Sadržaj

Uvod

Tijekom zadnjih nekoliko godina, mogli bismo primjetiti kako je došlo do brojnih promjena u svijetu. Jedna od najznačajnijih promjena jest nagli razvoj tehnologije. Kroz zadnjih dvadesetak godina izašlo je nekolicina stvari o kojima prije nismo uopće razmatrali da su moguće. Razvoj tehnologije, potaknuo je nadalje povećanu proizvodnju i brzinu stvaranja novih inovacija što je uvelike utjecalo na razvoj raznih tržišta pa tako i automobilske industrije.

Te sve tvrdnje, može nam potvrditi činjenica kako je u Hrvatskoj, u razdoblju od 2012. do 2022. godine broj osobnih automobila na tisuću stanovnika povećan za 44,8 posto. To je trend koji prati cijelu Europsku uniju pa i veći dio svijeta. Nadalje se postavlja pitanje gdje će i kako automobili putovati na prometnicama, kako regulirati promet da bi se izbjegle gužve.

Za ljude koji žive i rade u Zagrebu i većim mjestima, uvjeren sam kako su barem jednom zapali u gužvu na nakoj od prometnica. Sada se postavlja pitanje možemo li mi promijeniti određene prometne znakove i pravila na raskrižjima i prometnicama kako bismo ne nužno optimizirali tok prometa, već kako bismo uzeli u obzir neke od parametara o kojima se nije razmišljalo prilikom izrade prometnica i određivanja prometnih pravila na tim prometnicama.

Upravo to je cilj ovog rada; uzeti dio gradske mreže te pokušati implementirati algoritam kontrole toka koji će uzimati u obzir nekoliko ključnih parametara za raspoređivanje pravila na raskrižju te izraditi jednostavnu simulaciju koja bi nam omogućila vizualizaciju toka na tom dijelu mreže.

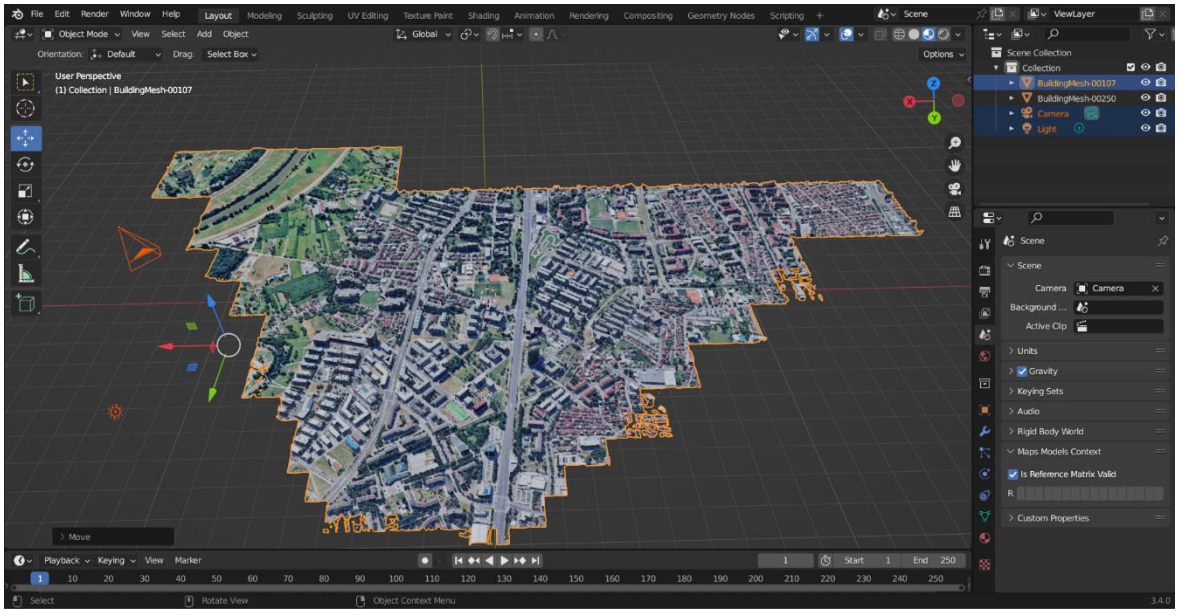
Rad se sastoji od učitavanja modela gradske mreže u Blender, izrade i označavanja prometnica, prebacivanje modela u Unity te zatim izrade algoritma kontrole toka i implementacija nad modelom u Unity-u. Kroz ovaj rad dati ću i opis pojedinih dijelova programskog koda te svoj zaključak o potrebi, mogućnosti i uspješnosti razvoja takvog algoritma i programa.

1. Opis aplikacije

Kao što je već napomenuto u uvodu, aplikacija se sastoji od modela gradske mreže učitanom putem google maps API-ja, algoritma kontrole toka i same simulacije kretanja objekata kao vozila kroz mrežu. Kroz ovu cjelinu, proći ćemo kroz Blender i opisati postupak kojim umetnuli google maps područje grada u sam Blender te neke mogućnosti koje smo iskoristili za daljnje oblikovanje.

1.1. Modeli

Modeli u ovom projektu sastoje se od više dijelova. Prvo je model gradske mreže učitan sa google maps-a. Zatim imamo modele cesta koji su ručno izrađeni u Blender-u kako bismo mogli izvoditi simulacije. Osim toga imamo i kontrolne točke koje predstavljaju pojedina raskrižja na pojedinim trakama ili prometnicama te ulazne i izlazne točke koje predstavljaju početak i kraj vožnje nekog od vozila. Potom, kako bismo mogli simulirati izvođenje na prometa na tim djelovima mreže imamo i modele vozila koji će u ovom slučaju za potrebe simulacije biti primitivni modeli.



Slika 1.1. Model dijela gradske mreže u Blender-u

1.2. Algoritam kontrole toka

Kao bazu algoritma koristimo Ford-Fulkersonov algoritam pronalaska najvećeg toka kroz graf. Kroz Ford-Fulkersonov algoritam, potrebna nam je i implementacija Dijkstrinog algoritma. U nastavku ovog dijela pojasniti ćemo bazne algoritme kroz primjer implementacije u C++ programskom jeziku.

Prvo čemo opisati dijkstrin algoritam koji smo preoblikovali za potrebe implementacije u Ford-Fulkersonov algoritam. U nastavku priložen je dio koda s modificiranim dijkstrinim algoritmom.

```
void modificirana_dijkstra(int n, vector<int>& tok,
vector<vector<int> >& graf, vector<int>& roditelj){
    vector<int> bio(n, 0); // vektor za provjeru prolaska
    for (int i = 0; i < n - 1; i++){
        int tko, koliko = -1;
        for (int j = 0; j < n; j++){
            if (!bio[j] && tok[j] > koliko){
                tko = j;
                koliko = tok[j];
            }
        }
        bio[tko] = 1;
        for (int j = 0; j < n; j++){ //provjera toka i odabir puta
            if (min(koliko, graf[tko][j]) > tok[j]){
                tok[j] = min(koliko, graf[tko][j]);
                roditelj[j] = tko;
            }
        }
    }
}
```

Ako poznajemo standardni Dijkstrin algoritam, onda znamo kako svrha Dijkstrinog algoritma je pronalazak najkraćeg puta na nekom grafu i to na način da zapisuje duljine putova do čvora te uvijek kao sljedeći čvor s kojim počinje je onaj na kojem je duljina puta do tog čvora najkraća.

Razlika sa našim modificiranim Dijkstrinim algoritmom je da se pamti brid najlakše težine na trenutnom putu te se provjerava je li moguće preko ovog brida doći do konačnog čvora tako da tok bude najveći.

U nastavku, prikazan je dio koda za Ford Fulkerson algoritam u c++ programskom jeziku koji se veže na gore dani Dijkstrin algoritam.

Algoritam radi na način da se u while petlji svakim prolaskom pomoću dijkstre izračunava tok, na način da se odabire brid tako da je na odabranom putu najlakši brid što teži te potom na čvorove grafa, puta koji smo odabrali oduzimamo vrijednosti toka grafa, dok na inverzne čvorove dodajemo tu količinu toka.

Kao što sam već napomenuo, ovo je primjer baznih algoritama koji nam služe za izradu konačnog algoritma koji je u C# programskom jeziku te sami dokazi ovih i algoritama koji su izvedeni iz ovih, biti će u budućim poglavljima kao i objašnjenje konačnog koda za tok kroz graf.

```
int ford_fulkerson(int n, int m, int start, int end,
vector<vector<int> >& graf){
    const int inf = 1e9;
    int networkFlow = 0;
    vector<int> roditelj(n, 0);
    vector<int> tok;

    while(1){
        tok.clear();
        tok.insert(tok.begin(), n, 0);
        tok[start]=inf;

        modificirana_dijkstra(n, tok, graf, roditelj);

        int flow = tok[end];
        if (flow == 0) break;
        networkFlow += flow;

        int tko = end;

        while( tko != start){
            graf[roditelj[tko]][tko] -= flow;
            graf[tko][roditelj[tko]] += flow;
            tko = roditelj[tko];
        }
    }
    return networkFlow;
}
```

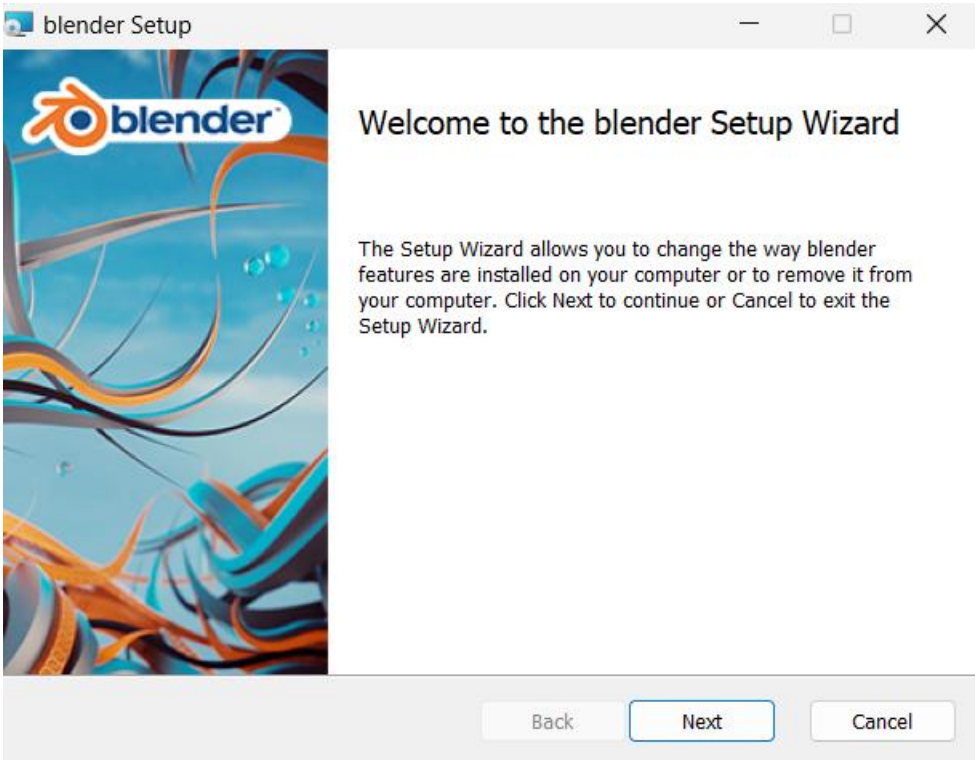
2. Instalacija potrebne programske podrške

Za instalaciju osim nekih poznatih nam alata potrebno je i instalirati neke posebne biblioteke korištene u ovom projektu. Kroz sljedećih par podcjelina opisati ću što je sve potrebno instalirati te neke napomene vezane za instalaciju.

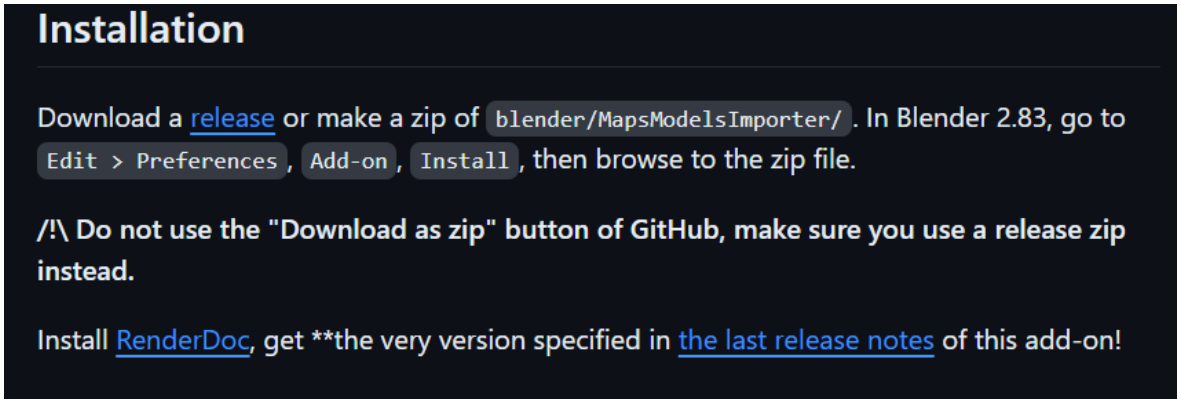
Za potrebe ovog projekta, nije moguće instalirati najnoviju verzije alata već nam je potrebno uzeti u obzir verzije alata koje su usklađene sa dodatnim ekstenzijama i bibliotekama koje koristimo.

2.1. Instalacija Blender-a i dodatka za Blender

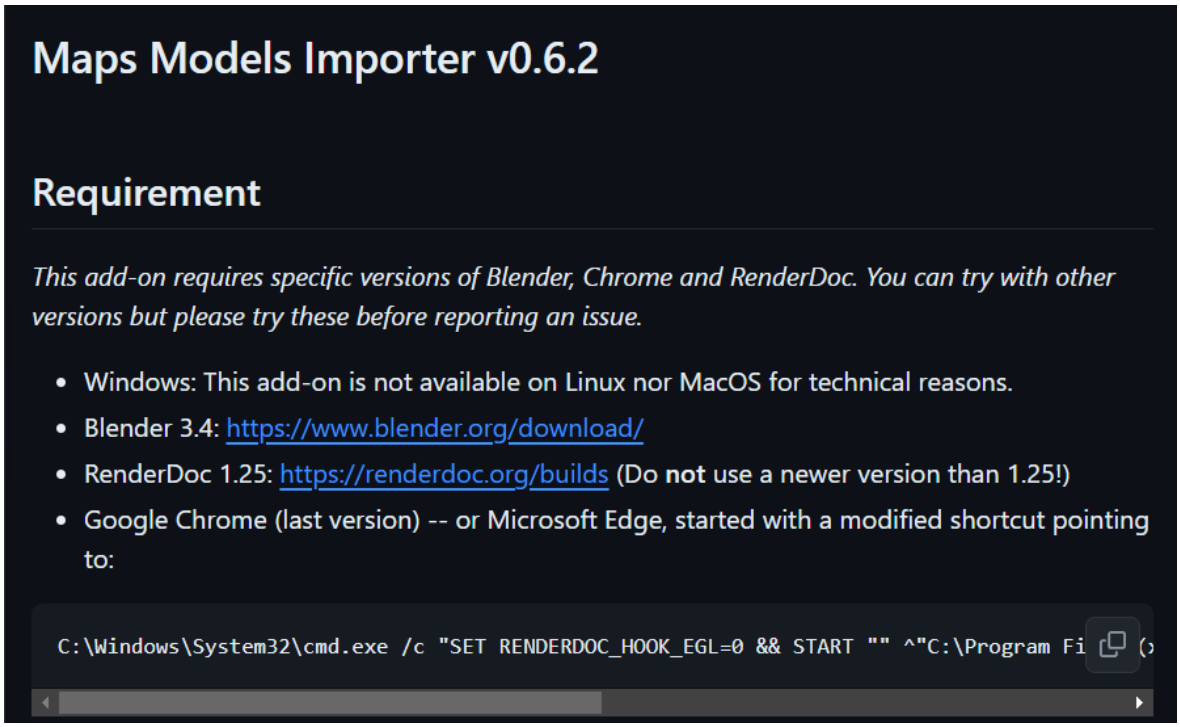
Kao što je navedeno, nećemo koristiti najnoviju verziju Blendera već ćemo koristiti verziju 3.4. Za instalaciju prethodnih verzija Blendera, idemo na sljedeći link: https://www.blender.org/download/previous-versions/ , pritisnemo „Download Any Blender“ i ondje odaberemo „Blender 3.4“ te potom za odaberemo verziju 3.4.0 za jedan od ponuđenih operacijskih sustava koje imamo. U našem slučaju to je windows. Pri završetku skidanja instalatora, pokrenemo ga te zadržimo sve standardne opcije.



Po završetku instalacije, potrebno je instalirati zasebnu ekstenziju sa githuba https://github.com/eliemichel/MapsModelsImporter/tree/master. Ondje idemo dolje i kliknemo na release.



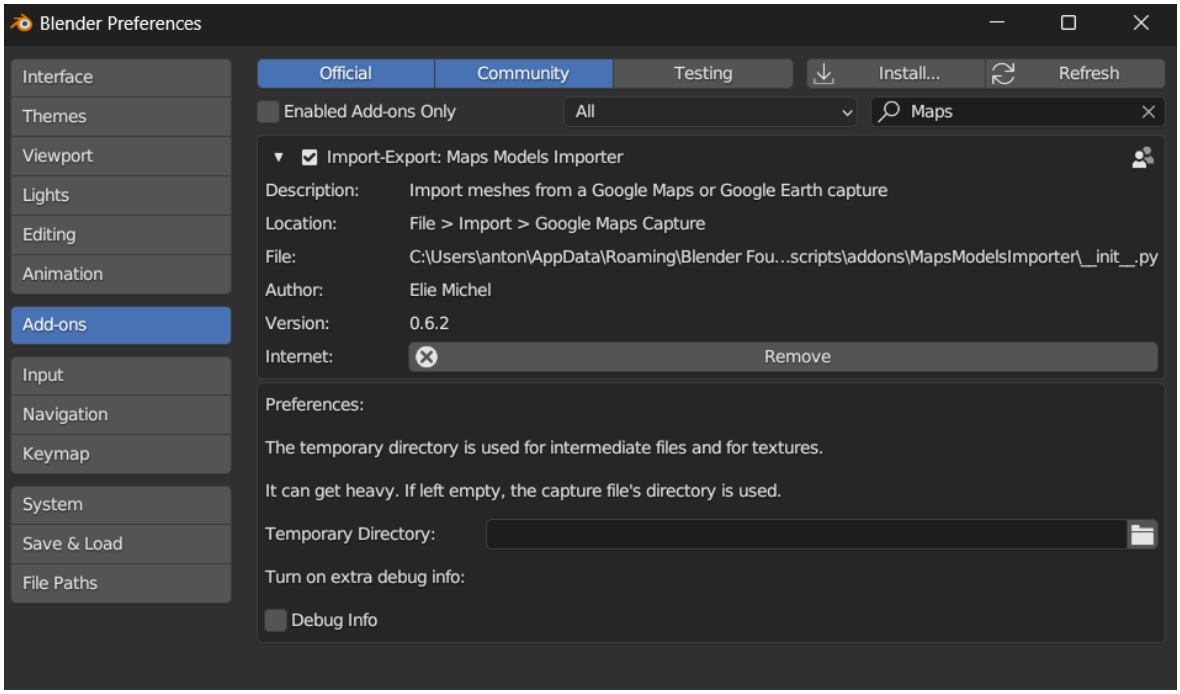
Nadalje, ponovno idemo dolje dok ne vidimo sljedeći zaslon.



Tada odaberemo na Assets i skinemo datoteku MapsModelsImporter-v0.6.2.zip.

Po završetku instalacije otvorimo Blender i u projektu idemo na Edit → Preferences → Add-ons → Install → Odaberemo skinutu datoteku → Install Add-on.

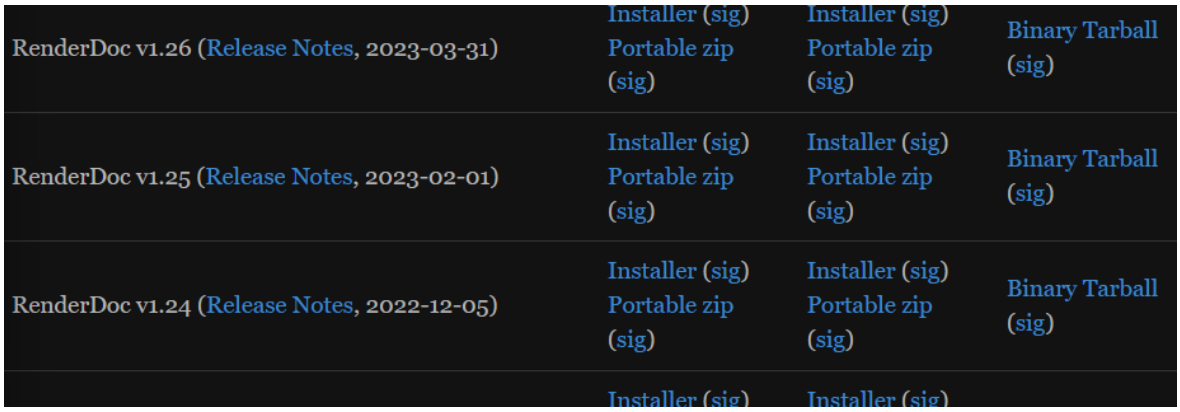
Nakon što smo to učinili idemo na Preferences i pretražimo taj add-on i označimo kučicu.



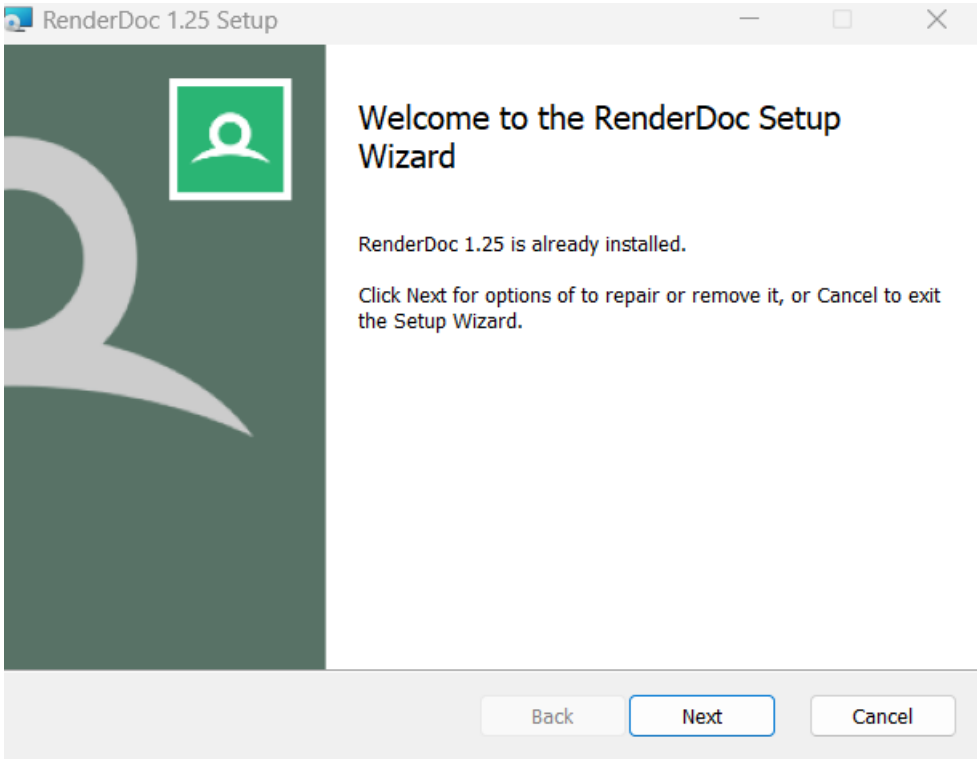
Nakon što smo to odradili Blender je spreman za rad.

2.2. Instalacija RenderDoc-a

Za RenderDoc, potrebno je instalirati verziju RenderDoc v1.25. Za instalaciju idemo na https://renderdoc.org/builds#stable , idemo prema dolje dok ne dođemo do verzije 1.25 kao u prikazu prema slici.



Zatim odaberemo jedan od ponuđenih operacijskih sustava te skinemo installer. Po završetku otvori nam se installer gdje dalje pritisnemo dalje i standardnu instalaciju.



Zaključak

Na kraju rada piše se kratak zaključak, duljine do najviše jedne stranice.

Literatura

[1] Napredno programiranje i algoritmi u C-u i C++-u, Domagoj Kusalić, 5. nepromijenjeno izdanje, Zagreb, 2014.

Sažetak