Vilniaus universitetas

Fizikos fakultetas

Taikomosios elektrodinamikos ir telekomunikacijų institutas

Steponas Pilkauskas

# Trimačių vektorinių duomenų taikymas daiktų interneto tinklų planavimui

BAKALAURO STUDIJŲ BAIGIAMASIS DARBAS

Elektronikos ir telekomunikacijų technologijų
studijų programa

| | |
|---|---|
| Studentas | Steponas Pilkauskas |
| Leista ginti: | 2023-05-25 |
| Darbo vadovas | doc. Rimvydas Aleksiejūnas |
| Instituto direktorius | prof. Robertas Grigalaitis |

Vilnius, 2023

# Turinys

Įvadas ..... 3

1 Literatūros apžvalga ..... 4

1.1 Laisvos erdvės sklidimas ..... 4

1.2 Hata modelis ..... 4

1.3 Atspindžiai ir difrakcija ..... 5

2 Sklidimo modelių įgyvendinimai ..... 6

2.1 Hata Ericsson 9999 modeliavimas ..... 6

2.2 Tiesioginio matomumo skaičiavimai ..... 6

2.3 Difrakcijos skaičiavimai ..... 9

2.4 Kombinuotas sklidimo modelis ..... 11

3 3D duomenų paruošimas ..... 13

3.1 CityJSON duomenų struktūra ..... 13

4 Vektorinis XDraw difrakcijos modelis ..... 16

5 Greitaveikos bandymai ..... 19

6 XDraw modelio taikymas daiktų interneto tinklų analizei ..... 22

Išvados ..... 24

Šaltiniai: ..... 25

Priedai ..... 26

# Įvadas

Daiktų internetas ( angl. Internet of Things, IoT ) yra svarbi 5G kartos ryšio technologijų dalis. Daiktų interneto technologijos, tokios kaip LoRa, SigFox, NB – IoT, leidžia tūkstančiams baterijomis maitinamų įrenginių porą kartų per dieną siųsti nedidelį kiekį duomenų (pavyzdžiui sensorius dirvos rūgštingumo vertę) į vieną stotį[8]. Privalumai – baterijas įrenginiams reikia keisti kas keletą metų, vienos stoties aprėpties spindulys skaičiuojamas kilometrais. Pavyzdžiui LoRa atstumo rekordas tarp taškų žemėje - virš 200km[1], tikėtinas aprėpties spindulys tarp 3-15km, kitų minėtų technologijų irgi panašūs atstumai, priklauso nuo kliūčių. Miesto scenarijuje dėl pastatų aukščio ir tankumo nuostoliai didesni. Kai aprėpties atstumų skirtumai tokie didžiuliai, sėkmingai plėtrai užtikrinti reikia tikslių ir greitų metodų radijo signalų nuostolių apskaičiavimui. Prieš du metus Nyderlandų Delfto technikos universiteto 3D geoinformacijos tyrimų grupė išleido visų Nyderlandų pastatų vektorinius 3D duomenis[2], kurie leis dar tiksliau apskaičiuoti nuostolius, nei senesni, rastriniai duomenys. Skaičiavimų greitį padidinti leis XDraw algoritmas, kurį pritaikysime vektoriniams duomenims. Galiausiai aptarsime keletą IoT tinklų scenarijų galimybių.

Šio darbo tikslas yra pritaikyti tikslius 3D vektorinius pastatų duomenis radijo bangų difrakcijos modeliui, tinkamam IoT tinklų planavimui. Darbe naudojama Jupyter Notebooks programavimo aplinka, Python 3.10 paketas.

# 1 Literatūros apžvalga

Šiame skyriuje susipažinsime su naudotais radijo bangų sklidimo modeliais.

## 1.1 Laisvos erdvės sklidimas

Laisvos erdvės sklidimas ( angl. Free space propagation ) – tai paprasčiausias sklaidos modelis, naudotinas kai tarp imtuvo ir siųstuvo nėra kliūčių, yra tiesioginis matomumas ( angl. Line of Sight, LoS ). Šiame modelyje elektromagnetinės bangos sklinda iš siųstuvo į visas puses tiesiomis linijomis. Signalo galia priklauso nuo atstumo tarp siųstuvo ir imtuvo *d* dėsniu $\frac{1}{d^2}$ .

Perduota signalo galia gaunama panaudojus Friso formulę ( angl. Friis transmission equation )(1.1) [3]:

$$P_i = \frac{P_s G_s G_i \lambda^2}{(4\pi d)^2} \tag{1.1}$$

Čia $P_i$ – galia perduota imtuvui, $P_s$ – siųstuvo išspinduliuoto signalo galia, $G_s$ ir $G_i$ – siųstuvo ir imtuvo antenų stiprinimai, λ – bangos ilgis.

## 1.2 Hata modelis

Hata sklidimo modelis – tai empirinis modelis, į kurį įskaičiuoti atspindžių, difrakcijos efektai[3,4]. Yra daug modelio versijų, mes pasirinkome Ericsson 9999 modelį, kuris naudojamas radijo dažniams iki 1900 MHz. Hata modeliai paprasti ir lengvai koreguojami – formulėje esančius koeficientus galima koreguoti pagal vietovės ypatybes.

Hata Ericsson 9999 nuostolių ( angl. Path Loss ) formulė[4]:

$$PL = a_0 + a_1 \log_{10}(d) + a_2 \log_{10}(h_s) + a_3 \log_{10}(h_s) \log_{10}(d) - 3{,}2(\log_{10}(11{,}75))^2 + 44{,}49 \log_{10}(f) - 4{,}78\,(\log_{10}(f))^2 \tag{1.2}$$

Čia $h_s$ – siųstuvo aukštis, $f$ – dažnis.

Įprastiniai koeficientai $a_0 = 36{,}2$ ; $a_1 = 30{,}2$ ; $a_2 = -12$ ; $a_3 = 0{,}1$

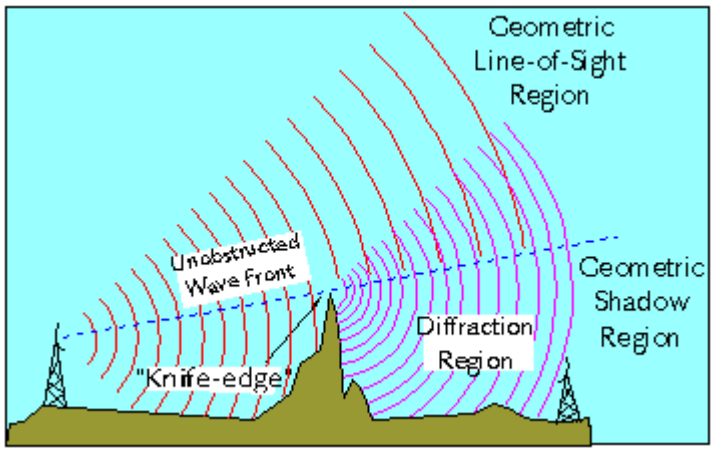
## 1.3 Atspindžiai ir difrakcija

Kai radijo banga keliaudama sutinka kliūtį, jai gali nutikti keli dalykai:

1. Dalinis ar pilnas atspindys
2. Difrakcija

Atspindžiai nutinka kai banga kelyje sutinka kokią nors lygią plokštumą, pavyzdžiui dangoraižių stikliniai langai. Laidžių medžiagų plokštumos geriau atspindi, mažesni signalų nuostoliai.

Difrakcija arba peilio ašmenų difrakcija ( angl. knife-edge diffraction ), yra efektas, kai bangos užsilenkia už reliatyviai smailių kliūčių kampų ( žr. 1.1 pav. ). Ją nusako Hoigenso-Frenelio ( angl. Huygens-Fresnel Principle ) modelis[3,7].



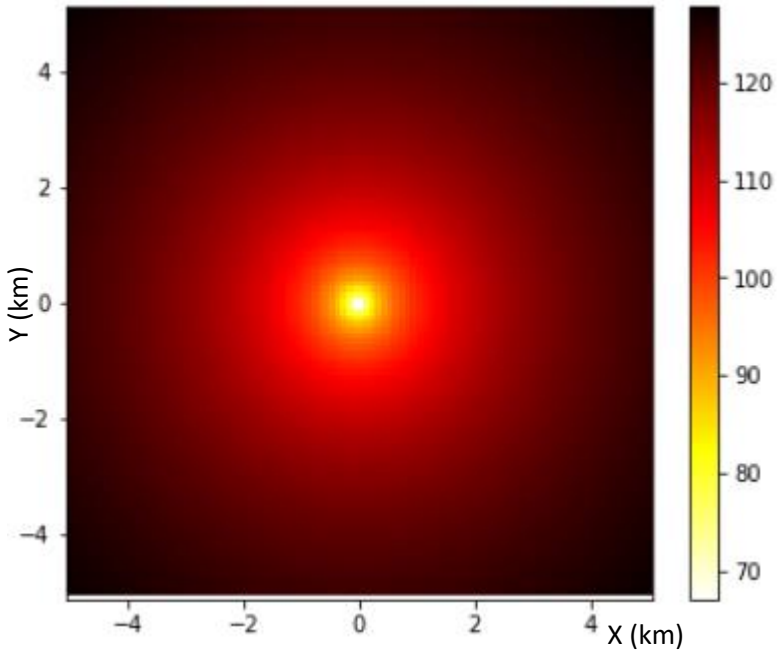

1.1 pav. Peilio ašmenų difrakcija [9]

# 2 Sklidimo modelių įgyvendinimai

Šiame skyriuje susipažinsime su naudotais algoritmais, kuriuos naudojome simuliuodami anksčiau aptartus sklaidos modelius.

## 2.1 Hata Ericsson 9999 modeliavimas

Naudodami NumPy biblioteką sukūrėme atstumo rastrą ( angl. raster ), pagal Hata nuostolių formulę (1.2) ir atstumų rastrą gauname Hata nuostolių rastrą ( žr. 2.1 pav. ).




2.1 pav. Hata nuostolių rastras, nuostoliai decibelais (f= 870 MHz (ISM juosta))

## 2.2 Tiesioginio matomumo skaičiavimai

Tiesioginio matomumo skaičiavimams buvo naudojami du būdai:

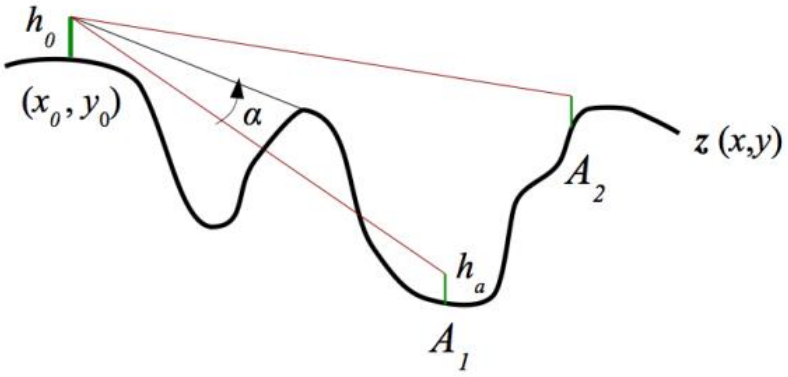
1. R3 algoritmas
2. XDraw algoritmas

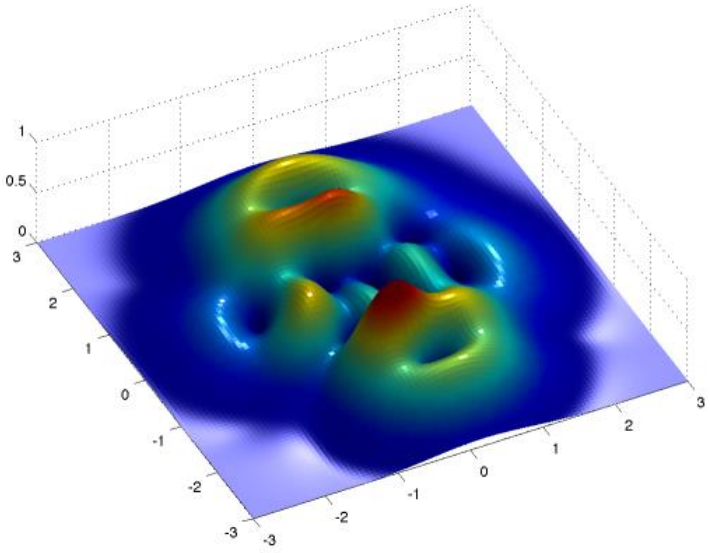
R3 algoritmas [5] – algoritmas, randantis, kurios vietos bus tiesiogiai matomos stočiai, pastatytai konkrečioje vietoje, kiekvieną tašką skaičiuojantis iš naujo, todėl vadinamas nepernaudojamu ( angl. nonreusable algorithm ). Šio tipo algoritmai – truputį tikslesni, bet daug lėtesni, jų trukmė nuo taškų skaičiaus priklauso trečiuoju laipsniu. Veikimo principas ( žr. 2.2 pav. ) – atkarpoje tarp siųstuvo ir imtuvo skaičiuojamas kampas $\alpha$ tarp siųstuvo-imtuvo ir siųstuvo-kliūties, jeigu kliūtis

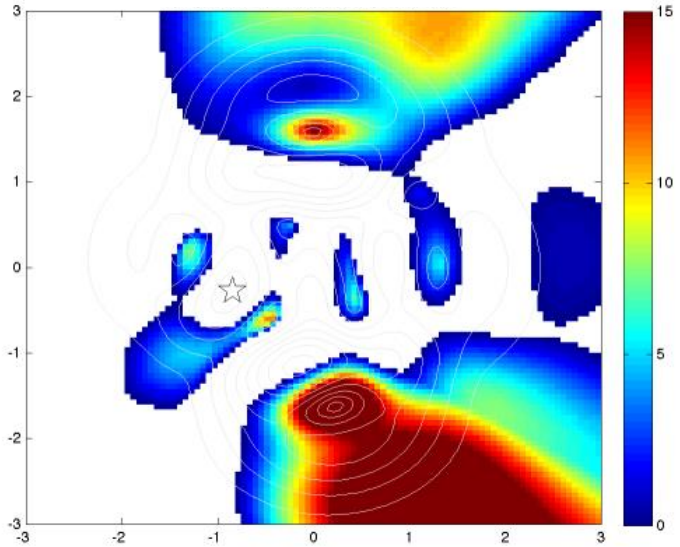
uždengia imtuvo poziciją – $\alpha > 0$, taškas įgauna $\alpha$ vertę. Jei kliūtis imtuvo neuždengia – $\alpha \leq 0$, taškas gauna vertę 0, tiesioginis matomumas yra.

$A_1$ = Užmaskintas, nematomas taškas $\alpha > 0$

$A_2$ = Matomas taškas ($\alpha \leq 0$)



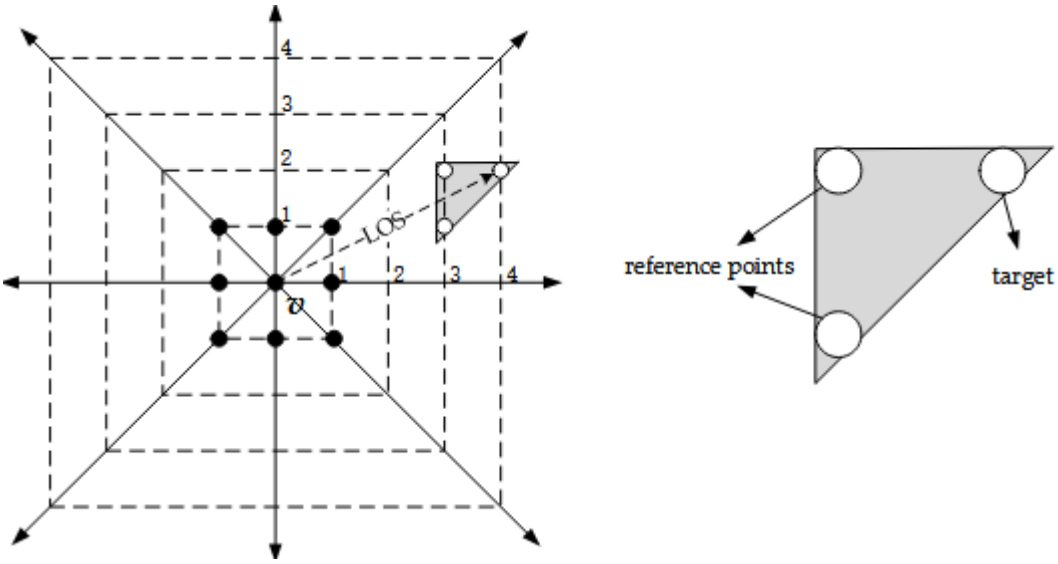


2.2 – 2.4 pav. R3 algoritmas, užduota topologija ir pavyzdinis rezultatas [5].

Išbandėme daug realizacijos variantų:

a. Numba yra kompiliatorius, išverčiantis Pythono ir NumPy kodą į greitą mašinų kodą.
b. Cython yra programavimo kalba, leidžianti kartu naudoti Python ir C kalbas, o paleidus kodą, viską išverčia į greitesnį C kodą.
c. Pandas yra greita, duomenų analizei pritaikyta Python biblioteka

XDraw algoritmas pradeda skaičiuoti matomumą nuo artimiausių siųstuvui taškų, vėliau pernaudodamas rezultatus tolimesnių taškų skaičiavimams, todėl yra vadinamas pernaudojamu ( angl. reusable algorithm ) [6,10]. Pernaudojami algoritmai ne tokie tikslūs kaip nepernaudojami, bet yra daug greitesni – jų trukmė priklauso nuo taškų skaičiaus antruoju laipsniu. Veikimo principas – rastras su siųstuvu centre yra suskirstomas į 8 dalis, šių dalių skaičiavimai visiškai atskiri ir nepriklausantys nuo kitų dalių. Pirmiausia priimama prielaida kad artimiausi siųstuvo 8 taškai yra matomi ir suskaičiuojamas matomumas R3 algoritmo būdu pagrindinėse 8 linijose kurios kertasi centre ir dalina visą rastrą į 8 dalis (vertikaliai, horizontaliai ir įstrižai). Pradedama skaičiuoti matomumą iki artimiausio rato taškų aplink siųstuvą. Jas suskaičiavus imamas didesnis žiedas, jo taškų matomumai skaičiuojami pagal anksčiau suskaičiuotų dviejų taškų matomumą( žr. 2.5 pav. ) naudojant IDW ( angl. Inverse Distance Weighting ) interpoliaciją[12]. Dėl atskirų skaičiavimų atskiroms rastro dalims, galima efektyviai paralelinti skaičiavimus ir taip juos dar pagreitinti.




2.5 pav. XDraw algoritmo skaičiavimo procedūra [10]

$$u(x) = \begin{cases} \frac{\sum_{i=1}^{N} w_i(x) u_i}{\sum_{i=1}^{N} w_i(x)} & \text{kai } d(x,x_i) \neq 0 \\ u_i & \text{kai } d(x,x_i) = 0 \end{cases}, \quad (2.1)$$

$$w_i(x) = \frac{1}{d(x,x_i)^p} \quad (2.2)$$

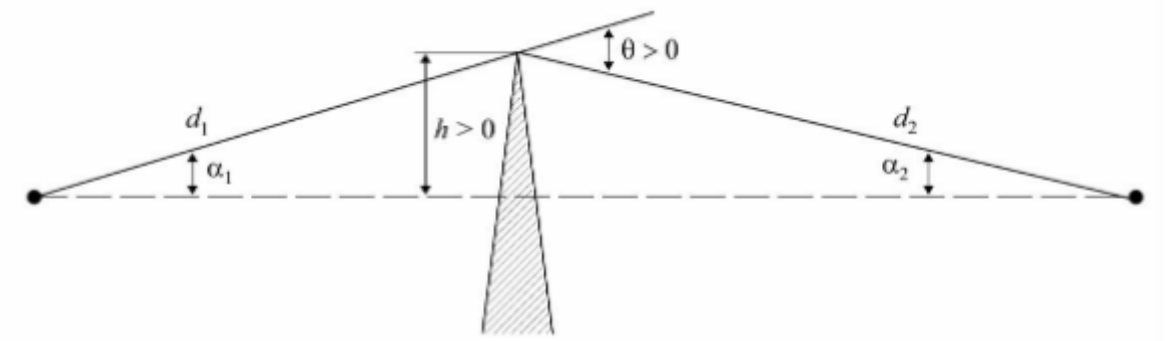
IDW interpoliacija[12], čia $u$ – ieškomas taškas, $u_i$ – turimi taškai, $d$ – atstumas tarp ieškomo ir turimo taškų, $p$ – kelimo laipsniu parametras.

## 2.3 Difrakcijos skaičiavimai

Skaičiuojant difrakcijos nuostolius, dėl paprastumo priimsime prielaidą kad visos kliūtys siaurėja iki peilio ašmenų siaurumo. Naudodami vieno peilio ašmenų kliūties modelį ( angl. single knife-edge obstacle ) [3,7], tereikia kiekvienam signalui surasti aukščiausios kliūties aukštį virš linijos, jungiančios siųstuvą ir imtuvą $h$ ir atstumus iki siųstuvo ir imtuvo $d_1$ ir $d_2$. Šis paprastas modelis leidžia naudoti formulę:

$$v = h\sqrt{\frac{2}{\lambda}\left(\frac{1}{d_1} + \frac{1}{d_2}\right)} \quad (2.3)$$

Čia $v$ – bedimensinis parametras, nusakantis įvairių formų apibendrintą įtaką difrakciniams nuostoliams.




2.6 pav. Difrakcijos skaičiavimų parametrai [7]

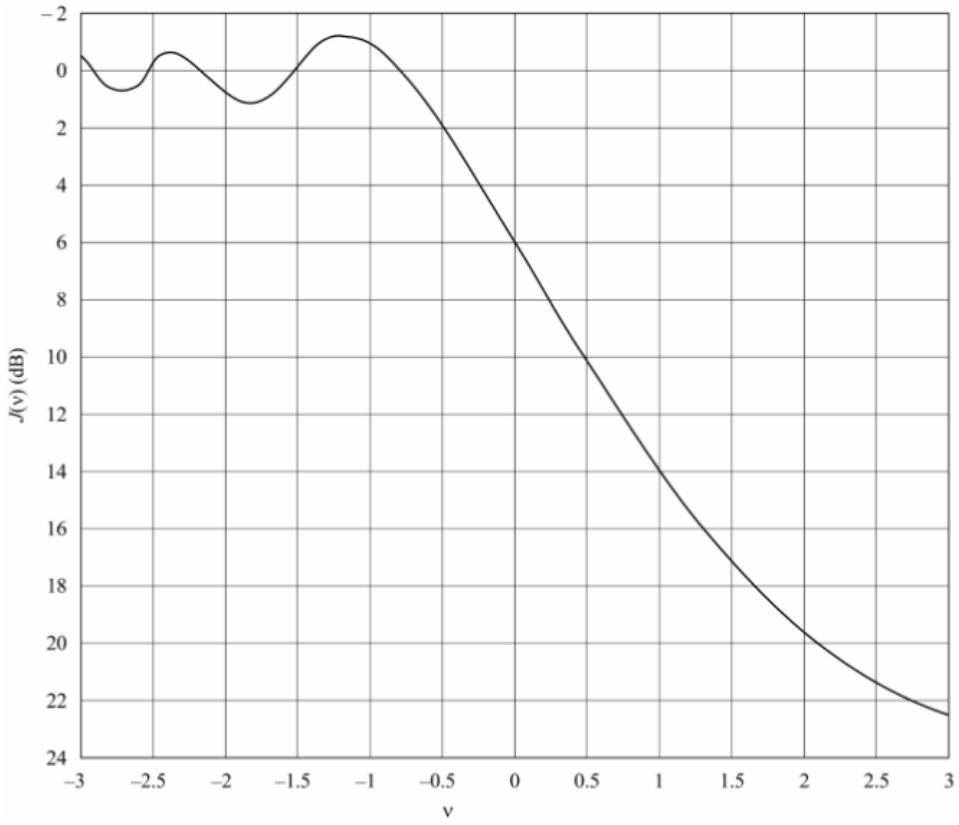
Tada difrakcijos nuostoliai bus:

$$J(v) = -20\log_{10}\left(\frac{\sqrt{(1-C(v)-S(v))^2+(C(v)-S(v))^2}}{2}\right) \quad (2.4)$$

Čia $C(v)$ ir $S(v)$ – Frenelio ( Fresnel ) integralai.

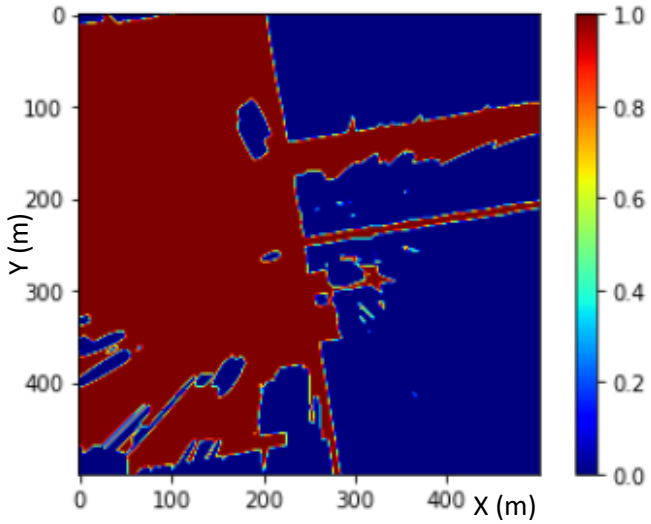
$$C(v) = \int_0^v \cos\left(\frac{\pi s^2}{2}\right) ds \quad (2.5)$$

$$S(v) = \int_0^v \sin\left(\frac{\pi s^2}{2}\right) ds \tag{2.6}$$

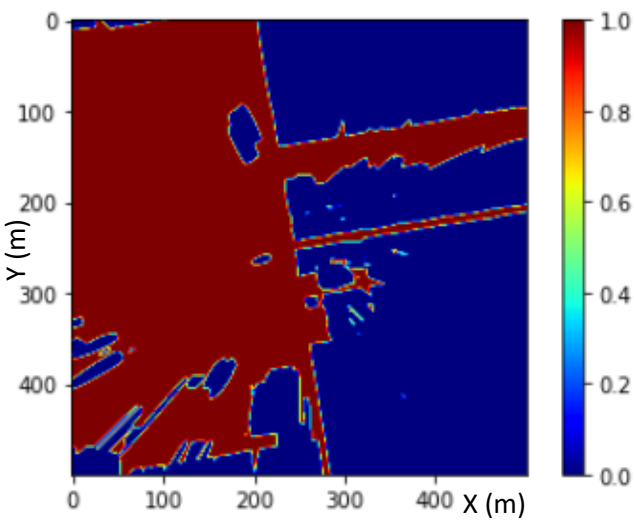



2.7 pav. Difrakcinių nuostolių priklausomybė nuo *v* [7]

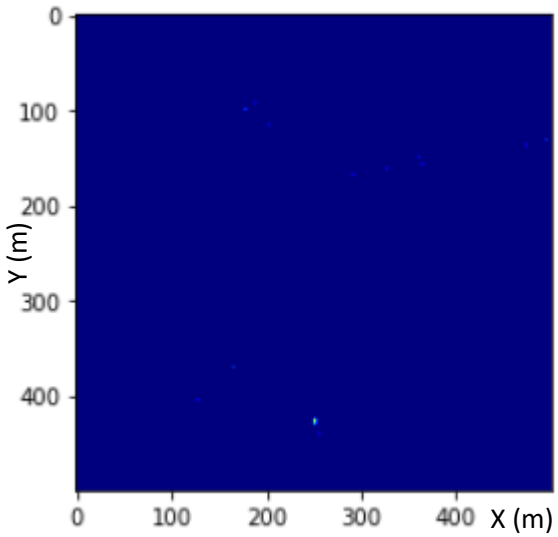
Čia matome ( žr. 2.7 pav. ) , kad ties 6dB nuostoliais *v* lygus 0, tai reiškia kad kliūtis neiškyla virš linijos, einančios tiesiai nuo siųstuvo iki imtuvo. Taigi difrakcijos nuostolių rastre visi taškai su mažesniais nei 6dB nuostoliais yra tiesiogiai matomi.




2.8 pav. su XDraw apskaičiuotas matomumas



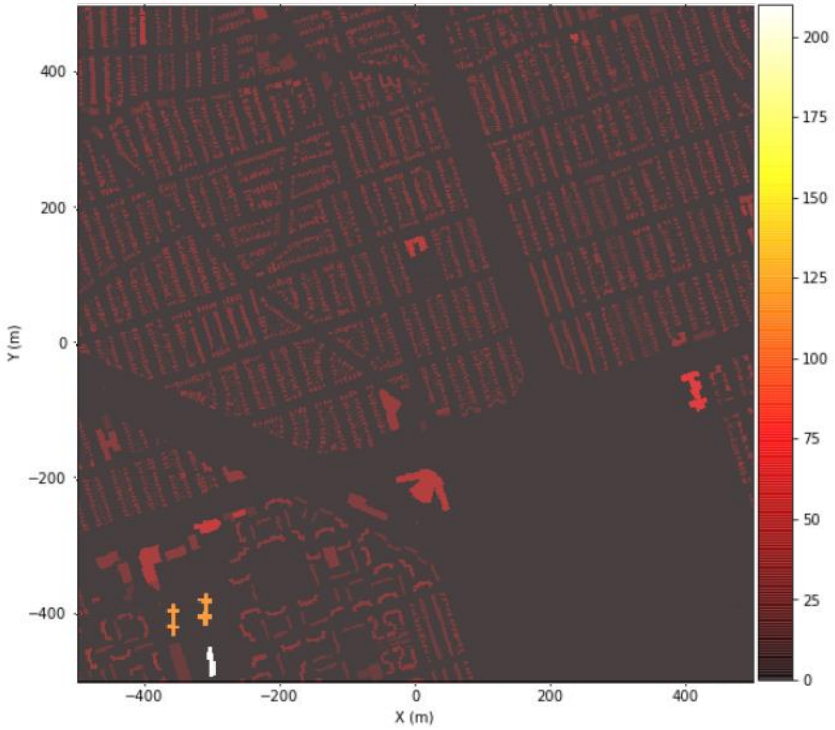

2.9 pav. *J*(*v*) < 6dB rastras



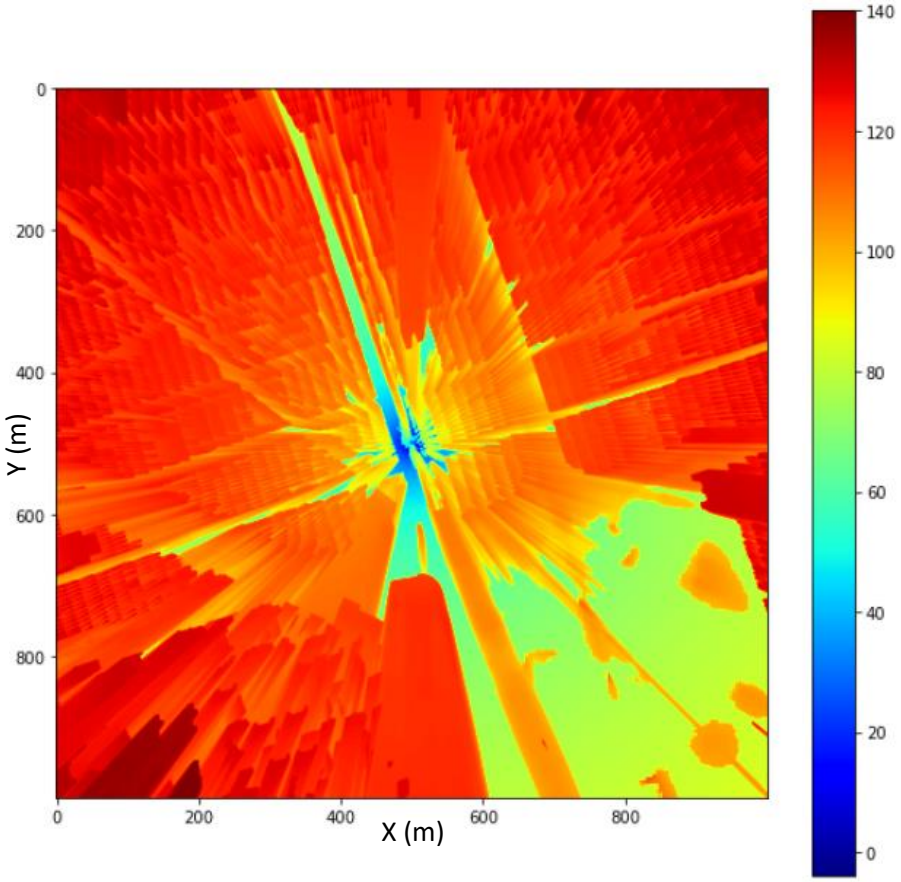

2.10 pav. Nevienodi taškai (24 iš 250 tūkst.)

## 2.4 Kombinuotas sklidimo modelis

Galutiniam modeliui naudojame kitų modelių kombinaciją – laisvos erdvės sklidimą kur yra tiesioginis matomumas, Hata Ericsson 9999 kartu su difrakcijos nuostoliais kur tiesioginio matomumo nėra. Naudojame iš difrakcijos skaičiavimų gaunamą matomumo rastrą.




2.11 pav. Pastatų rastras (aukštis metrais)

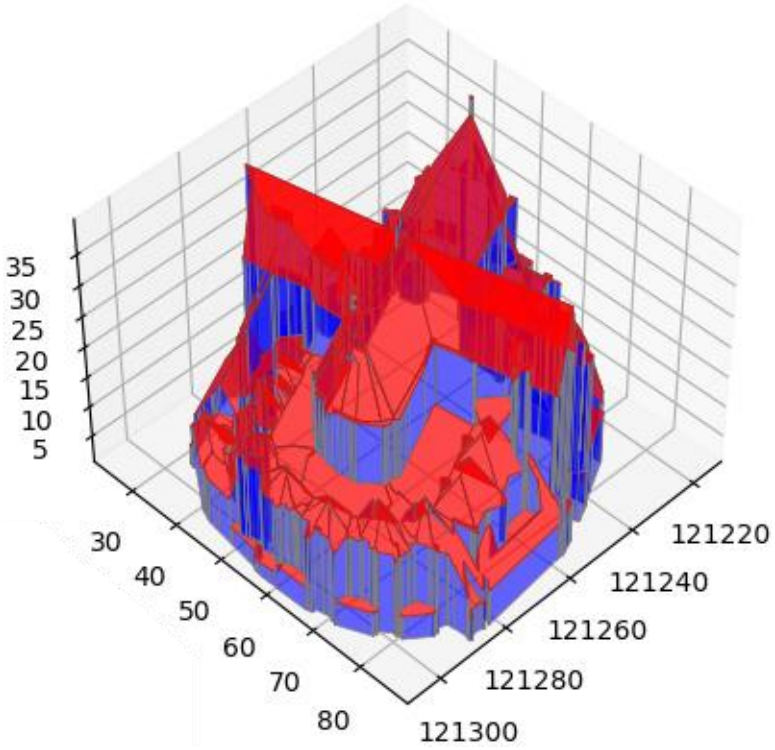



2.12 pav. Kombinuoto modelio rezultatas (nuostoliai decibelais)

# 3 3D duomenų paruošimas

Anksčiau aprašyti modeliai buvo pritaikyti rastriniams ( angl. raster ) Niujorko duomenims. Šie duomenys yra tiesiog didžiulė dvimatė matrica, kurios vertės nusako vieno 1mx1m „pikselio“ maksimalų aukštį, iš kurių susideda visos teritorijos topografija.

Toliau aptarsime tikslesnius, vektorinius duomenis. Tam panaudojome CityJSON formatu laisvai prieinamą Amsterdamo modelį[ŠALTINIS]. Vektoriniai duomenys reiškia, jog vietoj rastrinių duomenų „pikselių“, turime konkrečias sienų ir stogų plokštumas( žr. 3.1 pav. ). Mūsų naudojami Nyderlandų duomenys sudaryti ortofotografinių nuotraukų ir lazerinio skenavimo iš oro (LiDaR) kombinacija, duomenų tikslumas yra 30cm.




3.1 pav. De Nieuwe Kerk (Naujoji bažnyčia). Amsterdamo CityJSON duomenų vizualizacija (LoD2.2)

## 3.1 CityJSON duomenų struktūra

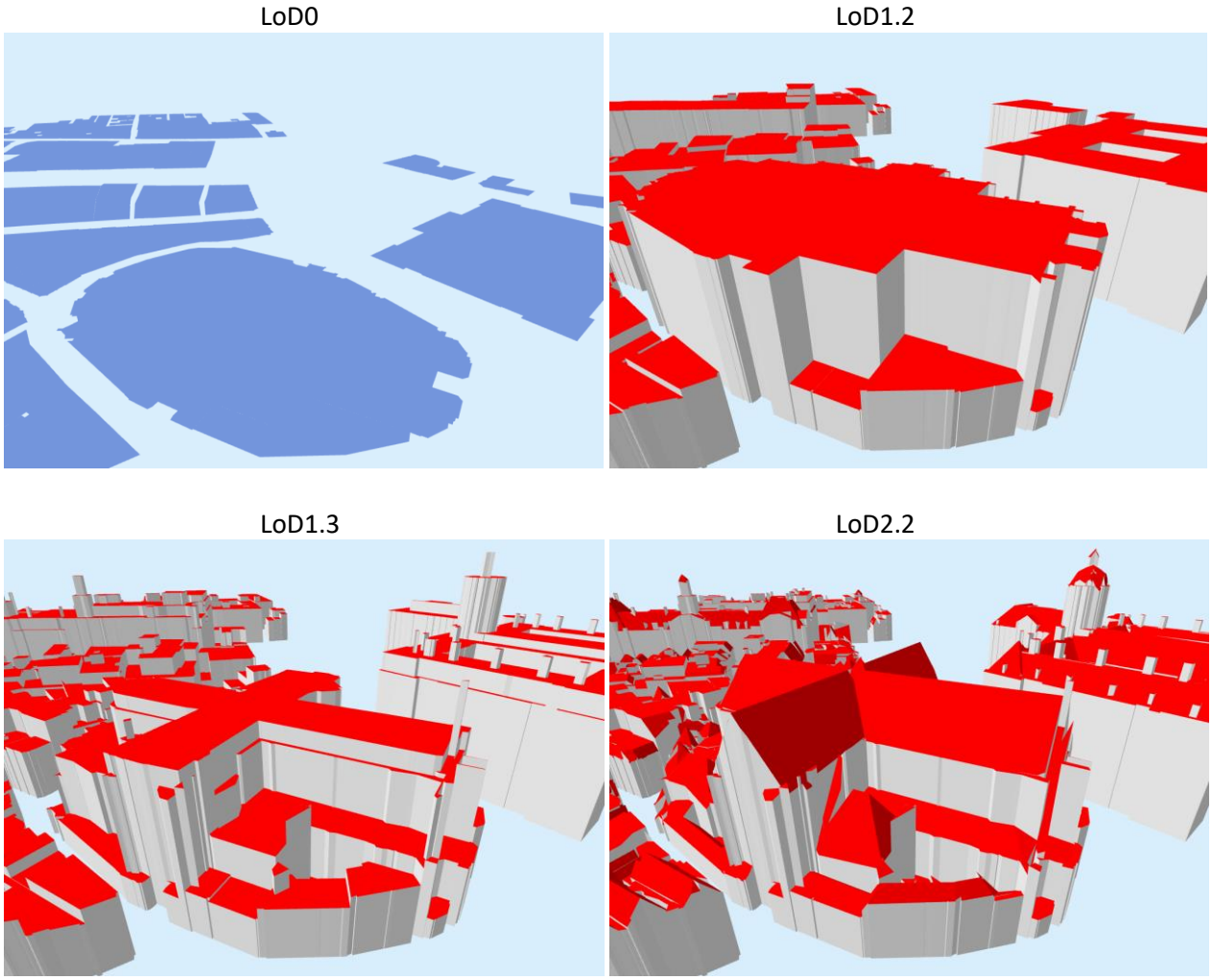
CityJSON yra JSON struktūros formatas, kuriame galima užrašyti daug įvairių objektų. Trumpai aptarsime failo struktūrą[11].

```
{
  "type": "CityJSON",
  "version": "1.1",
  "transform": {
    "scale": [1.0, 1.0, 1.0],
    "translate": [0.0, 0.0, 0.0]
  },
  "metadata": {},
  "CityObjects": {},
  "vertices": [],
}
```

```
"NL.IMBAG.Pand.0363100012165051-0": {
        "geometry": [{
            "boundaries": [[6, 7, 8, 9], [1, 2, 3]],
            "lod": 1.2,
            "semantics": {
              "surfaces": [{
                  "type": "GroundSurface"
                }, {
                  "type": "RoofSurface"
                }, {
                  "type": "WallSurface"
                }],
              "values": [[0, 2]]
            },
          }
```

3.2 pav. CityJSON failo struktūra

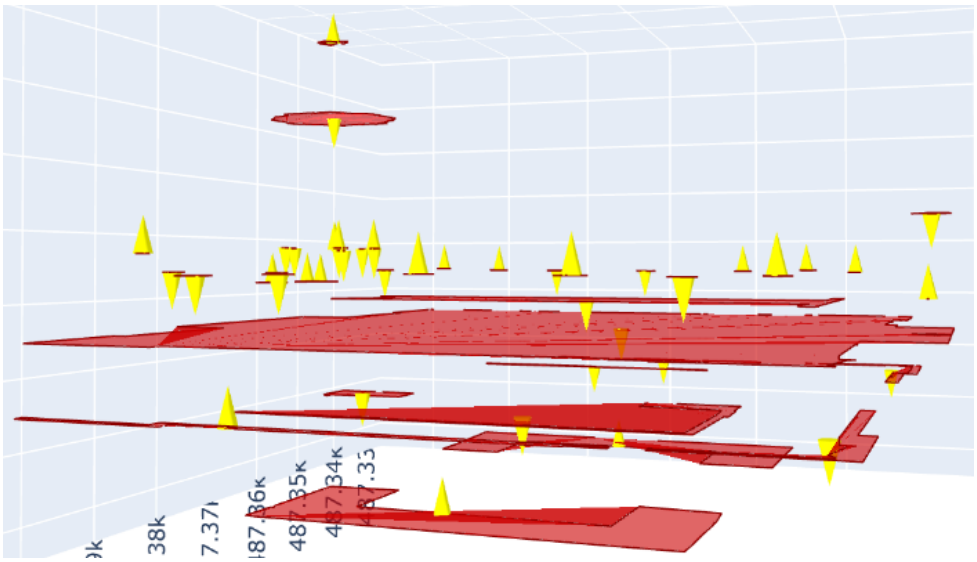
Kairėje matome minimalų **CityJSON Object**, dešinėje – mažą pavyzdinį **City Object**. Abiejuose gali būti ir daugiau narių, bet aptarsime tik mums svarbius. **CityJSON Object** viduje matome jo tipą, naudojamą CityJSON versiją, **metadata** naryje galima užrašyti duomenų autorių, atskaitos sistemą, kitus dalykus. **Vertices** naryje užduodami visi taškai, kuriuos naudosime šitame **CityJSON Object**. Vieną tašką sudaro trijų koordinačių masyvas, šiuos taškus naudosime dešinėje matomame **City Object boundaries** naryje, kuriame yra užduotos dvi plokštumos, pima turi keturis taškus, antra tris, taškai tiesiomis linijomis sujungti paeiliui ir paskutinis su pirmu. Plokštumos gali turėti ir sudėtingesnę geometriją, daugiau taškų, tie patys taškai gali būti naudojami keliose plokštumose. Vieną tašką čia aprašo tik vienas skaičius indeksas, kuris nurodo į **CityJSON Object vertices** naryje esantį trijų dimensijų tašką, pradedant indeksuoti nuo 0. Toliau dešinėje **semantics** naryje matome **surfaces** ir **values** narius, pirmame yra aprašyti trijų tipų plokštumų paviršiai, antrame kiekvienai **boundaries** naryje užduotai plokštumai suteikiamas vienas iš paviršių tipų, t.y. pirmoji plokštuma yra **GroundSurface**, antroji – **WallSurface**. **Lod** ( angl. Level of Detail ) narys nusako kokio detalumo yra šis plokštumų rinkinys, esantis **geometry** naryje. Vienas **City Object** gali turėti keletą **geometry** variantų, kurie skirsis objekto detalumu, pavyzdžiui Nyderlandų duomenyse kiekvienas **City Object** turi po keturias skirtingų LoD geometrijas. Paskutinis neaptartas liko kairėje **CityJSON Object transform** narys, kuris nusako kaip taškai skiriasi nuo realių objektų, naudojamas norint sumažinti CityJSON failo dydį.



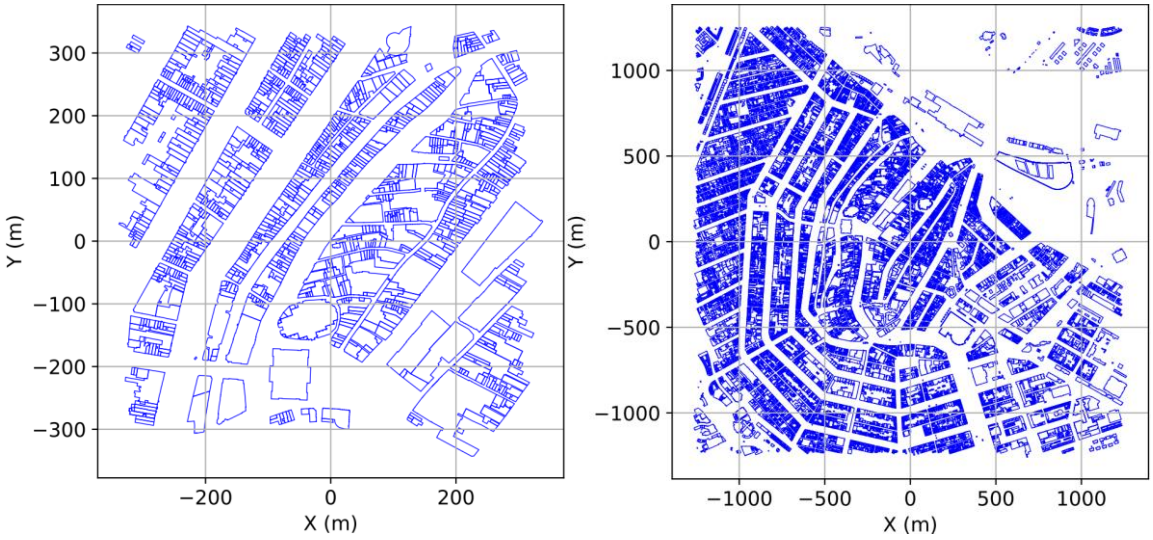

3.3 pav. Skirtingų LoD vizualizacija naudojant CityJSON Ninja[13].

Nyderlandų duomenys sudaryti iš skirtingų dydžių kvadratinių dalių, kvadrato dydis priklauso nuo pastatų tankio. Naudosime 3 skirtingus duomenų rinkinius. Pirmasis yra 7190 (dydis 700mx700m, plotu mažiausia imtis), antrasis T10 (2.8kmx2.8km, sudarytas iš dešimties gretimų imčių) ir didžiausias T300 (20.3kmx30.1km, sudarytas iš 293 imčių). Skaičiavimams naudojome detaliausią LoD2.2 geometriją.

Duomenys nebuvo idealiai tvarkingi, prieš skaičiuodami ištrynėme visas „plokštumas“ su mažiau nei 3 taškais. Siekiant pagreitinti skaičiavimus supaprastinome plokštumas, neįtraukėme taškų turinčių didesnį nei 178° vidinį kampą su kaimyniniais taškais, taip dvi linijas, suformuojančias beveik tiesią briauną pakeisdami viena. Dar bandėme skaičiavimuose pasinaudoti plokštumų normalėmis ir naudoti tik pastatų plokštumas, atsisukusias bazinės stoties kryptimi, bet radome, kad visų plokštumų normalių kryptys duomenyse yra atsitiktinės.



3.4 pav. Vieno iš pastatų stogų normalės




3.5, 3.6 pav. 7190(kairėje) ir T10(dešinėje) pastatų LoD0 geometrija

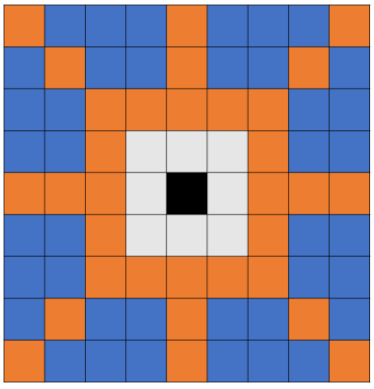
1 lentelė. Duomenų rinkinių dydžiai

| | 7190 | T10 | T300 |
|---|---|---|---|
| Pastatų kiekis | 1,110 | 13,141 | 357,792 |
| Stogų plokštumų kiekis | 10,240 | 99,347 | 1,123,381 |
| Sienų plokštumų kiekis | 98,916 | 953,702 | 11,385,358 |
| Plotas | 0.5km$^2$ | 7.8km$^2$ | 611km$^2$ |

# 4 Vektorinis XDraw difrakcijos modelis

Tam, kad galėtume panaudoti XDraw principą vektoriniams duomenims, reikia pakeisti metodo vietą, kur naudojant IDW interpoliaciją gaunamas kliūties aukštis, kadangi norime užduoti kliūtis ne taškais o plokštumomis. Mūsų sprendimas yra sukurti duombazę su matrica, turinčią norimą taškų skaičių, kurios kiekviename taške užregistruotos visos tą tašką kertančios plokštumos. Algoritmas veikia šitaip:

Užduodame duombazę su plokštumomis, rastrą, kurio vertės yra plokštumų indeksai, užduodame stoties vietą. Darome prielaidą, kad žiedas aplink stotį yra matomas, su R3 suskaičiuojamas matomumas visomis pasaulio kryptimis(vertikaliai, horizontaliai, įstrižai)( žr. 4.1 pav. ) ir antrojo žiedo matomumas. Toliau nebenaudojamas R3, sukasi ciklas į vis tolimesnius žiedus nuo stoties, ir kiekvienas taškas skaičiuojamas remiantis dvejais kaimyniniais taškais. Kaimyniniai taškai atsineša dvi vertes - dominuojančios kliūties aukštį ir dominuojančios kliūties indeksą. Jei kaimyninis taškas yra matomas, jo dominuojančios kliūties aukštis yra to taško aukščiausios plokštumos aukštis. Jei kaimyninis taškas nėra matomas, jo dominuojančios kliūties aukštis yra ankstyvesnės kliūties aukštis. Pirmiausia kaimyniniai taškai siūlo savo dominuojančios kliūties aukštį naujam taškui, pasirenkamas aukštesnis. Nubrėžiama linija tarp stoties ir ieškomo taško, suskaičiuojamas pasirinktos kliūties aukštis vietoje, kur linija kerta plokštumą. Taško aukštis lyginamas su kliūties aukščiu, jei taškas aukštesnis - jis matomas, jei kliūtis aukštesnė – nematomas. Ciklas tęsiamas, kol visi taškai bus surasti.

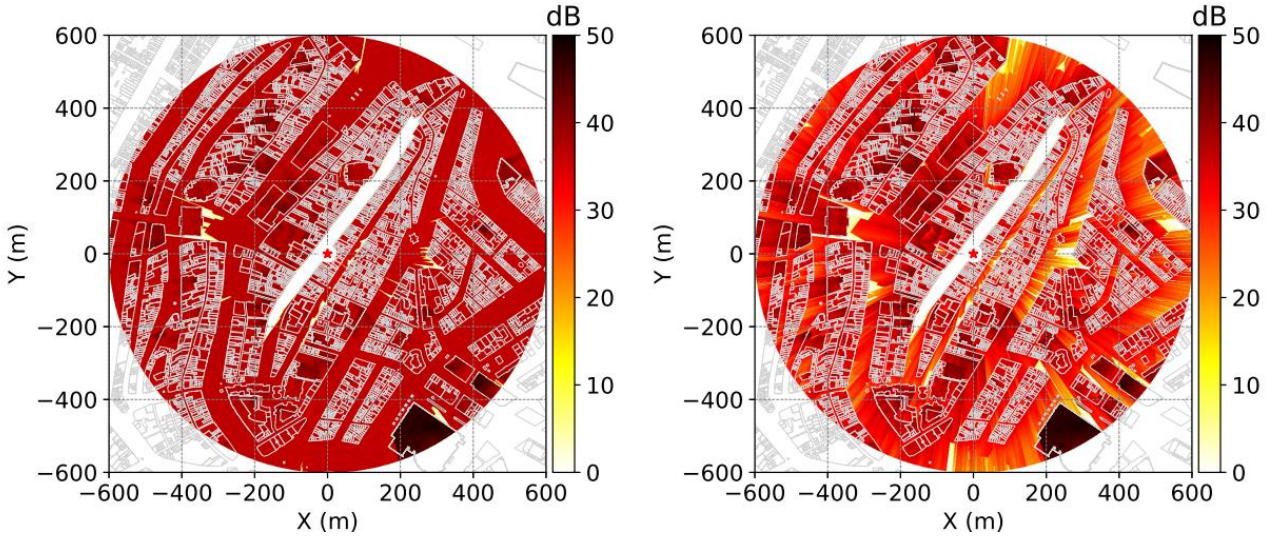


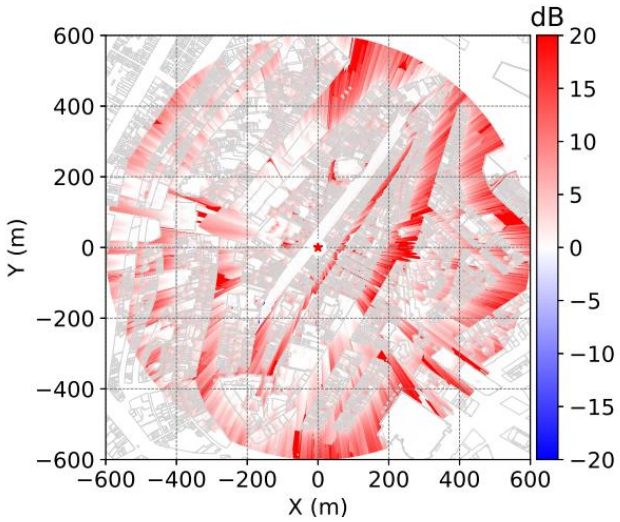
4.1 pav. XDraw algoritmo taškai: juodas – stotis, balti – pirmas žiedas, oranžiniai – suskaičiuoti R3 algoritmu, mėlyni – suskaičiuoti iš kaimyninių taškų

Aptarsime du pagrindinius naudotus skaičiavimo metodus, maksimalios kliūties (max-obst) ir maksimalaus stogo (max-roof). Max-obst metodu kiekvieno duombazės matricos taško

pastatų plokštumos analizuojamos tik vieną kartą, kai XDraw žiedas pasiekia tą tašką, žiūrima į dabartinį ir du ankstesnius kaimyninius taškus( žr. 2.5pav. ), apskaičiuojamos trys nuostolių vertės, pasirenkame didžiausią ir difrakcijos nuostolių rastre išsaugomi nuostoliai bei dominuojančios kliūties duombazės indeksas. Max-roof veikia panašiu būdu, tik skaičiavimams naudojamos vien stogų plokštumos. Dar abu šie metodai turi savo variacijas su skirtingais $k_{max}$ dydžiais. $K_{max}$ nurodo kiek dominuojančių kliūčių iš ankstesnių skaičiavimų išsaugoma. Čia aprašyti max-obst($k_{max}$=1) ir max-roof($k_{max}$=1). Didesnių $k_{max}$ verčių algoritmai veiktų tuo pačiu principu, tik kai renkamasi didžiausia nuostolių vertė, kiekvienas kaimyninis taškas pasiūlo ne vieną, o $k_{max}$ didžiausių nuostolių verčių.

Tikslumo patikrinimui sugeneravome 600m spindulio tikslų difrakcijos nuostolių rastrą (jo apskaičiavimas užtruko beveik savaitę), kuris atsižvelgė į visas plokštumas, su juo lyginome kitais metodais gautus rastrus. Išbandėme max-obst ir max-roof variacijų, ir radome, kad geriausias yra max-roof $k_{max}$=3. Toliau didinant kmax iki 20 išlošiamas vos 0.1 dB MAE( angl. Mean Absolute Error ) ir RMSE( ang. Root Mean Square Error ) tikslumo( žr. 2 lentelę ), o max-obst dėl didelio kiekio plokštumų palyginus lėtas, todėl po kelių išbandytų variacijų buvo jo atsisakyta. $MAE = \frac{1}{n}\sum_{i=1}^{n}|L_i - L_{aproxi}|$ $RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}|L_i - L_{aproxi}|^2}$ , čia $n$ – visi taškai, $L_i$ – tikslaus rastro nuostolių vertė, $L_{aproxi}$ – max-roof metodu gauta vertė.






4.2 – 4.4 pav. Tikslus sugeneruotas difrakcijos rastras(kairėje), max-roof $k_{max}$=3(dešinėje), rastrų skirtumas(apačioje), bazinės stoties aukštis 45m, mobilios antenos 2m.

Kaip matome 4.4 pav., vietomis, ypač už pastatų toliau nuo bazinės stoties, max-roof metodas nuvertina difrakcijos nuostolius. Panašu, kad algoritmas sutikdamas skirtingų aukščių kliūtis „pamiršta“ ankstesnes, globaliai reikšmingesnes kliūtis dėl tolimesnių, tik porai taškų aktualių kliūčių, todėl tolimesni nuo bazinės stoties taškai netiksliai skaičiuojami. $K_{max}$ didinimas virš 3 nepadeda išspręsti šios problemos.

# 5 Greitaveikos bandymai

Šiame skyriuje aptarsime matomumo skaičiavimų realizacijų trukmes ir tikslumus. Skaičiavimams naudototi du serveriai, kurių specifikacijos:

Intel Xeon E5-2670 CPU, 128Gb RAM, Linux Ubuntu OS.

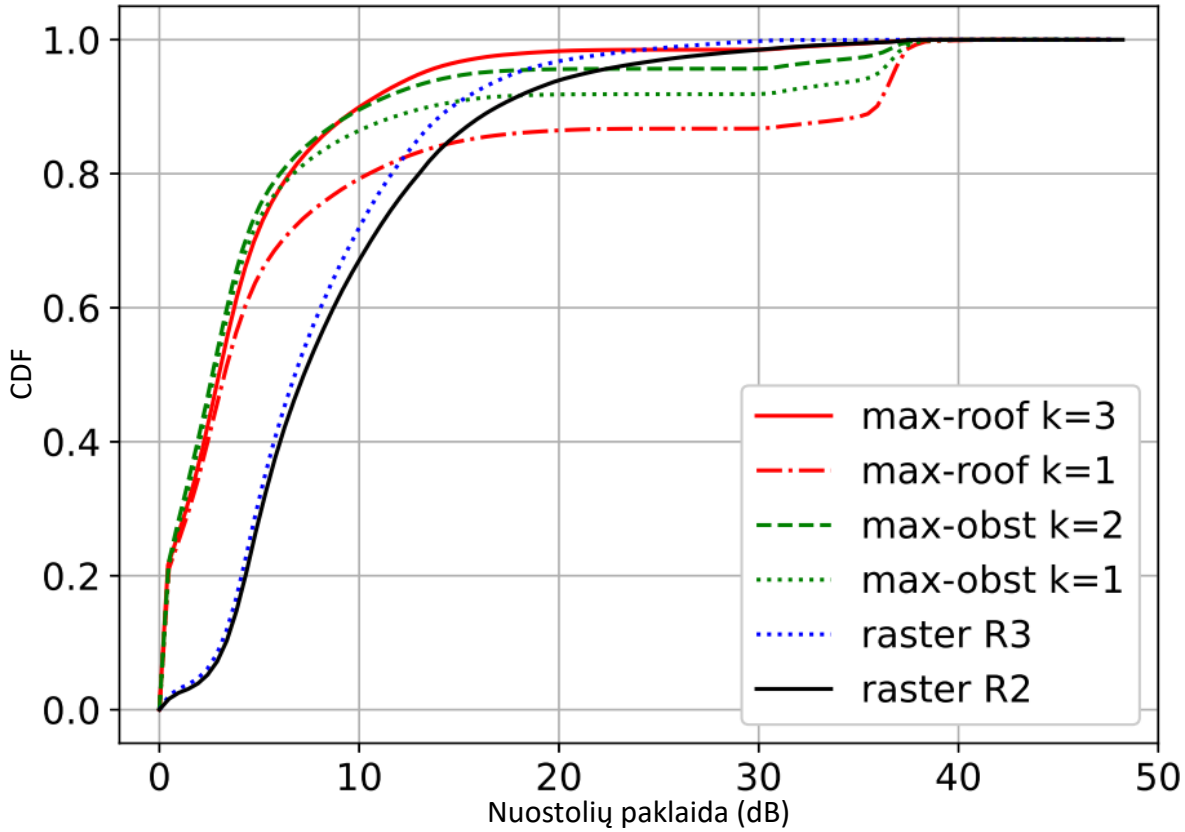
Intel i9-10900 CPU, 64Gb RAM, Linux Ubuntu OS.

Buvo naudojamos Python 3.10 bibliotekos su Numba JIT (angl. Just In Time) kompiliuotomis funkcijomis. Apskaičiuotų nuostolių rastro rezoliucija 1m.

2 lentelė. Skirtingais metodais apskaičiuotų nuostolių rastrų nukrypimai nuo tikslaus rastro

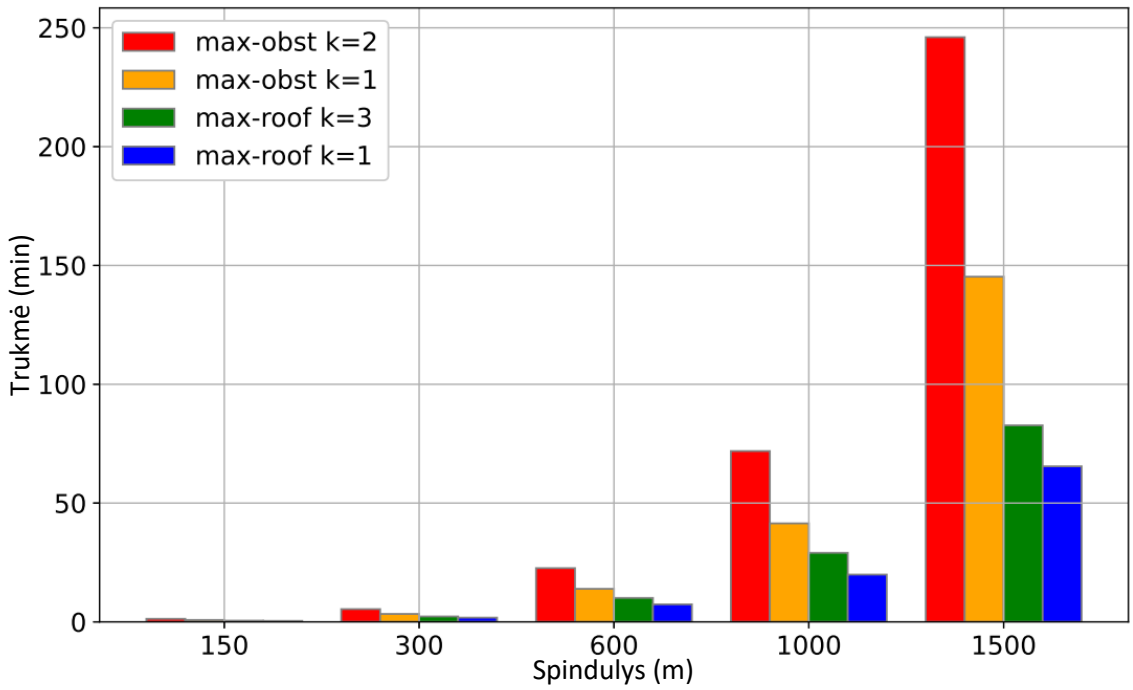
| Metodas | MAE(dB) | RMSE(dB) |
|---|---|---|
| Max-roof k=20 | 4.2 | 6.7 |
| Max-roof k=5 | 4.3 | 6.8 |
| Max-roof k=3 | 4.3 | 6.8 |
| Max-roof k=2 | 4.5 | 7.3 |
| Max-roof k=1 | 8 | 14.1 |
| Max-obst k=2 | 4.7 | 8.7 |
| Max-obst k=1 | 5.9 | 11.1 |
| Raster R3 | 8.1 | 9.5 |
| Raster R2 | 9 | 10.9 |

2 lentelėje Raster R3 ir R2 buvo skaičiuoti vektorinius duomenis pavertus rastriniais, t.y. duomenys buvo padalinti į 1mx1m aukščių rastrą ir R3 buvo naudojamas tikslus algoritmas, be XDraw, R2 buvo naudojamas greitesnis, ne toks tikslus, paremtas XDraw. Matome, jų tikslumai patys prasčiausi, net su tiksliu algoritmu, kuris lėtas, nuo taškų skaičiaus priklausantis trečiuoju laipsniu. Max-obst su tokiais pačiais $k_{max}$ kaip ir max-roof pasiekė panašius rezultatus, tik truputį prastesnius rezultatus, bet kartu buvo ir lėtesnis, todėl nusprendėme jo atsisakyti. Max-roof rezultatai geriausi, tik galima pastebėti, kad dabartinio algoritmo tikslumas aukščiau RMSE 6.8dB žymiai nekyla didinant $k_{max}$ vertes dėl anksčiau minėto duomenų „užmiršimo”.



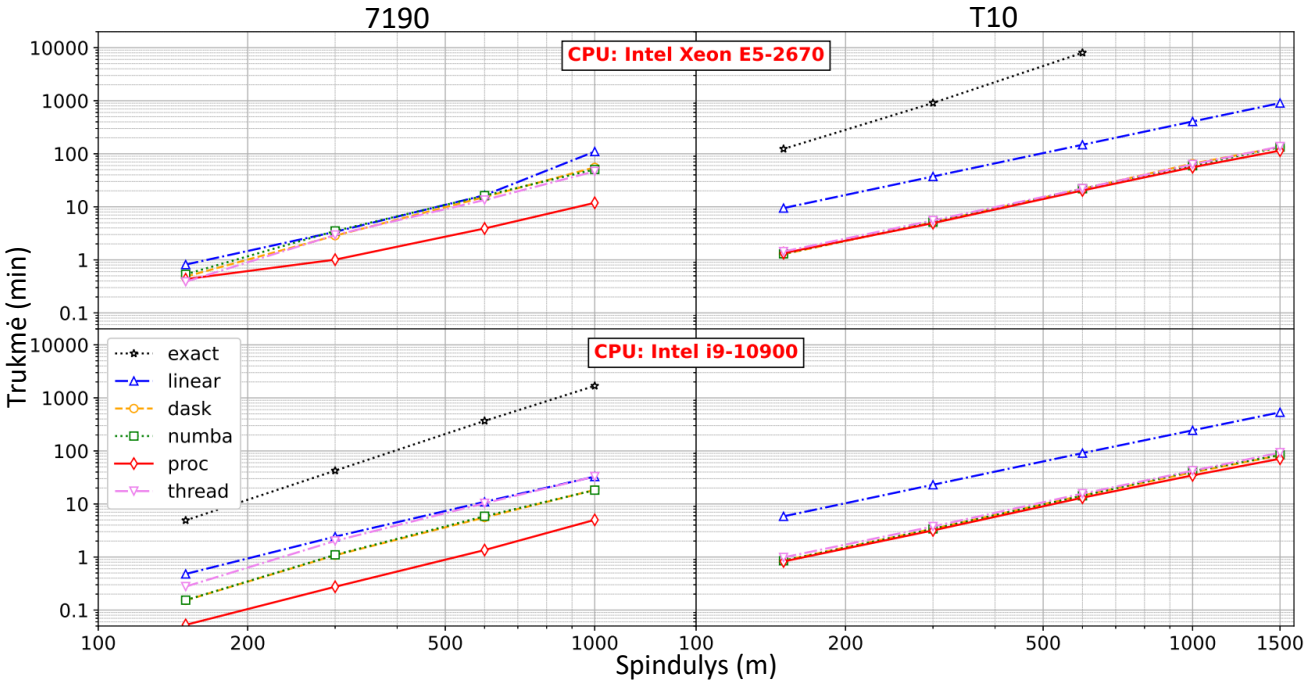

5.1 pav. Difrakcijos nuostolių kumuliatyvinė pasiskirstymo funkcija

5.1 grafike matome kumuliatyvinę pasiskirstymo funkciją CDF( angl. Cumulative Distribution Function ), kitais žodžiais kiek procentų ploto yra sugeneruojama su kokia paklaida. Max-roof $k_{max}$=3 90% ploto suskaičiuoja su mažesne negu 10dB paklaida.




5.2 pav. Skirtingų difrakcijos skaičiavimų metodų greičio palyginimas (T10 duomenys ir i9-10900 CPU, naudojant 8 branduolius)

Siekiant pagreitinti max-roof algoritmą buvo išbandyta daug paralelizavimo būdų. Pagrindinė problema yra Python‘o GIL( angl. Global Interpreter Lock ), kuris dėl stabilumo sumetimų leidžia vienu metu tik vienai procesoriaus gijai( angl. thread ) naudoti Python‘o interpretatorių, kas faktiškai paverčia mūsų 8 ar 10 branduolių procesorius į vieno branduolio. Tai galima apeiti keliais būdais. Galima naudoti **multiprocessing** modulį, tokiu atveju kiekvienas procesas turi atskirą Python‘o interpretatorių ir atskirą GIL, atskirai naudojantys paskirtą atmintį. Kitas variantas naudoti **threading** modulį, veikia tuo pačiu principu, tik visos gijos dalinasi atmintimi ir duomenimis. Dar išbandėme Dask biblioteką, ypač efektyvią dirbant su dideliais duomenų kiekiais, ir Numba **for loop** paralelinimą naudojant LLVM kompiliatoriaus biblioteką. XDraw algoritmai lengvai paralelizuojami, nes galima labai paprastai padalinti skaičiavimo plotą į lygias dalis, kiekvienai paskiriant atskirą procesorių.

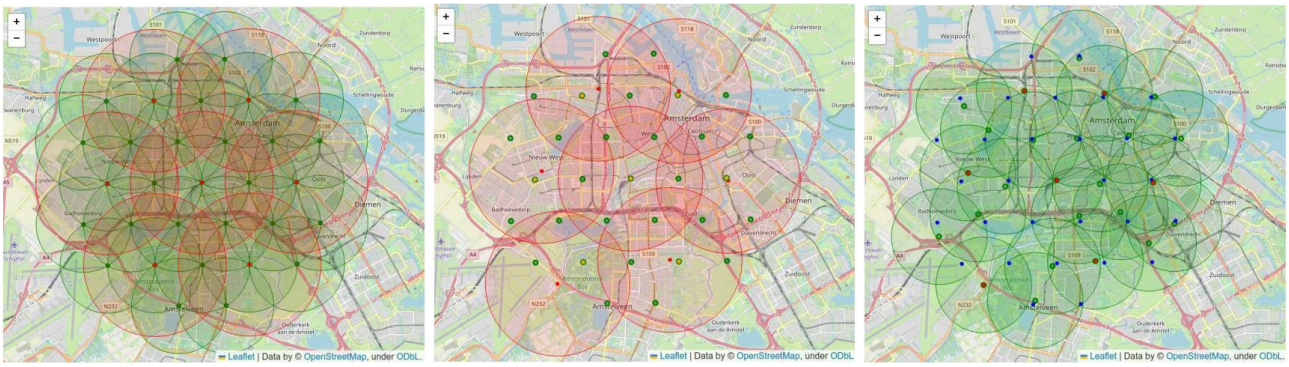



5.3 pav. Skirtingų paralelinimo būdų greitaveikos palyginimas (max-roof $k_{max}$=3), exact – tikslus algoritmas, linear – nuoseklus algoritmas.

Didesni skirtumai pasirodo prie mažesniojo 7190 duomenų rinkinio, bet ir prie ilgesnių skaičiavimų, su abiem procesoriais visur pirmauja algoritmas paralelintas su multiprocessing biblioteka. Pythono kalba sukurti rastriniai panašaus dydžio skaičiavimai už vektorinius greitesni 8-10 kartų, o naudojantys kompiliuotas rastrines bibliotekas nuo 50 iki 100 kartų greitesni, bet vektorinių skaičiavimų tikslumo pranašumas nenuginčijamas.

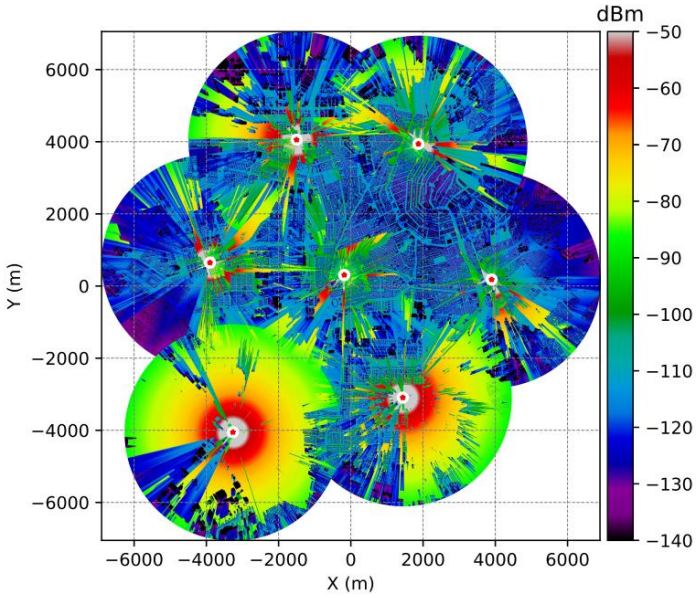
# 6 XDraw modelio taikymas daiktų interneto tinklų analizei

Tam kad parodytume vektorinių skaičiavimų vertę tinklų planavime, sukūrėme du skirtingus stočių išdėstymus. Naudodami vektorinius ir tradicinius rastrinius nuostolių skaičiavimo metodus paskaičiuosime, kiek skiriasi gaunami rezultatai.



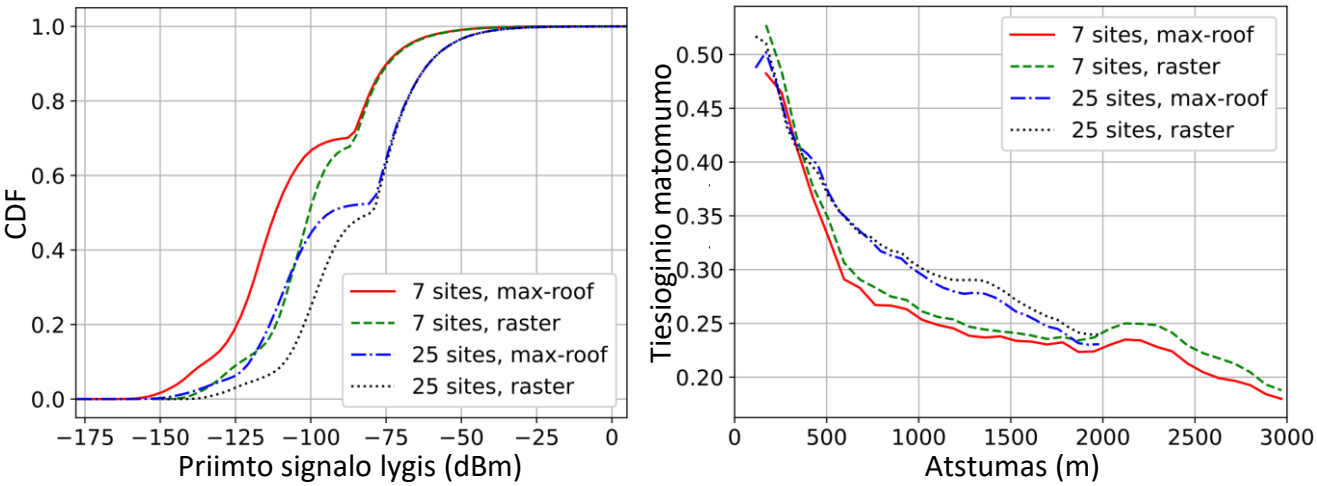

6.1 – 6.3 pav. Idealus šešiakampinis antenų išdėstymas(kairėje), pirmas išdėstymas su 7 stotimis(centre), antras išdėstymas su 25 stotimis(dešinėje).

Siūlome du šešiakampinius išdėstymus, pirmasis su 7 stotimis ir 4km tarpstotiniais atstumais( angl. Inter-Site Distance, ISD )( žr. 6.2 pav. ), antrasis tankesnis su 25 stotimis ir 2km tarpstotiniais atstumais( žr. 6.3 pav. ). Nyderlandų antenų biuras ( oland. Antennebureau ) leidžia atsisiųsti visas Nyderlandų bazinių stočių vietas, su antenų aukščiais. Savo išdėstymus pritaikėme prie artimiausių realių nuo 20m iki 50m aukščio antenų( algoritmas priede nr.1 ). Pirmasis išdėstymas su 3km stočių maksimaliu ryšio spinduliu( angl. cell radius ), antrasis su 2km stočių maksimaliu ryšio spinduliu.




6.4 pav. NB-IoT tinklo priimamo signalo simuliacija 7 stočių išdėstymui(max-roof metodas).

Skaičiavimams naudosime tipinį NB-IoT siųstuvo konfigūraciją. Siųstuvo galia 23dBm, maksimalūs sujungimo nuostoliai( angl. maximum coupling loss ) 164dB, imtuvo jautrumas -141dBm. Naudota tipinė 800MHz 3GPP B20 LTE dažnių juosta, mobilios antenos aukštis 2m, į žemės topologiją neatsižvelgta, rezultato rastro rezoliucija 1m.




6.5, 6.6 pav. Skirtingų metodų priimamų signalų lygių statistiniai pasiskirstymai(kairėje), skirtingų metodų tiesioginio matomumo (LoS) tikimybių priklausomybės nuo atstumo(dešinėje).

6.5 pav. skirtumas aiškiai matomas. Daugmaž sutampa grafikai tik kol yra tiesioginis matomumas, kai jo nebelieka, stebime apie 10dB skirtumus priimamojo signalo lygiui.

# Išvados

1. Pademonstruota galimybė radijo bangų difrakcijos nuostoliams dėl pastatų naudoti vektorinius trimačius duomenis. Naudojant vektorinį XDraw difrakcijos modelį MAE tikslumas pagerėja 4.7dB, o RMSE pagerėja 4.1dB lyginant su rastriniu XDraw( žr. 2 lentelė ).

2. Ištirta vektorinio XDraw difrakcijos modelio greitaveika naudojant lygiagrečius skaičiavimus( žr. 5.3 pav. ). Nors vektorinis XDraw yra lėtesnis už rastrinį modelį 50-100 kartų, vektorinio modelio greitį dar galima padidinti optimizuojant difrakcijos geometrijos skaičiavimus, naudojant papildomą duomenų filtravimą bei kompiliuotas C/C++ bibliotekas.

3. Vektorinis XDraw difrakcijos modelis pritaikytas daiktų interneto tinklų aprėpties analizei ir nustatyti apie 10dB difrakcijos ir apie 2% LoS tikimybės skirtumai( žr. 6.5, 6.6 pav. ).

# Šaltiniai:

[1] A.Spiess. #120 LoRa / LoRaWAN Range World Record Attempt. Will I succeed?, 2017. [žiūrėta 2023-03-15] Prieiga per internetą: https://www.youtube.com/watch?v=adhWIo-7gr4

[2] Balázs Dukai, Jordi van Liempt, Ravi Peters, Jantien Stoter, Stelios Vitalis, Teng Wu. 3D BAG, 2021. [žiūrėta 2022-10-15] Prieiga per internetą: https://3dbag.nl/en/download

[3] C. Akkaşlı. Methods for Path loss Prediction, 2009, p. 7-55. [žiūrėta 2022-10-15] Prieiga per internetą: http://lnu.diva-portal.org/smash/get/diva2:273839/FULLTEXT01.pdf

[4] Noman Shabbir, Muhammad T. Sadiq, Hasnain Kashif, Rizwan Ullah. Comparison of radio propagation models for Long Term Evolution (LTE) network, 2011, p. 30-32. [žiūrėta 2022-10-15] Prieiga per internetą: https://arxiv.org/ftp/arxiv/papers/1110/1110.1519.pdf

[5] F. Beauducel. Radiocover, 2016. [žiūrėta 2022-10-15] Prieiga per internetą: https://github.com/IPGP/mapping-matlab/blob/master/radiocover/README.md

[6] J. Zhang, S. Zhao, and Z. Ye. Spark-Enabled XDraw Viewshed Analysis, 2021, p. 1-3. [žiūrėta 2022-10-15] Prieiga per internetą: https://ieeexplore.ieee.org/document/9320526/

[7] ITU-R. Recommendation P.526: Propagation by diffraction, 2018, p. 3-19. [žiūrėta 2022-10-15] Prieiga per internetą: https://www.itu.int/rec/R-REC-P.526-15-201910-I/en

[8] K.Mekki, E.Bajic, F.Chaxel, F.Meyer. A comparative study of LPWAN technologies for large-scale IoT deployment, 2017, p. 1-5. [žiūrėta 2022-10-15] Prieiga per internetą: https://www.sciencedirect.com/science/article/pii/S2405959517302953

[9] https://standards.tiaonline.org/market_intelligence_/glossary/index.cfm?term=%26%23T%5F%5BQB%2BK%0A

[10] G. Zhu, J. Li, J. Wu, M. Ma, L. Wang, and N. Jing. HiXDraw: An Improved XDraw Algorithm Free of Chunk Distortion, 2019, p. 1-5. [žiūrėta 2022-10-15] Prieiga per internetą: https://www.mdpi.com/2220-9964/8/3/153

[11] CityJSON Specifications 1.1.3. [žiūrėta 2022-10-15] Prieiga per internetą: https://www.cityjson.org/specs/1.1.3/

[12] D. Shepard. A two-dimensional interpolation function for irregularly-spaced data, 1968m., p. 1-3. [žiūrėta 2022-10-15] Prieiga per internetą: https://dl.acm.org/doi/10.1145/800186.810616

[13] CityJSON Ninja Prieiga per internetą: https://ninja.cityjson.org/#

# Priedai

## Priedas Nr.1

```
from scipy.spatial import cKDTree

def find_closest_sites_reuse(coords, df_sites, df_sites1, d=1, delta=0.5, max_offset=900, min_distance=700):

@brief funkcija, randanti artimiausias realias stotis norimam išsidėstymui, su papildomomis sąlygomis

@param coords (np.array): idealaus šešiakampinio išsidėstymo stočių pozicijos.
@param df_sites (pd.DataFrame): visos galimos stočių pozicijos.
@param df_sites1 (pd.DataFrame): aukštesnio prioriteto galimos stočių pozicijos.
@param d (float): atstumai tarp šešiakampio taškų.
@param delta (float): jei per šį atstumą nuo sites stoties egzistuoja sites1 stotis, sites1 stotis bus pasirinkta.
@param max_offset (float): spindulys per kurį nuo coords taško ieškoma stočių.
@param min_distance (float): mažiausias leidžiamas atstumas tarp stočių.

@return closest_sites (pd.DataFrame): realios stotys, artimiausios idealiam šešiakampiniam išsidėstymui


kdtree = cKDTree(df_sites[['X', 'Y']].values)  # Sukuriamas kdtree su koordinatėmis iš df_sites
kdtree1 = cKDTree(df_sites1[['X', 'Y']].values)  # Sukuriamas kdtree1 su koordinatėmis iš df_sites1
final_sites = []  # Inicializuojamas tuščias listas galutiniam rezultatui saugoti

for i, point in enumerate(coords):  # Ciklas per kiekvieną coords tašką
    distance, index = kdtree.query(point, k=1)  # Užklausiam kdtree, kad surastų arčiausiai dabartinio taško esančias stotis
    if distance <= max_offset:  # Jei artimiausia stotis arčiau negu max_offset, naudojam tą stotį
        site = df_sites.iloc[index]
        distance1, index1 = kdtree1.query(np.array([site['X'], site['Y']]), k=1)  # Užklausiam kdtree1, kad surastų arčiausiai dabartinio taško esančias stotis
        if distance1 <= delta:  # Jei artimiausia kdtree1 stotis yra per atstumą delta, pakeičiam kdtree stotį šia kdtree1 stotimi
            site = df_sites1.iloc[index1]
            distance = np.linalg.norm(np.array([site['X'], site['Y']]) - point)
    else:
        print(f"Point {i} at coordinates {point} couldn't find a matching site within max_offset.")
        continue
    if final_sites:  # Patikrinam, ar nauja stotis nėra per arti esamų stočių
        existing_sites_kdtree = cKDTree(np.array([site[['X', 'Y']] for site in final_sites]))
        min_distance_to_existing_sites, _ = existing_sites_kdtree.query(np.array([site['X'], site['Y']]), k=1)
        if min_distance_to_existing_sites < min_distance:
            print(f"Point {i} at coordinates {point} found a site too close to existing sites.")
            continue

    final_sites.append(site)  # Jei visos sąlygos įvykdytos, pridedam stotį prie rezultato
closest_sites = pd.DataFrame(final_sites).reset_index(drop=True)  # Sukuriamas DataFrame su stotimis
return closest_sites
```

Steponas Pilkauskas

## Application of 3D vector data for IoT network planning

IoT technologies such as LoRa, SigFox, NB - IoT allow thousands of battery-powered devices to send small amounts of data (e.g. a sensor to measure soil acidity) to a station a couple of times a day. Advantages - batteries for devices need to be replaced once in a few years, coverage radius of a single station is in the kilometer range. For example LoRa's distance record between points on the ground is over 200km, with a more likely coverage radius of between 3km to 15km, the other technologies mentioned also have similar distances, depending on obstacles. In an urban scenario, losses are higher due to building height and density. With such a huge difference in coverage distances, successful deployment requires accurate and fast methods to calculate radio signal losses. We will use 3D vector data of buildings in the Netherlands, which will allow to calculate more accurate loss estimates than with traditional raster data. The XDraw algorithm, which we have applied to vector data, will increase the speed of the calculations. Finally, we will discuss some possibilities for IoT network scenarios.

The aim of this work is to apply accurate 3D vector data of buildings to a radio wave diffraction model suitable for planning IoT networks.

The possibility of using vector three-dimensional data for radio wave diffraction losses is demonstrated. Using a vector XDraw diffraction model improves the mean absolute error accuracy by 4.7dB and the root mean square error by 4.1dB compared to a raster XDraw model.

The speed of the vector XDraw diffraction model has been investigated using parallel computing. Although the vector XDraw is 50-100 times slower than the raster model, the speed of the vector model can still be improved by optimising the calculation of the diffraction geometry, using additional data filtering, and using the compiled C/C++ libraries.

The vector XDraw diffraction model was applied in analysing the coverage of IoT networks and differences of about 10dB in diffraction and about 2% in Line of Sight probability were found.