2017. 1

[thr] Three.js. https://threejs.org/. Dostopano 12.1.2017. 2

[zli] Zlib. https://zlib.net/. Dostopano 12.1.2017. 2