Tampereen yliopisto

Roope Suominen

TERÄSRUNGON AUTOMAATTINEN GENEROINTI

Kandidaatintyö
Rakennetun ympäristön tiedekunta
Tarkastaja: Kristo Mela
Toukokuu 2023

TIIVISTELMÄ

Roope Suominen: Teräsrungon automaattinen generointi (Automatic Generation of Steel Framed Building)
Kandidaatintyö
Tampereen yliopisto
Rakennustekniikan tutkinto-ohjelma
Toukokuu 2023

Tämän työn tavoitteena on tutkia algoritmisen mallintamisen kyvykkyyttä teräsrunkoisen rakennuksen automaattiseen generointiin. Tutkimusmenetelminä käytetään kirjallisuustutkimusta, sekä tapaustutkimusta. Kirjallisuuden kautta perehdytään algoritmisen suunnittelun määrittelyyn, sen toimintatapaan, sekä nykyiseen käyttöön rakennesuunnittelussa. Tapaustutkimuksen avulla tarkastellaan työn aikana luodun algoritmin toimivuutta käytännössä kuvitteellisen kohteen tietomallin luomisessa. Valmiista tietomallista rajattiin tarkempaan algoritmin kyvykkyyden tarkasteluun vasemman sivun runkorakenteet, joista luotiin valmis FEM-laskentamalli kuormituksien, sekä tuentojen osalta algoritmisesti. Tämän jälkeen rakenneanalyysin tuottamia tuloksia vertailtiin algoritmisen mallin, sekä perinteisesti tuotetun mallin välillä.

Rakenteiden mitoittamiseen tai optimointiin ei tässä työssä perehdytä, vaan keskitytään rungon generointiin liittyvään logiikkaan. Työssä käytettävät ohjelmistot ovat Rhinoceros3D, Grasshopper, Tekla Structures ja Robot Structural Analysis. Algoritmin toimintaan vaaditut lisäosat Grasshopperille ovat Grasshopper – Tekla Live Link, Salamander3 ja Telepathy.

Tutkimuksen aikana onnistuttiin luomaan algoritmi, joka kykenee tuottamaan rakennuksen rungosta tietomallin käyttäjän syöttämien parametrien mukaan. FEM-laskentamallien vertailussa saatiin samat rakenneanalyysitulokset algoritmisesti tuotetun, sekä perinteisesti tehdyn mallin välillä. Algoritmin tuottamissa lopputuloksissa ei ilmennyt ongelmia. Rakennemallia muunnettiin eri tilanteiden mukaan, onnistunein tuloksin.

Tutkimuksen perusteella huomataan kokonaisen rungon mallintamisen olevan mahdollista käyttäen pelkästään algoritmiavusteisia menetelmiä. Tietomallin tuottamisessa ilmenee kuitenkin haasteita muokkausmahdollisuuksien laajuuden kanssa. Algoritmeja hyödyntämällä rakenneosien yksilöintikin on mahdollista, mutta syötettävien parametrien määrä kasvaa nopeasti erittäin pitkäksi kokonaisuudeksi. Tällöin käyttöliittymän teko algoritmin toteutukselle on välttämätöntä selkeyden ja nopeuden takia. Rakenneanalyysituloksien yhtäläisyyden perusteella voidaan myös todeta algoritmin olevan kykeneväinen FEM-laskentamallin luomiseen. Haasteena ilmenee kuitenkin tässäkin tilanteessa mahdollisten muokkausparametrien laajuus täydellisen mallin luomista varten. Algoritmiavusteisia menetelmiä hyödynnettäessä onkin tärkeää tunnistaa mitkä lähtötiedot ovat algoritmin toiminnan kannalta välttämättömiä, ja mistä kohtaa olisi helpompi jatkaa toisen ohjelmiston avulla loppuun.

Avainsanat: algoritmi, algoritmiavusteinen suunnittelu, algoritminen mallintaminen, parametrinen mallintaminen, Grasshopper

Tämän julkaisun alkuperäisyys on tarkastettu Turnitin OriginalityCheck –ohjelmalla.

SISÄLLYSLUETTELO

1. JOHDANTO 1
2. ALGORITMIAVUSTEINEN SUUNNITTELU 3
2.1 Taustaa 3
2.2 Algoritmin toiminta 5
2.3 Rhino3D ja Grasshopper 6
2.4 Hyödyntäminen rakennesuunnittelussa 11
3. RUNGON GENEROINTI 15
3.1 Kohteen esittely 15
3.2 Lähtökohdat ja rajoitukset 17
3.3 Algoritmin toiminta 19
3.3.1 Gridi, runkopilarit ja runkoanturat 20
3.3.2 Jäykisteet ja runkopalkit 25
3.3.3 Kerrosten luonti ja kerrostuennat 27
3.3.4 Tietojen keruu ja RSA-exporttaus 30
3.4 Jatkokehitykset ja huomioitavaa 35
4. YHTEENVETO 37
LÄHTEET 39

KUVALUETTELO

Kuva 1. Esimerkki I-profiilipalkin mallista AutoCad-ohjelmistolla ja saatavan tiedon määrä (Teittinen 2022). 3
***Kuva 2**. Esimerkki BIM-ohjelmistolla luodun kolmiulotteisen I-profiilipalkin tietomalli ja saatavan tiedon määrä (Teittinen 2022).* 4
***Kuva 3.** Algoritmin vaiheet tiivistetysti (Tedeschi 2011).* 5
***Kuva 4.** NURBS-pinnan ja mesh-mallin tarkkuuksiin vaadittu pistemäärä (Tanska & Österlund 2014, s. 30).* 6
***Kuva 5.** Esimerkki rakennuksen ulkolinjasta, jossa jokaisen sivun pituus olisi 5.* 7
***Kuva 6.** Uudet rakennuksen ulkolinjat, jossa muunnetut parametrit on ympyröity.* 8
***Kuva 7.** Esimerkki Grasshopperin datapuurakenteesta. Tässä tilanteessa linjat muodostuvat jokaiselle kolmiolle erikseen. (Tedeschi 2011, muokattu)* 9
***Kuva 8.** Flatten -komponentin vaikutus puurakenteeseen. Tässä tilanteessa linjat muodostuvat kaikkien pisteiden välille. (Tedeschi 2011, muokattu)* 10
***Kuva 9.** The Morpheus Hotel. (Permarini et al. 2016, s. 57)* 12
***Kuva 10.** The Morpheus hotellin kantava runko. (Permarini et al. 2016, s. 57)* 12
***Kuva 11.** Varastohallin tukirakenteiden pohjapiirros.* 15
***Kuva 12.** Varastohallin leikkauskuva A-A.* 16
***Kuva 13.** Varastohallin leikkauskuva B-B.* 16
***Kuva 14.** Ylempänä on tutumpi johtojen liittäminen haluttuun komponenttiin. Alempana Telepathyn toiminta, eli vasen lähettäjä (sender) yhdistyy samalla nimellä olevaan vastaanottajaan (receiver).* 17
***Kuva 15.** Tässä työssä käytetyn algoritmin canvas. Oikealla alhaalla esitelty näkymä, jos kaikki johdot olisivat näkyvissä.* 18
***Kuva 16.** Algoritmin vaiheet.* 19
***Kuva 17**. Algoritmissa käytetyt syötteet.* 20
***Kuva 18.** Runkopilareiden algoritmin prosessikaavio.* 21
***Kuva 19.** Esimerkki puurakenteen ongelmasta ehtolauseilla.* 21
***Kuva 20.** Clean Tree -komponentin toiminta (komponentti ympyröity).* 22
***Kuva 21.** Tekla-näkymä ja Grasshopper syötteet pituuslinjan pilareille.* 23
***Kuva 22.** Pituuslinjan pilarien attribuuttien valinnat oikealla ja vasemmalla nurkkapilarin properties-näkymä. Nurkkapilariin liittyvät tiedot on nuolilla yhdistetty.* 23
***Kuva 23.** Anturoiden syötteet ja näkymä Teklassa.* 24
***Kuva 24**. Runkopilarit ja -anturat valmiit Tekla-näkymässä.* 24
***Kuva 25.** Yleisimpiä ristikkojäykisteratkaisuja. (Elementtisuunnittelu 2023, muokattu)* 25
***Kuva 26.** Jäykisteiden generoinnin prosessikaavio.* 25
***Kuva 27.** Ristikkojäykisteiden syöte ja Tekla-näkymä.* 26
***Kuva 28.** Runkopalkkien generoinnin prosessikaavio.* 27
***Kuva 29**. Kerroslaattojen syötteet ja Tekla-näkymä.* 27
***Kuva 30.** Kerrospilarien ja -palkkien generoinnin prosessikaavio.* 28
***Kuva 31.** Rhino -näkymä ylhäältä kerrostuentojen lähtöpisteistä. Punaisella merkityt pisteet on generoidut runkopilarit ja vihreällä merkityt tulevat kerrostuennat.* 29
***Kuva 32.** Kerrostuentojen syötteet ja Tekla-näkymä.* 30
***Kuva 33.** Vasemmalla case-kohteen valmis tietomalli. Oikealla puolella näkymä ylhäältä katsottuna, jossa ylin laatta piilotettuna kerrostuentojen näkemistä varten.* 31
***Kuva 34.** Analyysimallin luonti Salamander3 lisäosan komponenteilla.* 32
***Kuva 35.** RSA-mallien vertailu.* 33

Kuva 36. Analyysimallin laskennan tuottamat kuvaajat leikkausvoimalle Fz *ja sivusuuntaiselle siirtymälle.* *33*
Kuva 37. *Case-kohteen RSA-malli. Vasemmalla profiilit näkyvissä ja oikealla lineaariset elementit.* *34*
Kuva 38. *Human UI -lisäosalla luotu käyttöliittymä.* *35*

LYHENTEET JA MERKINNÄT

Globaali analyysimalli	Koko rakennuksen analysointia varten oleva malli
Lokaali analyysimalli	Rakennuksessa olevan osan tarkempi analyysimalli
RSA	Robot Structural Analysis -rakenneanalyysiohjelmisto

1. JOHDANTO

Tietomalleja hyödynnetään rakennushankkeissa vuosi vuodelta enemmän. Aiempien piirustusten ja laskelmien tueksi voidaan nykyään tuottaa kolmiulotteisia rakennemalleja, jotka visualisoivat paremmin tulevaa rakennusta. Tietomallin avulla voidaan myös selvittää rakennukseen liittyviä tietoja, esimerkiksi käytettävien materiaalien lukumäärää tai ylipäätään tarkastella ratkaisujen toimivuutta. Tietomallin luontiin tarvitaan kuitenkin suunnittelija, jonka tehtävänä on löytää paras, sopivin ja tehokkain rakenneratkaisu hankkeeseen (Arokoski 2020, s. 1). Rakenneratkaisun valinta ei kuitenkaan ole pelkästään yhteen tekijään sidottu, esimerkiksi kustannustehokkaimpaan, vaan siihen voi vaikuttaa myös uudelleenkäytettävyys ja hiilijalanjälki. Uusimpana tekijänä alalla pyritäänkin kehittämään algoritmisen suunnittelun mahdollisuuksia, jotka nopeuttaisivat edellä mainittujen ratkaisujen löytämistä. Suunnitteluun liittyviä vaiheita pyritään automatisoimaan algoritmisella ajattelulla, jolloin suunnittelijalle jäisi enemmän aikaa muuhun insinöörityöhön (Paukkeri 2018, s. 25).

Karjalainen (2021) esittääkin, miten Helsingin Jätkäsaaren Satamaparkkitalossa hyödynnettiin algoritmiavusteisia menetelmiä. Parkkitalo koostui kahdeksasta kerroksesta, ja ulkomuodoltaan se oli sylinterin muotoinen. Haastavan geometrian myötä mallintaminen perinteisin menetelmin olisi ollut työlästä ja aikaa vievää, etenkin mahdollisten muutoksien ilmetessä. Taustalla toimivan algoritmin kautta kuitenkin pystyi muuttamaan haluttua parametria, esimerkiksi kerroskorkeutta tai rampin leveyttä, jonka mukaan malli päivittyi. Tämän myötä voitiin käydä nopeasti läpi eri runkovaihtoehtoja hankkeessa ja pohtia niiden toteutettavuutta tai kustannuksia. Algoritmin tarkempaa toimintatapaa ei kuitenkaan kuvailtu.

Tämän tutkimuksen päätavoitteena on luoda algoritmi, joka mallintaisi rakennuksen pilari-palkkirungon jäykisteineen käyttäjän asettamien parametrien kautta. Algoritmin luomisen kautta perehdytään mallin geometrian muodostamisen tarvitsemaan logiikkaan. Geometrian luomisen jälkeen osa mallista siirretään rakenneanalyysiohjelmistoon ja tarkastellaan sen tuottamia tuloksia kuormituksen alla.

Tutkimus koostuu kirjallisuuskatsaukseta ja case-tutkimuksesta. Kirjallisuuskatsauksessa perehdytään algoritmiavusteisen suunnittelun taustaan, sen toimintatapaan sekä aiheeseen liittyvään kirjallisuuteen.

Case-tutkimuksessa tarkastellaan työn aikana kehitetyn algoritmin toimivuutta kuvitteellisen varastohallin avulla. Mallin luomisen jälkeen tarkastellaan sen siirtämistä rakenneanalyysiohjelmistoon sekä tutkitaan ohjelmiston laskemia rasituksia rakenteille. Tutkimuksessa käytetään Rhinoceros3D, Grasshopper, Tekla Structures ja Robot Structural Analysis -ohjelmistoja. Vaikka tässä tutkimuksessa ei rakenteiden mitoittamiseen tai optimointiin perehdytä, mallin tuottaminen tarvitaan kuitenkin osaksi edellä mainittuja prosesseja.

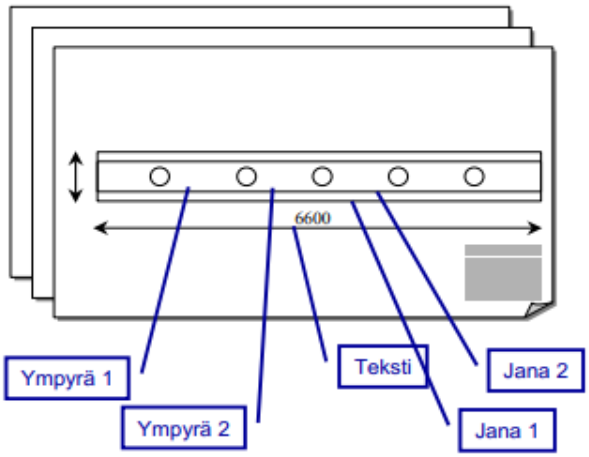
Tutkimuksessa tuotetulla algoritmilla luodun mallin tarkkuus rajataan luonnosmalliin. Tietomallista löytyy tässä vaiheessa perustukset sekä runkorakenteiden perusgeometria sijainnin osalta oikein. Raudoitteita tai liitoksia ei kuitenkaan oteta huomioon. (Karjalainen 2018, s. 21)

2. ALGORITMIAVUSTEINEN SUUNNITTELU

2.1 Taustaa

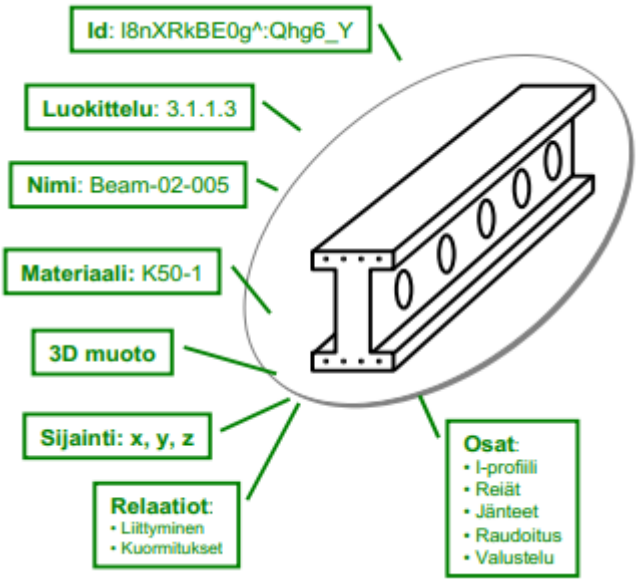
Computer-Aided Design (CAD) -ohjelmistojen käytön yleistyttyä 1970-luvulla rakennusala sai erittäin suuren aputyökalun suunnittelutoimistoihin. Se mahdollisti tarkkojen suunnitelmien luomisen jopa useista eri näkökulmista syrjäyttäen manuaalisen suunnittelun. Piirroksia ei myöskään tarvinnut aloittaa alusta, vaan uudet muutokset pystyttiin hoitamaan tietokoneavusteisesti. Suunnittelijan mahdolliset piirrosvirheet myös vähentyivät, ja syntyi mahdollisuus monimutkaisempien rakenteiden mallintamiseen. Yksi tunnetuimpia ohjelmistoja tähän tarkoitukseen on AutoCAD, jota käytetään pääosin piirrosten luomiseen. CAD-ohjelmistoilla pystyy myös tekemään kolmiulotteisia mallinnoksia, mutta niissä on omat rajoitteensa, esimerkiksi tietojen puutteellisuus. (Storrier 2021)

Tämän parantamiseksi 1990-luvulla esiteltiinkin uudenlainen tietomallinnustyyppi eli BIM (Building Informational Modelling). BIM-tietomallilla luodut 3D-mallit sisältävät huomattavasti enemmän informaatiota CAD-malleihin verrattuna, esimerkiksi tiettyyn rakenneosaan kootuista tiedoista. (Storrier 2021) Yksinkertaisena esimerkkinä voidaan käyttää betonirunkoisessa rakennuksessa olevaa I-profiilipalkkia. AutoCADilla tehdystä 2D-mallista voidaan nähdä palkin pituus, korkeus sekä mahdolliset liitokset muihin osiin. Rajoitteena piirustuksessa kuitenkin on, että se on leikkaus tietystä kohdasta rakennusta ja tarkemmat tiedot täytyy selvittää useammista piirroksista. Kuvassa 1 esitetään esimerkki AutoCAD-ohjelmistolla luodusta leikkauskuvasta ja saatavan tiedon määrästä.



Kuva 1. *Esimerkki I-profiilipalkin mallista AutoCad-ohjelmistolla ja saatavan tiedon määrä (Teittinen 2022).*

BIM:llä luodussa mallissa taas nähdään suoraan palkin kolmiulotteinen muoto, sen sijainti rakennuksessa muihin osiin suhteutettuna ja sen yhteneväisyydet niihin. Tietomalleissa voidaan myös yksilöllistää rakenneosia omilla tunnuksilla, mikä helpottaa haluttujen osien löytämistä. Kuvassa 2 esitetään esimerkki BIM-ohjelmistolla luodun palkkimallin sisältämän tiedon määrästä.

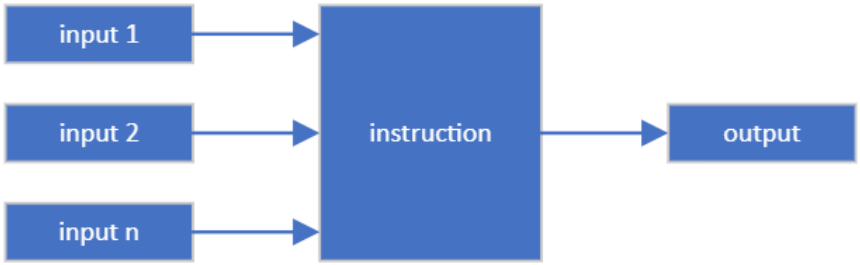


***Kuva 2**. Esimerkki BIM-ohjelmistolla luodun kolmiulotteisen I-profiilipalkin tietomalli ja saatavan tiedon määrä (Teittinen 2022).*

Molemmat mallit siis sisältävät tietoa rakenteesta, mutta siitä, kenen niitä on tarkoitus ymmärtää, löytyy suurin eroavuus. Kaksiulotteisten piirustuksien luomisessa toisen ihmisen täytyy ymmärtää, mitä siinä tarkoitetaan, joten ulkoasun muotoilu selkeäksi on erittäin tärkeää. Tietomallin luomisessa taas ihmisen visuaalisen ymmärtämisen lisäksi mallissa olevan datan täytyy olla koneen ymmärrettävissä, jotta sitä voidaan hyödyntää mallinnusprosessissa. Teittinen (2022) lyhentääkin ymmärrettävyyden hyvin kaksiulotteisissa malleissa muotoon *"Human Readable"* ja kolmiulotteisissa tietomalleissa muotoon *"Machine Readable".* Tietomalleissa olevan datan muotoilu tietokoneen ymmärtämäksi onkin erittäin kriittinen tekijä uusimmalla suunnittelun osa-alueella, nimittäin algoritmiavusteisessa suunnittelussa (*Algorithms-Aided Design, AAD*).

2.2 Algoritmin toiminta

Algoritmin tarkka määritelmä ei välttämättä ole kaikille selkeä. Käsitteenä Boberg (2023) kuvailee algoritmin olevan kuvaus tai ohje, jota seuraamalla saadaan tietty tehtävä, prosessi tai ongelmanratkaisu toteutettua. Hyvänä arjesta tuttuna esimerkkinä algoritmista voidaankin käyttää ruokareseptiä. Reseptin toteuttaja kirjaa ylös tietyn ruuan valmistukseen tarvittavat ainekset sekä niiden määrät ja vaiheet. Ohjetta seuraamalla saadaan lopputuloksena haluttu ruoka-annos. Algoritmin vaiheita havainnollistetaan kuvassa 3. Se siis koostuu lähtötiedoista *(input),* joita käsitellään ennalta asetetulla tavalla *(instruction)*, josta syntyy lopputulos *(output)*. Lähtötietoja voi olla sen mukaan, kuinka määritetty algoritmi osaa käsitellä sen.



Kuva 3. *Algoritmin vaiheet tiivistetysti (Tedeschi 2011).*

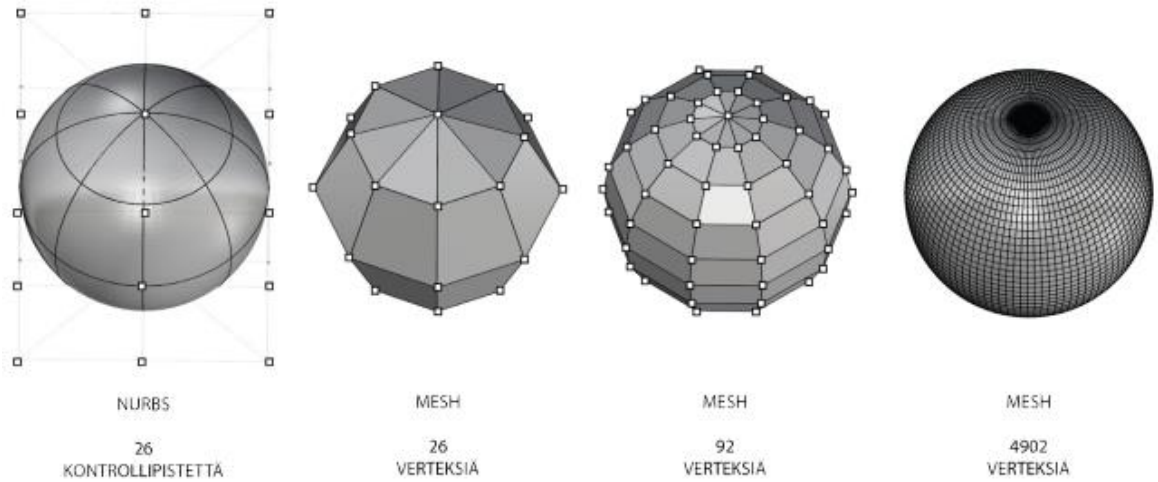
Syötteistä *(inputeista)* käytetään myös nimitystä *parametrit*. Muissa lähteissä algoritmisesta suunnittelusta saattaakin nähdä toisen ilmaustavan, nimittäin *parametrinen suunnittelu*. Lallan (2017, s. 4) mukaan vaikka ne ovat lähellä toisiaan, eroavuus löytyy siinä, mitä vaiheen osa-aluetta halutaan korostaa. Parametrisuudella pyritään korostamaan lopputuloksen muutosmahdollisuutta lähtötietojen (*inputtien*) avulla, kun taas algoritmiavusteisessa korostetaan itse lopputuloksen tuottamaa algoritmin osuutta (*instruction*). Tässä työssä käytetään kuitenkin algoritmiavusteisen suunnittelun nimitystä sekaannusten välttämiseksi.

Algoritmiavusteisen suunnittelun vahvuudet ilmenevät ratkaisun laajuuden kautta. Kuvitellaan ongelma, joka voidaan rajata tiettyyn luokkaan, ja luodaan sille algoritmi, joka tuottaa halutun lopputuloksen. Lopputulos ei kuitenkaan rajaudu pelkästään tähän tiettyyn tilanteeseen, vaan algoritmi kykenee antamaan vastauksen kaikkiin samaan luokkaan kuuluviin ongelmiin. Yksinkertaisimmillaan esimerkkinä voisi käyttää kokonaislukujonon laskemista yhteen. Kun syötteenä on 1+2+3, voisi algoritmi laskea kaikki yhteen ja tulosteena olisi 6. Tämän jälkeen syötteenä olisi 1+2+3+4, niin tulosteena olisikin 10. Samaa algoritmin toimintaperiaatetta pystyy siis hyödyntämään kaikissa mahdollisissa tilanteissa, joissa kokonaislukujono pitäisi laskea yhteen.

Algoritmit eivät kuitenkaan rajaudu pelkästään laskutoimituksien tekoon, vaan mahdollistavat esimerkiksi erilaisten geometristen muotojen luomisen. Tässä kandidaatintyössä perehdytäänkin McNeel & Associates -yhtiön tuottamaan 3D-mallinnusohjelmaan Rhinoceros 3D (*Rhino 3D*) ja siihen kuuluvaan lisäosaan Grasshopperiin, jonka avulla voidaan tehdä algoritmista mallintamista.

2.3 Rhino3D ja Grasshopper

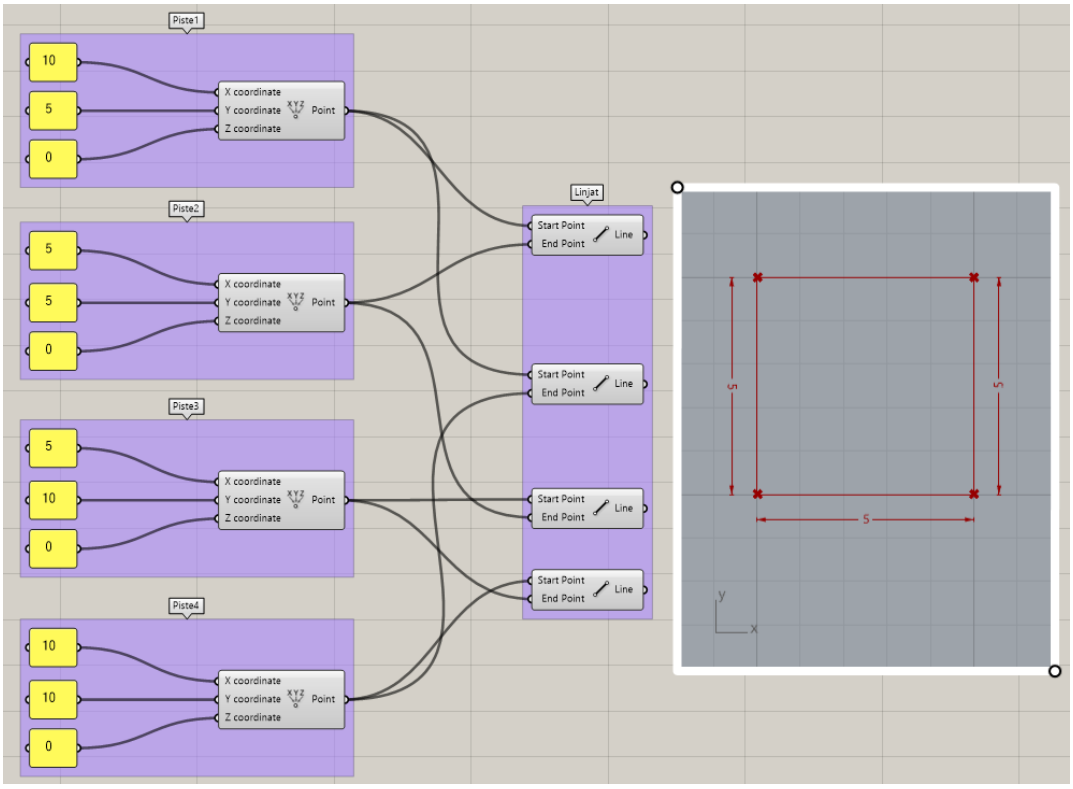
Rhino 3D on AutoCad-pohjaan perustuva 3D-mallinnusohjelma, jonka yksi vahvuusalue tulee NURBS (*Non-Uniform Rational B-Splines*) -geometrioiden hyödyntämisessä (Rhino3d 2023). NURBS-linjojen ja pintojen luominen perustuu matemaattiseen määrittelyyn, jonka seurauksena malli on hyvinkin tarkka. Tämän takia tarkan geometrian luomiseen tarvitsee huomattavasti vähemmän tietoa kuin vertailukohteena olevaan mesh-pohjaiseen malliin. (Tanska & Österlund 2014, s. 30) Mesh-malli rakentuu geometrian päälle luodusta verkosta (*mesh*), joka koostuu solmupisteistä (*vertices*), niiden välillä olevista linjoista (*edges*) ja linjojen välille tehdyistä pinnoista (*faces*) (Autodesk 2015). Koska malli rakentuu solmupisteiden välillä, halutun tarkkuuden luominen on suoraan riippuvainen pisteiden määrästä. Paremman tarkkuuden saavuttaminen johtaa myös suurempaan määrään tietoa käsiteltäväksi. Kuvassa 4 esitellään tarvittavien pisteiden määrää pallomaisen geometrian luomiseen. Vasemmalla puolella oleva NURBS-pinta saavuttaa hyvän tarkkuuden 26 kontrollipisteellä, kun taas vertailukohteena olevat mesh-mallit vaativat huomattavasti useamman. Mesh-malleissa huomataan tarkkuuden parantuminen solmupisteitä lisäämällä, mikä havainnollistaa hyvin mallin rakentumisen toimintaperiaatteen. Saman tarkkuuden saavuttamiseen NURBS-pinnalle riittää siis huomattavasti pienempi tietomäärä, joka on koneellisesti myös vähemmän kuormittavaa.



Kuva 4. *NURBS-pinnan ja mesh-mallin tarkkuuksiin vaadittu pistemäärä (Tanska & Österlund 2014, s. 30).*

Grasshopper on visuaalinen algoritmieditori, jonka tuottama algoritminen malli on linkattu Rhinoon nähtäväksi (Grasshopper3d 2023). Perinteisemmin algoritmin luominen on ollut mahdollista tuottaa koodaamalla, mutta se vaatisi käyttäjältä ohjelmointikokemusta. Grasshopper tarjoaakin tähän ratkaisun *visuaalisella skriptauksella*. Toiminta perustuu siis valmiiksi määritettyjen ohjelmakomponenttien liittämisellä toisiinsa, eikä ohjelmointikoodia itsessään tarvitse osata kirjoittaa. Monien erilaisten komponenttien valikosta käyttäjän on mahdollista luoda oma kokonaisuutensa oikeita yhdistelmiä tuottamalla.

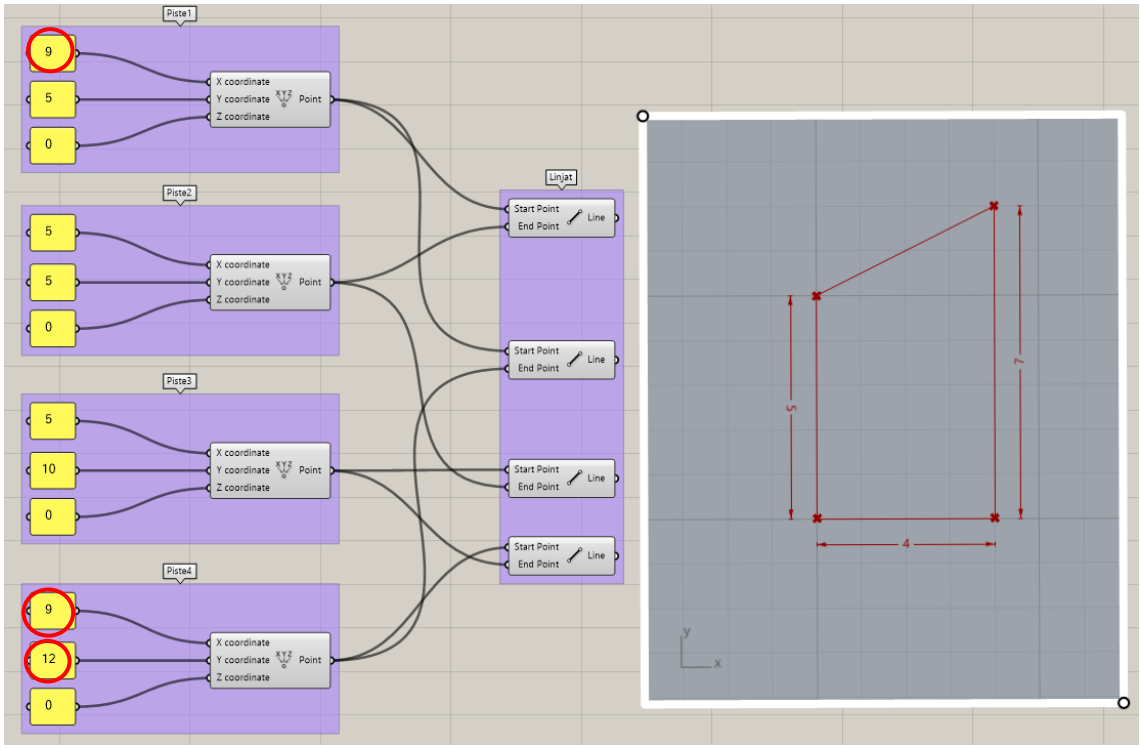
Aiemmin mainittuun algoritmin ratkaisun laajuuteen Grasshopper tarjoaakin hyvät lähtökohdat. Kuvitellaan ongelmaksi rakennuksen ulkolinjan muodostaminen neljän pisteen kautta, mutta pisteen tai pisteiden koordinaatit saattavat muuttua. Kuvassa 5 esitellään esimerkki Grasshopperin *skriptistä*, kuinka ulkolinja saataisiin muodostettua tunnettujen pisteiden koordinaattien välille.



Kuva 5. *Esimerkki rakennuksen ulkolinjasta, jossa jokaisen sivun pituus olisi 5.*

Vasemmalla on Grasshopperissa luotu algoritmi, jonka tuotoksena on oikealla puolella Rhinossa näkyvä rakennuksen ulkolinja. Tulosteen vaihe alkaa koordinaattien syöttämisellä, jotka nähdään vasemmalla keltaisissa *Panel* -komponenteissa. Tämän jälkeen käytetään *Construct Point* -komponenttia, joka luo tulevien koordinaattien perusteella pisteen. Lopuksi pisteiden välille luodaan suorat linjat *Line* -komponentilla. Ratkaisu ei

siis rajaudu pelkästään tähän, vaan parametreja muuttamalla, eli tässä tilanteessa koordinaatteja, saadaan muodostettua uusi ulkolinja pisteiden välisiä relaatioita rikkomatta. Uusi tuotos nähdään kuvassa 6.

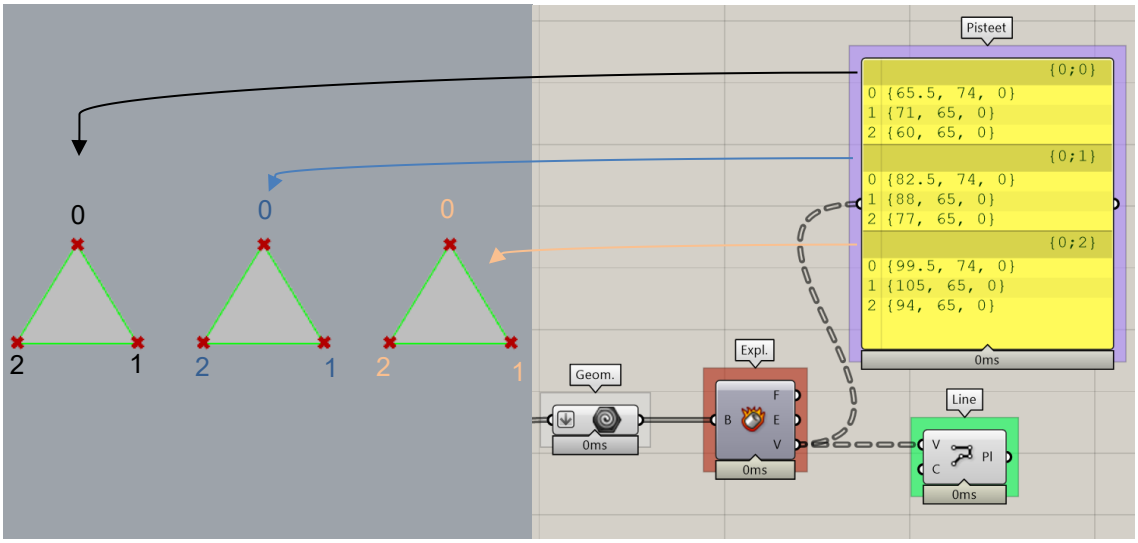


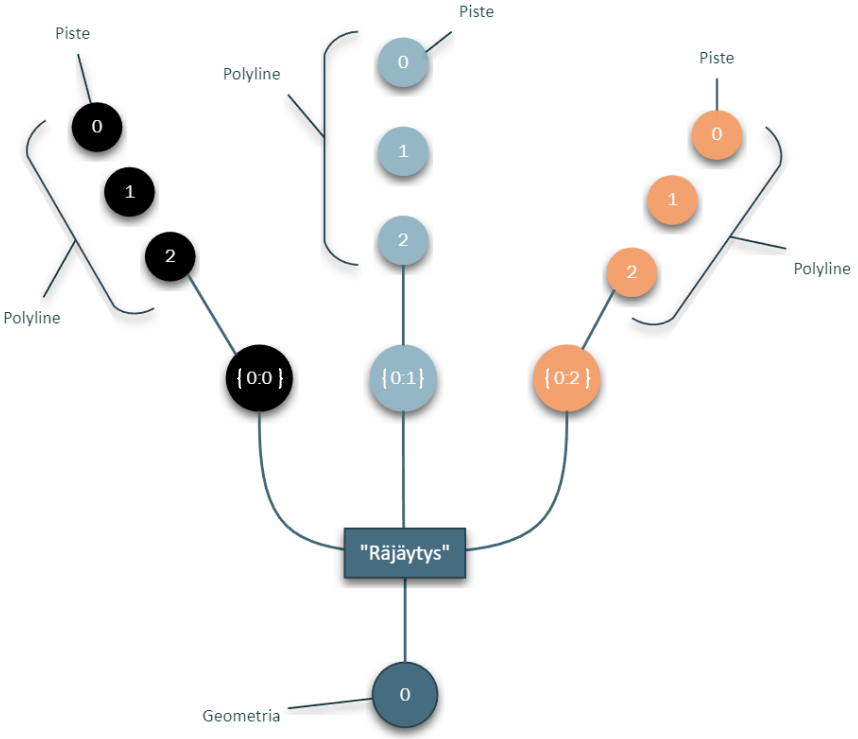
Kuva 6. *Uudet rakennuksen ulkolinjat, jossa muunnetut parametrit on ympyröity.*

Visuaalisen ohjelmoinnin hyvä puoli ilmenee myös komponenttien välisen yhteyden hahmottamisen kanssa, sillä käyttäjän on helppo seurata datan kulkemista algoritmissa niin kutsuttujen *johtojen* avulla. Hyvänä tapana on myös rakentaa algoritmin kulku loogisesti vasemmalta oikealle ja ryhmitellä toimintoja, jotta ymmärtää datan käyttäytymisen.

Tärkeä huomioitava tekijä Grasshopperissa onkin sen tapa käsitellä dataa, joka perustuu puurakenteeseen. Aloitteleva käyttäjä sivuuttaa asian helposti aihepiirin haastavuuden takia, mutta ennemmin tai myöhemmin tulee vastaan algoritmeja tehdessä. Datapuurakenne koostuu alkuun siis listasta *(trunk)*, eli *rungosta*. Komponenttien läpikäymisen jälkeen runkoon alkaa kehittyä alilistoja *(branch)*, eli *oksia*. Algoritmin edetessä Grasshopper rakentaa omaa *datapolkua* puurakenteella, joka mahdollistaa suurempienkin tietomäärien käsittelemisen. (Tedeschi 2011) Komponenttien käyttäytyminen halutulla tavalla on tähän puurakenteeseen sidottu ja miten sen oksakohdissa oleva data on muodostettu. Kuvassa 7 esitellään esimerkki tilanteesta, jossa ennalta määritettyjen kolmioiden geometriasta haluttaisiin erottaa ulkolinjat. Vasemmalla harmaalla pohjalla oleva ensimmäinen komponentti sisältää kolmioiden geometrian, jonka jälkeen se *”räjäytetään”* osiin punaisella pohjalla olevalla komponentilla. Tästä saadaan kärkipisteet, joiden välille linjat luodaan. Vasemmalla ylhäällä on Rhino-näkymä, jossa harmaalla pohjalla on kolmioiden

alkugeometria. Punaiset pisteet ovat *"räjäytys"* komponentin tuottamat kärkipisteet ja vihreällä merkattu niiden välille luodut linjat. Oikealla ylhäällä on Grasshopper -näkymä ja näiden alla kyseisen prosessin datapuurakenne.

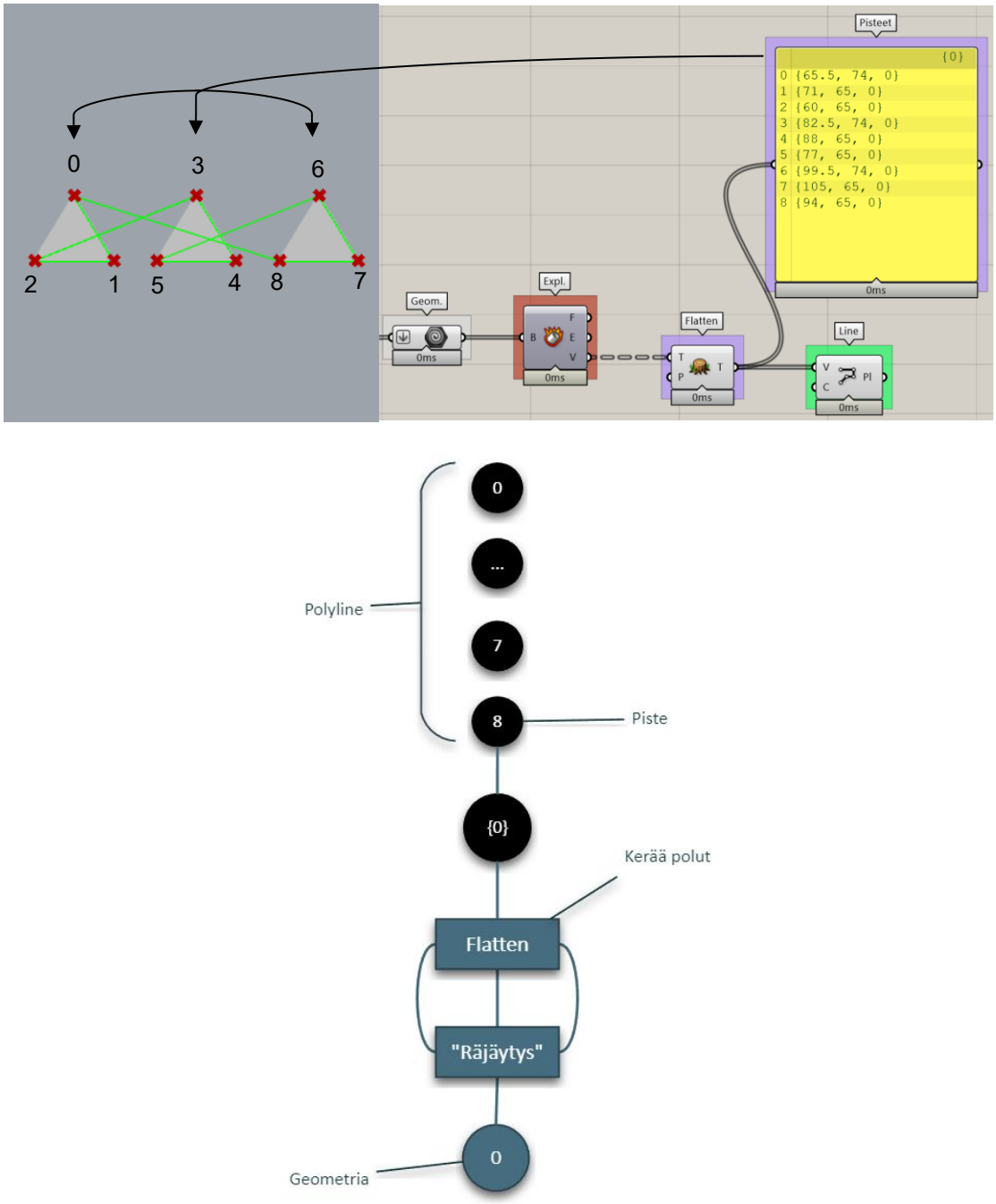




Kuva 7. *Esimerkki Grasshopperin datapuurakenteesta. Tässä tilanteessa linjat muodostuvat jokaiselle kolmiolle erikseen. (Tedeschi 2011, muokattu)*

Tässä huomataan Grasshopperin tapa rakentaa omaa datapolkuaan. Se jakaa vasemmanpuoleisimpaan kolmioon liittyvät pisteet polkuun {0;0}, keskimmäisessä polkuun {0;1} ja oikeanpuoleisimmassa polkuun {0;2}. Nämä polut ovat puurakenteen *oksakohdat*. Tässä tilanteessa linja rakentuu siis oksakohtaisesti niissä olevien pisteiden välille.

Puurakennetta joutuu kuitenkin välillä muokkaamaan, jotta haluttu toiminto toteutuu. Kuvassa 8 esitetään *flatten* -komponentti, joka yhdistää kaikki oksat yhteen polkuun ja miten sama algoritmi käyttäytyisi uudella puurakenteella.



Kuva 8. *Flatten -komponentin vaikutus puurakenteeseen. Tässä tilanteessa linjat muodostuvat kaikkien pisteiden välille. (Tedeschi 2011, muokattu)*

Huomataan että samalla algoritmilla, mutta puurakenteen muutoksella, saavutetaan aivan erilainen lopputulos. Linjan luomiseen sopii siis paremmin aiemmin esitelty puurakenne, mutta toisessa käyttötarkoituksessa tilanne voisi olla päinvastainen. Esimerkiksi

jos kaikista kolmioiden kärkipisteistä halutaan valita tiettyjä kohtia yksilöllisesti, niin kuvan 8 rakenne soveltuisi paremmin. Siinä tapauksessa listasta poimittuna indeksillä 2 valittaisiin pelkästään vasemmanpuoleisin piste. Kuvan 7 rakenteella taas indeksillä 2 tulisi valituksi jokaisesta *oksasta* kyseisellä indeksillä oleva arvo, eli yhden pisteen sijaan valituksi tulisikin kolme.

Grasshopper tarjoaa useita komponentteja puurakenteen muokkaukseen liittyen, joista käyttäjä valitsee omaan tilanteeseen parhaiten soveltuvia. Valinnassa on kuitenkin hyvä pysyä tarkkana ja seurata oman datan käyttäytymistä, sillä väärällä valinnalla voi helposti tulostaa saman asian useampaan kertaan. Jos ei ongelmaa heti huomaa, niin myöhemmässä vaiheessa algoritmia komponenttien käyttäytymisessä saattaa ilmetä ristiriitaisuuksia.

Grasshopperin käyttötarkoituksia on myös mahdollista laajentaa erilaisien lisäosien avulla. Tässä työssä esimerkiksi perehdytään Grasshopper-Tekla Live Link -lisäosaan, joka mahdollistaa algoritmisen mallintamisen Tekla Structures -ohjelmistossa.

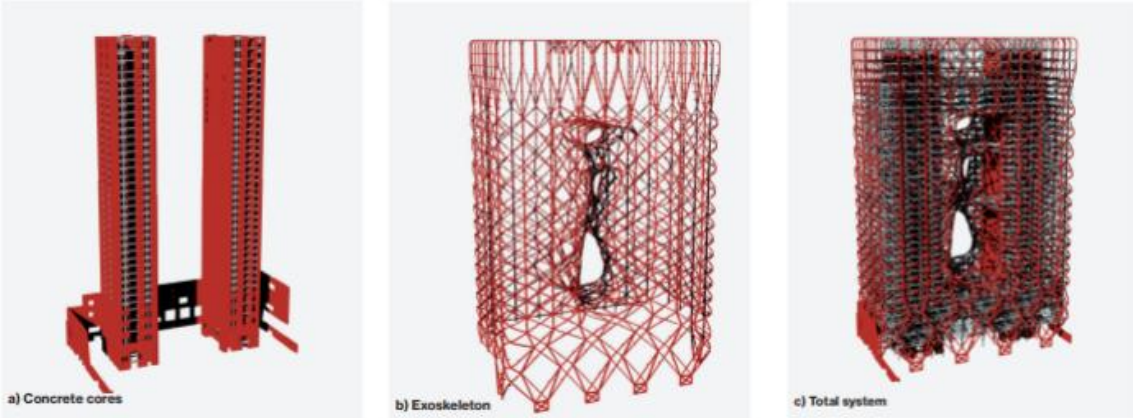
2.4 Hyödyntäminen rakennesuunnittelussa

Suurin käyttö algoritmisella suunnittelulla rakennusalalla on tällä hetkellä arkkitehdeilla niiden työnkuvansa myötä. Lalla (2019) tekee karkean yleistyksen arkkitehdin roolista, jossa heidän tehtävänsä olisi suunnitella rakennuksen tilat ja muoto. Pääasiallisen työtehtävän rajauksella voidaan paremmin havainnollistaa mitä algoritmisella suunnittelulla tarkoitetaan. Erilaisten rakennuksien muotojen ja tilojen sijoittelun visualisointi *perinteisellä tietomallintamisella* olisi työlästä eri vaihtoehtojen laajuuksien myötä. Algoritmisessa suunnittelussa taas otetaan jo alusta lähtien huomioon rajaehdot rakennukselle, joka antaa mahdollisuuden käymään nopeammin erilaisia vaihtoehtoja läpi parametreja muuttamalla.

Algoritminen suunnittelu ei kuitenkaan rajaudu pelkästään arkkitehteja varten. Sen vahvuudet tulee esille haastavampien geometrioiden mallintamisessa, sekä niiden mitoittamisessa. Tästä erittäin vaikuttava esimerkki on Kiinan Macaussa sijaitseva The Morpheus Hotel, joka on esitetty kuvassa 9. Hotellin suunnittelussa hyödynnettiin pääosin Rhinoceros3D ja Grasshopper -ohjelmistoja.

Kuva 9. *The Morpheus Hotel. (Permarini et al. 2016, s. 57)*

Hotellin haastavan julkisivun geometrian lisäksi liitosten määrän suunnittelussa, olisi ollut huomattavan paljon työtä ilman algoritmisen suunnittelun tuottamia mahdollisuuksia. Piermarini (et al. 2016) kertovatkin hyvin teräsrungon suunnittelun vaiheista heidän artikkelissaan. Rakennuksen kantavan rungon visualisoinnin valmistuttua, päädyttiin kahteen betoniseen ytimeen, jonka ympärille luotiin vahvistukseksi teräksinen runko. Yhdessä ne toimivat vahvana elementtinä tulevia sivuttaiskuormia vastaan. Rungon visualisointi esitellään kuvassa 10.



Kuva 10. *The Morpheus hotellin kantava runko. (Permarini et al. 2016, s. 57)*

Teräsrungon yksityiskohtaisille liitoksien dokumentoinnille ja suunnittelulle jäi lopulta 12 kuukautta aikaa. Työn laajuuden takia olisi pitänyt toteuttaa keskimäärin 50 liitosta joka

viikko, jotta aikataulullisiin tavoitteisiin olisi päästy. Tällöin suunnittelutiimi vetosi aikaisempaan kokemukseensa algoritmisessa suunnittelussa ja jakoi ongelman viiteen pääosaan:

1. samanlaisten liitosten tunnistaminen
2. liitoksen kytkentäjärjestelyn kehittäminen
3. globaalissa mallissa olevien kuormien löytäminen ja kartoittaminen
4. liitoksen analysointi
5. analyysiraporttien, sekä rakentamisasiakirjojen luonti.

Rakennuksen globaali malli, joka analysoi rakenteiden rasituksia oli luotu MIDAS Structural Analysis-ohjelmistolla. Laskentaprosessi kesti jo 12 tuntia ilman liitosten analysointia, joten uusi ratkaisu piti kehitellä. Geometrian alkuperäiseen hahmottamiseen oli kuitenkin tehty jo Rhinoceros 3D malli, jota voitiin hyödyntää liitoksien määrityksessä Grasshopperissa. Ensimmäisen kohdan ratkaisuna luotiinkin Grasshopper *skripti*, joka selvitti samantyyppisten liitosten lukumäärän. Tällä prosessilla saatiin vähennettyä runkoon tarvittavien uniikkien liitosten lukumäärä alkuperäisestä 2500:sta uuteen 400:aan.

Seuraavaksi piti miettiä liitostyyppi, joka on mahdollista valmistaa tehtaalla ja asentaa työmaalla. Näiden lisäksi annettiin arkkitehdillisia rajoituksia liitoksen tyyliin liittyen. Ratkaisuna Grasshopperiin syötettiin alustavana parametrina liitoksen todennäköisimmät mitat täyttääkseen ehdot. Grasshopperin ja Rhinon vahvuuksina tässä projektissa olikin mallin automaattinen päivittyminen parametreja muuttamalla, sillä mahdollinen ristiriita huomattiin heti alustavien parametrien kanssa. Näitä muuttamalla pystyttiin nopeasti luomaan ehdot täyttävä liitos analysointia varten.

Ennen analysointia piti kuitenkin saada liitoksiin kohdistuvien voimien suuruudet määritettyä globaalista mallista. Kuormitustapauksia ollessa 105 erilaista ja yhdessä liitoksessa jopa 9 eri osaa kiinni, ilmeni haasteita oikeiden rasitusten löytämisessä ja sijoittamisessa liitoskohtaisesti. Tässä kehiteltiinkin *skripti*, jossa globaalista mallista tunnistettiin ja tuotiin Grasshopperiin liitoksien solmupisteiden numeroinnit, solmupisteissä yhtenevien linjojen profiilit, sekä linjojen suunnat. Profiileissa vaikuttavat voimat ja momentit tuotiin myös, jotka sitten lisättiin suunniteltuihin liitoksiin. Voimakomponenttien todellisuuden tarkistamisessa auttoi ohjelmistojen ominaisuus näyttää visuaalisesti vektoreita mallissa, jolla voitiin todentaa voimakomponenttien suunnat.

Neljäntenä osa-alueena oli liitokseen syntyvien jännitysten laskeminen FEM-ohjelmistolla. Laskenta suoritettiin Robot Structural Analysis -ohjelmistolla (RSA), jossa käytettiin hyödyksi Live-Link toimintoa Grasshopperin ja RSA:n välillä. Skriptillä luotiin siis lokaali

malli liitoksesta RSA-ohjelmistoon sen hyväksymään muotoon, lisättiin malliin liittyvät rajaehdot, suoritettiin analysointi ja kerättiin tulokset. RSA:n tuottamasta jännitysjakaumasta voitiin nähdä, syntyykö millään alueella rajojen yli meneviä jännityksiä. Jos korjattavaa ilmeni, niin liitoksen asettelutyyliä voitiin muokata, kunnes saatiin hyväksytty jännitysjakauma.

Viimeisenä kohtana oli analyysiraporttien dokumentointi ja rakennusasiakirjojen tuottaminen. Koska kaikki datan käsittely suoritettiin Grasshopperin puolella, niin skriptin luominen, joka siirtäisi tulokset suunnittelun etenemisestä Excel-tiedostoon oli looginen ratkaisu.

The Morpheus Hotel on hieno taidonnäyte algoritmisen suunnittelun kyvykkyydestä, joka antaa hyvän esimerkin ajankäytön säästämisestä ja hyötykäyttämisestä oikeaan insinöörityöhön. Useat toistoa vaativat toimenpiteet siis siirrettiin tietokoneen hoidettavaksi, jonka jälkeen lopullisten tulosten analysointi jäi suunnittelijoiden tehtäväksi.

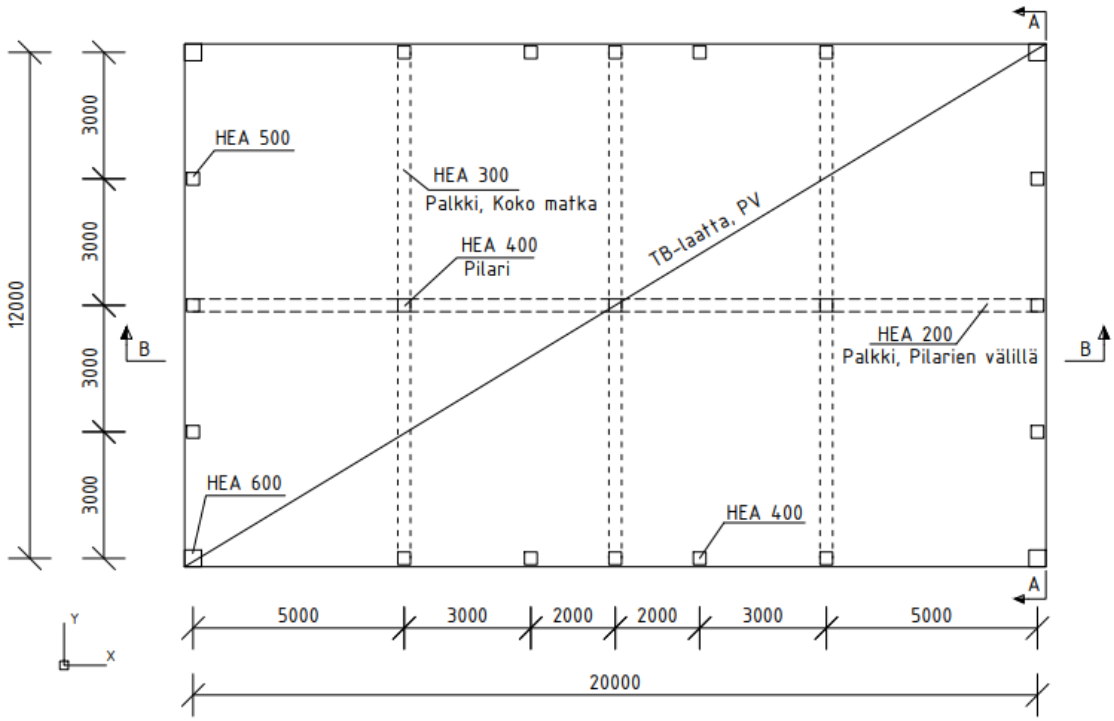
3. RUNGON GENEROINTI

3.1 Kohteen esittely

Tässä työssä perehdytään teräsrunkoisen pilari-palkkirakennuksen generointiin Grasshopperin avulla Tekla Structures -ohjelmistoon. Viimeisenä vaiheena luodaan osasta runkoa valmis rakenneanalyysimalli RSA-ohjelmistoon, sekä vertaillaan laskennan tuloksia perinteisesti tuotettuun malliin. Käyttäjän on mahdollista määrittää seuraavat rakenneosat:

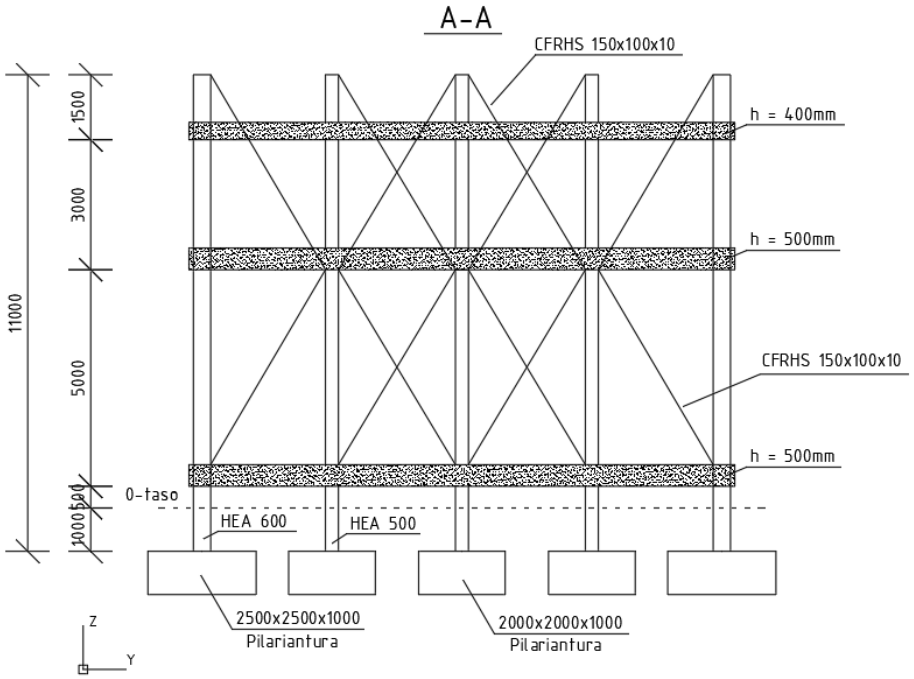
1. runkopilarit ja -palkit
2. anturat
3. ristikkojäykisteet seinällä
4. kerroslaatat
5. kerrostuennat (pilarit ja palkit).

Käyttäjän on myös mahdollista muokata edellä mainittuja rakenneosia. Esimerkki toteutetaan kuvitteellisella kaksikerroksisella varastohallilla, joka esitellään kuvassa 11. Molemmissa kerroksissa on samanlainen kerrostuenta.

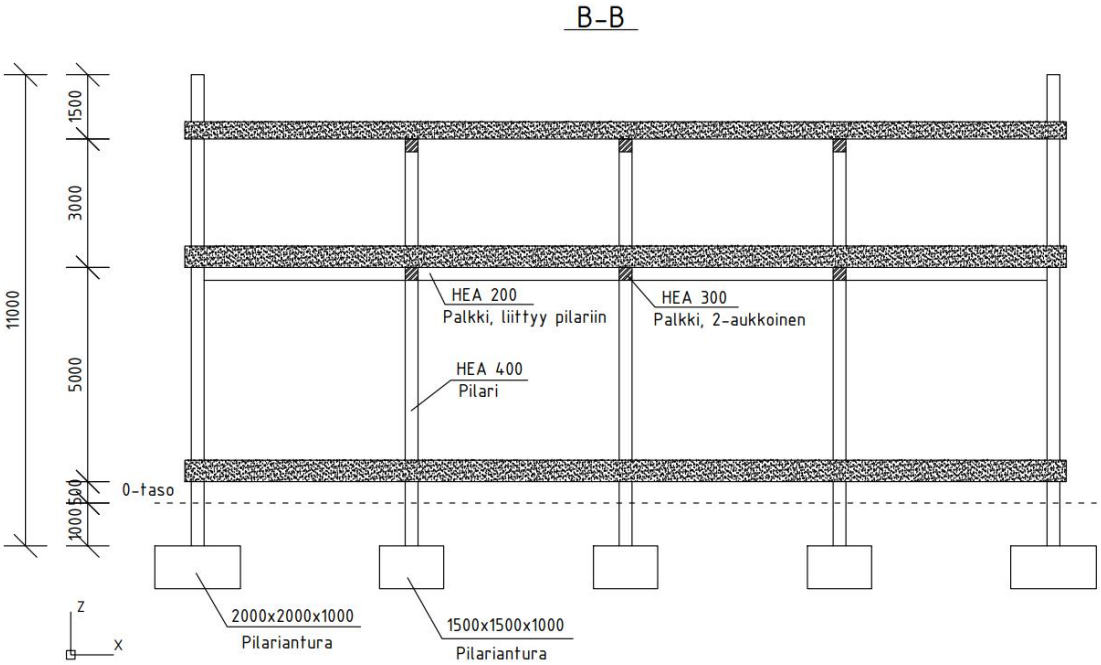


Kuva 11. *Varastohallin tukirakenteiden pohjapiirros.*

Kuvan selkeyttämisen takia jokaisen rakenneosan profiilia ei olla erikseen merkattu. Kuvaan 9 merkatut profiilit tarkoittavat siis kaikkia samaan ”tyyppiin” tarkoitettuja rakenneosia. Esimerkiksi kaikki nurkkapilarit ovat HEA 600 -profiileja, mutta merkintä löytyy vaan vasemmasta alanurkasta. Vastaavasti pituussuunnan runkopilarit ovat HEA 400 -profiileja ja leveyssuunnan runkopilarit HEA 500 -profiileja. Rakennuksen leikkauskuvat on esitelty kuvissa 12 ja 13.



Kuva 12. *Varastohallin leikkauskuva A-A.*

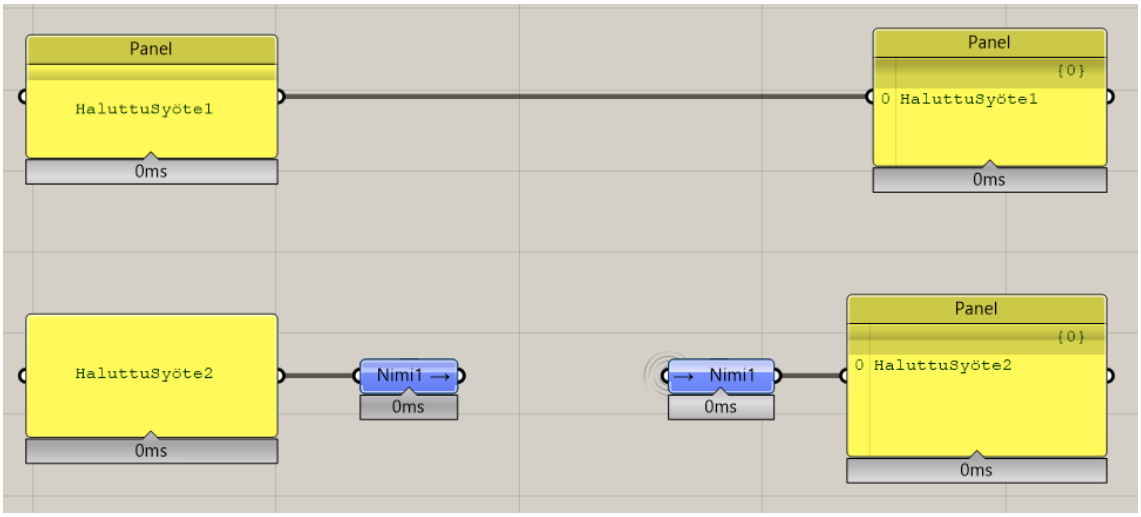


Kuva 13. *Varastohallin leikkauskuva B-B.*

Varastohallissa on turhan paljon rakenneosia sen kestävyyden kannalta, mutta tarkoituksena on esittää mahdollisimman laajasti algoritmin mahdollisuudet. Rajoitteena on, että rungon ulkolinjan tulee olla suorakulmion muotoinen, kerrokset ovat samassa tasossa ja kerrostuenta on samanlainen. Rakenteiden välisten liitosten generointi jätetään huomiotta niiden laajuuden takia.

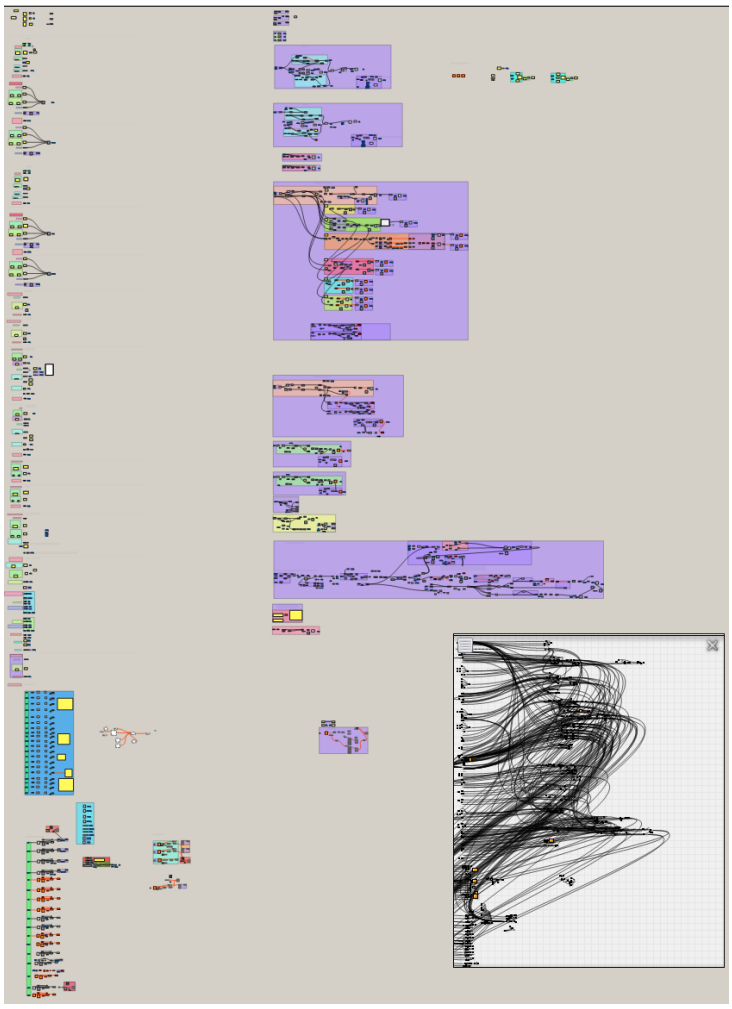
3.2 Lähtökohdat ja rajoitukset

Käytettyinä ohjelmistoina on Rhino3D, Grasshopper, Tekla Structures ja Robot Structural Analysis. Taustalla toimivina lisäosina on käytetty Grasshopper-Tekla Live Linkin lisäksi Telepathya ja SnappingGeckoa. Telepathy on Marc Sypin ja Andrew Heumannin kehittämät komponentit, jotka selkeyttävät Grasshopperin *canvas*-näkymää vähentämällä *johtojen* lukumäärää. Toiminnaltaan se yhdistää samalla nimityksillä olevat syötteet toisiinsa ilman *johdon* liittämistä, joka esitellään kuvassa 14. SnappingGecko on weArk yhtiön kehittämä toiminto, joka auttaa komponenttien järjestelyssä. (Food4Rhino)



Kuva 14. *Ylempänä on tutumpi johtojen liittäminen haluttuun komponenttiin. Alempana Telepathyn toiminta, eli vasen lähettäjä (sender) yhdistyy samalla nimellä olevaan vastaanottajaan (receiver).*

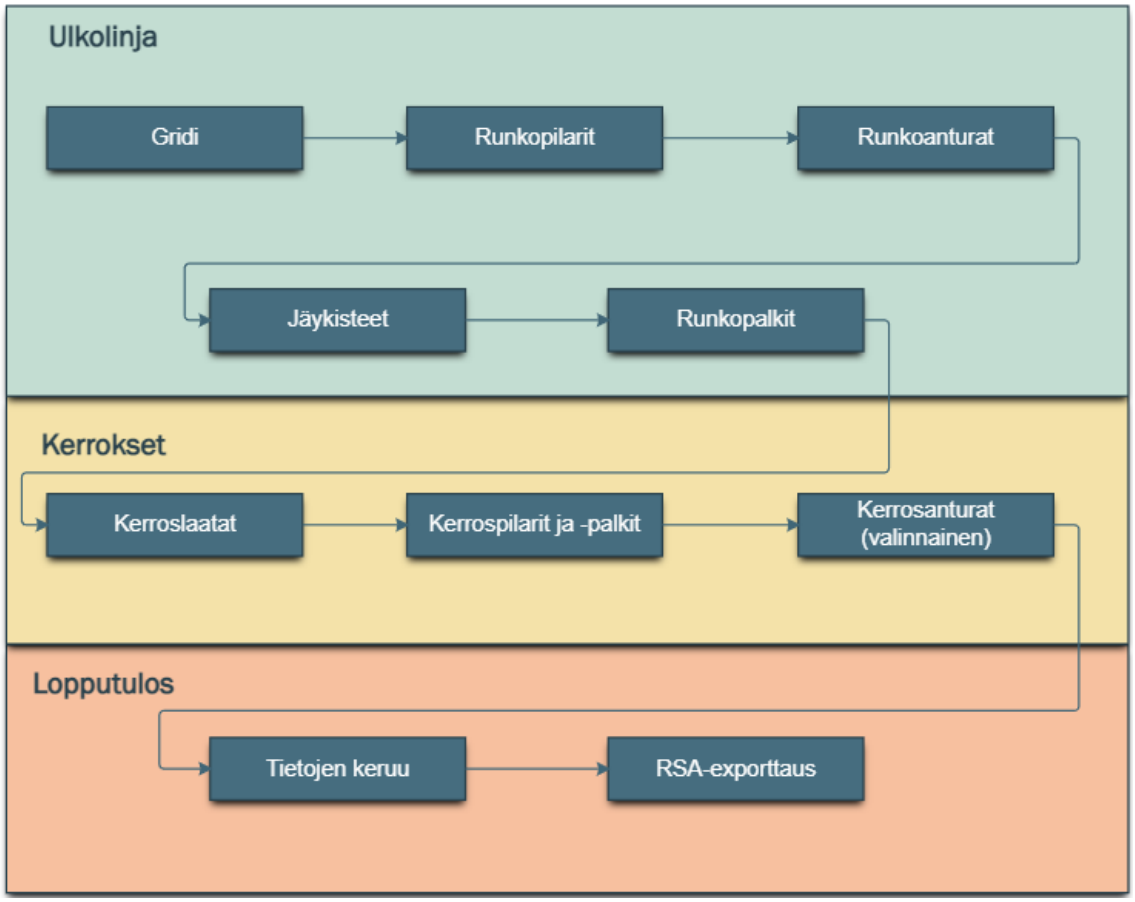
Suurissa tiedostoissa algoritmin selkeys on erittäin tärkeää, jotta jälkikäteen on helpompi tehdä tarpeen mukaan lisäyksiä tai korjauksia. Muokkaukset on huomattavasti helpompi toteuttaa, jos pystyy erottamaan eri kokonaisuuksien toiminnan ja niistä rajaamaan yksittäisten komponenttien käyttäytymisen. Grasshopperin johdoista kertyy tosi nopeasti sekavan oloinen kokonaisuus, ellei pidä järjestystä komponenttien välillä. Kuvassa 15 esitetään tässä työssä käytetyn algoritmin *canvas*-näkymä Grasshopperissa järjesteltynä, sekä vertailuna näkymä, jos kaikki johdot olisivat nähtävillä.



Kuva 15. *Tässä työssä käytetyn algoritmin canvas. Oikealla alhaalla esitelty näkymä, jos kaikki johdot olisivat näkyvissä.*

Tärkeänä huomiona on ohjelmistojen käynnistysjärjestys. Jotta linkin saa muodostettua Grasshopperin ja Teklan välille, niin täytyy käynnistää ensin Tekla ja tämän jälkeen Rhino. Toisena tärkeänä huomiona on, että jos mallia muokkaa Teklassa niin linkki katkeaa. Mallia ei kuitenkaan ole sidottu pelkästään Grasshopperin algoritmiin, vaan on mahdollista käyttää *bake* toimintoa, joka lisää generoidut rakenteet pysyvästi malliin. Tämän jälkeen on mahdollista jatkaa mallin käsittelyä Teklassa tai lähettää tiedosto muille eteenpäin.

Käyttäjän syötemahdollisuudet Grasshopperissa on esitelty loogisesti ylhäältä alaspäin etenevässä järjestyksessä. Rhinon ikkunaa ei ole välttämätöntä pitää näkyvissä samaan aikaan, sillä Grasshopperissa tehdyt muutokset generoituvat automaattisesti Tekla -malliin. Rakenneosien luomisen voi jakaa kolmeen osakokonaisuuteen, ulkolinjaan, kerroksiin ja lopputulokseen. Mallin generoinnin vaiheet on esitetty kuvassa 16.



Kuva 16. *Algoritmin vaiheet.*

Generoinnin vaiheet on syytä toteuttaa kuvan 16 esittämässä järjestyksessä, sillä aiemmin määritettyjä tietoja hyödynnetään tulevissa kokonaisuuksissa. Ulkolinjan muokkausmahdollisuuksissa on pituus- ja leveyslinja eroteltu toisistaan, vaikka niitä ei kuvaan 16 olla erikseen merkattu.

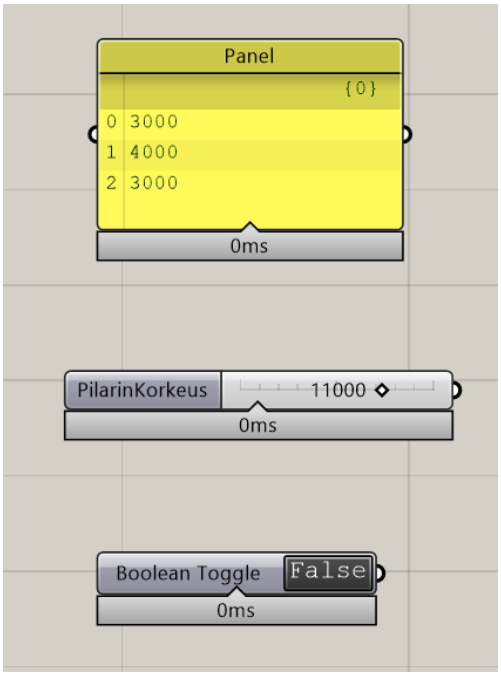
3.3 Algoritmin toiminta

Ohjelmistojen avaamisen jälkeen täytyy ensin poistaa Teklan generoima automaattinen gridi-linja, sillä uusi tuotetaan Grasshopperissa. Seuraavissa luvuissa käsitellään aiemmin mainitun prosessikaavion vaiheet ja muokkausmahdollisuudet niissä. Haasteita ilmeni useiden eri muokkausmahdollisuuksien tuottamisessa käyttäjälle ilman, että syötteiden määrä vaikuttaisi liian monimutkaiselta tai haastavalta. Tämän seurauksena rakenneosien yksityiskohtaiset muokkausmahdollisuudet esitellään tarkemmin pelkästään reunapilareiden luomisessa, vaikka se olisi mahdollista toteuttaa jokaiselle vaiheelle. Käyttäjän helpottamiseksi jokaisen syötekohdan vasemmalle puolelle on eroteltu esimerkki tilanteesta ja selostettu miten se vaikuttaa lopputulokseen.

Parametrien syöttäminen tapahtuu kolmella eri tyylillä niiden toiminnallisuuden myötä. *Panel* -komponentilla syötetään yksilölliset tiedot ja muuttuvilla etäisyyksillä olevat rakenteet, koska sillä saa helposti listamuodossa käsiteltyä haluttua rakennetta.

Toisena syöttötapana käytetään *Slider* -komponenttia, jolla voidaan nopeasti muokata lukumäärää ja visualisoida rakenteen sijainnin muutosta muihin osiin suhteutettuna.

Kolmas syöttötapa ei varsinaisesti ole muokkaukseen perustuva, vaan ehtolauseisiin perustuva, jolla käyttäjä voi säätää minkälainen rakenne tullaan generoimaan. Se toteutetaan *Boolean Toggle* nimisellä komponentilla, jossa on True ja False vaihtoehdot. Vaihtoehdoilla pystyy siis rakentamaan algoritmin sisälle ehtolauseisiin perustuvia kokonaisuuksia. Käytetyt syötetavat on esitelty kuvassa 17.



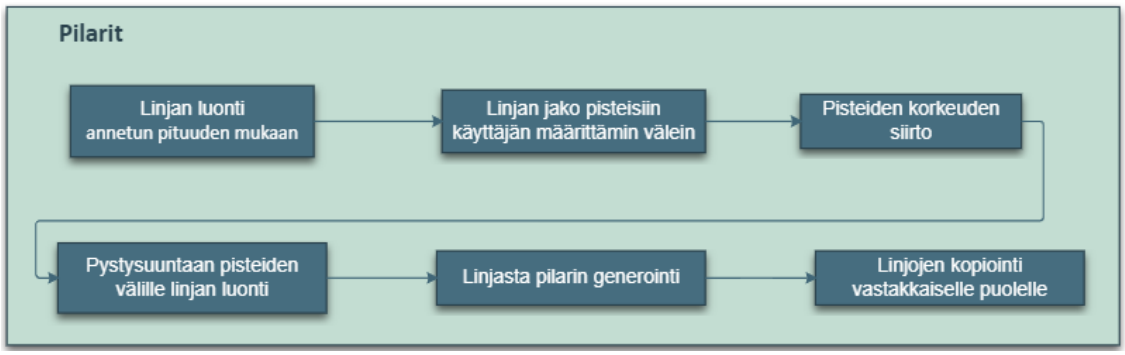
***Kuva 17**. Algoritmissa käytetyt syötteet.*

3.3.1 Gridi, runkopilarit ja runkoanturat

Rakennuksen rungon generoinnin ensimmäinen vaihe koostuu gridi-linjojen luomisesta. Linjojen määritys tapahtuu käyttäjän asettamilla väleillä, jonka jälkeen ne siirretään Live Linkin *Grid* -komponenttiin. Asetettuja välejä käytetään rakennuksen pituuden ja leveyden muodostamiseen, jota hyödynnetään jatkossa rakenneosille

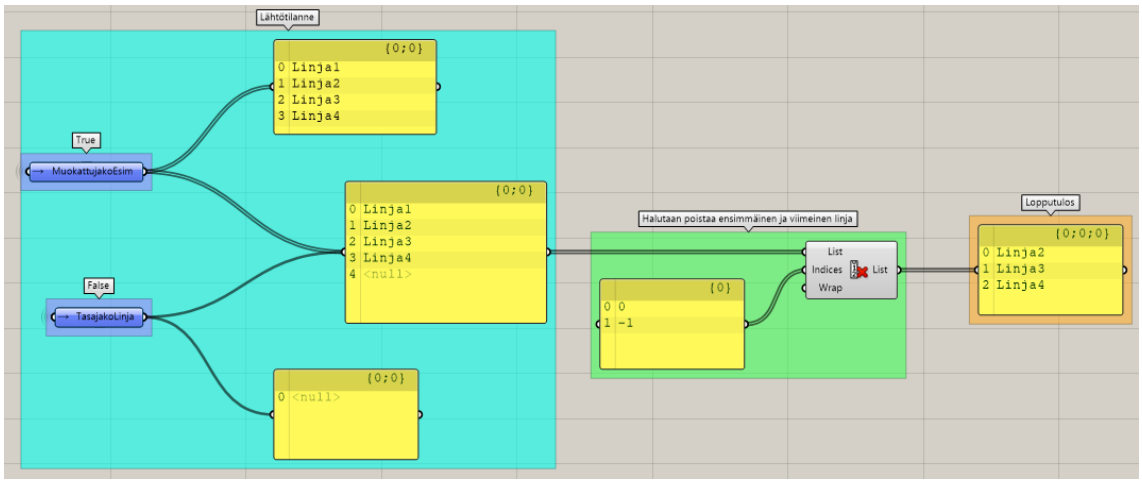
Gridin luomisen jälkeen siirrytään rakennuksen pituusakselilla (*x-linjalla*) olevien pilareiden määrittämiseen, jonka prosessikaavio esitellään kuvassa 18. Määritys alkaa yksinkertaisella *True/False* -komponentilla, jolla käyttäjä valitsee, tuleeko pilarit tasajaolla vai

muokatulla jaolla. Ehtolause on linkitetty algoritmissa Grasshopperin *Cull Pattern* -komponenttiin, jolla voi muokata mitä osia tulevasta listasta poistetaan. Valitsemalla pelkästään toisen näistä voi valita jatkaako tuloste eteenpäin (*True*) vai ei (*False*).



Kuva 18. *Runkopilareiden algoritmin prosessikaavio.*

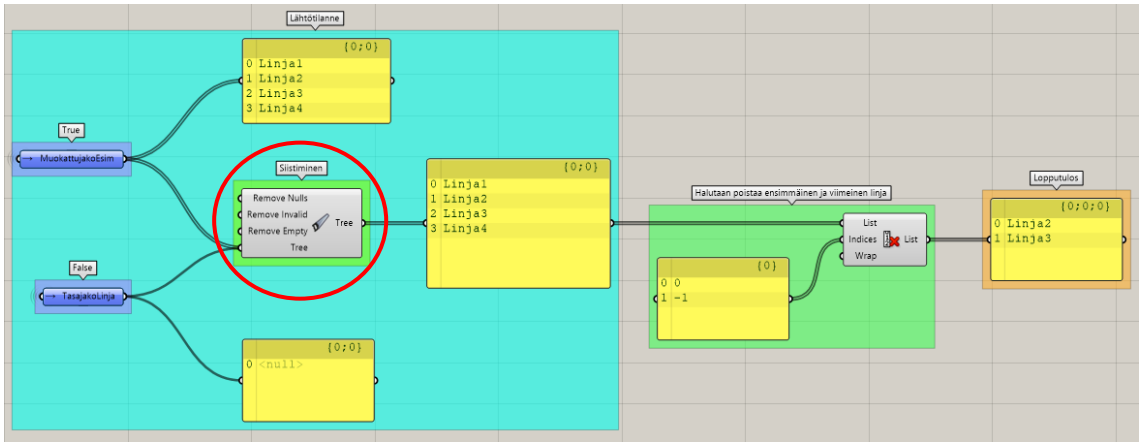
Reunapilarien generoinnissa ehtolause on määritetty algoritmin alkuun, sillä ainoa eroavaisuus löytyy niiden jaossa. Muutoin mallinnus toimii samalla tavalla, joten on turhaa tehdä skriptistä sekavampi tekemällä molemmille omat algoritmit. Tämän seurauksena joutuu hieman siistimään tiedostossa kulkevaa dataa komponenttien toimivuuden kannalta. Vaikka toisen valinnan jättäisi False muotoon, niin algoritmin vaiheissa syntyy silti lista, jonka arvona on *null* tai *empty*. Tämä saattaa vaikuttaa harmittomalta, koska syötteessä ei ole arvoa, mutta todellisuudessa näin ei ole. Grasshopper käsittää sen silti listana, jossa on arvo ja sen seurauksena muokkaa datapuurakennetta. Ongelma ilmenee etenkin komponenteissa, joilla pyritään muokkaamaan tulevan listan tietoja. Asiaa havainnollistetaan kuvassa 19, jossa on pilarijako valittu muokatulla jaolla. Jaon valinnasta huolimatta tulevasta listasta halutaan poistaa ensimmäinen *(indeksi 0)* ja viimeinen *(indeksi -1)* linja.



Kuva 19. *Esimerkki puurakenteen ongelmasta ehtolauseilla.*

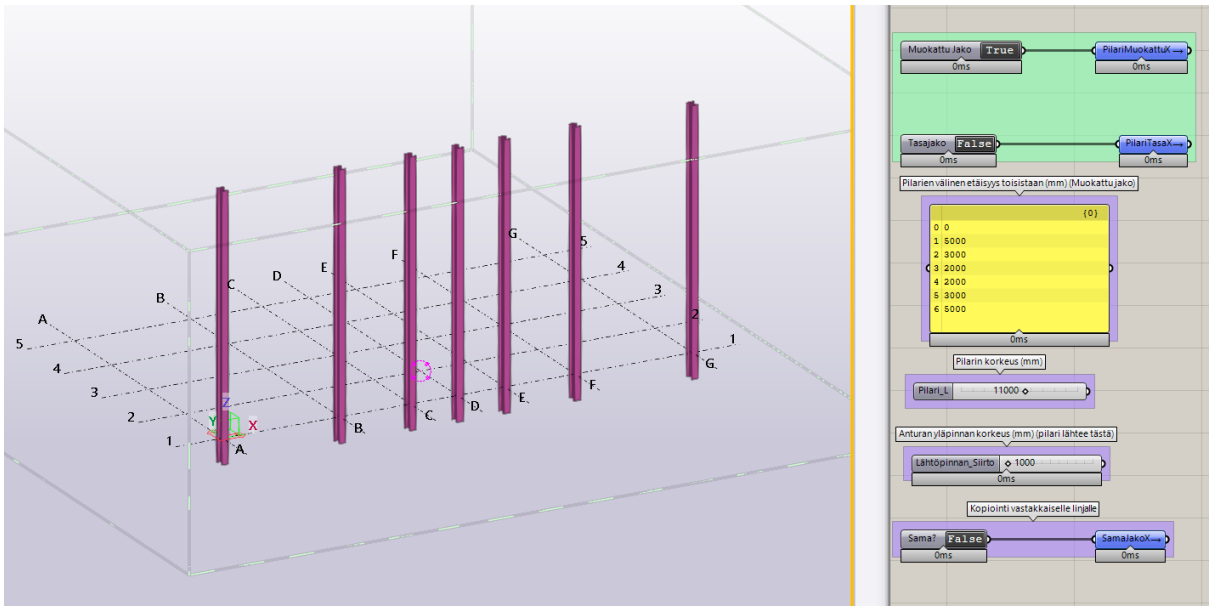
Kuvan 19 tilanteeseen on lisätty *Panel* -komponentteja havainnollistamaan tiedon kulkua. Tässä tilanteessa haluttaisiin siis poistaa vasemmalla ylhäällä muokatusta jaosta tulevasta listasta *Linja1* ja *Linja4* tiedot, eli ensimmäinen ja viimeinen linja. Tämän alapuolella tasajaosta aiheutuva *null* lista muuttaa puurakennetta kuitenkin siten, että keskellä vihreällä pohjalla oleva *Cull Index* -komponentti poistaa *Linja1* ja *null* tiedon aiheuttaen ei toivotun lopputuloksen.

Tämä ongelma korjataan *Clean Tree* -komponentilla, jolla voidaan siistiä tulevaa listaa ei toivotuista syötteistä. Tässä tilanteessa halutaan poistaa tyhjät listat ja listat, joissa on arvo *null* tai *empty*. Edellä mainitun esimerkin lopputulos esitellään kuvassa 20 siistimisen jälkeen.



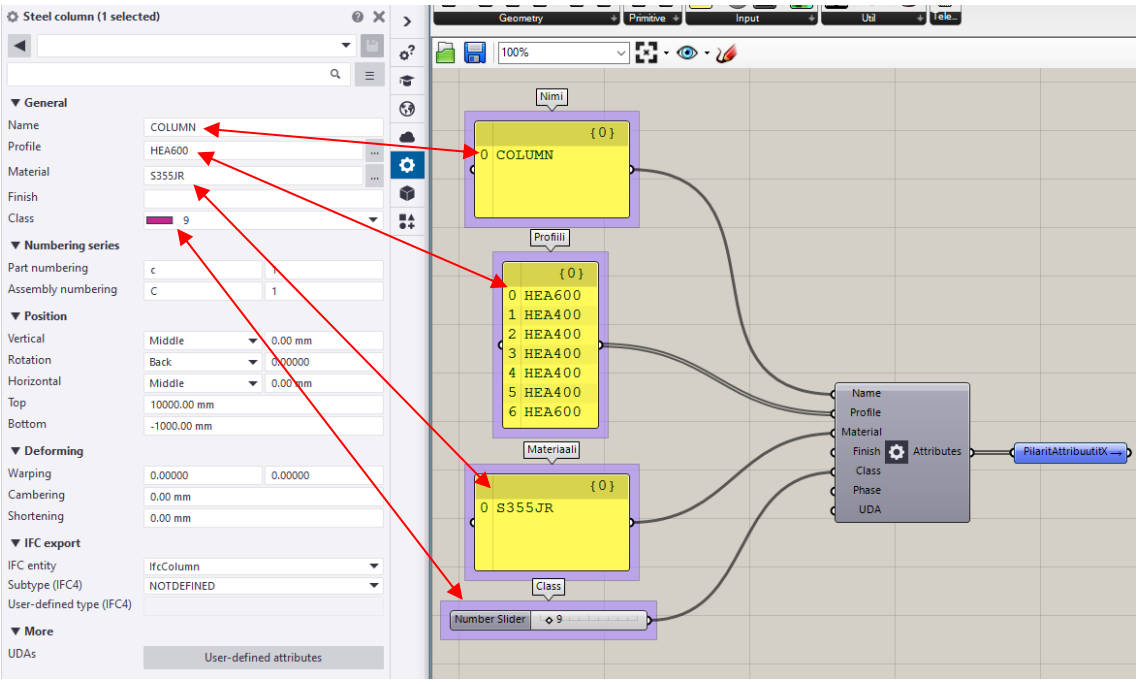
Kuva 20. *Clean Tree -komponentin toiminta (komponentti ympyröity).*

Pilarijaon tyylin valinnan jälkeen käyttäjä valitsee millä väleillä ne tullaan sijoittamaan. Jako toteutuu erottamalla gridillä luodun rakennuksen pituuslinja osapisteisiin, joiden korkeutta on mahdollista muuttaa. Korkeuden muuttamisella luodaankin käyttäjän määrittämä korkeus pilareille. Lähtökorkeuden säätö toimii samalla tavalla, pisteitä siirtämällä. Viimeisenä kohtana kysytään *True/False* -komponentilla onko vastakkaisella puolella samanlainen tyyli. Kuvassa 21 esitellään case-kohteen pituuslinjan runkopilarien syötteet Grasshopperissa ja näkymä Teklassa.



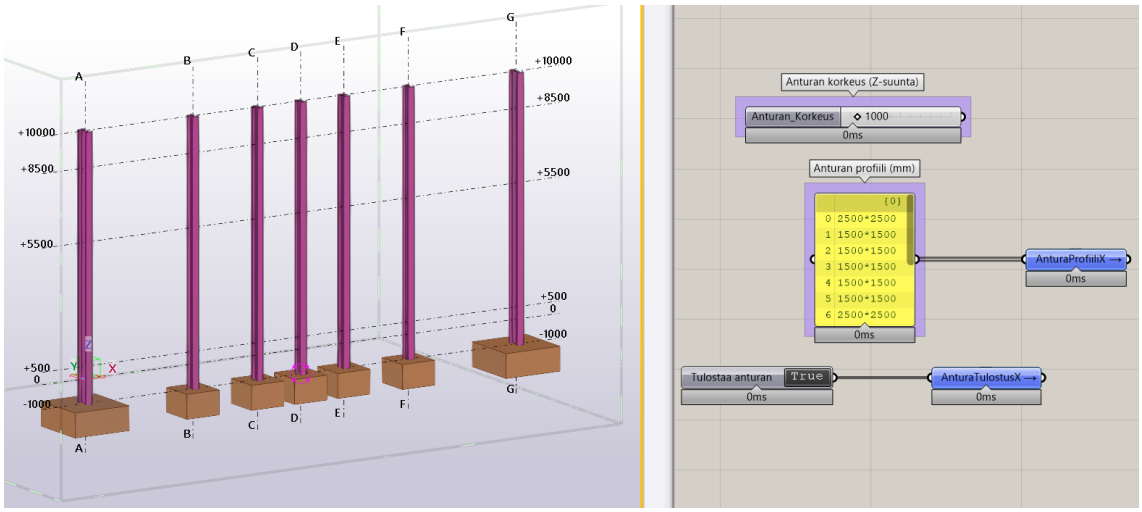
Kuva 21. *Tekla-näkymä ja Grasshopper syötteet pituuslinjan pilareille.*

Runkopilarien luomisen jälkeen käyttäjän on mahdollista yksilöllistää niihin liittyviä tietoja, eli attribuutteja. Tämä toteutetaan Live Linkin *Part Attributes* -komponentilla. Case-kohteen profiilien määritys Grasshopperissa ja nurkkapilarin tiedot Tekla-näkymässä on esitelty kuvassa 22.



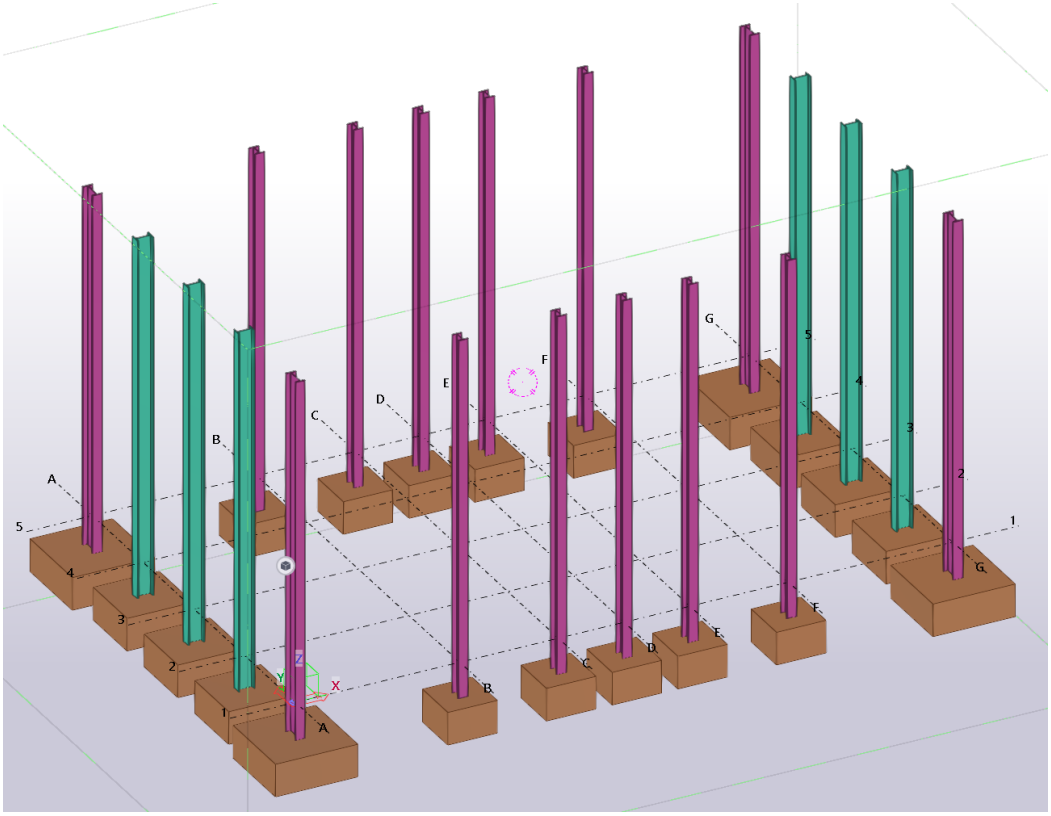
Kuva 22. *Pituuslinjan pilarien attribuuttien valinnat oikealla ja vasemmalla nurkkapilarin properties-näkymä. Nurkkapilariin liittyvät tiedot on nuolilla yhdistetty.*

Seuraavana vaiheena tulee anturaperustukset. Niissä hyödynnetään aiemmin määritettyjä pilarin lähtötason pisteitä, joiden korkeutta muokkaamalla saadaan anturoiden korkeus. Pituuden ja leveyden määritys tulee *Part Attributes* -komponentilla. Kuvassa 23 esitellään pituuslinjan anturoiden syötteet ja näkymä Teklassa.



Kuva 23. *Anturoiden syötteet ja näkymä Teklassa.*

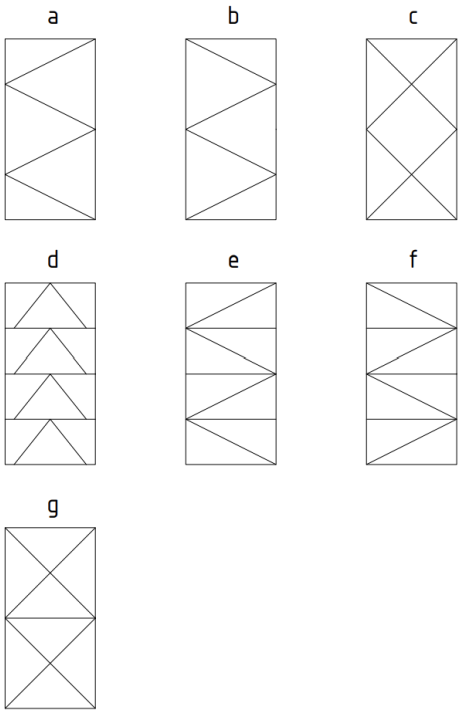
Samat vaiheet toteutetaan leveysakselin *(y-linjan)* runkopilarilinjoille. Selkeyden vuoksi määrätään leveysakselin pilareille eri *class*, eli väri, jotta ne voidaan erottaa pituusakselin pilareista. Runkolinjojen valmis tuotos on esitelty kuvassa 24.



Kuva 24*. Runkopilarit ja -anturat valmiit Tekla-näkymässä.*

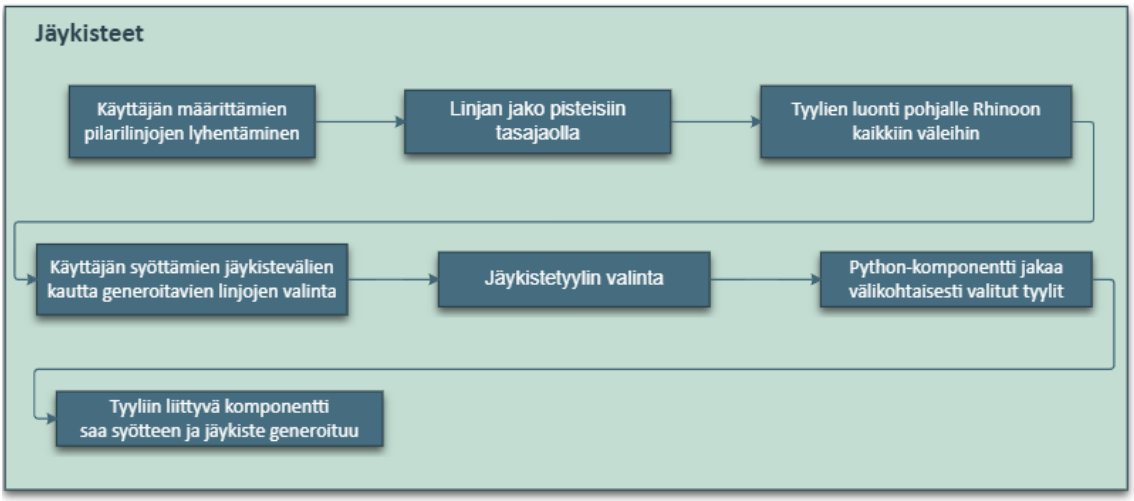
3.3.2 Jäykisteet ja runkopalkit

Tässä työssä jäykistys rajataan ristikkojäykisteisiin. Käyttäjän on mahdollista valita jäykistysjärjestelmä seitsemästä yleisimmin käytetystä ristikkoratkaisusta ja määrittää mihin väleihin ne tulee. Vaihtoehdot on esitelty kuvassa 25.



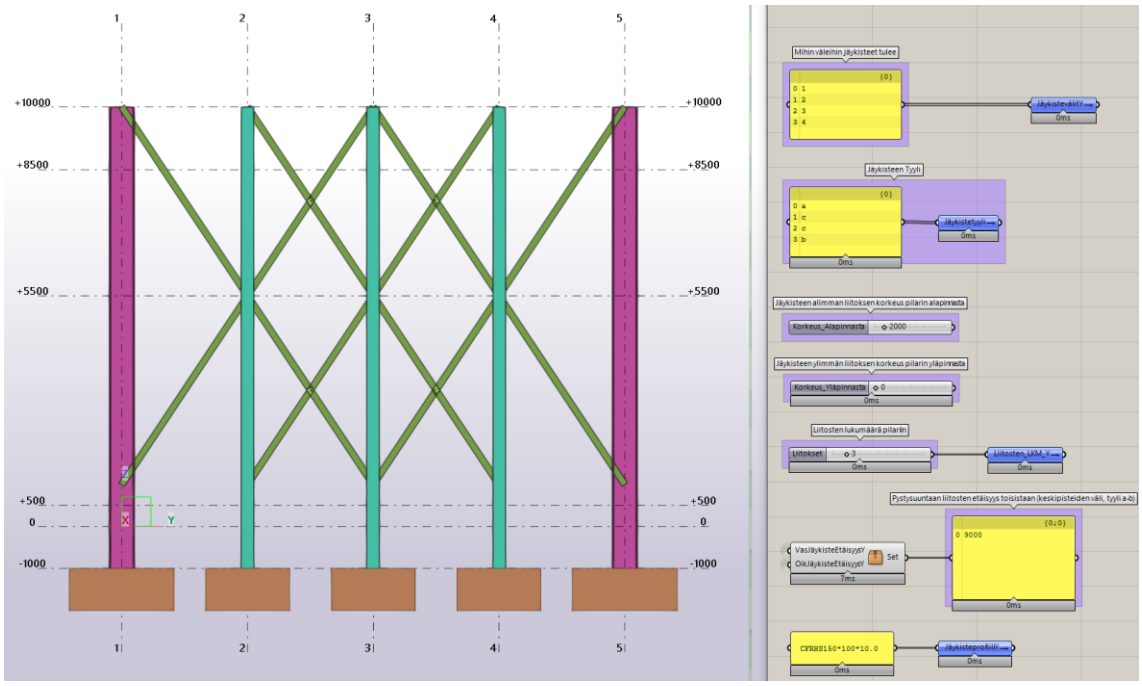
Kuva 25. *Yleisimpiä ristikkojäykisteratkaisuja. (Elementtisuunnittelu 2023, muokattu)*

Kaikki tyylit luodaan pohjalle valmiiksi, joka mahdollistaa nopeammin eri tyylien läpikäymisen pelkästään kirjainta vaihtamalla. Algoritmin prosessikaavio jäykisteille on esitelty kuvassa 26.



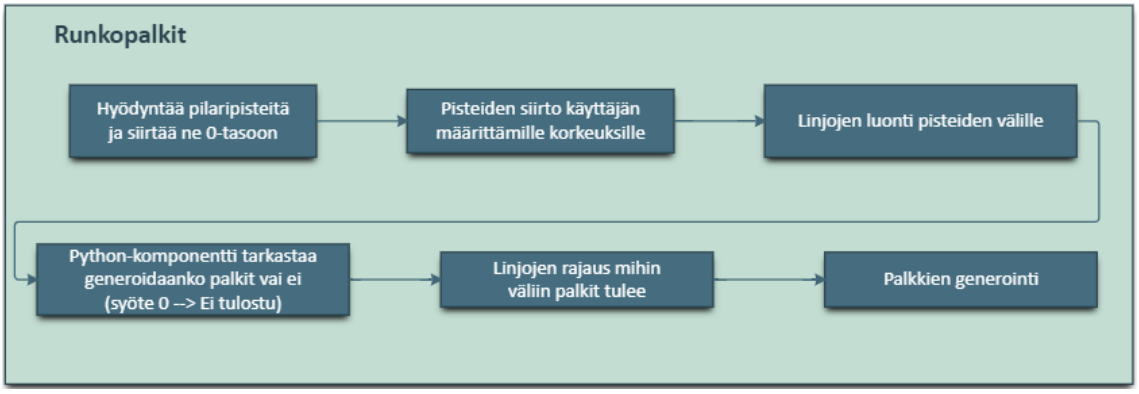
Kuva 26. *Jäykisteiden generoinnin prosessikaavio.*

Käyttäjä päättää ensin mihin väleihin ja millä tyylillä ristikot tullaan generoimaan. Ristikkojäykisteiden variaatiot on eritelty kirjaimilla toisistaan, kuten kuvassa 25 nähdään. Kirjaimien merkitsemisen jälkeen syöte kulkee *Python* -komponentin läpi, johon kirjoitettu koodi yhdistää listojen indeksejä hyödyntämällä käyttäjän valitseman tyylin haluamaansa väliin. Koodi määrittelee myös, että jos kaikkiin väleihin tulee samanlainen tyyli, niin yhden kirjaimen merkintä riittää. Jos tyylejä tulee kuitenkin erilaisia, niin jokainen väli pitää merkata erikseen. Näiden jälkeen käyttäjän on mahdollistaa siirtää jäykisteiden aloitus- ja lopetuspisteen korkeutta, jossa hyödynnetään aiemmin luotuja runkopilarilinjoja. Aloitus- ja lopetuspisteen välillä oleva linja on jaettu osapisteisiin, joiden välille ristikot tullaan luomaan. Pisteiden lukumäärää voidaan tihentää halutun lopputuloksen saamiseksi. Case-kohteen ristikkojäykisteiden syötteet ja Tekla-näkymä on esitelty kuvassa 27.



Kuva 27. *Ristikkojäykisteiden syöte ja Tekla-näkymä.*

Jäykisteiden generoinnin jälkeen seuraa runkopalkkien määritys, jonka prosessikaavio esitellään kuvassa 28. Tässä hyödynnetään käyttäjän aiemmin valittuja runkopilareiden pisteitä, joiden välille palkit tullaan luomaan käyttäjän määrittämin välein ja korkeuksin.

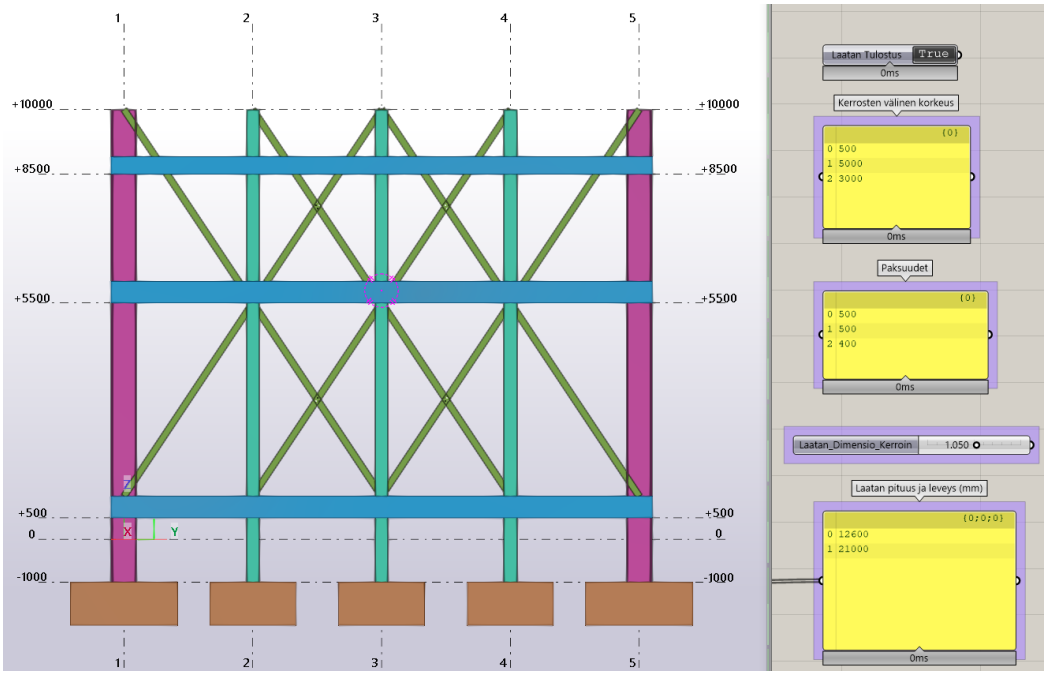


Kuva 28. *Runkopalkkien generoinnin prosessikaavio.*

Tämä vaihe ei ole välttämätön jatkon kannalta. Eli jos palkkeja ei tule, käyttäjä merkkaa haluttuihin väleihin ”*0*”, josta algoritmi tunnistaa, ettei generoi mitään. Case-kohteessa ei runkopalkkeja ole, joten tämä vaihe voidaan ohittaa.

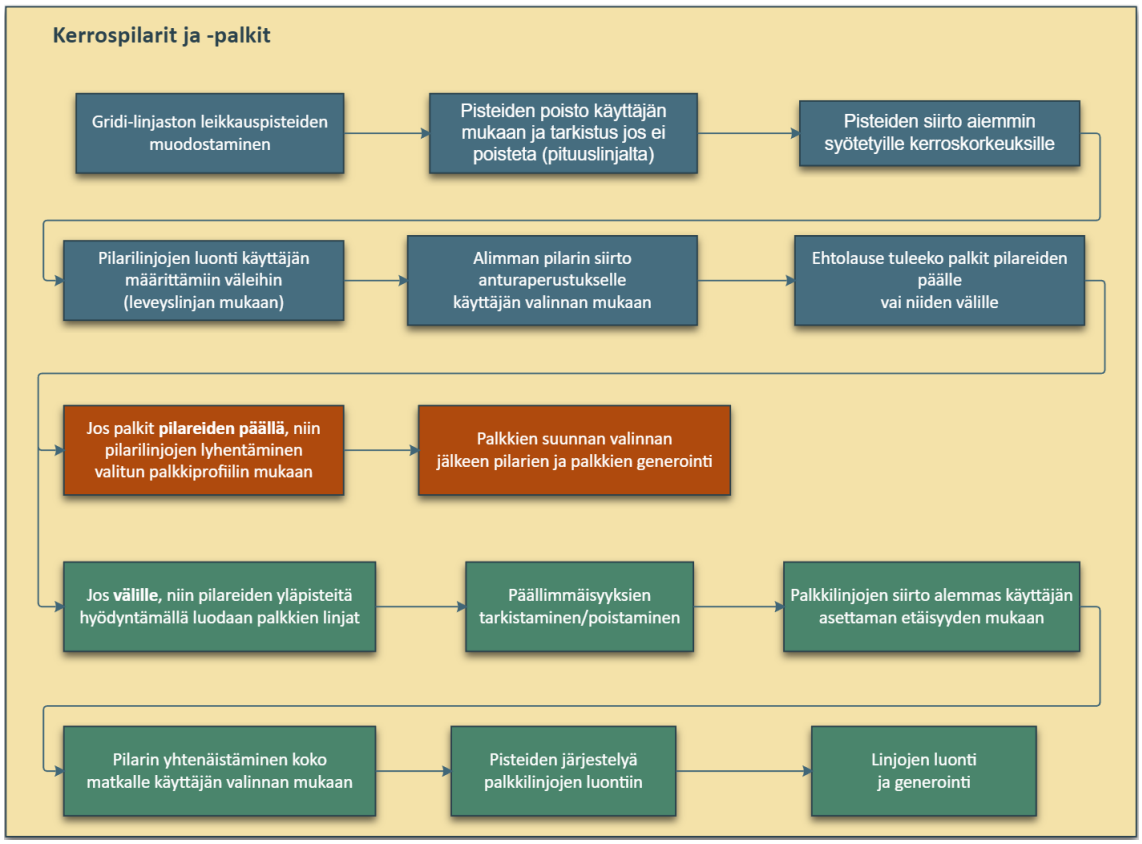
3.3.3 Kerrosten luonti ja kerrostuennat

Gridi-linjoista muodostettua rakennuksen ulkolinjaa hyödynnetään laattojen alustavan rungon luomisessa. Runko on siis luotu jo pohjalle, jota käyttäjä pystyy monistaa asettamilleen kerrosväleille. Tämän jälkeen laattojen paksuudet määritetään, jonka jälkeen on mahdollista säätää pituutta ja leveyttä. Case-kohteen kerroslaattojen syötteet ja Tekla-näkymä esitellään kuvassa 29.



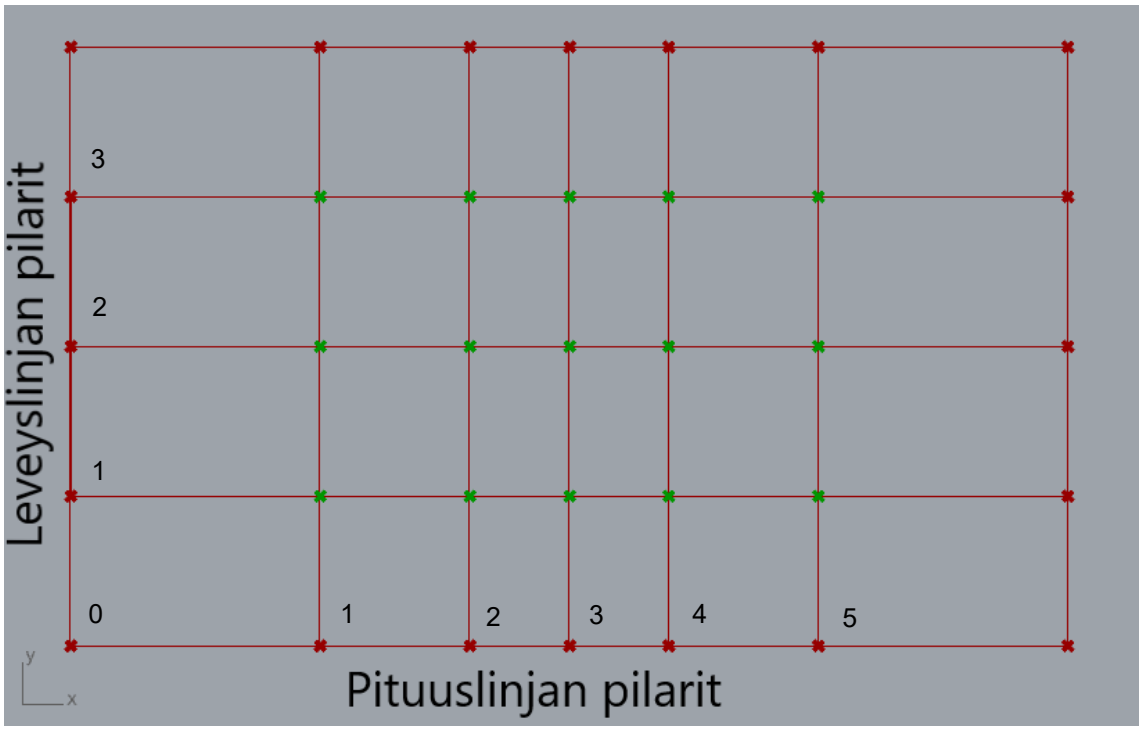
Kuva 29*. Kerroslaattojen syötteet ja Tekla-näkymä.*

Algoritmin haastavin osio oli kerrospilareiden ja -palkkien luominen. Haasteita ilmeni niiden monimuotoisuuden myötä ja muokkausmahdollisuuksien tuottamisessa. Algoritmin prosessikaavio kerrostuennoille esitellään kuvassa 30.



Kuva 30. *Kerrospilarien ja -palkkien generoinnin prosessikaavio.*

Algoritmin rajoitteena on, että generoitavien kerrostuentojen pitää olla runkopilarien kanssa samassa linjassa. Rajoitteen syytä havainnollistetaan kuvassa 31, sillä lähtöpisteet muodostetaan pilarilinjojen leikkauskohdista. Kuvassa ulkolinjalla olevat punaiset rastit ovat käyttäjän määrittämät sijainnit runkopilareille ja vihreät rastit ovat leikkauspisteet, joita käytetään kerrostuentojen luomisessa.



Kuva 31. *Rhino -näkymä ylhäältä kerrostuentojen lähtöpisteistä. Punaisella merkityt pisteet on generoidut runkopilarit ja vihreällä merkityt tulevat kerrostuennat.*

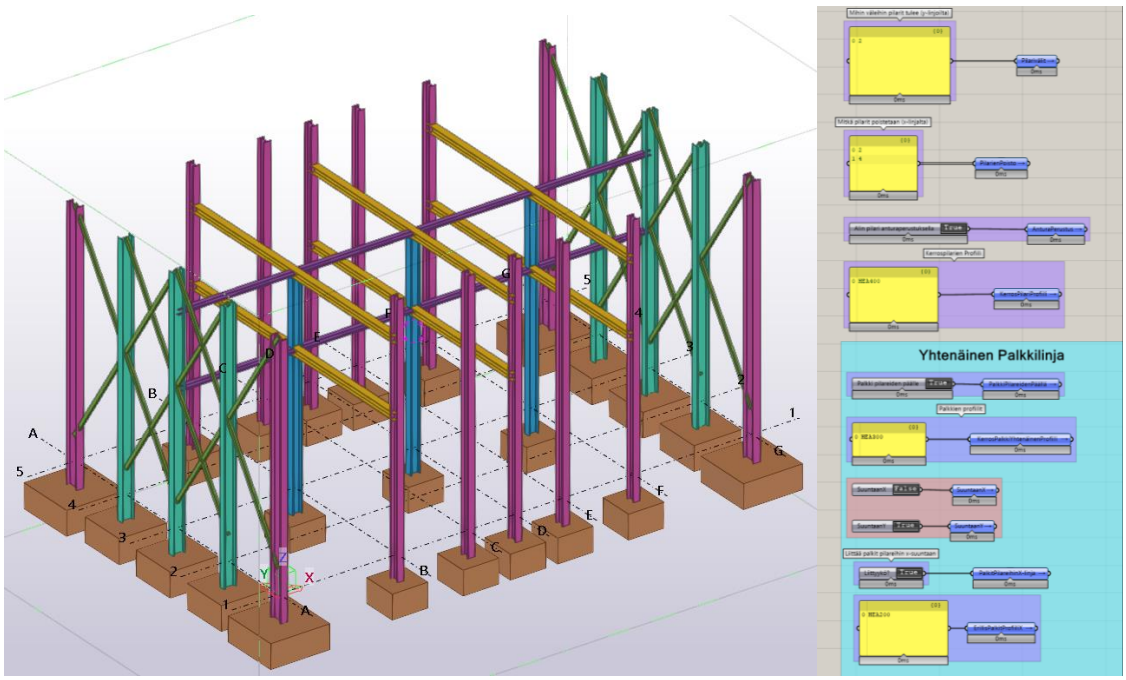
Lähtöpisteillä käyttäjä pystyy muokata tuentoihin liittyviä tekijöitä, esimerkiksi mihin väleihin niitä tulee. Ensimmäisenä vaiheena käyttäjä valitseekin mihin väleihin kerrostuennat luodaan leveyslinjan (*y-linjan*) pilareista katsottuna. Tämän jälkeen hän pystyy poistaa kerrospilareita pituuslinjalta (*x-linjalta*) katsottuna oman syötteen mukaan.

Kuvassa 31 vasemman alanurkan pilari toimii lähtöpisteenä valinnoille, niin sanotusti indeksillä ”*nolla*”. Leveyslinjan generoinnin vaihtoehdot ovat siis nurkkapilarista katsottuna ensimmäisen, toisen ja kolmannen linjan välillä. Pituuslinjan poistovaihtoehdot ovat vastaavasti välillä 1–5. Valittujen sijaintien pisteet siirretään sitten kerroslaatoissa määritetyille korkeuksille ja pystysuuntaiset linjat luodaan kerroskohtaisesti.

Tulevien linjojen päätöksen jälkeen käyttäjä valitsee, tuleeko alimmat pilarit anturaperustuksella. Valinnan mukaan linjaa pidennetään samalle syvyystasolle muiden perustusten kanssa, tai jätetään alimman laatan korkeudelle. Anturoiden muokkausvaihtoehdot löytyvät myöhemmin valinnaisena kohtana.

Seuraavassa ehtolauseessa päätetään, tuleeko kerrospalkit pilareiden päälle, vai niiden välille. Jos palkit tulee pilareiden päälle, niin se on yhtenäinen linja koko matkalla. Pilareiden pituutta lyhennetään valitun palkkiprofiilin korkeuden mukaan siten, että palkki tulee pilarin ja laatan väliin ilman päällekkäisyyksiä.

Case-kohteen kerrostuentojen syötteet ja Tekla-näkymä esitellään kuvassa 32. Selkeyttämisen vuoksi laatat on piilotettu, jotta kerrostuennat erottuvat paremmin.

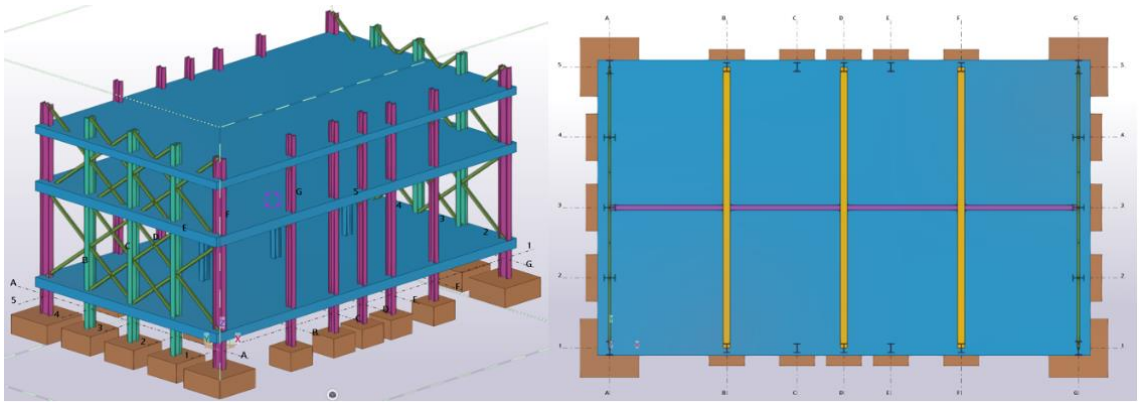


Kuva 32. *Kerrostuentojen syötteet ja Tekla-näkymä.*

Kuvassa 32 syötteiden valinnasta on jätetty anturan muokkaukseen liittyvä osio pois, mutta se löytyy kerrostuentojen jälkeen ja toiminnallisuus samanlainen kuin runkopilareiden anturoiden määrityksessä.

3.3.4 Tietojen keruu ja RSA-exporttaus

Jos käyttäjä huomaa tässä vaiheessa ristiriitaisuuksia mallissa, niin osia on mahdollista muokata järjestyksestä riippumatta. Täytyy kuitenkin muistaa, että runkopilareiden lukumäärää muuttaessa valitut välit täytyy mahdollisesti korjata. Valmis tietomalli nähdään kuvassa 33.



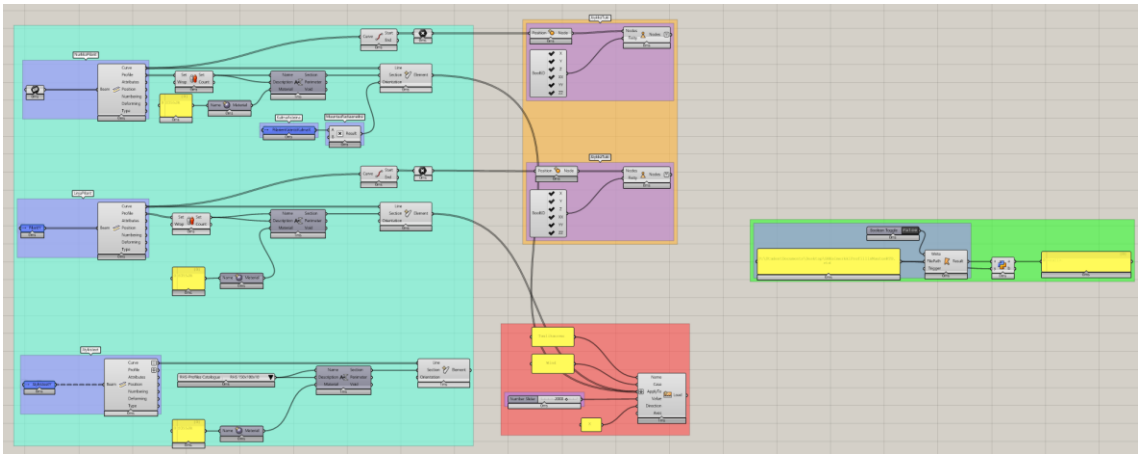
Kuva 33. *Vasemmalla case-kohteen valmis tietomalli. Oikealla puolella näkymä ylhäältä katsottuna, jossa ylin laatta piilotettuna kerrostuentojen näkemistä varten.*

Viimeisenä vaiheena käyttäjän on mahdollista luoda rakenneanalyysia varten oma tiedostonsa. Tässä vaiheessa ilmeni rajoitteita mahdollisten ohjelmistojen kanssa, sillä algoritmin kehittämiseen käytetyllä tietokoneella oli rajattu tiedostojen asennusoikeus. Työssä olisi voinut tutkia suoraan *Grasshopper – Tekla Structural Designer Live Link* -lisäosan käytettävyyttä, jossa malli olisikin luotu Teklan sijaan suoraan saman yhtiön kehittämään rakenneanalyysiohjelmistoon. Tämä ei kuitenkaan ollut mahdollista rajoitusten myötä. Useampia Grasshopperiin liittyviä FEM-laskentaan tarkoitettujen lisäosien toimivuutta selvitettiin ja yritettiin huonolla tuloksella asennusrajoitteiden seurauksena.

Käytettäväksi ohjelmistoksi päädyttiin Paul Jeffriesin kehittämään Salamander3-lisäosaan, joka mahdollistaa tiedoston muokkaamisen Robot Structural Analysis -ohjelmiston (RSA) hyväksymään muotoon. Tämäkään lisäosa ei ole täydellinen, sillä mallin generointi RSA-ohjelmaan tai sen päivittäminen on niin kuormittava, että RSA kaatuu väistämättä. Tämän seurauksena RSA-ohjelmisto joudutaan sulkemaan aina kun Grasshopperissa tallennetaan uusi FEM-laskentamalli. Ongelmaa helpotettiin lisäämällä *Python* -komponentti, joka mahdollistaa tiedoston avaamisen heti tallentamisen jälkeen.

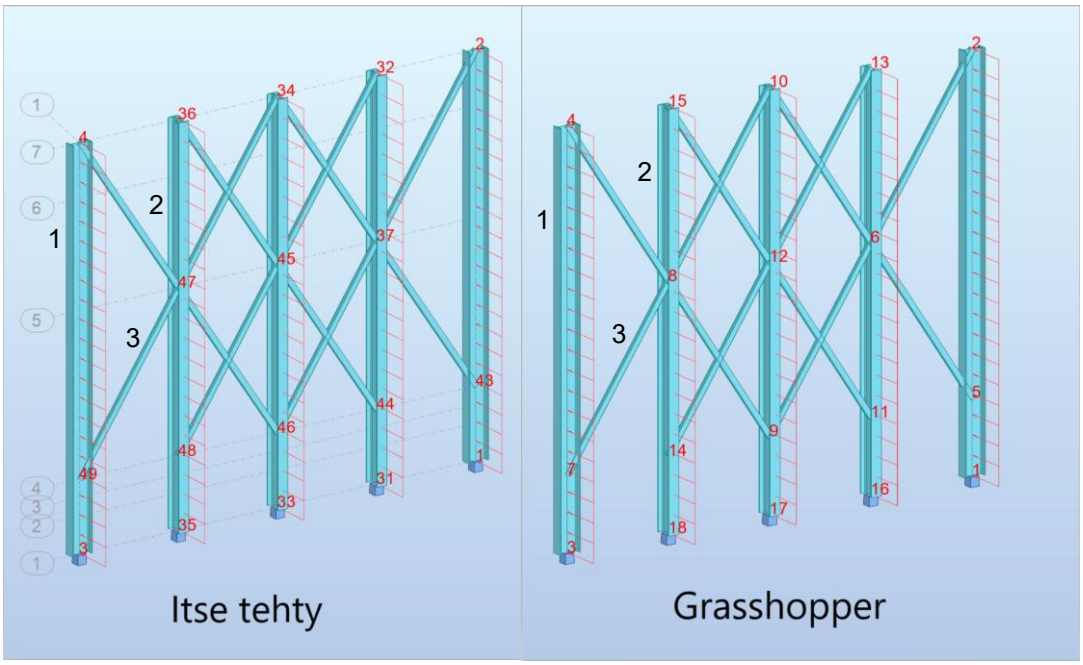
Analyysimallin luomisessa ilmeni haasteita valintojen laajuuden kanssa, esimerkiksi liitoksen tyypin määrityksessä. Jos jokaiselle osalle annettaisiin erikseen mahdollisuus vaikuttaa liitäntään tai vapausasteiden määrään, niin osuudesta tulisi erittäinkin sekava. Toiminnallisuuden esittämisessä rajataan liitokset kuitenkin siten, että anturaliitokset ovat jäykkiä ja kaikki muut liitännät ovat *node* solmupisteillä toteutettuja. Mahdollisien kuormituksien lisääminen on myös haastavaa rakenneosakohtaisesti eri kuormitusyhdistelmien laajuuden kanssa. Jos tietyn tilanteen kuormitukset tuntee jo entuudestaan, niin ne olisi kuitenkin mahdollista lisätä Salamander3 -ohjelmiston avulla.

Analyysimallin tuloksien todellisuutta tutkitaan rajaamalla tarkasteluun varastohallin vasemman sivun runkopilarit ja jäykisteet. Tarkastelua varten luodaan kaksi eri mallia, toinen Grasshopperin kautta ja toinen manuaalisesti. Molemmissa malleissa pilareihin vaikuttaa 2 kN/m suuruinen vaakasuuntainen viivakuorma, jonka aiheuttamia rasituksia vertaillaan mallien välillä. Grasshopperissa tehdyn analyysimallin luonti nähdään kuvassa 34. Turkoosilla pohjalla vasemmalla olevassa osiossa määritetään lineaariset elementit, eli pilarit ja jäykisteet. Siinä määritetään myös kyseisten elementtien attribuutit, eli profiilit ja käännösasteet. Keskellä oranssilla pohjalla olevassa osiossa luodaan pilarien tuenta anturaan, joka tässä tapauksessa on jäykästi kiinnitetty ja sen alla punaisessa osiossa määritetään pilareihin kohdistuva viivakuorma. Vihreällä pohjalla olevassa osiossa määritetään tallennuspolku, eli mihin tiedosto tallennetaan ja missä muodossa.



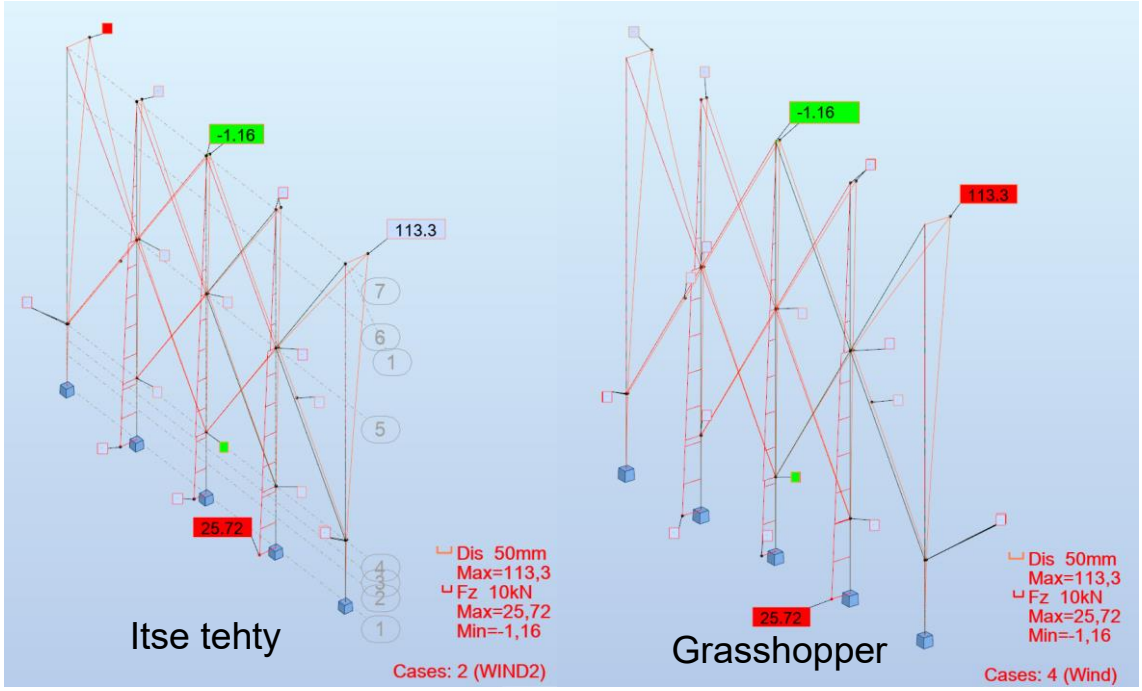
Kuva 34. *Analyysimallin luonti Salamander3 lisäosan komponenteilla.*

Algoritmissa käyttäjän määritettäväksi tulee siis tuennat, kuormituksen suuruus ja tallennetun tiedoston sijainti. Muissa osioissa hyödynnetään Tekla-malliin aiemmin määritettyjä tietoja. Kuvassa 35 esitellään analyysimallit, jossa vasemmanpuoleinen on manuaalisesti luotu ja oikeanpuoleinen Grasshopperissa.



Kuva 35. *RSA-mallien vertailu.*

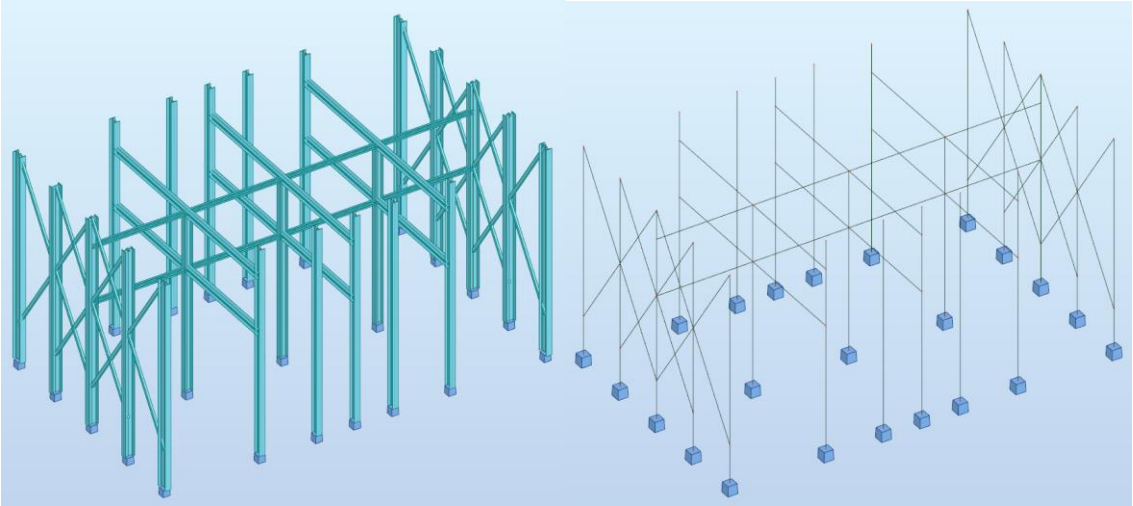
Jokaiseen pilariin vaikuttaa 2 kN/m suuruinen viivakuorma vaakasuuntaan. Laskenta suoritetaan RSA-ohjelmistossa ja tutkitaan leikkausvoiman F_z maksimi- ja minimiarvot, sekä sivusuuntaisen siirtymän maksimi. Molempien tilanteiden kuvaajat esitellään kuvassa 36.



Kuva 36. *Analyysimallin laskennan tuottamat kuvaajat leikkausvoimalle* F_z *ja sivusuuntaiselle siirtymälle.*

Huomataan, että kuvaajat ovat samat molemmissa malleissa, sekä maksimi- ja minimiarvot täsmäävät. Grasshopperilla voidaan siis luoda rakenneanalyysimalli, joka saavuttaa samanlaiset tulokset verrattuna perinteisellä menetelmällä tuotettuun.

Tuloksia tarkasteltiin vain kuvan 36 tapauksessa, mutta todellisuudessa Grasshopper luo taustalla RSA-tiedostoa valmiiksi käyttäjän tallentamista varten. Case-kohteen tapauksessa taustalle siis on luotu koko varastohallin runkorakenteet valmiiksi, jotka ovat käyttäjän syötteiden mukaisia. Kuvassa 37 esitellään case-kohteen RSA-malli, jossa kaikki pilariliitokset anturoihin ovat jäykkiä.



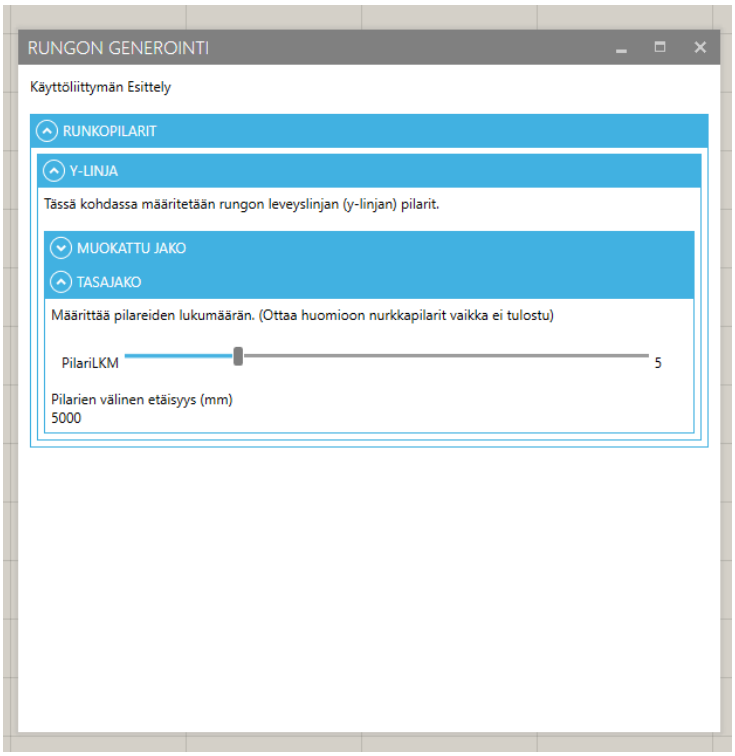
Kuva 37. *Case-kohteen RSA-malli. Vasemmalla profiilit näkyvissä ja oikealla lineaariset elementit.*

Rakenneosien liitoksien lisäys olisi myös mahdollista toteuttaa Grasshopperissa, mutta sen esittely jätetään tässä työssä pois.

Rakenteiden profiilien siirtämisessä ilmeni pieni ongelma RSA-mallissa. Profiili määrittyy oikein, mutta jos tiedostossa on useampi samanlainen poikkileikkaus, niin sen nimen perään tulee numerointi. Kuvassa 37 ensimmäisen nurkkapilarin profiili olisi HEA600, toisen nurkkapilarin HEA600 2, kolmannen HEA600 3 ja viimeisen HEA600 4. Tuloksien analysoinnin kannalta se ei kuitenkaan ole ongelma, sillä profiili määrittyy oikein jälkinumeroinnista huolimatta.

3.4 Jatkokehitykset ja huomioitavaa

Algoritmin syötteet toteutettiin tässä työssä pelkästään Grasshopperin *canvas*-näkymässä. Osioiden lukumäärän ja niissä olevien muokkausmahdollisuuksien takia käyttöliittymästä tulee nopeasti sekava ja laaja, eikä käyttäjällä ole selkeää käsitystä vaiheiden sijainnista. Ensimmäisenä kehityskohteena olisikin siistimmän ja selkeämmän käyttöliittymän toteuttaminen Andrew Heumannin kehittämällä *Human UI* -lisäosalla. Tämä mahdollistaisi huomattavasti paremman käsityksen siitä, onko muokkausmahdollisuuksia liian paljon vai olisiko järkevämpää tehdä malli perinteisesti. Kuvassa 38 esitellään leveyslinjan runkopilarien syöttäminen *Human UI*:n käyttöliittymällä.



Kuva 38. *Human UI -lisäosalla luotu käyttöliittymä.*

Käyttöliittymässä pystyisi siis erilaisilla toiminnoilla tekemään huomattavasti selkeämmän ja tiiviimmän kokonaisuuden malliin syötettävistä tiedoista. Kuvan 38 tapauksessa tämä toteutettiin alasvetovalikoilla, Tässä työssä käyttöliittymän luominen ei kuitenkaan ollut tärkeässä roolissa, joten se jätettiin pois.

Toisena kehityskohteena olisi voinut tutkia kuinka mallin olisi voinut viedä analyysiohjelmiston läpi ja saada tulokset suoraan Grasshopperiin takaisin. Salamander3 -lisäosan avulla pystyttiin kuitenkin toteamaan algoritmisesti luodun mallin tuloksien täsmäävän perinteisesti tehdyn mallin kanssa. Tuloksien avulla Grasshopperissa olisi voinut tutkia rakenteiden mitoittamista ja optimointia. Nämä tekijät olisivat kuitenkin kasvattaneet työn määrää huomattavasti.

Mitoittamisessa ja optimoinnissa huomioitavaksi tekijäksi olisi tullut algoritmin toteuttamiseen kuluvan ajan suuruus. Etenkin jos siinä hyödyntää Grasshopperissa valmiiksi olevaa Galapagos-työkalua. Galapagoksen toimintaperiaate perustuu iteraatiokertoihin, jossa se muokkaa syötettyjä parametreja, suorittaa algoritmin ja kerää lopputuloksen. Se siis käy erilaisia vaihtoehtoja läpi parametrien antamissa rajoissa, kunnes alkaa lähestymään tavoiteltua lopputulosta. Käytävien rakenneosien määrä vaikuttaa erittäin paljon tämän prosessin kestoaikaan, joten oikeiden parametrien rajaaminen on tärkeää.

Tämän työn Tekla -mallin tarkkuudessa ilmenee helposti päällekkäisyyksiä muiden osien kanssa, esimerkiksi jäykisteiden ja pilarien välillä. Niiden *leikkaaminen* haluttuun tasoon algoritmin avulla on mahdollista, mutta ei koettu tärkeäksi tekijäksi esittelyä varten. Koettiin myös, että tietomallin täydellinen detaljointi ei tässä suunnitteluvaiheessa ole tärkeimmässä roolissa.

4. YHTEENVETO

Työn tavoitteena oli luoda algoritmi, joka mahdollistaisi teräsrunkoisen pilari-palkkirakennuksen generoimisen käyttäjän syöttämien parametrien kautta. Lisäksi tutkittiin algoritmin tuottamaa laskentamallia ja sen tarkkuutta perinteiseen mallintamiseen verrattuna. Tavoitteissa onnistuttiin, eikä suurempia rajoituksia tullut vastaan. Rakennemallia muunnettiin eri tilanteille, onnistunein tuloksin.

Automaattisen generoinnin ongelmakohtana on kuitenkin muokkausmahdollisuuksien laajuus. Kuvassa 15 esiteltiin työn aikana toteutettu algoritmi, mutta siinä ei olla jokaiseen kohtaan lisätty rakenteiden attribuuttien täydellistä määrittämistä. Se lisättiin tarkemmin pelkästään runkopilareille, attribuuttien lisäämisen esittelyä varten. Toisena asiana algoritmista jätettiin pois käyttäjän muokkausmahdollisuudet FEM-laskentamalliin liittyen. Pelkästään näistä kahdesta tekijästä parametrien määrä kasvaisi erittäin pitkäksi kokonaisuudeksi, jolloin käyttöliittymän teko olisi välttämätöntä selkeyden ja nopeuden takia. Vaikka toteutus olisikin mahdollinen, niin käyttöliittymä pitäisi pystyä rakentamaan siten, että se pysyisi selkeänä ja käyttäjäystävällisenä.

Laskentamallin tuottamiseen algoritmi kykenee yhtä hyvin, kuin Tekla-mallinkin luomiseen. Ongelmana ilmenee kuitenkin sama tilanne kuin edellä mainittu, eli muokkausmahdollisuuksien laajuus. Täydellisen laskentamallin tuottaminen olisi haastavaa ilman suurta määrää parametreja rakenneosakohtaisesti, ellei sitä rajata järkevään valmiusasteeseen. Etenkin kuormien ja liitoksien vapausasteiden määritys olisi huomattavasti nopeampi, sekä varmempi tehdä RSA -ohjelmistossa.

Algoritmisessa mallintamisessa onkin tärkeää tunnistaa missä vaiheessa sitä on hyödyllistä käyttää ja miten tarvittavat lähtötiedot rajataan. Case-kohteen mallintaminen perinteisesti olisi varmasti ollut nopeampaa verrattuna algoritmin luomiseen kulutettuun aikaan. Algoritmin muokkausmahdollisuuksien myötä se ei kuitenkaan rajaudu pelkästään tähän tilanteeseen, vaan sillä pystyy toteuttamaan useammankin eri rakenneratkaisun nopeasti.

Tärkeänä tekijänä voisi nostaa visuaalisen ohjelmoinnin tuottamat hyödyt algoritmin luomiseen liittyen. *Visuaalisella skriptaamisella* näkee heti komponenttien käyttäytymisen Rhinon käyttöliittymässä, jolloin käyttäjä saa suoraan palautteen algoritmin toiminnasta. Aikaisempi ohjelmointikokemus ei myöskään ole välttämätöntä ohjelmiston käytön kannalta, joten se ei ole rajoite algoritmien toteuttamisessa. Tästä hyvänä esimerkkinä on kirjoittaja itse, joka onnistui rakentamaan työssä käytetyn algoritmin ilman suurempaa

taustaa ohjelmistoihin liittyen. Grasshopperiin liittyvien lisäosien myötä, myös käyttökohteiden laajuus kasvaa. Suurempia algoritmeja rakentaessa ohjelmointikokemuksen omaaminen olisi kuitenkin hyödyksi, jotta pystyy tekemään tarpeen mukaan omiakin komponentteja.

Jatkotutkimuksia aiheeseen liittyen voisi olla kattorakenteen generointi, sekä käyttöliittymän luominen. Käyttöliittymän valmistuttua voisi haastatella suunnittelutoimistojen mielipidettä algoritmiin liittyen, sekä kehittää sitä pidemmälle. Toinen tutkimusaihe liittyy Salamander3-lisäosan toimintatapaan luoda RSA -laskentamalli. Se käyttää mallin luomisessa API (*Application Programming Interface*) ja COM (*Component Object Model*) toimintoja, jotka siis mahdollistavat koodaamalla ohjelman käyttämisen ulkoisella sovelluksella. Toisin sanoen tällä hetkellä komponentti pystyy luomaan mallin, mutta ei toteuta laskentaa sillä. Aiheeseen perehtymällä voisi olla mahdollista luoda komponentti, joka suorittaa laskennan RSA-ohjelmistolla, tuo tulokset Grasshopperiin ja käyttää niitä esimerkiksi rakenteen mitoittamiseen tai optimointiin. Tämä vaatisi kuitenkin jo suurempaa ymmärrystä ohjelmointiin liittyen, sekä käytettävään rakenneanalyysiohjelmistoon.

LÄHTEET

Arokoski, J. (2020). Algoritmisen suunnittelun opetuksen kehittäminen. Diplomityö. Tampereen yliopisto. Tampere. 89 s.

Autodesk (2015). About Creating 3D Meshes. Saatavissa (viitattu 18.2.2023): https://knowledge.autodesk.com/support/autocad/learn-explore/caas/CloudHelp/cloudhelp/2016/ENU/AutoCAD-Core/files/GUID-A6232957-5039-4AB7-8B1D-8FD0AD98F77B-htm.html

Boberg, J. Tietojenkäsittelytieteiden perusopinnot: Esittely ja itseohjaavan testin materiaali. Turun yliopisto. Saatavissa (viitattu 13.3.2023) https://ville.cs.utu.fi/avoin/mat/TestiMat.pdf

Elementtisuunnittelu (2023). Jäykistysjärjestelmät. Saatavissa (viitattu 24.3.2023): https://www.elementtisuunnittelu.fi/rakennejarjestelmat/rakennuksen-jaykistys/jaykistysjarjestelmat

Food4Rhino, Grasshopperin lisäosille tarkoitettu sivu. Saatavissa (viitattu 21.2.2023): https://www.food4rhino.com/en

Grasshopper3d viralliset sivut (2023). Saatavissa (viitattu 18.2.2023): https://www.grasshopper3d.com/

Karjalainen, P. (2018). Kantavien rakenteiden algoritmiavusteisen rakennesuunnitteluprosessin kehittäminen. Diplomityö. Tampereen teknillinen yliopisto. Tampere. 106 s.

Karjalainen, P. (2021). Paras runkoratkaisu parkkitaloon hahmottui algoritmien avulla. A-insinöörit. Saatavissa (viitattu 30.1.2023): https://www.ains.fi/asiantuntija-artikkelit/paras-runkoratkaisu-hahmottuu-algoritmien-avulla

Lalla, A. (2018). Kantavien rakenteiden parametrinen suunnittelu ja mallintaminen. Diplomityö. Tampereen teknillinen yliopisto. Tampere. 113 s.

Paukkeri, S. (2018). Algoritmiavusteinen rakennesuunnittelu. Kandidaatintyö. Tampereen teknillinen yliopisto. Tampere. 35 s.

Piermarini, E., Nuttall, H., May, R. & Janssens, V. (2016). City of Dreams, Macau – making the vision viable, The Structural Engineer, Vol. 94(3), pp. 56–67.

Rhinon viralliset sivut. Saatavissa (viitattu 16.2.2023): https://www.rhino3d.com/

Storrier, A. (2021). How CAD and BIM Changed the Construction Industry. Hexagon Saatavissa (viitattu 13.3.2023): https://resources.hexagonppm.com/articles-blogs/how-cad-and-bim-changed-the-construction-industry

Tanska, T. & Österlund, T. (2014). Algoritmit puurakenteissa: menetelmät, mahdollisuudet ja tuotanto, B 32, 1. painos. DigiWoodLab, Oulun yliopisto, Arkkitehtuurin tiedekunta.

Tedeschi, A. (2011). Parametric Architecture With Grasshopper. Tecnostampa, Italia. 208 s.

Teittinen, T. (2022). 01 - Rakennusten Tietomallit. Mittaustekniikan ja tietomallintamisen perusteet. Yliopisto-opettajan luento 1.4.2022 Tampereen yliopistossa.