LAPIN AMK
Lapland University of Applied Sciences

LISÄOSAN LUOMINEN BLENDERIIN

Proseduraalinen 3D-generointityökalu

Li Onni

Opinnäytetyö

Tieto- ja viestintätekniikka
Insinööri (AMK)

2021

LAPIN AMK
Lapland University of Applied Sciences

Opinnäytetyön tiivistelmä

Tieto- ja viestintätekniika
Insinööri (AMK)

Tekijä	Onni Li	Vuosi	2021
Ohjaaja	Maisa Mielikäinen, Jarkko Piippo		
Toimeksiantaja	LapinAMK, Frostbit - ohjelmistotekniikan laboratorio		
Työn nimi	Lisäosan luominen Blenderiin Proseduraalinen 3D-generointityökalu		
Sivumäärä	62		

Opinnäytetyössä luodaan lisäosa 3D-ohjelmaan Blender. Lisäosan ideana on luoda proseduraalisesti 3D-kenttiä modulaarisista 3D-malleista. Opinnäytetyössä tutkitaan miten Python-ohjelmoinnilla voi ohjata Blenderin toimintoja. Aihetta lähestytään rakentamalla omaan Blender-instanssiin uusi lisäosa.

Idea syntyi kirjoittajan omista kokemuksista ja kiinnostuksesta 3D-teollisuutta kohtaan. Lisäosan lopullinen tarkoitus on kannustaa Blender-ohjelman oppimista tarjoamalla valmiita rakennuspalikoita, jotka nopeuttavat projektien aloittamista. Tällä tavalla uusi käyttäjä pääsee helposti kokeilemaan ohjelman muita ominaisuuksia, jotka eivät ole mahdollisia ilman jonkinlaista olemassa olevaa 3D-kenttää.

Blender on open-source ohjelma, joka julkaistaan GNU General Public License -lisenssillä. Se on siis ilmainen ohjelma, jonka ominaisuuksia muokataan Python-ohjelmointikielellä. Opinnäytetyö voitiin siksi toteuttaa täysin Blenderissä aina 3D-mallien luomisesta Python-ohjelmointiin. Opinnäytetyöstä syntyneellä prototyypillä pystyy luomaan proseduraalisesti käytäväkomplekseja, mikä vastaa tyylillään dungeon-generaattoreita.

Avainsanat 3D-mallinnus, ohjelmointi, python

LAPIN AMK
Lapland University of Applied Sciences

Abstract of Thesis

Bachelor of engineering
Degree Programme in Information
and Communication Technology

Author	Onni Li	Year	2021
Supervisor	Maisa Mielikäinen, Jarkko Piippo		
Commissioned by	Lapland UAS, Frostbit - Software Lab		
Subject of thesis	Creating a Blender add-on		
Number of pages	62		

The aim of this thesis was to examine how an individual user can create their own add-on for the 3D program Blender. A prototype add-on will be developed in this study. This prototype add-on functions as a procedurally generative 3D-modeling tool that creates corridor complexes similar to a dungeon generator. The goal will be achieved by examining how programming with Python works with Blender's own API framework by locking the entire workflow to Blender's ecosystem. This is a viable method due to the Blender's open-source nature and its GNU General Public License release form. This means that everything that is modified and created using Blender is the author's or the artist's sole property. The employer of this thesis is Frostbit Software Lab, which is an R&D laboratory that develops software and platforms using game engines.

In the first part of the study it was examined how to create modular 3D-objects using Blender's 3D-manipulation tools for the generation algorithm. In the second part it was examined how the procedural generation algorithm functions as the handler of the modular 3D-objects and how it incorporates them in its spawning logic. The add-on creation process was carefully examined and integral observations were made during the study.

This thesis provides a basic guideline in both 3D-modeling workflow and add-on creation. The result is a working prototype add-on called O-Gen that is highly customizable and adaptable with great potential for further development. O-Gen's users can generate complex scenes with a click of a button and modify the generated results with simple tweaks.

Key words 3D-modeling, programming, python

SISÄLLYS

1 JOHDANTO 8

2 3D-TYÖNKULKU 10

2.1 3D-mallintaminen 10

2.1.1 Mallinnustyökalut 11

2.1.2 3D-mallin pinnan normaalit 12

2.1.3 UV-mappaus 13

2.2 Node editor 15

2.3 PBR-teksturointi 16

2.4 Valaistus 23

2.5 Jälkiprosessointi 24

3 PROSEDURAALINEN 3D-GENEROINTI 25

3.1 Muuntajat 25

3.2 Noise-algoritmit 26

3.3 Proseduraalinen mallintaminen muuntajilla 28

3.4 Proseduraalinen mallintaminen geometrianodeilla 28

3.5 Proseduraalinen teksturointi 30

4 LISÄOSAN LUOMINEN BLENDERIIN 31

4.1 Ohjelmointirajapinta 32

4.2 Graafinen käyttöliittymä 33

4.3 Generointilogiikka 34

4.4 Modulaariset 3D-mallit 35

4.5 Kuutioverkoston Generointialgoritmi 37

4.5.1 Generate_cells-funktio 38

4.5.2 Populate_branch-funktio 39

4.5.3 Neighbour_check-prosessi 43

4.5.4 Module_conditions-prosessi 46

4.6 Lopputulos 49

4.7 Laajennuksen käyttöönotto 51

5 POHDINTA 53

5.1 O-Genin kehittäminen 53

5.2 Huomioitavia asioita 54

5.3 Jatkokehitysideoita 55
5.4 Lisäosien potentiaali 57
LÄHTEET 60

KÄYTETYT LYHENTEET JA TERMIT

3D-scene	3D-kenttä, kolmiulotteinen tila, jossa 3D-mallit sijaitsevat
CGI	Computer Generated Imagery, tietokoneella tuotettua kuvamateriaalia
Compositor	Blenderin moduuli, jossa muokataan node-pohjaisessa muokkausikkunassa renderoitua kuvaa
HDRi	High Dynamic Range Imaging, 360-asteinen kuva, jota käytetään 3D-kentän valaistuksessa (Herland 2020)
Mesh	Tahkoverkosto (Joensuu 2016), koostuu edgeistä, faceista sekä vertexeistä eli särmistä, tahkoista sekä kärkipisteistä.
Node	Solmu, noodi, yksittäinen elementti visuaalisessa asetuksien muokkausikkunassa eli node editorissa
Node editor	Nodejen muokkausikkuna, visuaalinen asetuksien muokkausikkuna
Path-tracing	Säteenseuranta, renderointitekniikka, jossa lasketaan kentän valaistus ampumalla säteitä 3D-kenttään (Carlson 2017)
PBR	Physically Based Render, teksturointitekniikka, jossa käytetään informaatiota monesta lähteestä yhden materiaalin tuottamiseen
Principled BSDF	Principled Bidirectional Scattering Distribution Function, yksi Blenderin käyttämistä shadereista
Rendering	Renderointi (Joensuu 2016), operaatio, jossa suoritetaan esimerkiksi säteenseurannan simulaatio
Rigging	Riggaaminen, riggaus, prosessi, jossa kiinnitetään 3D-objektiin luita, joiden avulla 3D-malli animoidaan
Shader	Varjostin, 3D-ohjelman renderointitekniikan perusrakenteellinen elementti, mikä kertoo tietokoneelle, miten materiaalia tulisi käsitellä tietokoneen prosessorissa tai näytönohjaimessa (Vivo & Lowe 2015b)
Shader editor	Muokkausikkuna, Blenderin moduuli, jossa muokataan tekstuureita ja shadereita node-pohjaisessa muokkausikkunassa

UV-mapping	UV-kartoitus, prosessi, jossa visualisoidaan 3D-elementin rakennetta 2D-tasokuvana, jossa U ja V vastaavat 2D-tason X ja Y-akseleita
VFX	Visual Effects, visuaalinen tehoste

1 JOHDANTO

Uudet käyttäjät kohtaavat monia vaikeuksia 3D-mallinnuksessa. Esimerkiksi ohjelman oppiminen voi olla haastavaa, sillä ensimmäinen ikkuna tuo kokemattomalle käyttäjälle loputtoman määrän erilaisia ominaisuuksia ja toimintoja. Internetistä löydettävät ohjeet keskittyvät yleensä klassiseen työnkulkuun, joka aloitetaan 3D-mallinnuksesta. Ohjelman perustoimintojen oppiminen voi viedä jopa useamman kuukauden. Blender tarjoaa verkkosivuillaan demokenttiä, jotka ovat renderointivalmiita. Tämä on erityisen hyödyllistä varsinkin 3D-artisteille, jotka ovat käyttäneet aikaisemmin muita 3D-ohjelmia ja haluavat kokeilla tai verrata Blenderin ominaisuuksia tai toimintoja esimerkiksi vastaaviin 3D-mallinnusohjelmiin, kuten Mayaan tai Cinema4D:hen.

Opinnäytetyössä kuvataan lisäosan luomista 3D-mallinnusohjelmaan Blender. Lisäosan tarkoituksena on nopeuttaa 3D-kentän luomista. Kentät luodaan proseduraalisesti generoimalla. Opinnäytetyön lopputuotteena syntyy prototyyppi, jolla luodaan proseduraalisesti generoituja käytäväkomplekseja.

Tutkimuksesta syntyneen prototyypin tarkoituksena on ensisijaisesti kannustaa uusia käyttäjiä kokeilemaan ohjelman eri ominaisuuksia ja toiminnallisuuksia. Opinnäytetyön toissijainen tarkoitus on luoda toimiva laajennus, jota muut käyttäjät pääsevät lataamaan itselleen ja käyttämään omassa Blender-ohjelmassaan. Opinnäytetyöstä syntynyt tuote on viralliselta nimitykseltään Blenderin lisäosa eli add-on, mutta toinen epävirallisempi nimitys on plugin eli laajennus tai työkalu. Opinnäytetyössä kaikilla näillä nimityksillä viitataan samaan asiaan. Opinnäytetyössä tutkitaan ensisijaisesti suoralla havainnoinnilla sekä case study -metodilla, mitä haasteita, ongelmia tai huomioitavaa yksittäiselle Blender-käyttäjälle tulee vastaan lisäosaa kehittäessä.

Blender-ohjelman suosio on kasvanut kesäkuun 2019 versio 2.80 päivityksen jälkeen (Price 2019; Siddi 2019). Päivitetyssä versiossa on työstetty käyttöliittymää uudestaan käyttäjäystävällisemmäksi. Tämän lisäksi Blenderin GNU Open Public License -lisenssin ansiosta kaikki käyttäjien tekemät muutokset ohjelmaan ovat

muokkaajan omaisuutta. Tämä on mahdollistanut Blender-käyttäjien omien laajennuksien sekä taiteen myymisen erilaisissa 3D-painotteisissa nettikaupoissa, kuten Artstationissa, Blendermarketissa ja Gumroadissa. (Blender.org 2021.)

Yksityisten käyttäjien lisäksi indie-elokuva- ja pelistudiot ovat ottaneet muiden ohjelmien ohessa Blenderin käyttöönsä. Esimerkiksi Bungie on mallintanut Blenderillä objekteja peliin Destiny 2 (Choi 2020), A46 Studio on mallintanut aseita peliin Metro: Exodus (A46 Studio 2020), Ubisoft on hyödyntänyt Blenderiä Assassin's Creed: Valhallan konseptitaiteissa (Tan 2020) ja CD Projekt Red on käyttänyt Blenderin työkaluja hahmosuunnittelussaan (Blaszczak 2021).

CGI-töiden ja pelisisällön lisäksi 3D-tekniikkaa hyödynnetään nykypäivänä entistä enemmän elokuvateollisuudessa, taiteessa, mainoksissa sekä erilaisten prototyyppien visualisoinnissa. Esimerkiksi Digital Twin -visualisointitekniikka, erilaisten tuotteiden CAD-mallit tai mainoksien liikegrafiikkaillustraatiot vaativat 3D-osaamista.

Blenderin monipuolisuuden ja joustavuuden ansiosta tutkimukseni aihe on jatkuvassa nousussa, minkä takia tutkimukseni pysyy ajankohtaisena myös tulevina vuosina. Opinnäytetyössäni kehitetty laajennus käyttää proseduraalisessa generoinnissa hyväkseen mallinnettuja, modulaarisia 3D-malleja, minkä takia opinnäytetyössä käydään läpi pintapuolisesti 3D-mallintamisen työvaiheita.

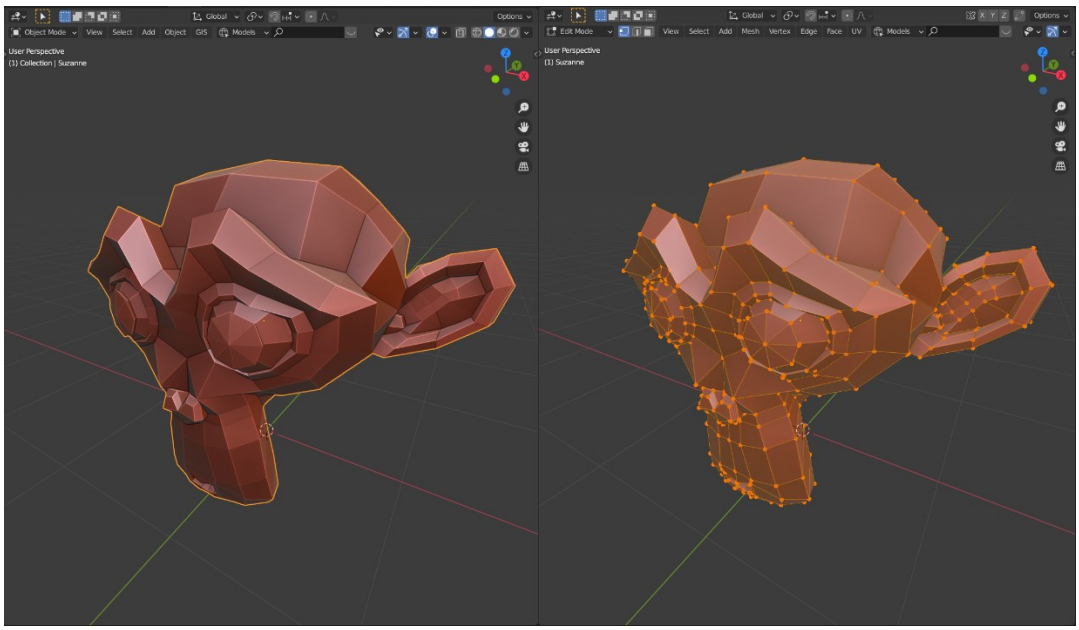
2 3D-TYÖNKULKU

Sopivan 3D-työnkulun valitseminen riippuu käyttökohteesta. Esimerkiksi liikkuvaan kuvaan mallinnettu 3D-hahmo on järkevämpi toteuttaa erilaisella työnkululla kuin pelimoottorissa käytettävä 3D-hahmo. Opinnäytetyössä käytetään hard surface modeling -metodia. 3D-mallinnus on työläs vaihe, sillä huonosti tai huolimattomasti tehty malli tuo tuleviin työvaiheisiin ongelmia. Näitä voivat olla esimerkiksi erilaiset artefaktit eli renderointivirheet valaistusvaiheessa, mallin pintaan väärin kääriytynyt tekstuuri teksturointivaiheessa tai väärin toisiinsa kiinnittyneet mallin pinnat riggaus- ja animointivaiheessa.

2.1 3D-mallintaminen

3D-mallinnuksella tarkoitetaan työvaihetta, jossa muokataan kolmiulotteista mallia ja sen pintojen ominaisuuksia eli kolmiulotteisen mallin geometrian ja topologian manipulointia. Kolmiulotteinen malli koostuu vertexeistä, edgeistä ja faceista eli kärjistä, särmistä ja tahkoista. Näitä osia muokkaamalla malli saadaan muistuttamaan haluttua lopputuotetta. Renderoitava 3D-malli vaatii kuitenkin jo mallinnusvaiheessa erilaisia toimenpiteitä, kuten saumojen merkitsemistä UV-mappausta varten. (Petty 2018.)

Kaikilla 3D-mallin osilla on tarkat koordinaatistot 3D-tilassa (Katsbits 2019a). Kuviossa 1 nähdään, että kärjistä lähtee särmiä muihin kärkiin ja niiden välissä on tahkot. Vasemmalla puolella on Blenderin Object-tila, jossa manipuloidaan kokonaisia objekteja. Oikealla puolella on Edit-tila, jossa manipuloidaan objektin yksittäisiä elementtejä, kuten vertexejä tai faceja. Tässä tilassa näkyvät mallin osat selkeämmin.

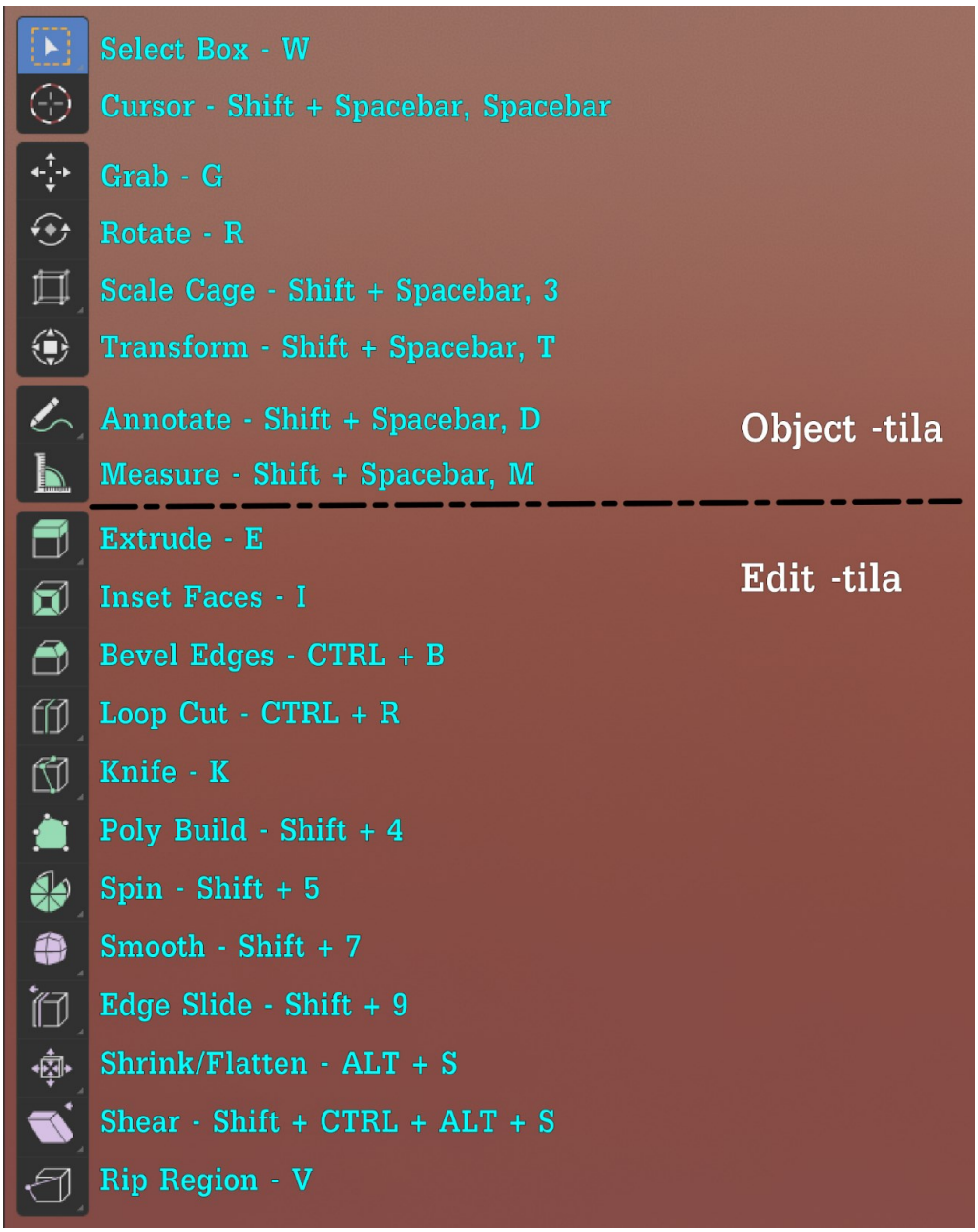


Kuvio 1. Suzanne-apinan pää, joka tulee Blenderin mukana

2.1.1 Mallinnustyökalut

3D-malleja ja niiden osia voi muokata erilaisilla toiminnoilla, joita kutsutaan operaatioiksi. Painamalla näppäintä T saadaan sivupaneeli esille. Sivupaneelissa näkyy erilaisia operaatioita, joita voi käyttää.

Moni Object-tilan operaatioista on universaaleja toimintoja, eli ne toimivat myös muissa ikkunoissa, joissa on liikutettava tai skaalattava elementti. Edit-tilan operaatiot toimivat vain Edit-tilassa. Kuviossa 2 nähdään Object ja Edit-tilan toiminnot sekä niiden pikanäppäinkomennot.

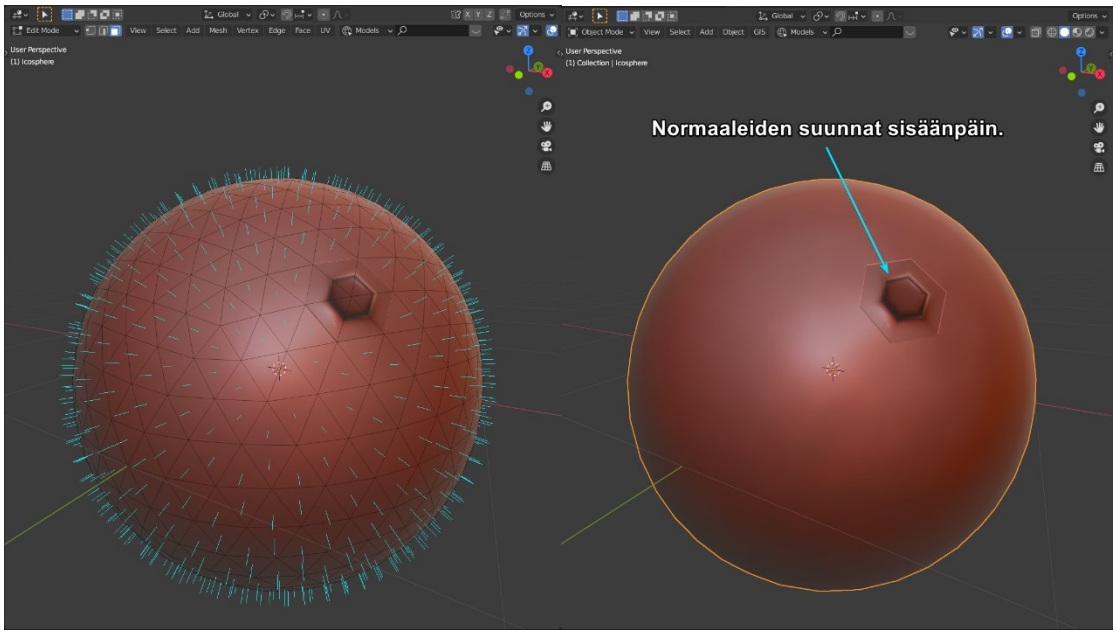


Kuvio 2. Blenderin 3D-viewportin T-paneeli

2.1.2 3D-mallin pinnan normaalit

Teksturointia varten 3D-mallin normaalit pitää saada oikean suuntaisiksi ja 3D-mallin UV:t pitää purkaa. Facen normaalit kertovat, mihin suuntaan face osoittaa. Tämä määrittää, miten mallin pinta valaistaan. Normaaleiden asetetaan sisäänpäin, jos objektia tarkastellaan sisäpuolelta. Normaaleiden suunnat asetetaan ulospäin, jos objektia tarkkaillaan ulkopuolelta. (Blender Manual 2020a.)

Edit-tilassa on mahdollista ottaa käyttöön asetus, joka näyttää pintojen normaaleiden suunnan. Normaaleiden suunnat nähdään sinisinä viivoina, jotka menevät kohtisuoraan facen pinnasta. Esimerkiksi kuviossa 3 on asetettu viiden pinnan normaalit toiseen suuntaan eli sisäänpäin, jolloin nähdään, miten pinta reagoi valaistukseen. Blenderissä normaaleiden väärä suunta vaikuttaa esimerkiksi valaistukseen, mutta esimerkiksi Unity ei näytä pintoja, joiden normaalit ovat kamerasta poispäin. Tämä johtuu Unityn kentän optimoinnin oletusasetuksista (Unity Manual 2020).

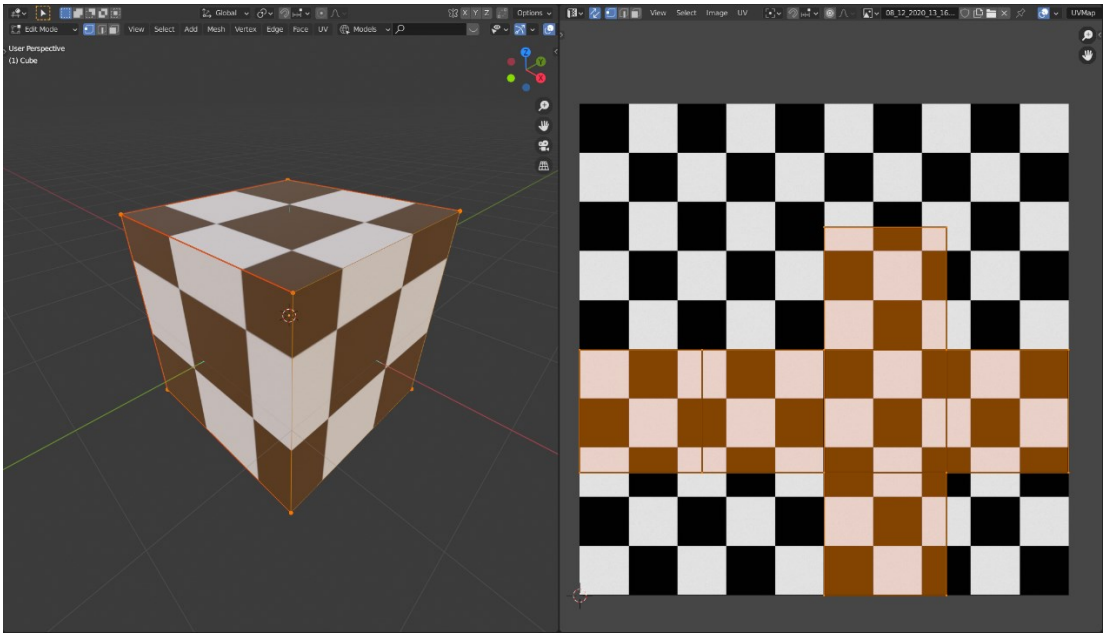


Kuvio 3. Normaalit

2.1.3 UV-mappaus

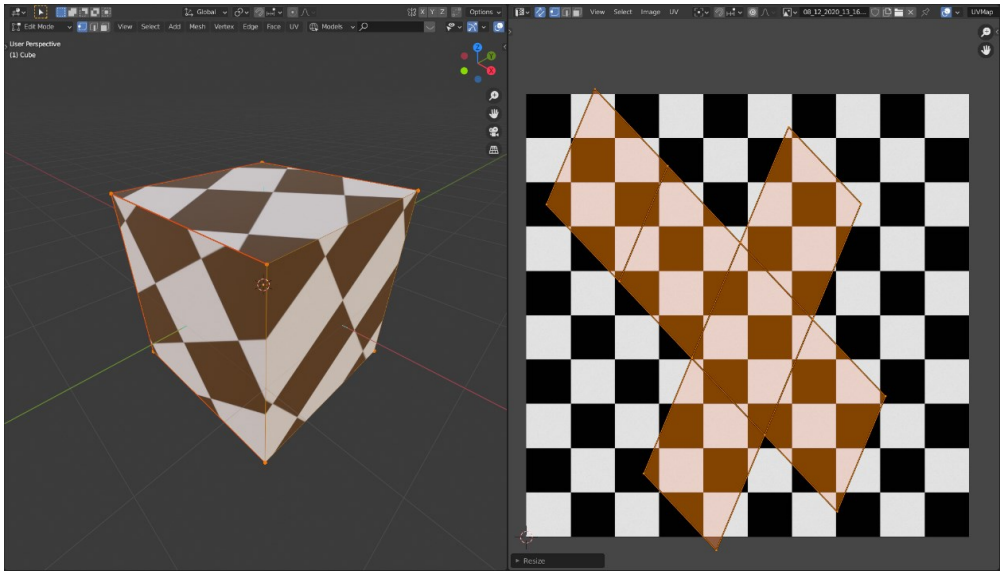
Teksturointi tarkoittaa prosessina käytännössä 2D-kuvien liimaamista 3D-mallin pintaan. UV-mappauksessa puretaan 3D-mallin facet 2D-tasokuvaksi eli UV:ksi. U ja V vastaavat 2D-tason X ja Y-akseleita. UV:t asetetaan tasolle siten, että ne täyttävät mahdollisimman hyvin olemassa olevan tilan. Purkua ohjataan merkitsemällä saumoja. UV-mappaus on ikään kuin käänteinen origami, jossa taitettu 3D-malli puretaan tasaiseksi. Huonosti purettu malli johtaa tekstuurin venymiseen ja saumojen alueella tekstuurin jatkumattomuuteen. Hyvin purettu malli mahdollistaa erilaisten tekstuureiden sopimisen esineen pintaan. Joissain tapauksissa mallin topologia on niin monimutkainen, että saumat jäävät selkeästi näkyville. Tällaisissa tapauksissa saumat asetetaan paikkoihin, joista esinettä

tarkastellaan harvoin. (Denham 2020.) Seuraavassa esimerkissä mapataan 3D-mallin UV:t manuaalisesti merkitsemällä sopiviin edgeihin saumoja. Kuvioissa 4 nähdään kuutio, joka UV-mapataan. Punaiset edget ovat käyttäjän merkitsemät saumat. Tekstuuri asettuu 3D-malliin järkevästi, jos 3D-mallin UV:t mapataan järkevästi 2D-tasolle.



Kuvio 4. Esimerkki hyvin tehdystä UV-purusta

Kuviossa 5 nähdään kuutio, jonka UV-mapit ovat sijoitettu huonosti. Tekstuuri asettuu 3D-mallin pintaan huonosti ja johtaa tekstuurin jatkumattomuuteen saumojen kohdalla. Sekavasti tai huonosti UV-purettu malli kannattaa korjata suorittamalla UV-mappaus uudestaan.



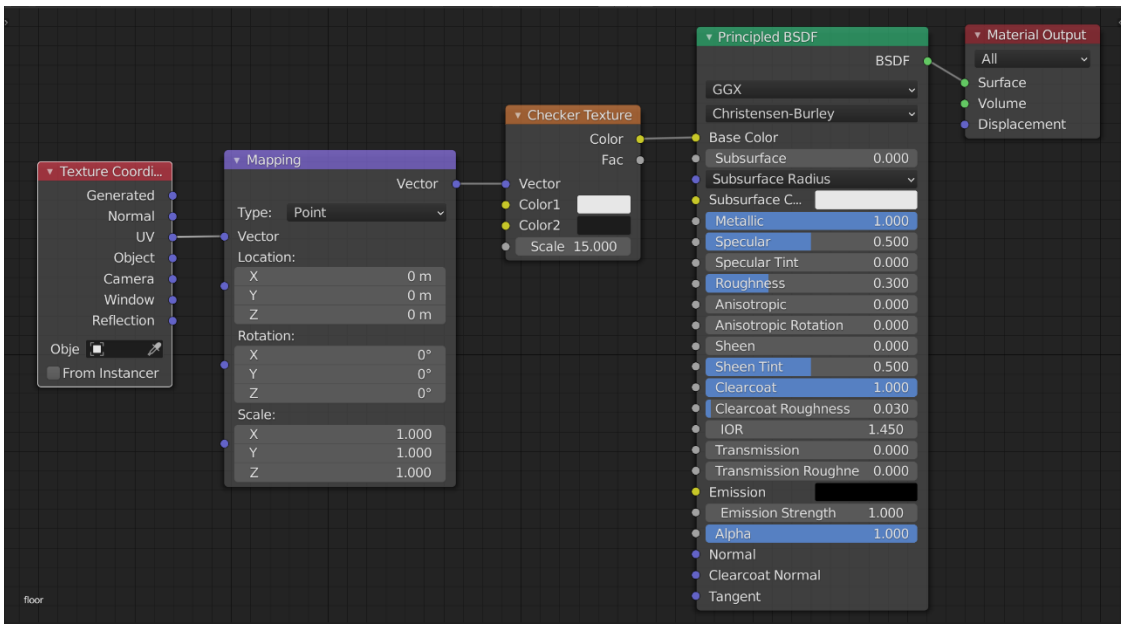
Kuvio 5. Esimerkki huonosti tehdystä UV-purusta

2.2 Node editor

Blenderissä on käytössä visuaalinen node-pohjainen muokkausikkuna esimerkiksi compositorissa sekä Shader-editorissa. Erilaiset nodet suorittavat erilaisen muokkauksen, ja niitä voi yhdistellä toisiinsa muodostaen jonkinlaisen asetuksien loppusumman. Asetuksien loppusumma näkyy käyttäjälle ainoastaan, jos node-ketju yhdistetään Output-nodeen. Noden kulkusuunta on vasemmalta oikealle. Noden vasemmalla puolella sijaitsee sisääntulot ja oikealla puolella ulostulot. Noden sisään- sekä ulostulot ovat värikoodattuja. Harmaa pallo ottaa vastaan ja lähettää matemaattisia tai numeerisia float-arvoja, keltaiset pallot RGBA-arvoja, siniset pallot vektoritietoa ja vihreät pallot shader-tietoa. (Blender Manual 2020e.)

Node-pohjaisissa muokkausikkunoissa pystyy käyttämään Blenderin universaaleja pikanäppäinkomentoja. Esimerkiksi Shift+A-pikanäppäinyhdistelmällä voi lisätä erilaisia nodeja tai G:tä painamalla voi liikutella nodeja. Blender pyrkii yleistämään operaatioita mahdollistaen samojen komentojen käytön erilaisissa ikkunoissa.

Kuvio 6 esittää Shader-editorin node-asetuksia, joilla saadaan kuvioissa 4 sekä 5 käytetty shakkiruututekstuuri. Texture Coordinate -node määrittää mistä lähteestä tekstuuri asettuu 3D-mallin pintaan. Tässä tapauksessa on käytetty 3D-mallin omia UV-mappeja. Mapping-node määrittää, millä koolla tai orientaatiolla tekstuuri asettuu UV-mappeihin. Checker Texture -node korvaa Principled BSDF-shaderin väriasetuksen, minkä takia 3D-objekti näyttää ruudulliselta.

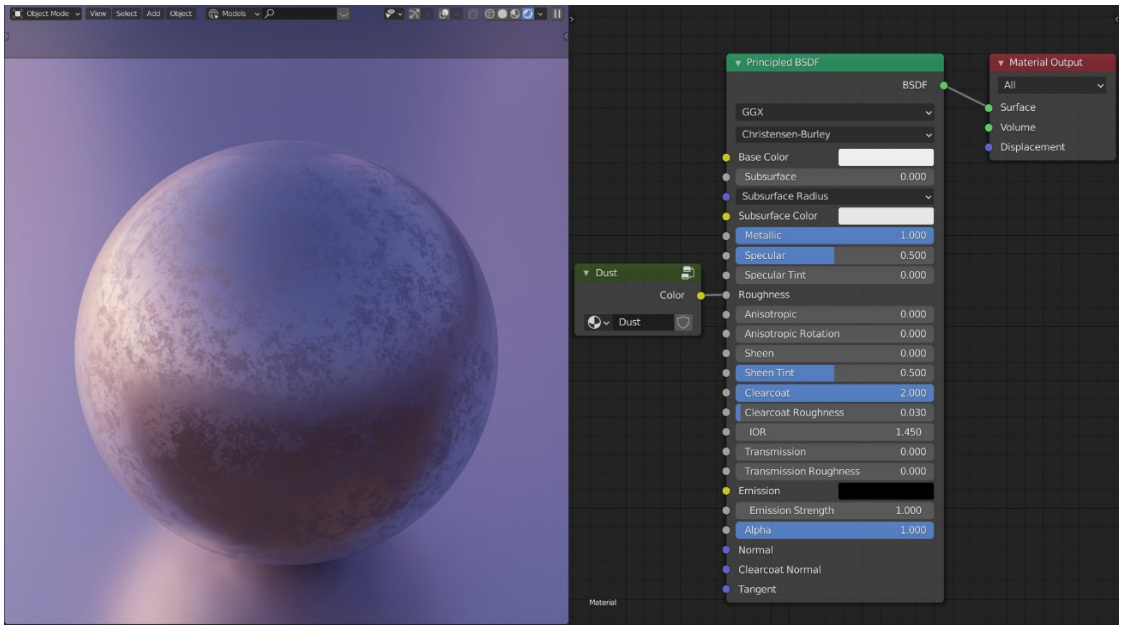


Kuvio 6. Shader editor -ikkuna

2.3 PBR-teksturointi

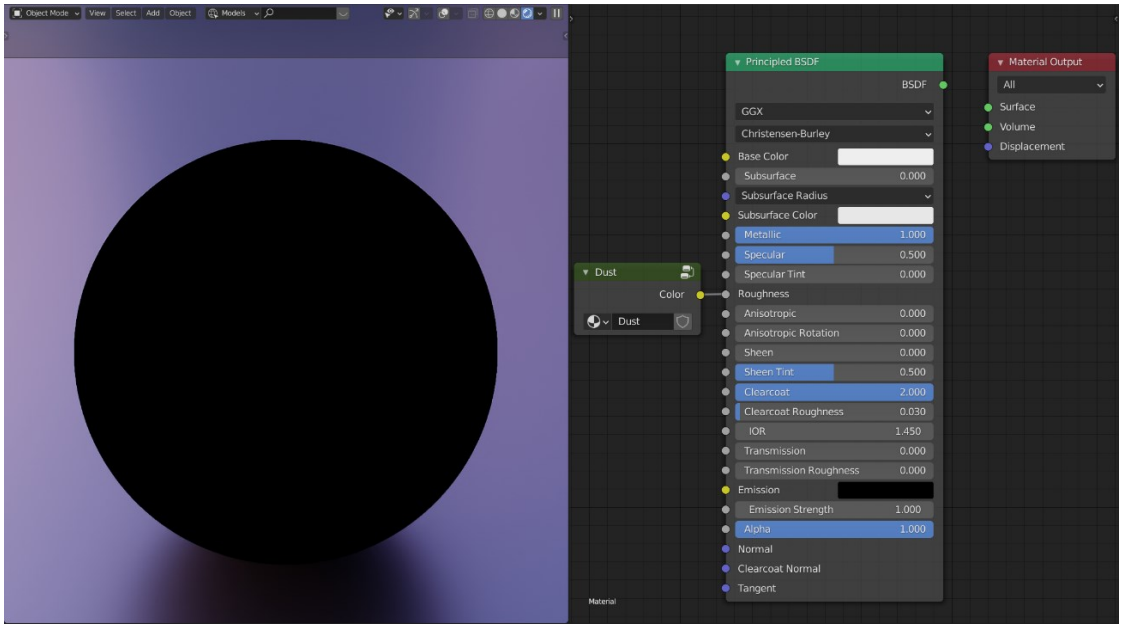
UV-mappauksen jälkeen siirrytään teksturointiin. Teksturoinnissa tuodaan haluttu tyyli selvästi esille. Tässä tapauksessa pyritään mahdollisimman fotorealistiseen tyyliin, joten käytetään PBR-tekniikkaa. Physically Based Rendering tarkoittaa valo-oppiin perustuvaa renderointitekniikkaa. Tällä tekniikalla yhdessä Cycles-säteenseurantarenderointimoottorilla saavutetaan mahdollisimman fotorealistisia tuloksia (Purcell 2004). Blenderissä on mahdollista ohjata materiaalin ulkonäköä esimerkiksi viidellä eri texture mapilla. PBR-teksturoinnissa käytetään tyypillisesti oikeasta maailmasta skannattuja 2D-kuvia eli bittikarttoja texture mappeina. Bittikartta voi olla esimerkiksi resoluutioltaan 6144 x 6144 pikseliä. Näitä kutsutaan lyhyesti 6K-tekstuureiksi. Samalla tavalla 2048 x 2048 -pikselin bittikartat ovat 2K-tekstuureita ja 4096 x 4096 -pikselin bittikartat ovat 4K-tekstuureita. (McDermott 2018.)

Blenderissä on Principled BSDF -shader oletuksena käytössä (Blender Manual 2020c). Renderointimoottori laskee shaderille annettujen ohjeiden ja informaation avulla valaistuksen (McDermott 2018). Blender laskee valaistuksen objekteille ainoastaan silloin, jos objektin shader on yhdistetty Material output -nodeen (Kuviot 7 ja 8).



Kuvio 7. Pallo, jonka shader on yhdistetty Material output -nodeen

Principled BSDF -shaderissa on mahdollista säätää erilaisia asetuksia. Näitä asetuksia muuttamalla pystyy jäljittelemään melko tarkasti mitä vain olemassa olevaa tunnettua materiaalia. Kuviossa 7 on Principled-shaderin roughness-arvolle annettu erityinen asetus, jonka takia pallo näyttää likaiselta.

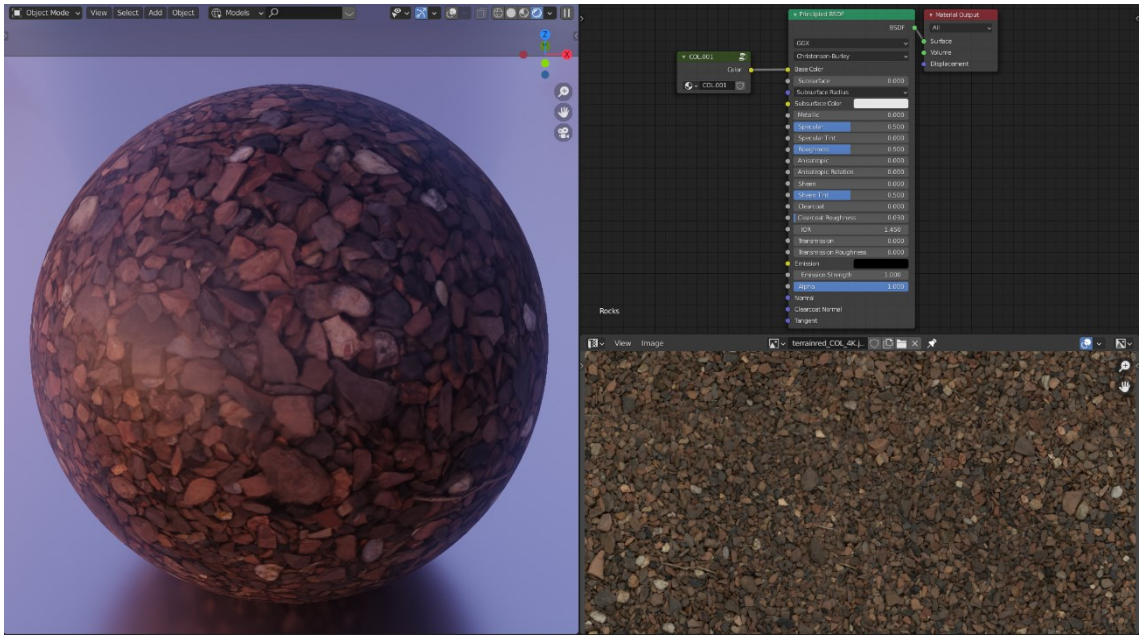


Kuvio 8. Pallo, jonka shader ei ole yhdistetty Material output -nodeen

Blenderin Cycles-säteenseurantamoottori sekä reaaliaikainen rasterointimoottori Eevee käyttävät oletuksena Principled BSDF -shaderia (Katsbits 2019b). Eri renderointimoottorit käyttävät eri shadereita, esimerkiksi AMD:n ProRender -renderointimoottori käyttää Principled BSDF -shaderin lisäksi Uber-shaderia (Advanced Micro Devices, Inc 2020).

PBR-teksturoinnissa tarvitaan erilaisia texture mappeja, jotka on skannattu oikeasta maailmasta. Blenderin Principled BSDF -shaderin kanssa voi käyttää specular tai metallic työnkulkua. Opinnäytetyössä käytetään metallic työnkulkua, jonka yleisimmät texture mapit ovat colour, metallic, roughness, normal, displacement sekä ambient occlusion map (Zgraggen 2019). Ilman näitä texture mappeja 3D-objektin pinta näyttäisi hyvin homogeeniselta ja yksinkertaiselta. Yhdistämällä näitä texture mappeja shaderin sisääntuloihin saadaan pinta heterogeeniseksi sekä luonnollisemmaksi korvaamalla shaderin omia ennalta määriteltyjä asetuksia texture mapin informaatiolla.

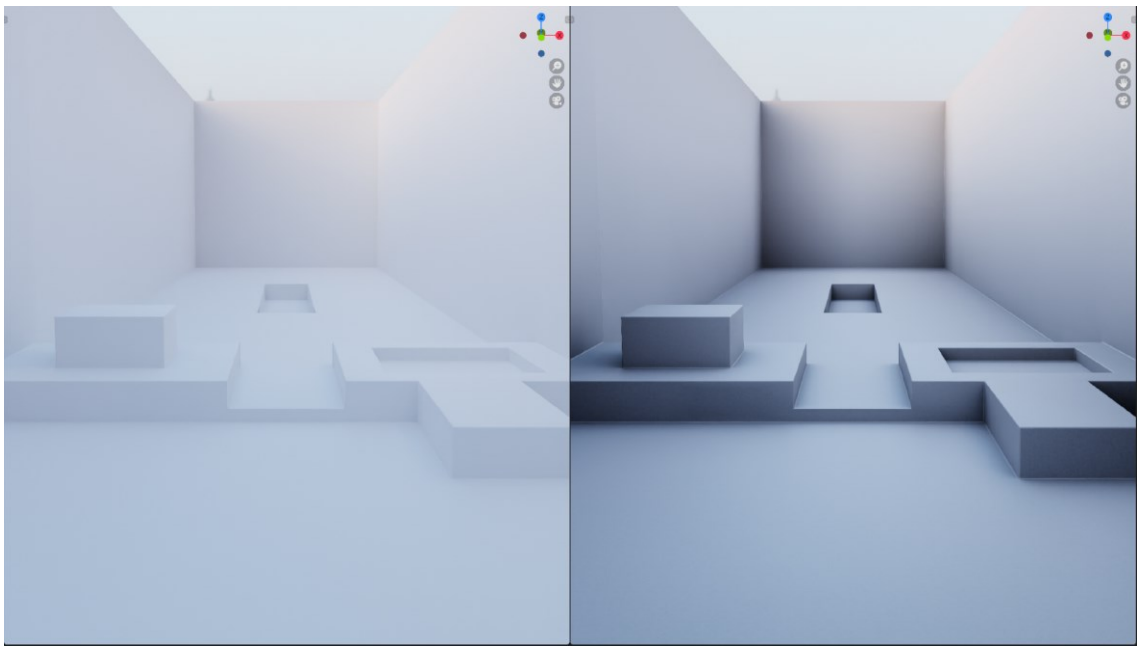
Colour map antaa shaderille väritiedon (Kuvio 9). Yleensä vain colour map antaa shaderille väritiedot, loput texture mapit asetetaan antamaan vain ei-väritietoja shaderille. (Zgraggen 2019.)



Kuvio 9. Colour mapin yhdistäminen shaderiin

Ambient occlusion map antaa 3D-mallille keinotekoisia pehmeitä varjoja. Kuviossa 10 nähdään vasemmalla käytävä, joka ei käytä ambient occlusion mappia

ja oikealla sama käytävä, joka käyttää ambient occlusion mappia. Blenderissä ambient occlusionin simulointi ei ole valosta riippuvainen. Se lasketaan asettamalla facejen vertexeihin näytepalloja ja tarkistamalla osuuko nämä näytepallot muuhun geometriaan. 3D-mallin pinta tummenee, jos muu geometria osuu näytepalloon. (Blender Manual 2020i.) Tämä keventää säteenseurantamoottorin renderointia, sillä pehmeät varjot syntyvät vain valon heijastuksista, toisin sanoen vain kimpoavista valon säteistä. Reaaliaikaisille rasterointimoottoreille pehmeiden varjojen saavuttaminen toteutuu vain joko esilaskemalla kentän valaistuksen tai ambient occlusion mappeja käyttämällä. Ambient occlusion mapin tarkoitus on nopeuttaa ja keventää renderointiprosessia. Yleensä on kätevämpää ja laskennallisesti kevyempää käyttää näitä mappeja, kuin alkaa fyysisesti simuloimaan efektejä. (Zgraggen 2019.)

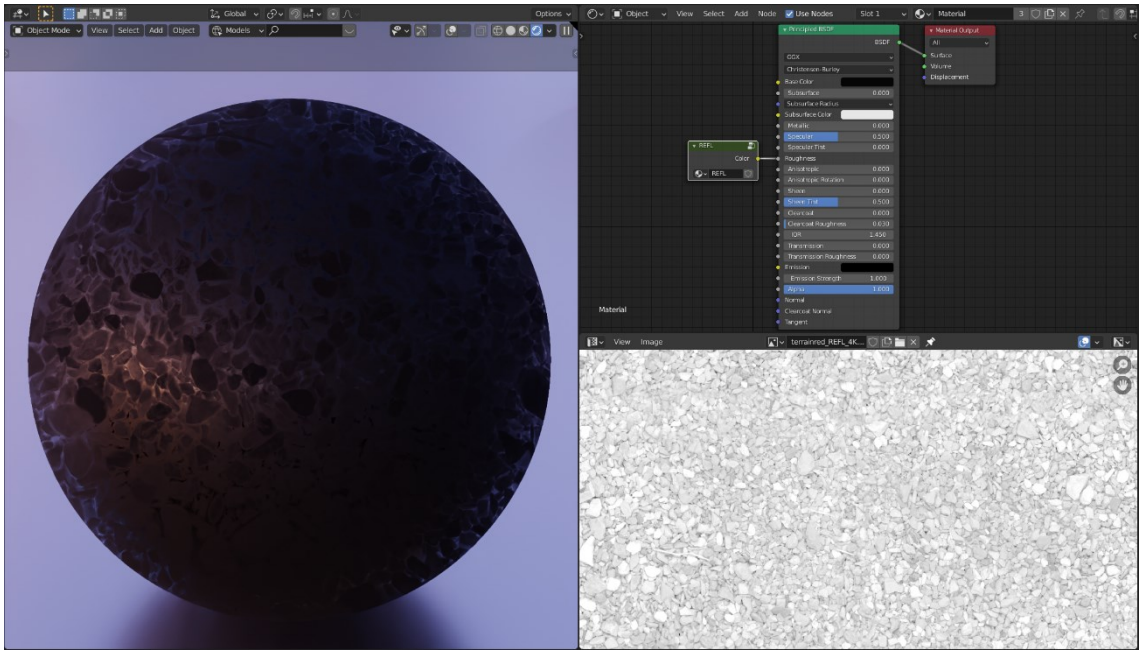


Kuvio 10. Ambient occlusion mapin vaikutus valaistukseen

Metallic map kertoo shaderille, mitkä kohdat pinnasta ovat metallisia ja mitkä eivät. Metallisuuskarttaa tarvitaan materiaaleissa, joissa esiintyy metallista ainetta. Kuvio 10 esittää kivistä materiaalia, joten metallic mappia ei tarvita. (Zgraggen 2019.)

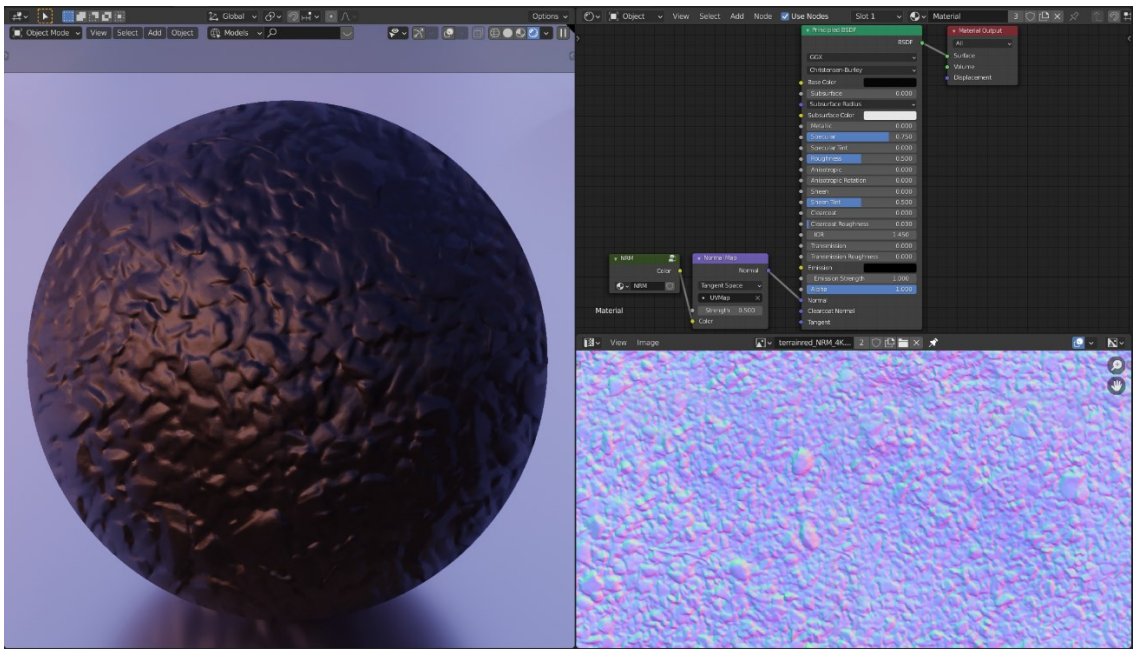
Roughness map kertoo shaderille, kuinka kiiltävä materiaali on (Zgraggen 2019). Kuviota 11 tarkasteltaessa huomataan, että roughness mappia käyttä-

mällä saadaan mallin pinnan heijastus monimutkaiseksi. Roughness map on yhdistetty Principledin roughness asetukseen. Cycles lukee roughness mapin vaaleat värit karheaksi ja tummat värit sileäksi pinnaksi. Tämän avulla saadaan materiaalin eri kohdat heijastamaan valoa eri lailla. Kuvio 11 pallon väri on asetettu tummaksi, jotta nähdään roughness mapin vaikutus selkeämmin.



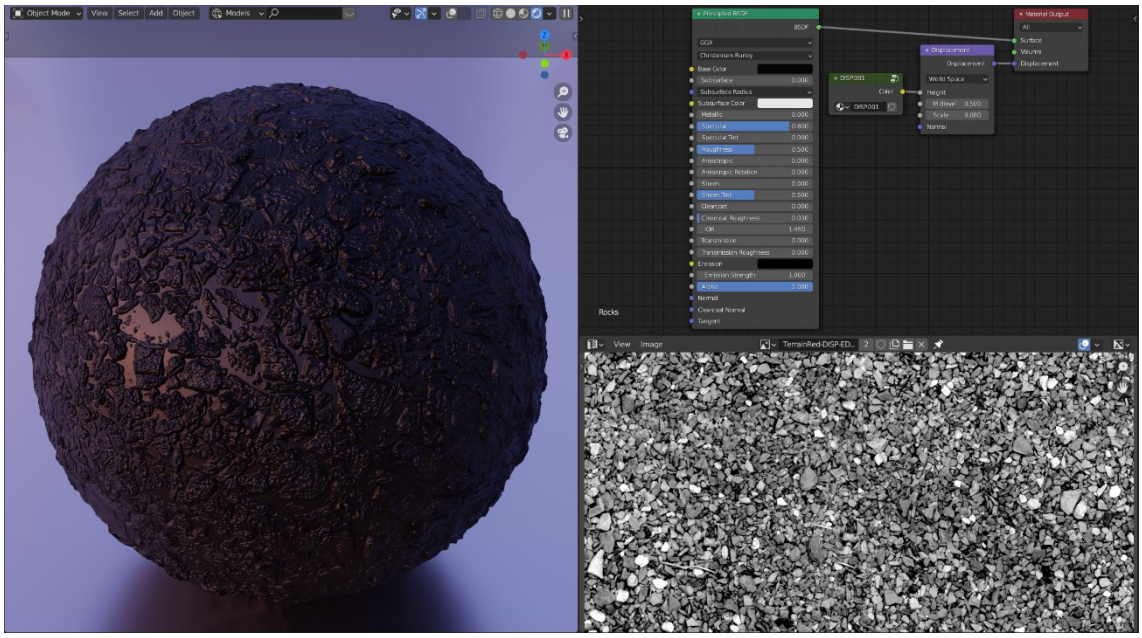
Kuvio 11. Roughness mapin yhdistäminen shaderiin

Normal map kertoo shaderille, mihin suuntaan 3D-mallin facen normaalit osoittavat. Normal mapin avulla saadaan yhden facen pinta vaikuttamaan monimutkaisemmalta geometrialta. Se huijaa valaistuksella 3D-mallin pintaan yksityiskohtia ja saa tasaiseen pintaan epätasaisuutta. Normal mapin RGB-arvot vastaavat Blenderin XYZ-koordinaatteja. R eli punaisen mapin arvo voi olla väliltä 0–255 ja se vastaa X-arvoa väliltä -1,0 – 1,0. G eli Green mapin arvo voi olla väliltä 0–255 ja se vastaa Y-arvoa väliltä -1,0 – 1,0. B eli Blue mapin arvo voi olla väliltä 0–255 ja se vastaa Z-arvoa väliltä 0,0 – 1,0. (Blender Manual 2020j.) Normal mapin RGB-arvot käännetään Blenderin XYZ-koordinaatteihin Normal map -nodella, joka tulee mapin ja shaderin väliin kuvion 12 mukaisesti.



Kuvio 12. Normal mapin yhdistäminen shaderiin

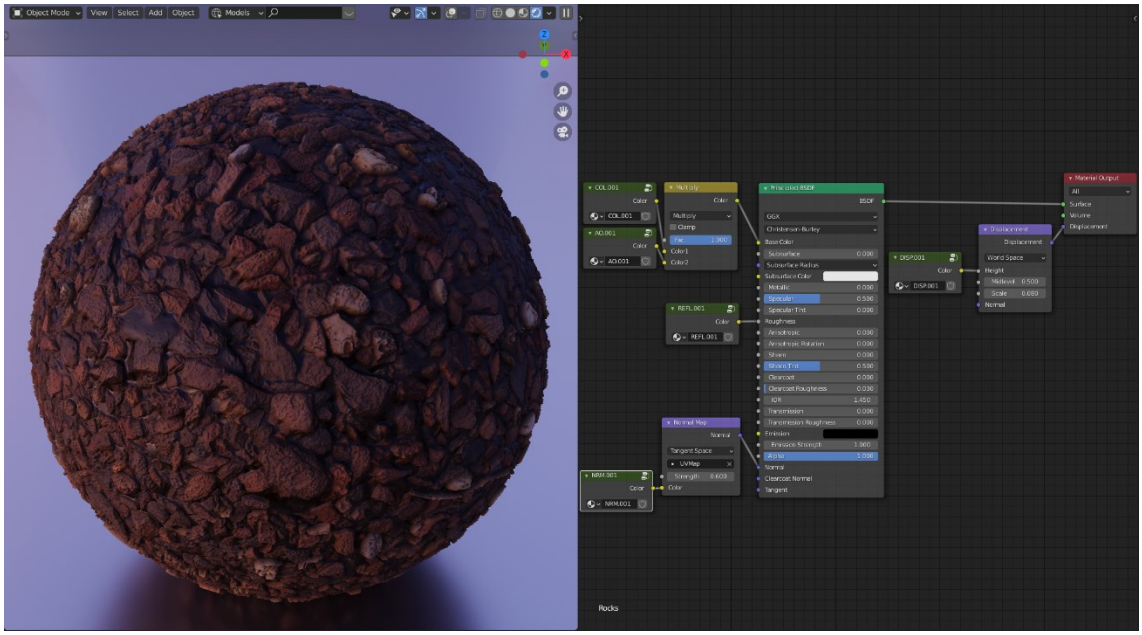
Displacement map kertoo, miten mallin geometrian tulisi siirtyä. Tämän avulla saadaan esimerkiksi ylimääräisiä yksityiskohtia mallin pintaan mikrotopologiana. Toisin kuin normal map, jossa yhden facen pintaan huijataan yksityiskohtia käyttämällä valaistusta, displacement map siirtää fyysisesti vertexejä 3D-tilassa ja se vaatii monimutkaisen meshin. (Zgraggen 2019.)



Kuvio 13. Displacement mapin yhdistäminen shaderiin

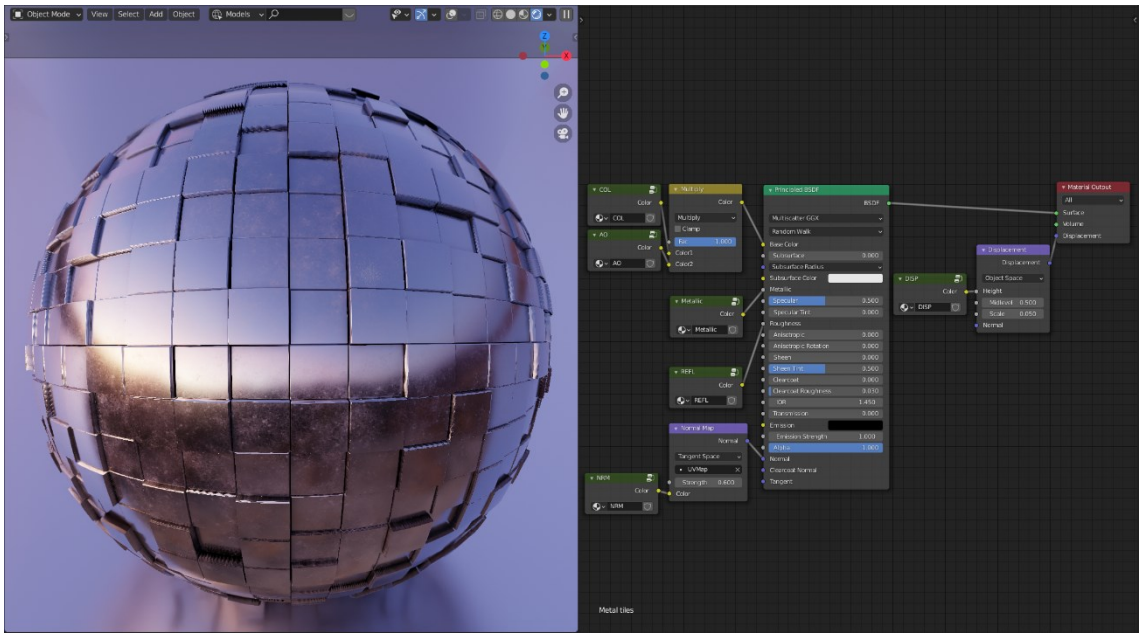
Kuviossa 13 nähdään, että displacement map yhdistetään shaderin sijaan Material output -nodeen. Material output -nodessa on Displacement-sisääntulo, joka

on vastuussa fyysisen geometrian muokkauksesta. Displacement mapin käyttö on laskennallisesti erittäin raskasta, koska se siirtää jokaisen kärjen koordinaatin 3D-kentässä aina, kun käyttäjä renderoi. Kuviossa 14 nähdään materiaali, joka on PBR-teksturoitu. Tässä materiaalissa on yhteensä viisi texture mappia käytettynä.



Kuvio 14. Viiden texture mapin yhdistäminen shaderiin

Kuviossa 15 näkyy metallinen materiaali, joka on PBR-teksturoitu. Materiaalissa on käytetty metallic mappia osoittamaan shaderille, mitkä kohdat ovat metallisia ja mitkä eivät. Kuvion 15 tapauksessa likaiset ja pölyiset kohdat ovat epämetallisia, kun taas puhtaat pinnat ovat metallisia. Materiaalissa on käytetty yhteensä kuutta texture mappia.

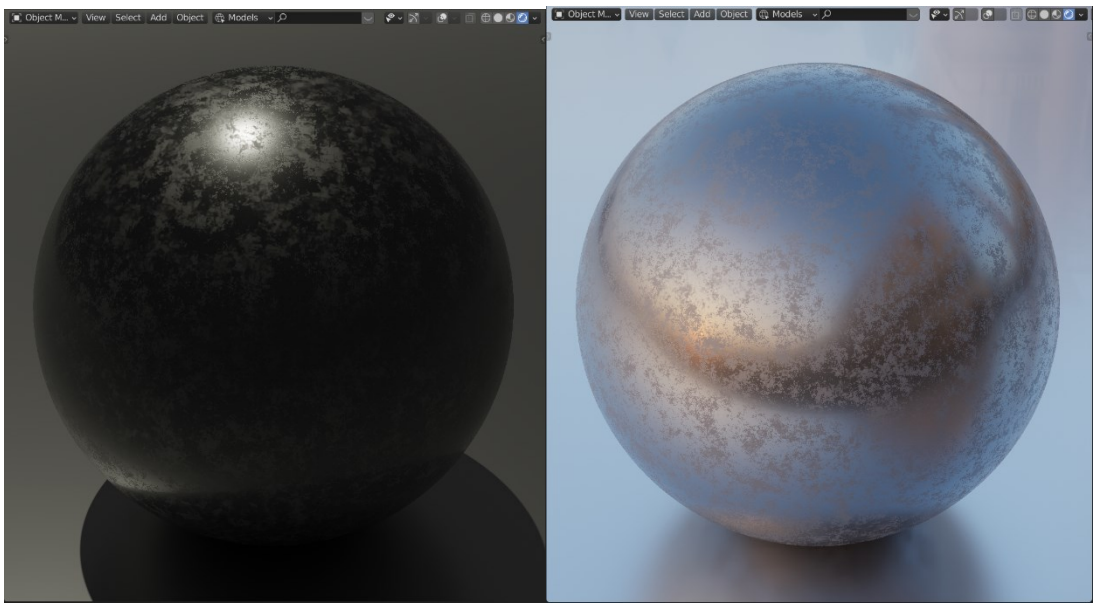


Kuvio 15. Metallinen materiaali, joka on PBR-teksturoitu

2.4 Valaistus

Blenderin Cycles jäljentää oikean maailman valaistusta melko tarkasti sen säteenseurantatekniikan ansiosta. Esimerkiksi pieni ja lähellä oleva valonlähde luo dramaattisen ja terävän varjon mallin taakse, kun taas iso ja kaukana oleva valonlähde luo pehmeämmän varjostuksen aivan, kuten oikeassa maailmassa. Realistisin valaistus saadaan kuitenkin ympäristövalaistuksella HDRi-mappia käyttäen. Oikeassa maailmassa valoa tulee monesta eri suunnasta erivärisenä ja -vahvuisena. HDRi-valaistus pyrkii jäljittelemään tätä valaisemalla 3D-kentän 360-asteisella kuvalla, jossa kuvan jokainen pikseli tuottaa eri vahvuudella valoa ja väriä. (Wengenmayer 2016; Carlson 2017.)

Kuviossa 16 on vasemmalla Blenderin aurinkolamppu ja oikealla leikkaamaton 32-bittinen HDRi-mappi käytössä. Blenderin aurinkolampusta lähtee yhdellä intensiteetillä yhteen suuntaan samanväristä valoa, jonka takia valaistus näyttää yksinkertaiselta. Vasen metallipallo näyttää mustalta, koska Blenderin ympäristövärin oletusasetuksena on homogeeninen tummanharmaa, joten heijastettavaa ei ole. Oikealla puolella nähdään heijastuksessa sininen taivas, ilta-aurinko sekä beigenvärinen rakennus. Oikean puolen HDRi-kuva on otettu italialaiselta kadulta.

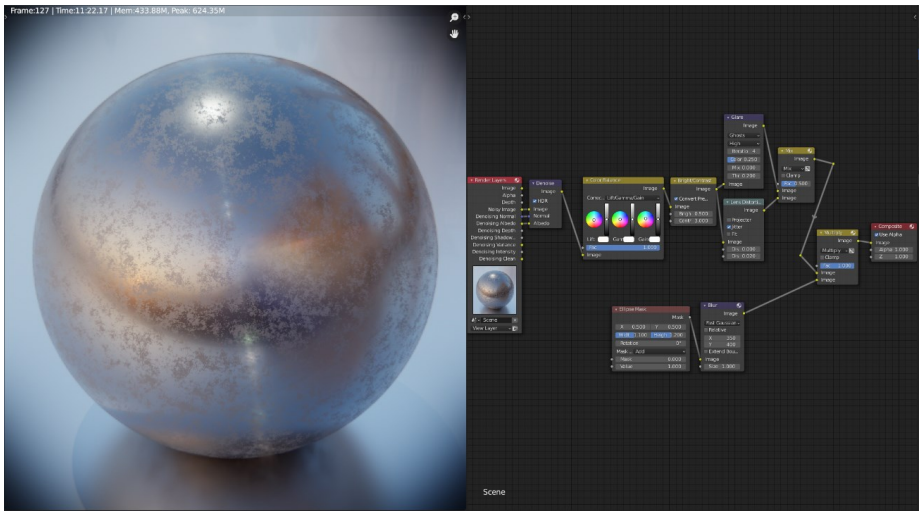


Kuvio 16. Vasemmalla Blenderin aurinkolamppu ja oikealla HDRi-valaistus

2.5 Jälkiprosessointi

Renderoinnin jälkeen on mahdollista injektoida renderoituun kuvaan ylimääräisiä efektejä, tehdä värimäärittely sekä yhdistää erilaisia elementtejä renderoituun kuvaan. Compositing-ikkunassa voi myös esimerkiksi simuloida oikean kameran linssin vääristymiä. Compositing-ikkunassa prosessoitu kuva on vahvasti käyttäjästä riippuvainen.

Kuviossa 17 käytetyt efektit ovat vain esimerkkejä. Kuviossa on käytetty muun muassa värikorjaus-, kirkkaus/kontrasti-, häikäisy- ja linssin vääristymä -efektejä sekä matemaattisia operaatioita efektien yhdistämistä varten.



Kuvio 17. Compositing-ikkunassa prosessoitu kuva

3 PROSEDURAALINEN 3D-GENEROINTI

Proseduraalinen generointi tarkoittaa datan luontia ennalta-määrätyillä säännöillä. Käyttäjä pystyy esimerkiksi nappia painamalla generoimaan 3D-kentän tai 3D-mallin, joka on yhdistetty satunnaisesti modulaarisilla osilla suureksi kokonaisuudeksi. Proseduraalisen generoinnin tärkein ominaisuus on sen kyky osata luoda uutta ja ainutlaatuista annetuilla ohjeilla. (Bycer 2015.)

Proseduraalisen generoinnin hyviä puolia ovat sen nopeus ja yksinkertaisuus. Laajennukselle annetaan 3D-mallit tai säännöt, jotka tekevät generoiduista kentistä ainutlaatuisia. Laajennuksen 3D-mallien tai sääntöjen määrä on ainutlaatuisuuden rajoite, toisin sanoen alkutekijöiden muuttujien määrä vaikuttaa lopputuloksen satunnaisuuteen. Huonona puolena tässä tekniikassa on sen joustamattomuus. Laajennus osaa generoida ainoastaan esimerkiksi käytäväkomplekseja, jos sille on mallinnettu ainoastaan käytävän osia ja se on ohjelmoitu osaamaan yhdistellä vain käytävän osia toisiinsa. (Bycer 2015.)

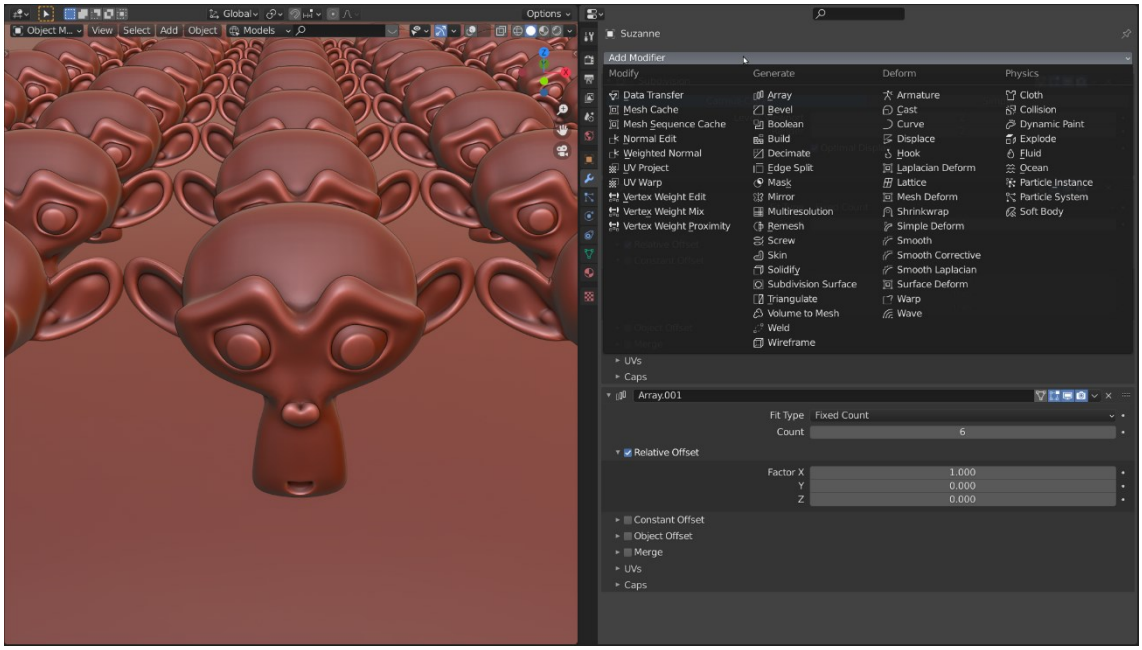
Blenderissä on mahdollista mallintaa proseduraalisesti muuntajien tai geometrianodejen avulla. Molemmat menetelmät käyttävät hyväkseen noise-algoritmeja, joiden ansiosta havaittu pseudo-satunnaisuus saavutetaan.

3.1 Muuntajat

Muuntajat ovat automaattisia operaatioita, jotka vaikuttavat 3D-mallin geometriaan tuhoamattomalla tavalla. Muuntajat muuttavat, miten 3D-malli esiintyy käyttäjälle object-tilassa sekä render-tilassa, mutta ei muuta käyttäjän muokkaamaa geometriaa edit-tilassa. Muuntajien käyttö on tuhoamaton 3D-mallinnustekniikka, koska muuntajia voi kytkeä päälle tai pois milloin tahansa. Käyttäjä voi asettaa yhdelle 3D-mallille monta muuntajaa, jotka vaikuttavat samanaikaisesti 3D-mallin ulkonäköön. (Blender Manual 2020b.)

Kuviossa 18 nähdään oikealla puolella Muuntajat-valikko, jossa voi pudotusvalikosta valita erilaisia muuntajia 3D-mallille. Sen alapuolella näkyy muuntajapino, jossa muokataan yksittäisen muuntajan asetuksia. Muuntajia otetaan huomioon järjestyksessä ylhäältä alas. Esimerkissä on käytetty array-muuntajaa, joka monistaa objektin. Muuntajan voi ottaa pois käytöstä, minkä jälkeen kenttään jäisi

vain yksi apinan pää. Array-muuntajan ansiosta muokkaus yhteen päähän näkyisi jokaisessa päässä, joten samaa muokkausta ei tarvitse tehdä jokaiseen päähän erikseen. Käyttäjä joutuisi tekemään muutokset jokaiseen päähän erikseen, jos päät kopioitaisiin ilman muuntajaa. (Blender Manual 2020b.)

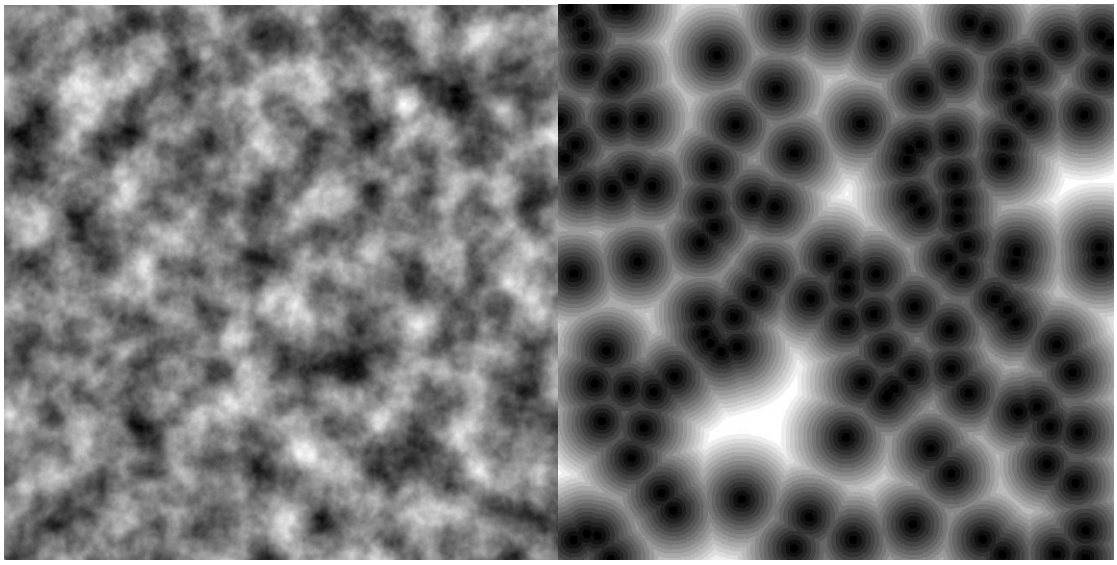


Kuvio 18. Muuntajien käyttö

3.2 Noise-algoritmit

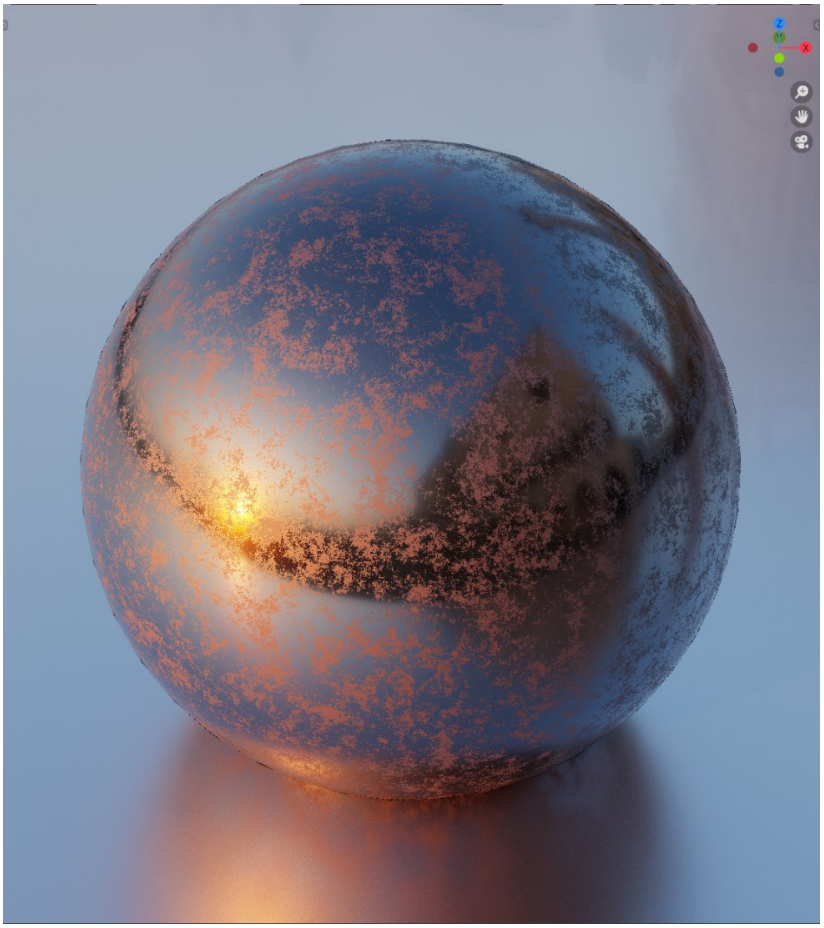
Noise-algoritmit ovat alun perin suunniteltu antamaan digitaaliselle taiteelle luonnollisemman ja orgaanisemman ulkonäön. Tämän takia osa proseduraalisen generoinnin muuntajista käyttää hyväkseen noise-algoritmeja niiden pseudo-satunnaisuuden saavuttamiseksi. Noise-algoritmit ovat matemaattisia funktioita, jotka muodostavat signaalin. (Vivo & Lowe 2015a.)

Blenderissä on mahdollista käyttää erilaisia noise-algoritmejä. Kuviossa 19 nähdään vasemmalla Ken Perlinin kehittämä Perlin noise -algoritimi ja oikealla Georgy Voronoyn kehittämä Voronoi noise -algoritmi. (Blender Manual 2020d.) noise-algoritmeja voi ohjata erilaisilla asetuksilla. Asetuksien muokkaaminen tapahtuu Shader editor -ikkunassa.



Kuvio 19. Vasemmalla Perlin noise -algoritmi ja oikealla Voronoi noise -algoritmi

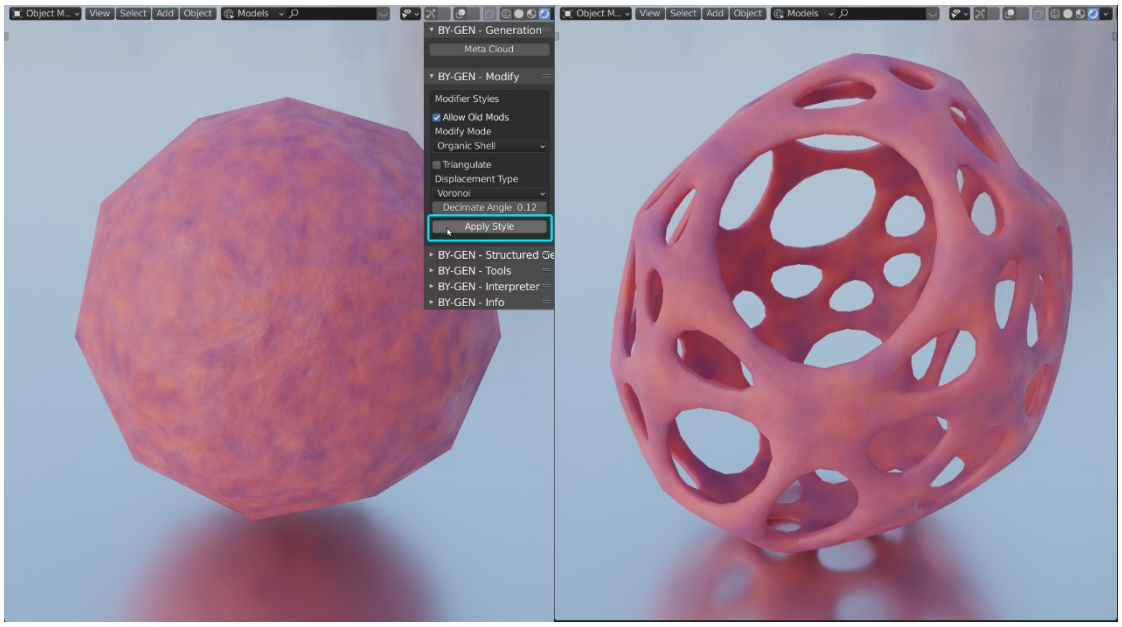
Kuviossa 20 pallon pinnalla oleva pöly ja lika on generoitu Perlin noise -algoritmilla. Kuviossa 21 (s. 28) nähdään 3D-malli, joka käyttää Voronoi noise -algoritmia generoinnissa. Generoidusta mallista huomaa, että reikien muodot vastaavat melko lähellä kuvion 19 Voronoi noise -algoritmin muotoja.



Kuvio 20. Perlin noise -algoritmilla generoitu pinnan epäpuhtaudet

3.3 Proseduraalinen mallintaminen muuntajilla

Curtis Holtin By-Gen-laajennus on proseduraalinen mallinnustyökalu, jonka voi ladata Blenderiin lisäosana. Se luo esimerkiksi napin painalluksella kiinnostavia muotoja ja ennalta-arvaamattomia lopputuloksia. (Holt 2020.) Holtin työkalu tarjoaa erilaisia generoinnin tyylejä. Generoinnin tyylin valitsemisen jälkeen By-Gen asettaa erilaisia muuntajia 3D-malliin. Muuntajien ansiosta 3D-malli saadaan esiintymään erilaisena. Kuviossa 21 nähdään vasemmalla yksinkertainen ikosaedri, johon asetetaan By-Genin proseduraalinen orgaaninen kuori -asetus. Oikealla näkyy proseduraalisesta generoinnista syntynyt lopputulos, jossa generoinnin pohjana on käytetty Voronoi noise -algoritmia. Kuviossa näkyvät materiaalit ovat manuaalisesti asetettuja.

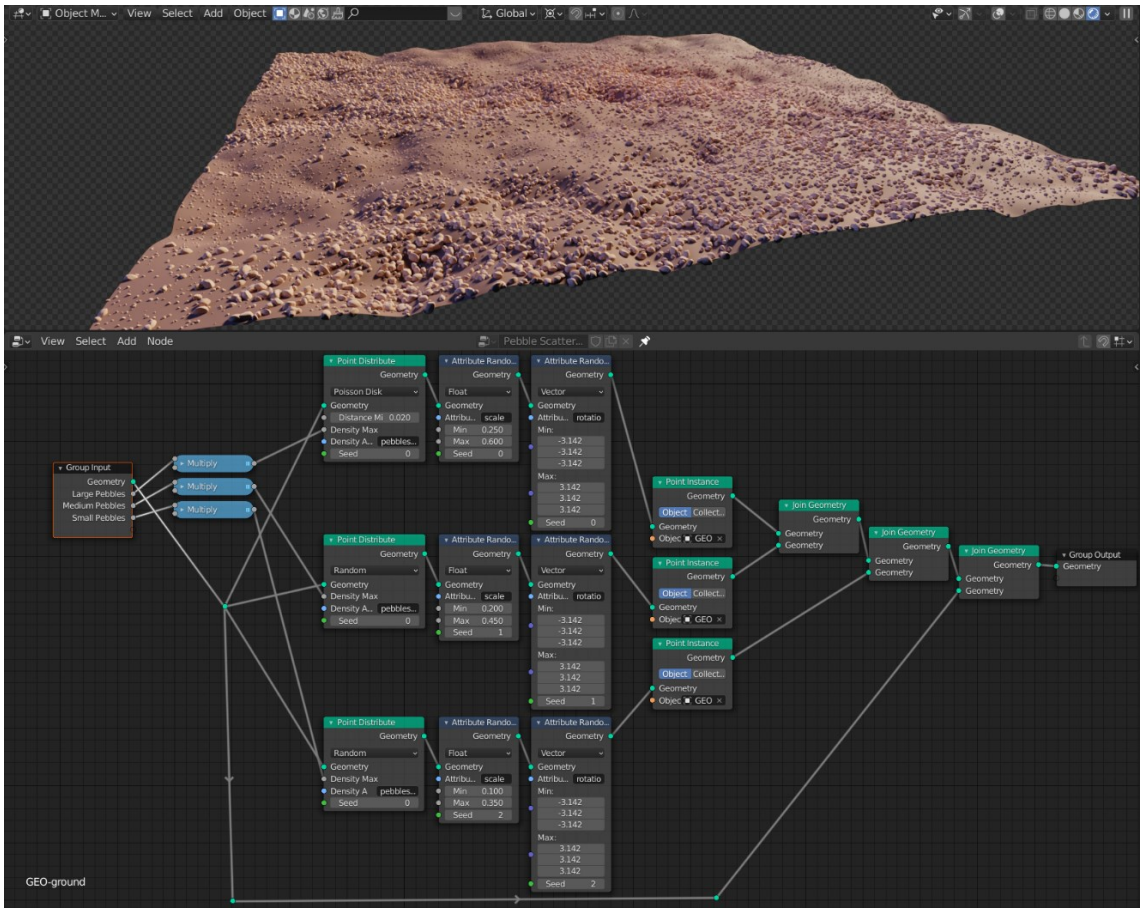


Kuvio 21. By-Genin proseduraalinen generointityökalu

3.4 Proseduraalinen mallintaminen geometrianodeilla

Geometrianodet tulivat Blender 2.92 -päivityksen mukana (Felinto 2020). Geometrianodejen avulla voidaan muokata 3D-mallin ulkonäköä käyttämällä Geometry node editor -ikkunaa. 3D-mallia voi muokata erilaisilla nodeilla samalla tavalla, miten tekstuureita muokataan Shader editor -ikkunassa. Geometrianodet ovat muuntajien tavoin tuhoamattomia toimenpiteitä. Kuviossa 22 nähdään Blenderin tarjoama demokenttä. Demokentässä nähdään deformoitu taso, johon on

siroteltu erikokoisia kiviä. Kivet ovat siroitettu luomalla pistedatainstanssi Point distribute -nodella. Tämä node luo näkymättömiä pisteitä tasolle. Pisteiden hajontaa ohjataan Attribute randomize -nodella. Pistedatainstanssille kerrotaan käyttämään kiven näköisiä 3D-objekteja Point instance -nodella. Tämä node täyttää näkymättömät pisteet 3D-objekteilla. Hajontaa voi hienosäätää muuttamalla Attribute randomize -noden parametreja. Samat toimenpiteet on toistettu kuvissa kolmeen kertaan. Tällä tavalla saadaan kolme erikokoista ja erinäköistä kiveä asettumaan eri tavoin tasolle. Kivet asettuvat päällekkäin, jos kaikki niistä käyttävät samaa pistedatainstanssia.



Kuvio 22. Geometrianodet

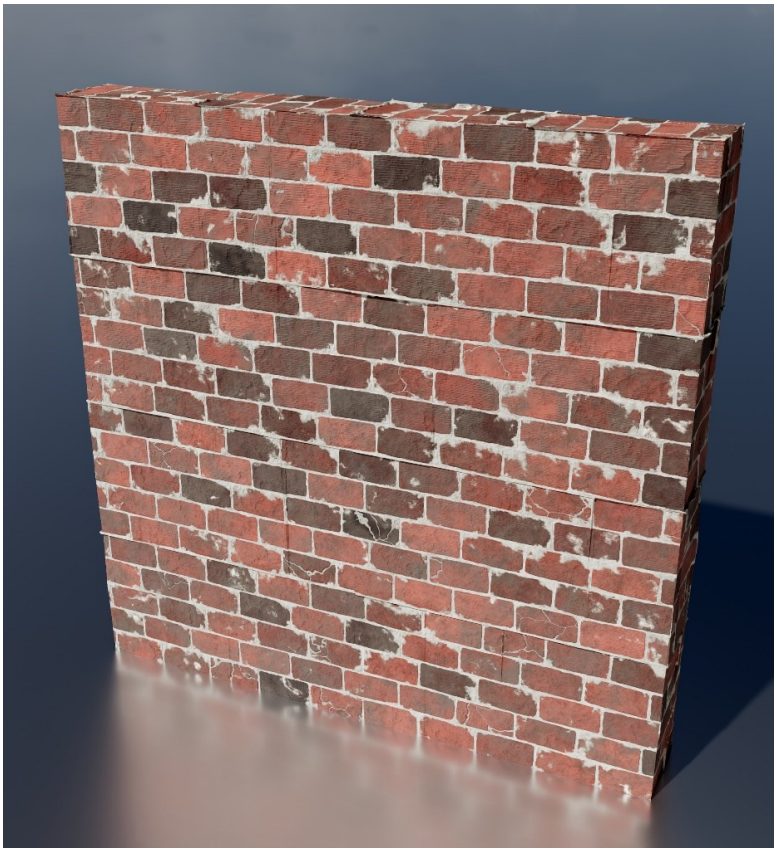
Demokentän lopputulos voidaan saavuttaa vaihtoehtoisesti perinteisellä menetelmällä, jossa käytetään nodejen sijaan Blenderin partikkelijärjestelmää. Tasolle luodaan oma partikkeli-instanssi, jossa määritellään partikkelien määrä ja miten partikkelit asettuvat tasolle. Partikkelit korvataan esimerkiksi kiven näköisillä 3D-objekteilla.

Blenderin partikkeleilla voi simuloida partikkelien vuorovaikutusta eri järjestelmien välillä. Partikkeleilla voi simuloida esimerkiksi erilaisia nestesimulaatioita, jossa partikkeleihin vaikuttaa painovoiman lisäksi muut partikkelit. Partikkelisimulaatio on raskaampi toimenpide verrattuna pistedataninstanssiin.

3.5 Proseduraalinen teksturointi

Proseduraalisessa teksturoinnissa käytetään generoituja textuureja. Generoituja tekstuureita ohjataan erilaisilla matemaattisilla ohjeilla. Tässä menetelmässä texture mapin resoluutiolla ja UV-mappauksella ei ole merkitystä, koska tekstuurit generoidaan joka kerta, kun käyttäjä renderoi. Tämä tarkoittaa, että generoitu tekstuuri skaalautuu äärettömästi. (McCombs 2010.)

Kuviossa 23 nähdään Simon Thommesin Br’cks -proseduraalinen tiilitekstuuri. Tekstuurissa on käytössä ainoastaan Blenderin sisäänrakennettuja noise-algoritmejä ja matemaattisia nodeja. Monimutkaisten ohjeiden ansioista tiilitekstuurin voi asettaa mihin 3D-malliin tahansa. (Thommes 2020.)



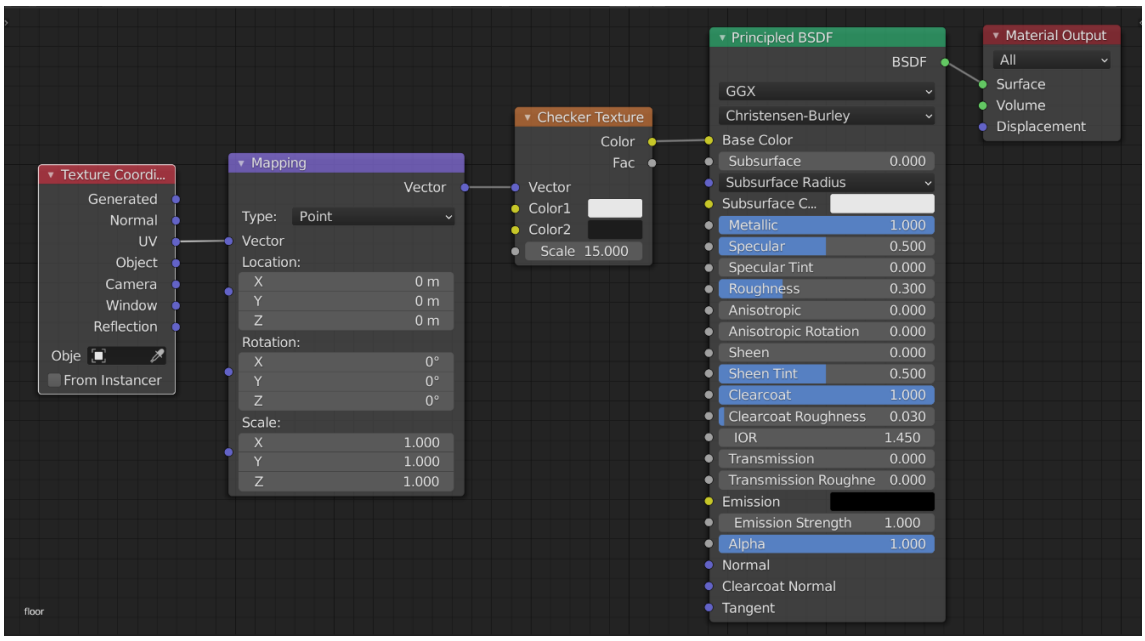
Kuvio 23. Simon Thommesin Br’cks -tiilitekstuuri

4 LISÄOSAN LUOMINEN BLENDERIIN

Blenderiä voi ohjelmoida Text editorilla, ja luotuja skriptejä voi ajaa Python consolessa. Molemmat moduulit löytyvät Blenderistä. Luodut skriptit tallentuvat .py-tiedostoina ja niitä voi asentaa muihin, vähintään Blender 2.80 -versioille. (Blender manual 2020f.)

Blenderissä jokaisella toiminnolla on oma luokka, jota voi kutsua. Asetuksista voi ottaa käyttöön Python tooltipsin ja Developer extrasin, jotka näyttävät toimintojen Blender Python -komennot. Tämän avulla on helppoa kirjoittaa toiminnallisuuksia omalle laajennukselle. Open source -luonteen takia Blenderistä löytyy muiden käyttäjien kirjoittamia laajennuksia, joita voi ottaa käyttöön. Oman laajennuksen lisääminen viralliseen julkaisuversioon edellyttää esimerkiksi kirjoittajan sitoutumista laajennuksen jatkuvaan ylläpitoon (Blender Manual 2020g).

Laajennuksien toiminnallisuus on yleensä tehty vähentämään toistuvia operaatioita ja helpottamaan erilaisia työnkulkuja. Esimerkiksi Blenderin sisäänrakennetulla Node Wrangler -laajennuksella pystyy muokkaamaan node-editorin nodeja tehokkaammin. Kuviossa 24 on luotu Texture Coordinates sekä Mapping-node valitsemalla Checker texture -node ja painamalla pikanäppäinkomentoa CTRL+T. Node Wrangleriin on kirjoitettu metodi *bpy.ops.node.nw_add_texture()*, jota kutsutaan aina, kun pikanäppäinkomentoa käytetään. Node Wranglerin metodi nw_add_texture() tarkistaa kutsuttaessa, että valittu node on oikean tyyppinen ja luo nodet oikeisiin paikkoihin oikeassa järjestyksessä kytkettynä.



Kuvio 24. Shader editor -ikkunan node-asetukset

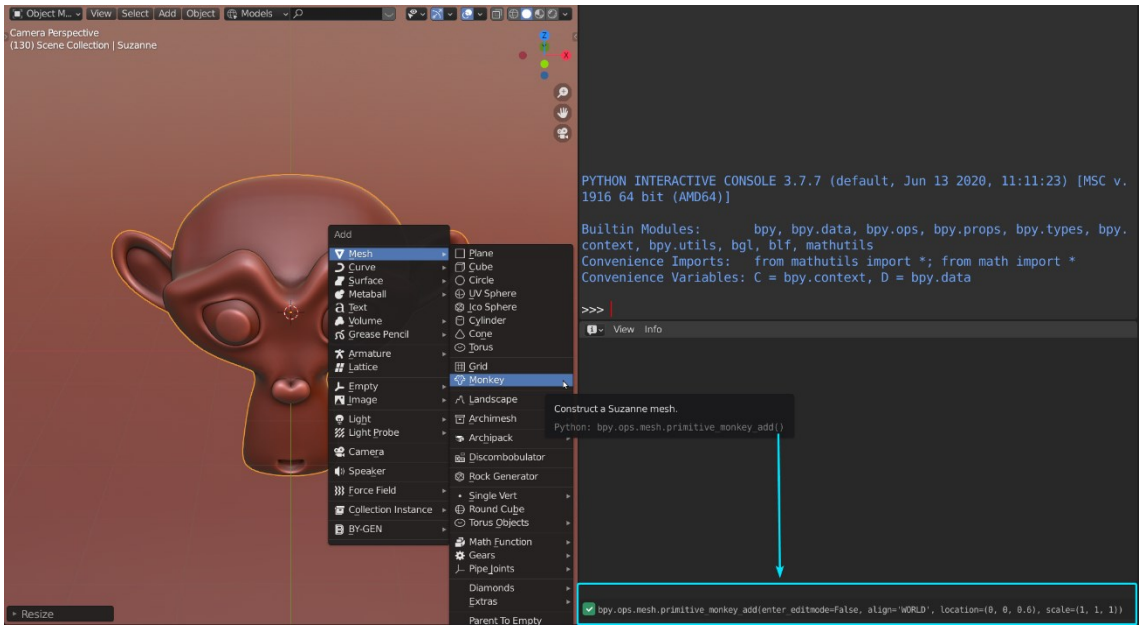
Opinnäytetyössä kehitetylle prototyypille annetaan työnimike O-Gen. Nimi saattaa muuttua julkaisuversiossa.

4.1 Ohjelmointirajapinta

Blender Pythonille on luotu oma ohjelmointirajapinta, jonka ansioista Blender-toimintojen ajaminen saadaan tehokkaammaksi. Ohjelmointirajapinnan moduuleita ovat muun muassa bpy, bpy.data, bpy.props, bpy.types, bpy.context sekä bpy.utils. Nämä moduulit ohjaavat kukin eri osapuolta Blenderissä. Esimerkiksi bpy.ops tarkoittaa Blender Python Operations, jolla suoritetaan erilaisia operaatioita, kuten objektin luomista tai muuntajien tai materiaalien lisäämistä objektiin. Bpy.data-moduulin komennoilla taas muokataan olemassa olevan objektin tietoja, kuten koordinaatteja, rotaatiota, kokoa 3D-kentässä tai materiaalin arvoja Shader editorissa. (Blender Manual 2020h.)

Kuviossa 25 nähdään, että 3D-malleja voi luoda painamalla pikanäppäinkomentoa Shift+A, jonka jälkeen valitaan listalta jokin objekti. Tässä esimerkissä valittiin listalta Monkey, joka muistuttaa apinan päätä. Python-komento näkyy vihjeikkunassa, kun käyttäjä tuo hiiren päälle. Blender ajaa Python-konsolissa komennon, kun käyttäjä valitsee listalta Monkeyn. Tämä tarkoittaa sitä, että kaikki käyttäjän tekemät toiminnot kutsuvat komentoa, jota ajetaan konsolissa. Käyttäjä pystyy

myös tarkentamaan suoraan komentorivin parametreihin, esimerkiksi mihin pää luodaan ja kuinka suuri pää on. Eri ikkunoiden ja asetuksien valitseminen ajaa myös komentoja konsolissa.

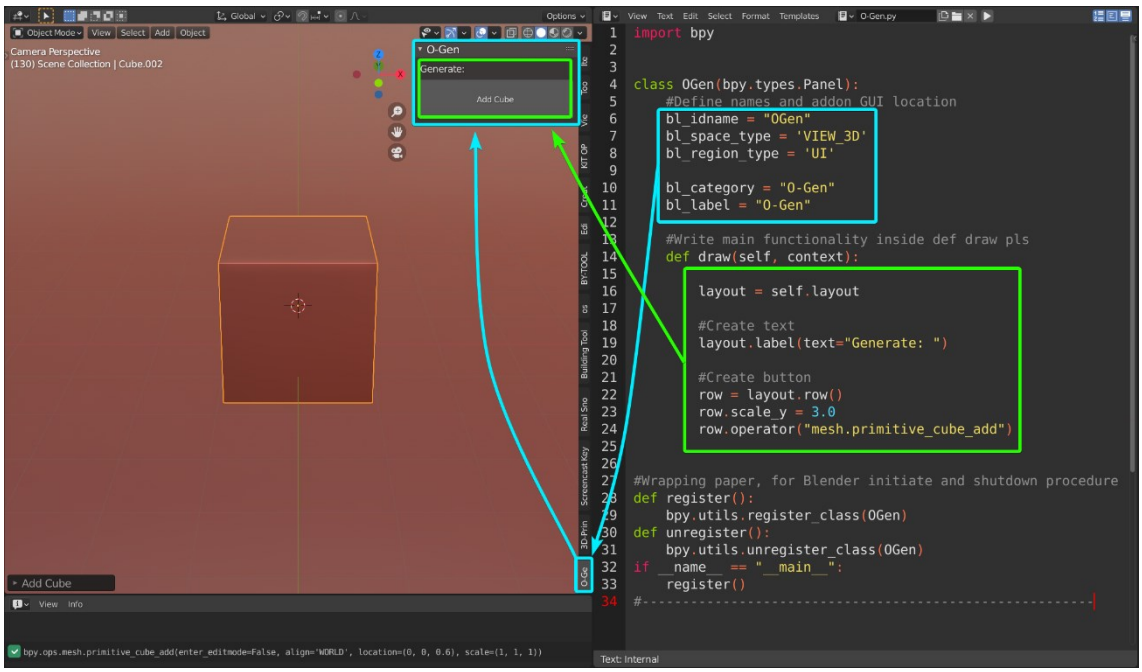


Kuvio 25. 3D-objektin luominen graafista ikkunaa tai Python-konsolia käyttäen

4.2 Graafinen käyttöliittymä

Blenderin tasaisen käyttöliittymän takia kaikki ikkunat tai moduulit jakavat saman tason näytössä, ja käyttäjä pääsee rajaamaan ja muuttamaan työtiloja oman mielensä mukaan. Blender tarjoaa ohjelmoijille erilaisia kohteita, minne voi upottaa oman laajennuksen tai toiminnon. Yleisin kohde kolmannen osapuolen laajennuksille on 3D-mallinnusikkunan N-sivupaneeli. O-Genin graafinen käyttöliittymä kirjoitetaan myös 3D-mallinnusikkunan N-sivupaneeliin jatkuvuuden takia. Graafinen käyttöliittymä helpottaa myös laajennuksen käyttöä, koska käyttäjän ei tarvitse ajaa toimintoja Python-konsolin kautta.

Laajennus löytyy 3D-Print-työkalun alapuolella (Kuvio 26). O-Gen-kategoriaa valitsemalla se saadaan avattua esille. Laajennuksen varsinaiset asetukset sekä toiminnot näkyvät 3D-mallinnusikkunan oikealla yläpuolella. Laajennuksen toiminnallisuudet toteutetaan pääasiassa painikkeina, kytkiminä ja liukusäätiminä, jotka määritellään def drawin alle. Toiminnallisuuksille luodaan omat funktiot, jotka sidotaan johonkin painikkeeseen tai säätimeen. Tällä tavalla käyttäjä kutsuu nappia painamalla siihen sidottua toimintoa, joka suorittaa tietyn operaation.



Kuvio 26. Laajennuksen upottaminen graafiseen käyttöliittymään

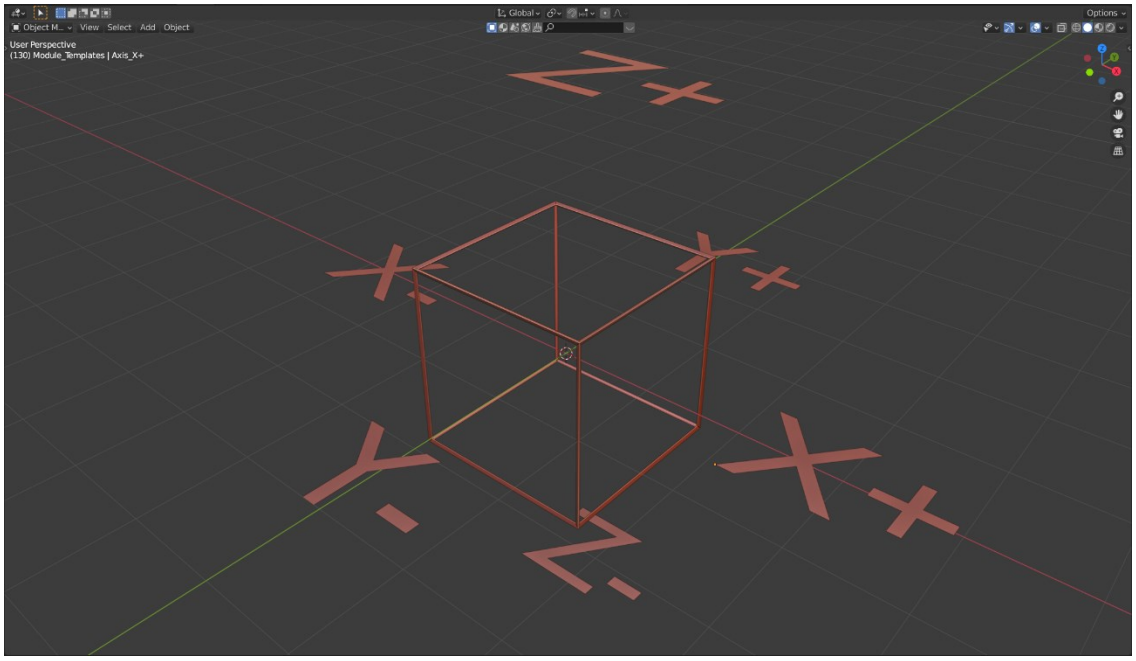
4.3 Generointilogiikka

O-Genin generointilogiikka perustuu tileset-menetelmään, jota on käytetty esimerkiksi 2D-videopelien maaston generoinnissa. Tileset-menetelmässä voi maalata ruudukkopohjaiseen pelikenttään maaston osia. Käyttäjä voi esimerkiksi asettaa ruudukkoihin hiekkatietä muistuttavan laatan. Ohjelma on voitu ohjelmoida täyttämään hiekkatielaattojen vieressä olevat tyhjät ruudut ruoholaatoilla tai muulla maastolla.

O-Gen luo kolmiulotteisen ruudukkojärjestelmän eli kuutioverkoston. Yksittäistä osaa kuutioverkostosta kutsutaan soluksi. Kuutioverkoston ensisijainen tarkoitus on ohjata generoitavien 3D-mallien sijoitusta sekä auttaa generoidun solun paikannusta suhteessa muihin soluihin. Solujen paikannusprosessia kutsutaan naapurustotarkistukseksi. Kuutioverkoston ja naapurustotarkistuksen takia ei tarvita törmäyksenhavaitsemisjärjestelmää, jossa varmistetaan, että generoidut 3D-mallit eivät mene päällekkäin. Päällekkäisyydet tarkistetaan vertaamalla olemassa olevien solujen koordinaatteja toisiinsa. Tämän lisäksi lineaarisempi prosessi avaa mahdollisuuden helpompaan jatkokehitysprosessiin.

Kuutioverkoston solulle annetaan tilavuudeksi kaksi kuutiometriä. Solun orientaatio asetetaan seuraavasti: Y+ on eteenpäin, Y- on taaksepäin, X+ on oikealle, X-

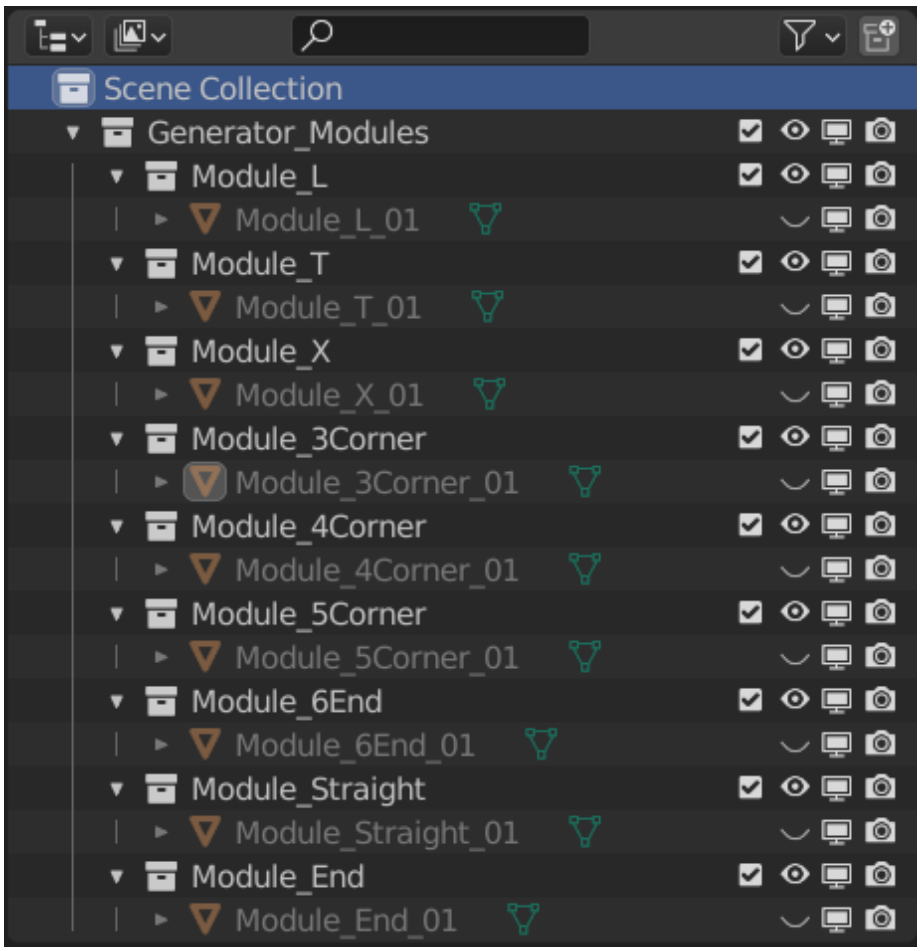
on vasemmalle, Z+ on ylöspäin, Z- on alaspäin. Kuviossa 27 esiintyy runkorakenne, joka kuvaa yhden solun rajoja. Jokainen generoitava modulaarinen 3D-malli tulee mallintaa siten, että se mahtuu solun rajojen sisään. Runkorakenne ja akselimerkinnät ovat näkymättömiä lopullisessa versiossa.



Kuvio 27. Solun orientaatio sekä tilavuus

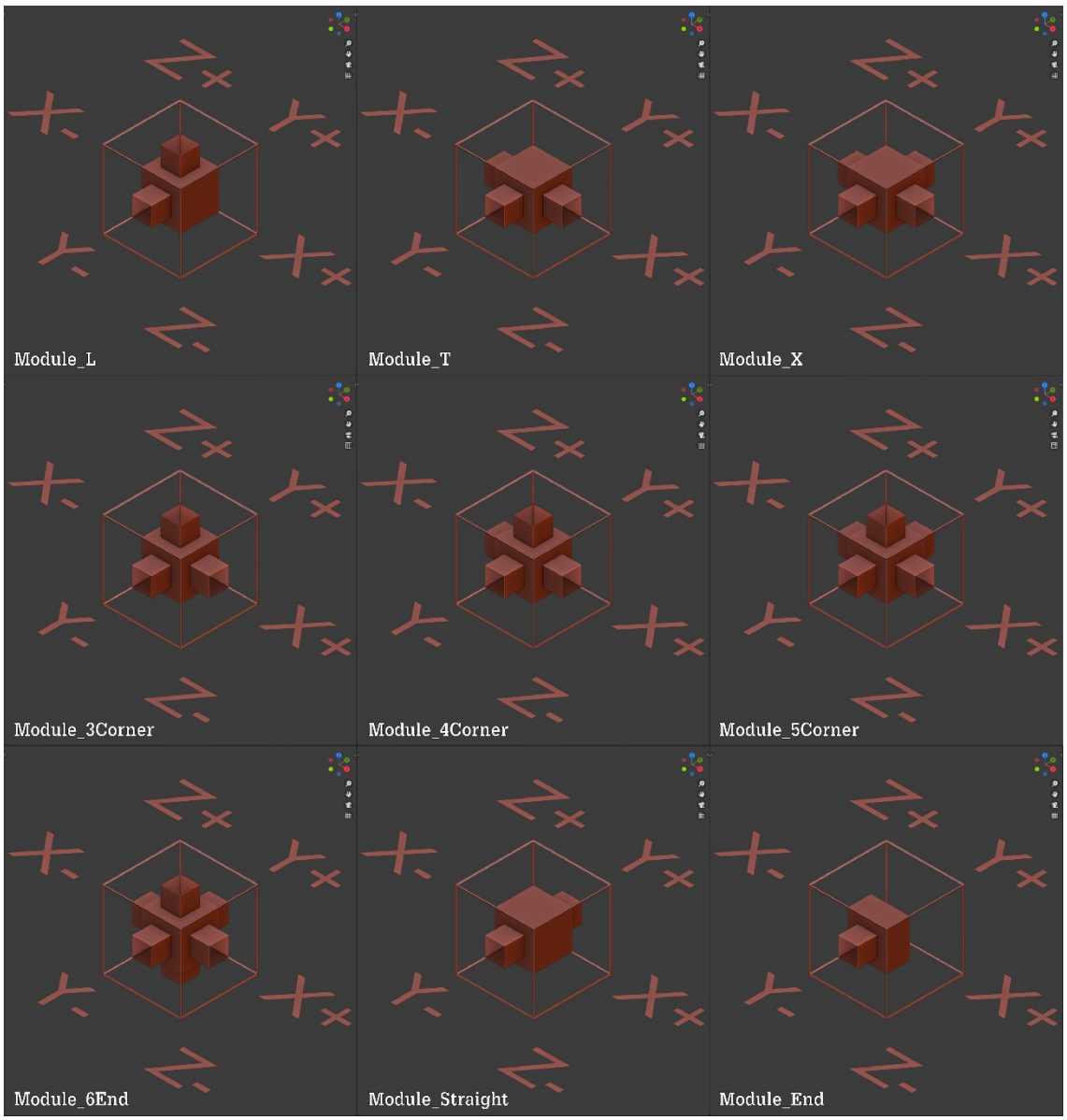
4.4 Modulaariset 3D-mallit

Laajennuksessa käytetään hyväksi uudistettua 3D-mallien hallintajärjestelmää Collections eli kokoelmia, jotka tulivat Blender 2.80 -päivityksen mukana. Kokoelmien avulla 3D-malleja voi kategorisoida niiden muotojen mukaan. Kuviossa 28 nähdään, että esimerkiksi kaikki 90 astetta kääntyvät kappaleet laitetaan Module_L-kokoelmaan ja kaikki suorat kappaleet Module_Straight-kokoelmaan. Visuaalisen toistuvuuden estämiseksi voidaan mallintaa esimerkiksi useampi 3D-malli jokaiseen kokoelmaan. Moduulit nimetään kokoelmatyypin mukaan, ja numerokoodi erottaa saman kokoelman moduulit toisistaan.



Kuvio 28. Kokoelmat

Kuviossa 29 nähdään työkalun alustavat moduulit. Näitä on yhteensä yhdeksän, jotta jokaiseen suuntaan on mahdollista kääntyä tai haarautua. Alustavissa moduuleissa on tuotu visuaalisesti esille niiden käyttötarkoitus pullistamalla keskirakenteet ja ohentamalla liitoskohdat. Tämän avulla voidaan tarkastella generoinnissa, osaako algoritmi asettaa ja kääntää moduulit oikein. Yksi näistä moduuleista on käytävän päättävä umpikujamoduuli. Näin varmistetaan, että käytävä ei lopu kesken. Laajennuksen käyttäjillä on mahdollisuus parantaa tai korvata moduulien ulkonäköä sekä tehdä vapaasti kokonaan uusia moduuleja kokoelmiin.



Kuvio 29. Alustavat moduulit

4.5 Kuutioverkoston Generointialgoritmi

Virtuaalisen kuutioverkoston ensimmäinen solu generoidaan aina 3D-kentän origoon, minkä jälkeen valitaan satunnaisesti akseli sekä suunta, eli X/Y/Z ja +/-. Ensimmäinen solu ja sille valitut suuntatiedot muodostavat yhden haaran. Yhden haaran pituus valitaan satunnaisesti branchMin- ja branchMax-arvojen väliltä. Haaraan generoidaan soluja arvottuun suuntaan, kunnes arvottu pituus saavutetaan. Jokaisella generoidulla solulla on myös mahdollisuus haarautua. Haarautu-

mistodennäköisyys riippuu käyttäjän syöttämästä chanceLevel-arvosta. Haarautuvien solujen koordinaatit tallennetaan muuttujaan branchList. Kaikki generoitujen solujen koordinaatit tallennetaan muuttujaan cellList.

Haarautumisen tapauksessa solulle valitaan uudestaan satunnaisesti akseli ja suunta, johon uusi haara kasvaa. Uusi suunta voi olla kaikki paitsi edellisen haaran eteenpäin-suunta. Uudelle haaralle arvotaan myös oma maksimipituus. Solujen ja haarojen generointi jatkuu niin pitkään, kunnes käyttäjän määrittämä maxModules-arvo saavutetaan. MaxModules-arvon saavuttamisen jälkeen generointiprosessi loppuu.

Generointia ohjaa pääasiassa funktiot generate_cells ja populate_branch sekä prosessit neighbour_check ja module_conditions. Skripteissä käytetään yleensä kolme muuttujaa. Yksi muuttuja alkuperäiselle arvolle ja toinen muuttuja iteraatioita varten, joka korvaa lopussa alkuperäisen muuttujan. Joissain tapauksissa tarvitaan kolmas muuttuja iteraatiomuuttujan argumentiksi.

4.5.1 Generate_cells-funktio

Generate_cells-funktio kutsutaan skriptin alussa. Funktio tarkistaa ensimmäisenä, onko maxModules-arvo saavutettu ja jatkaa, jos arvoa ei ole saavutettu. Funktio asettaa start_procedure-arvoon TRUE ja tallentaa current_cell-muuttujaan 3D-kentän origon koordinaatit. Funktio tallentaa kaikki generoidut solut cellList-muuttujaan ja lisää cell_count-arvoa yhdellä. (Kuvio 30.)

Muuttujaan branchLenght tallennetaan satunnaisesti valittu int-arvo branchMin- ja branchMax-arvojen väliltä. Tämän jälkeen valitaan myös satunnaisesti akseli. AxisPick-muuttujaan tallennetaan arvot ‘X’, ‘Y’, ‘Z’, jotka vastaavat numeroita samassa järjestyksessä ‘0’, ‘1’, ‘2’. PickAxis-muuttujalla valitaan satunnainen numero väliltä 0–2. Valittu numero muuntuu kirjaimeksi, kun se tallennetaan chosenAxis-muuttujaan parametrina.

Sama toimenpide tehdään akselin suunnan valitsemiselle. DirectionPick-muuttujaan tallennetaan arvot ‘POS’, ‘NEG’, jotka tarkoittavat akselin positiivista tai negatiivista suuntaa. PickDir valitsee sitten satunnaisen arvon väliltä 0–1 ja tallentaa valitun arvon chosenDir-muuttujaan.

Ensimmäisen solun generoinnin jälkeen start_procedure-muuttuja asetetaan arvoon FALSE, ettei laajennus luo uutta ensimmäistä solua. Seuraavaksi funktio kutsuu funktiota populate_branch, jossa generoidaan loput solut.

```
def generate_cells():
    global maxModules
    global branchMin
    global branchMax
    global chanceLevel
    global cell_count
    global start_procedure
    global cellList
    global branchList
    global branchRef

    #Test
    #global extDict
    #global portBin

    #Maxmimum cell count has been reached
    if cell_count >= maxModules:
        print("Ended generation due to maxModules reached.")
    #Maximum cell count has not been reached
    else:
        #If this is the first branch
        if start_procedure == True:
            current_cell = Vector((0,0,0))

            #Add the origin cell as a starting point and increment the total cell count.
            cellList.append(Vector((current_cell[0],current_cell[1],current_cell[2])))
            cell_count += 1

            # Get [x] rand from (branchMin, branchMax)
            branchLength = random.randint(branchMin,branchMax)

            # Pick random axis, X/Y/Z [0]/[1]/[2]
            axisPick = ['X','Y','Z']
            pickAxis = random.randint(0,2)
            chosenAxis = axisPick[pickAxis]

            # Pick random direction, positive [+] or negative [-]
            directionPick = ['POS', 'NEG']
            pickDir = random.randint(0,1)
            chosenDir = directionPick[pickDir]

            #Change the boolen start_procedure
            start_procedure = False

            #Call Populate Branch
            populate_branch(current_cell, branchLength, chosenAxis, chosenDir)
            generate_cells()
```

Kuvio 30. Generate_cells, osa 1

4.5.2 Populate_branch-funktio

Populate_branch luo generate_cellsin ensimmäisen solun ohjeiden mukaan haaraan loput solut. Funktio tarkistaa alussa jälleen, onko maxModules-arvo saavutettu. Tämän jälkeen haetaan branchLenght, jotta saadaan haaran maksimiarvo ja luodaan while-silmukka. While-silmukka on tosi, kunnes branchLenght-arvo

saavutetaan. Tässä funktiossa tallennetaan uudet vektoritiedot muuttujaan new_cell, johon tehdään myös tarvittavat muutokset. Muutoksien jälkeen new_cell-muuttujan tiedot korvaavat current_cell-muuttujan tiedoilla, jotka tallennetaan cellListiin. (Kuvio 31.)

While-silmukassa generoidaan haaraan solut. Jokaista solua liikutetaan move_offsetin verran, jonka käyttäjä on määrittänyt. Move_offsetin oletusarvo on 2, koska solun oletustilavuus on kaksi kuutiometriä. Esimerkiksi jos haaran suunnaksi valitaan X+, niin new_cell-muuttujan vektoritietoihin [0] lisätään move_offset. Samalla tavalla, jos haaran suunnaksi valitaan Z-, niin new_cell-muuttujan vektoritiedoista [2] vähennetään move_offset.

```
def populate_branch(current_cell, branchLength, chosenAxis, chosenDir):
    global maxModules
    global branchMin
    global branchMax
    global chanceLevel
    global move_offset
    global cell_count
    global cellList
    global branchList
    global branchRef

    #Test
    #global extDict
    #global portBin

    if cell_count < maxModules:
        print ("Branch Length: "+str(branchLength))
        #For the length of the branch
        i = 1
        while i <= branchLength:
            #increment current cell using axis and direction
            new_cell = Vector((current_cell[0],current_cell[1],current_cell[2]))
            if chosenAxis == 'X':
                if chosenDir == 'POS':
                    new_cell[0] += move_offset
                if chosenDir == 'NEG':
                    new_cell[0] -= move_offset
            if chosenAxis == 'Y':
                if chosenDir == 'POS':
                    new_cell[1] += move_offset
                if chosenDir == 'NEG':
                    new_cell[1] -= move_offset
            if chosenAxis == 'Z':
                if chosenDir == 'POS':
                    new_cell[2] += move_offset
                if chosenDir == 'NEG':
                    new_cell[2] -= move_offset
```

Kuvio 31. Populate_branch, osa 1

Jokaista luotua solua kohden tarkistetaan, ettei solu ole päällekkäin toisen solun kanssa ja arvotaan, haarautuuko solu. Tämä toteutetaan tallentamalla senhetkisen solun tiedot eli new_cell-muuttujan tiedot newVector-muuttujaan ja tarkistetaan, löytyvätkö vektoritietoja cellList-muuttujasta. Funktio jatkaa tallentamalla senhetkisen solun cellListiin, jos sitä ei löytynyt muuttujasta. (Kuvio 32.)

Solun haarautumista varten tallennetaan branchChance-muuttujaan satunnaisesti valittu arvo väliltä 1–100. Haarautuminen tapahtuu, jos valittu arvo on pienempi, kuin käyttäjän asettama arvo. Tällä tavalla käyttäjän syöttämä kokonaisluku vastaa prosenttiyksikköä. Haarautumisen tapauksessa valitaan solulle uusi akseli ja suunta, johon uusi haara kasvaa. Skriptissä on ehto, jossa eliminoidaan tapauskohtaisesti tietyt akselit. Esimerkiksi jos haaran alkuperäinen suunta on X-akseli, niin mahdolliset suunnat, johon uusi haara kasvaa on Y+, Y- tai Z+ ja Z-. Samalla tavalla, jos haaran alkuperäinen suunta on Z-akseli, niin mahdolliset suunnat, johon uusi haara kasvaa on X+, X- tai Y+ ja Y-. Uusi suunta arvotaan luomalla muuttuja axisList, johon lisätään mahdolliset akselit ja suunnat. AxisList-muuttujaan tallennetut neljä mahdollista arvoa sekoitetaan ja arvo valitaan satunnaisesti. Branch_entry-muuttujaan tallennetaan lopullisen uuden haaran tiedot muodossa koordinaatit/akseli:suunta eli esimerkiksi 0,1,2/X:POS. Tämä lisätään lopulta muuttujaan branchList.

Funktio nollaa muuttujat ja inkrementoi silmukan yhdellä riippumatta siitä, tapahtuuko haarautumista vai ei. Näin varmistetaan, että seuraavan solun generoinnissa ei tallenneta vääriä tietoja vääriin muuttujiin.

```
            #Check for overlap
            newVector = Vector((new_cell[0],new_cell[1],new_cell[2]))
            if newVector not in cellList:

                #Record current_cell in cellList
                cellList.append(Vector((new_cell[0],new_cell[1],new_cell[2])))
                cell_count += 1
                current_cell = new_cell

                #Get branchChance between 1,100
                branchChance = random.randint(1,100)
                if branchChance <= chanceLevel:

                    #randAxis = rand between 1,4
                    randAxis = random.randint(1,4)
                    axisList = []

                    #if chosenAxis is 'X':
                    if chosenAxis == 'X':
                        #axisList add Y+ Y- Z+ Z-
                        axisList.append("Y:POS")
                        axisList.append("Y:NEG")
                        axisList.append("Z:POS")
                        axisList.append("Z:NEG")
                    #if chosenAxis is 'Y':
                    if chosenAxis == 'Y':
                        #axisList add X+ X- Z+ Z-
                        axisList.append("X:POS")
                        axisList.append("X:NEG")
                        axisList.append("Z:POS")
                        axisList.append("Z:NEG")
                    #if chosenAxis is 'Z':
                    if chosenAxis == 'Z':
                        #axisList is X+ X- Y+ Y-
                        axisList.append("X:POS")
                        axisList.append("X:NEG")
                        axisList.append("Y:POS")
                        axisList.append("Y:NEG")

                    #For randAxis:
                    j = 1
                    while j < randAxis:
                        random.shuffle(axisList)
                        cell_key = str(new_cell[0])+","+str(new_cell[1])+","+str(new_cell[2])

                        branch_entry = cell_key + "/" + axisList[0]
                        #Store new branch starting point in branchList
                        branchList.append(branch_entry)

                        #remove it from axisList
                        axisList.pop(0)
                        #Increment loop
                        j += i
            current_cell = Vector((new_cell[0], new_cell[1], new_cell[2]))
            #Increment Loop
            i += 1
```

Kuvio 32. Populate_branch, osa 2

Start_proceduren FALSE-arvon ansiosta funktio ei luo uudestaan ensimmäistä solua ja jatkaa suoraan else-ehtoon. Else-ehdossa tarkistetaan branchList-muuttujasta solut, jotka ovat haarautuvia ja täydentävät niiden haarat kutsumalla populate_branch-funktiota. Skripti vaihtelee koko kuutioverkoston generoinnin aikana generate_cells sekä populate_branch -funktioiden välillä, kunnes maxModules-arvo saavutetaan. (Kuvio 33.)

```
            #Change the boolen start_procedure
            start_procedure = False

            #Call Populate Branch
            populate_branch(current_cell, branchLength, chosenAxis, chosenDir)
            generate_cells()

        else:
            #Try to kick off new branch loop
            if len(branchList) > 0:
                branchRef = []
                branchRef.extend(branchList)
                for newBranch in branchRef:
                    if cell_count < maxModules:
                        #Get axis and direction from newBranch
                        cell_string, axis_dir = newBranch.split("/",1)
                        chosenAxis, chosenDir = axis_dir.split(":",1)

                        #Get Vector from newBranch
                        cell_x, cell_y, cell_z = cell_string.split(",")
                        new_cell = Vector((float(cell_x),float(cell_y),float(cell_z)))

                        # Get [x] rand from (branchMin, branchMax)
                        branchLength = random.randint(branchMin,branchMax)


                        #if it works but ain't bumb, it ain't dumb
                        if new_cell in cellList:
                            #Call Populate Branch
                            populate_branch(new_cell, branchLength, chosenAxis, chosenDir)
                    else:
                        break
                #Now loop is done, remove from branchList
                for element in branchRef:
                    branchList.remove(element)
                generate_cells()
```

Kuvio 33. Generate_cells, osa 2

Kuutioverkoston luomisen jälkeen siirrytään varsinaisesti 3D-mallien generointiin. 3D-kentässä on tällä hetkellä näkymättömiä kahden kuutiometrin kokoisia soluja, joiden koordinaatit ovat tallennettu. Seuraavaksi skripti siirtyy naapurustotarkistukseen, jossa verrataan vierekkäisiä soluja toisiinsa. Tämän perusteella laajennus asettaa oikeanlaisen 3D-mallin soluihin.

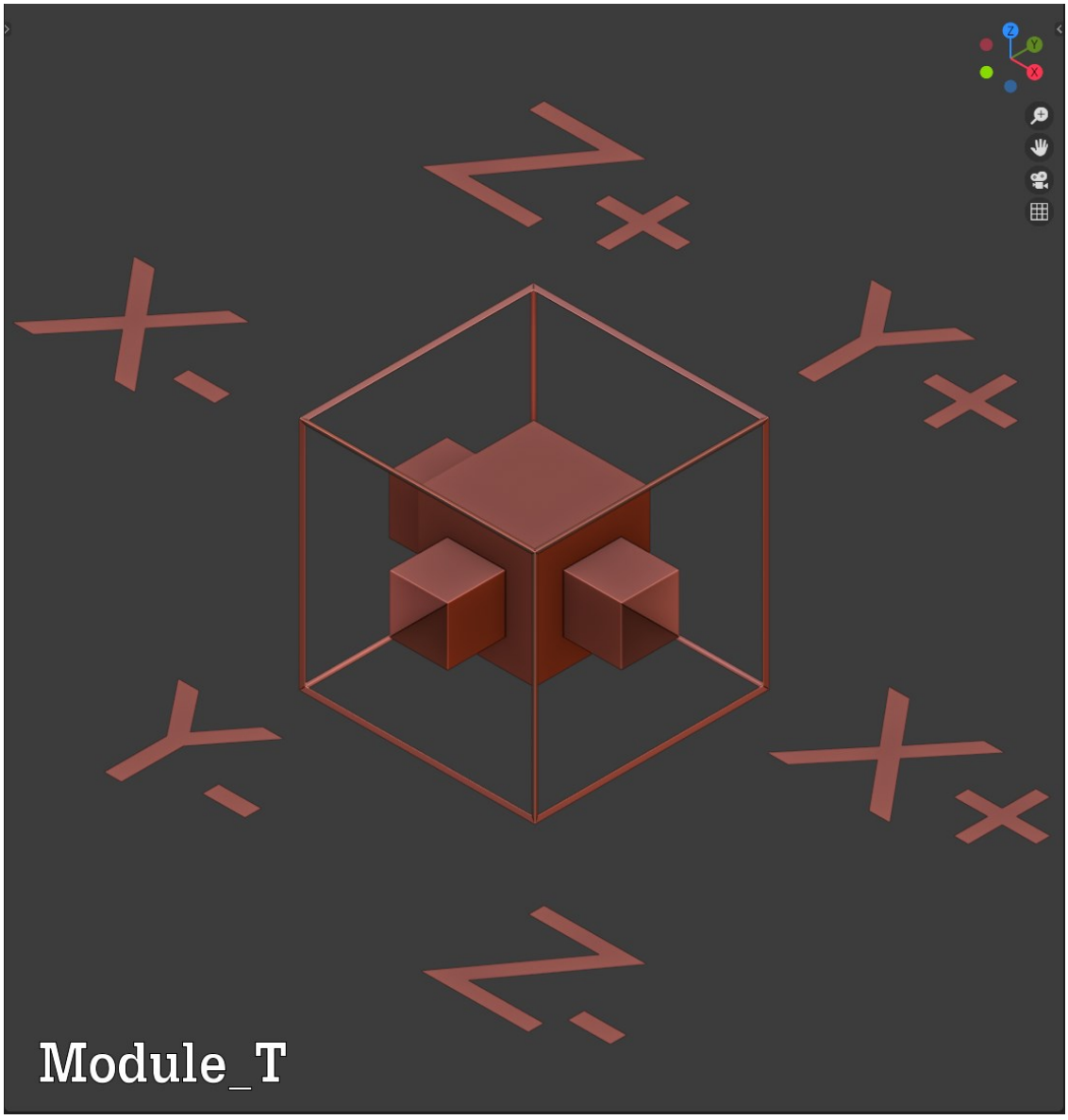
4.5.3 Neighbour_check-prosessi

Naapurustotarkistus toteutetaan kuuden bitin binäärimerkkijonolla. Merkkijonoon merkataan 0, jos naapurissa ei ole solua. Merkkijonoon merkataan 1, jos naapurissa on solu. Esimerkiksi jos solun ainoa naapurisolu on suuntaan Y+, niin soluun merkataan binäärimerkkijonoksi 001000. (Kaavio 1.)

Kaavio 1. Naapurustotarkistuksen binäärimerkkijonon selite

0	0	1	0	0	0
X+	X-	Y+	Y-	Z+	Z-

Esimerkiksi solu, jolla on kolme naapurisolua, vaatii 3D-mallin kokoelmasta Module_T tai Module_3Corner. Binäärimerkkijonon perusteella valitaan oikea 3D-malli, joka käännetään oikean suuntaiseksi. Kuviossa 34 nähdään solu, jossa on suunnissa X+, X- ja Y- naapurisolu. Tämän solun binäärimerkkijono on 110100, jonka mukaan valittiin Module_T-kokoelmasta 3D-malli täyttämään solu.



Kuvio 34. Naapurustotarkistuksen logiikka

Kuvion 35 koodi sijaitsee pääluokan O-Genin alla generate_cells()-funktion kutsun jälkeen. Koodissa määritetään naapurustotarkistuksen eli binäärimerkkijonon logiikka. Binäärimerkkijonon arvo määritetään tarkistamalla cellList-muuttujasta naapurisolun vektoritiedot.

```
        for active_cell in cellList:
            #Neighbour check on current cell
            is_xpos = is_xneg = False
            is_ypos = is_yneg = False
            is_zpos = is_zneg = False

            #Prepare binary to store neighbour check data
            nID = ""

            #X Positive
            check_vector = Vector((active_cell[0], active_cell[1], active_cell[2]))
            check_vector[0] += move_offset
            if check_vector in cellList:
                is_xpos = True
                nID = nID + "1"
            else:
                is_xpos = False
                nID = nID + "0"
            #X Negative
            check_vector = Vector((active_cell[0],active_cell[1],active_cell[2]))
            check_vector[0] -= move_offset
            if check_vector in cellList:
                is_xneg = True
                nID = nID + "1"
            else:
                is_xneg = False
                nID = nID + "0"

            #Y Positive
            check_vector = Vector((active_cell[0],active_cell[1],active_cell[2]))
            check_vector[1] += move_offset
            if check_vector in cellList:
                is_ypos = True
                nID = nID + "1"
            else:
                is_ypos = False
                nID = nID + "0"
            #Y Negative
            check_vector = Vector((active_cell[0],active_cell[1],active_cell[2]))
            check_vector[1] -= move_offset
            if check_vector in cellList:
                is_yneg = True
                nID = nID + "1"
            else:
                is_yneg = False
                nID = nID + "0"

            #Z Positive
            check_vector = Vector((active_cell[0],active_cell[1],active_cell[2]))
            check_vector[2] += move_offset
            if check_vector in cellList:
                is_zpos = True
                nID = nID + "1"
            else:
                is_zpos = False
                nID = nID + "0"
            #Z Negative
            check_vector = Vector((active_cell[0],active_cell[1],active_cell[2]))
            check_vector[2] -= move_offset
            if check_vector in cellList:
                is_zneg = True
                nID = nID + "1"
            else:
                is_zneg = False
                nID = nID + "0"

            #Decide which module to spawn
            new_module = None
```

Kuvio 35. Neighbour_check

4.5.4 Module_conditions-prosessi

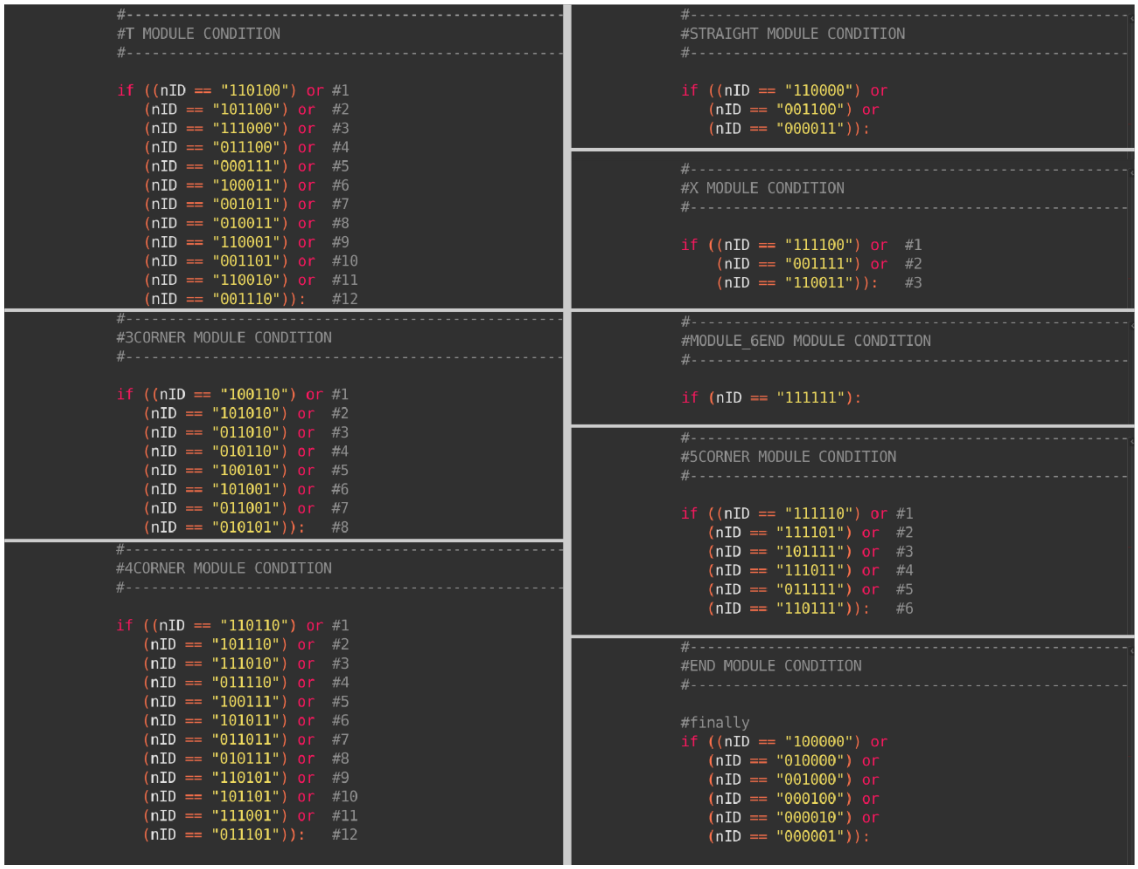
Naapurustotarkistuksen jälkeen jokaisella solulla on oma binäärimerkkijono, jonka mukaan valitaan oikea 3D-malli. Jokaiselle kokoelmalle määritetään ehdot, joiden mukaan 3D-mallit valitaan. Kuviossa 36 nähdään Module_T-kokoelman ehdot 3D-malleille. Module_T-kokoelman 3D-malli valitaan, jos solun binäärimerkkijono vastaa sen kokoelman mahdollisia vaihtoehtoja. Module_T-kokoelmalla on 12 mahdollista vaihtoehtoa, koska sen kokoelman 3D-mallien on mahdollista asettua 3D-kenttään 12 eri tavalla. Esimerkiksi jos solun binäärimerkkijono on 101100, niin 3D-malli valitaan kokoelmasta Module_T ja valittu malli asetetaan soluun siten, että sitä käännetään +90-astetta Z-akselissa.

Laajennus osaa valita myös satunnaisesti 3D-mallin kokoelmasta, jos kokoelmassa on enemmän kuin yksi 3D-malli. Tämän takia saman kokoelman alla sijaitsevat 3D-mallit nimetään eri numeropäätteellä. Kokoelmien 3D-mallit sijaitsevat oletuksena 3D-kentän origossa. Niiden näkyvyys on asetettu näkymättömäksi, kun taas Generator_Result-kokoelman näkyvyys on näkyvä. Tämän takia 3D-mallin valitsemisen jälkeen objekti kopioidaan kokoelmaan Generator_Result ja sille asetetaan senhetkisen solun koordinaatit. Tämän prosessin avulla asetetut objektit ovat aina näkyviä ja sijaitsevat oikeassa paikassa 3D-kentällä.

```
#-------------------------------------------------------------------------
#T MODULE CONDITION
#-------------------------------------------------------------------------

if ((nID == "110100") or #1
   (nID == "101100") or  #2
   (nID == "111000") or  #3
   (nID == "011100") or  #4
   (nID == "000111") or  #5
   (nID == "100011") or  #6
   (nID == "001011") or  #7
   (nID == "010011") or  #8
   (nID == "110001") or  #9
   (nID == "001101") or  #10
   (nID == "110010") or  #11
   (nID == "001110")):   #12

    #Get list of modules in Module_T collection
    object_names = []
    for obj_ref in module_t.objects:
        if 'pos' not in obj_ref.name:
            object_names.append(obj_ref.name)

    #select random object inside collection
    if len(object_names) > 0:
        randID = random.randint(0, len(object_names)-1)
        new_obj = module_t.objects[object_names[randID]]

        #create the object to cell if eligible
        new_module = new_obj.copy()
        new_module.data = new_obj.data.copy()
        #just in case, clear object keyframes
        new_module.animation_data_clear()

        #Link created object to Generation Result collection
        generation_result.objects.link(new_module)
        #Change visibility
        new_module.hide_viewport = False
        new_module.hide_render = False

        #Move object to the current cell
        new_module.location = active_cell

        #Check for rotation
        if nID == "110100": #1
            #Same as original, do nothing
            old_euler = new_module.rotation_euler
            old_euler[2] += 0
            new_module.rotation_euler = old_euler

        if nID == "101100": #2
            #X+ Y- Y+, 90+ Z
            old_euler = new_module.rotation_euler
            old_euler[2] += r90
            new_module.rotation_euler = old_euler
```

Kuvio 36. Module_conditions, osa 1

Kuviossa 37 nähdään jokaisen kokoelman ehdot. Module_6End kokoelman ainoa vaihtoehto on 111111, sillä tämän kokoelman 3D-mallin voi asettaa vain, jos jokaisessa suunnassa on naapurisolu.



```
#----------------------------------------------------
#T MODULE CONDITION
#----------------------------------------------------

if ((nID == "110100") or #1
   (nID == "101100") or  #2
   (nID == "111000") or  #3
   (nID == "011100") or  #4
   (nID == "000111") or  #5
   (nID == "100011") or  #6
   (nID == "001011") or  #7
   (nID == "010011") or  #8
   (nID == "110001") or  #9
   (nID == "001101") or  #10
   (nID == "110010") or  #11
   (nID == "001110")):   #12
```

```
#----------------------------------------------------
#3CORNER MODULE CONDITION
#----------------------------------------------------

if ((nID == "100110") or #1
   (nID == "101010") or  #2
   (nID == "011010") or  #3
   (nID == "010110") or  #4
   (nID == "100101") or  #5
   (nID == "101001") or  #6
   (nID == "011001") or  #7
   (nID == "010101")):   #8
```

```
#----------------------------------------------------
#4CORNER MODULE CONDITION
#----------------------------------------------------

if ((nID == "110110") or #1
   (nID == "101110") or  #2
   (nID == "111010") or  #3
   (nID == "011110") or  #4
   (nID == "100111") or  #5
   (nID == "101011") or  #6
   (nID == "011011") or  #7
   (nID == "010111") or  #8
   (nID == "110101") or  #9
   (nID == "101101") or  #10
   (nID == "111001") or  #11
   (nID == "011101")):   #12
```

```
#----------------------------------------------------
#STRAIGHT MODULE CONDITION
#----------------------------------------------------

if ((nID == "110000") or
   (nID == "001100") or
   (nID == "000011")):
```

```
#----------------------------------------------------
#X MODULE CONDITION
#----------------------------------------------------

if ((nID == "111100") or  #1
    (nID == "001111") or  #2
    (nID == "110011")):   #3
```

```
#----------------------------------------------------
#MODULE_6END MODULE CONDITION
#----------------------------------------------------

if (nID == "111111"):
```

```
#----------------------------------------------------
#5CORNER MODULE CONDITION
#----------------------------------------------------

if ((nID == "111110") or #1
   (nID == "111101") or  #2
   (nID == "101111") or  #3
   (nID == "111011") or  #4
   (nID == "011111") or  #5
   (nID == "110111")):   #6
```

```
#----------------------------------------------------
#END MODULE CONDITION
#----------------------------------------------------

#finally
if ((nID == "100000") or
   (nID == "010000") or
   (nID == "001000") or
   (nID == "000100") or
   (nID == "000010") or
   (nID == "000001")):
```

Kuvio 37. Kokoelmien binäärimerkkijonovaihtoehdot

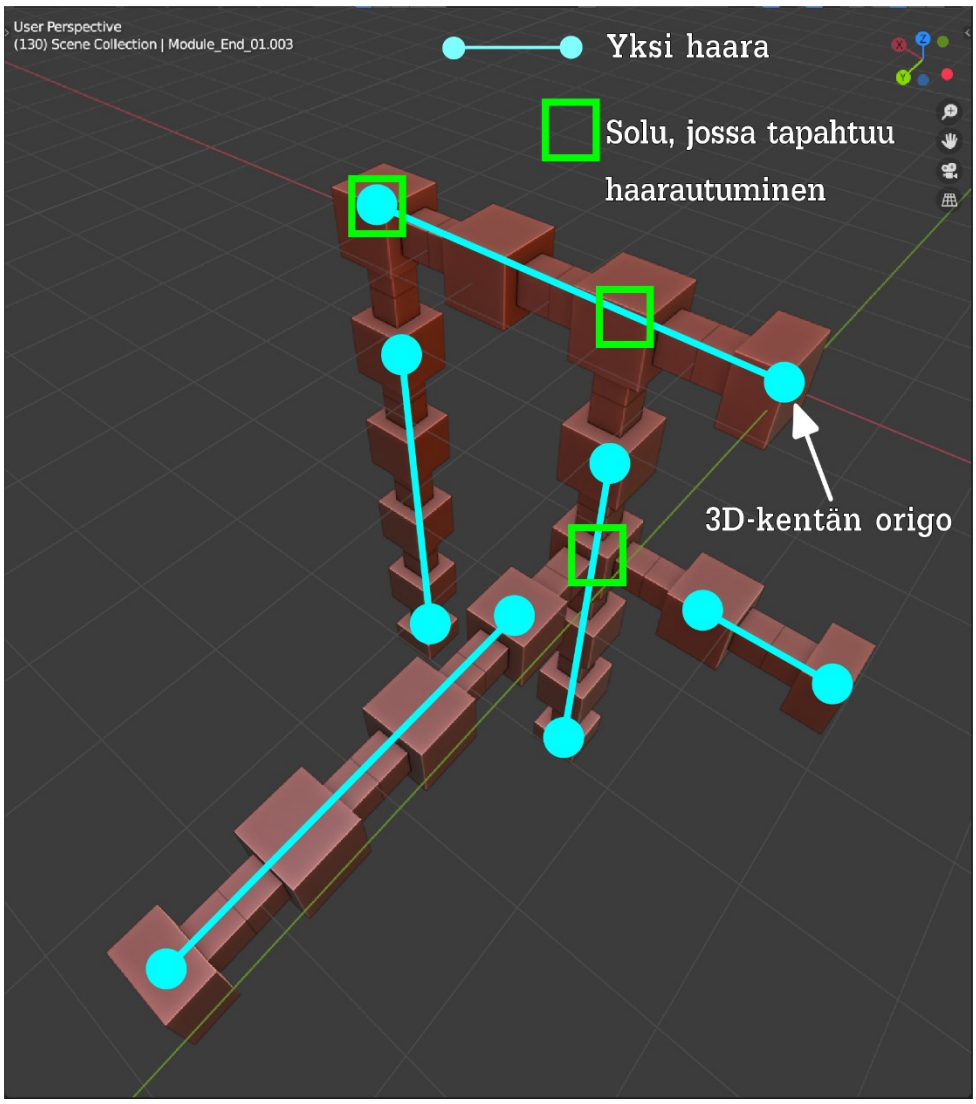
Jokaiselle binäärimerkkijonolle on kirjoitettu oma kääntämislogiikka erikseen. Kuviossa 38 nähdään, että kääntäminen on toteutettu lisäämällä tai vähentämällä akseleista r90- tai r180-muuttujalla. Skriptin alussa on määritelty, että r90 = 1.570796 ja r180 = 3.141593. Tämä on karkea, yksinkertainen radiaanien kääntäminen euler-kulmiin. 3D-mallien asettamisen jälkeen päivitetään näkymä, jotta muutokset tulevat käyttäjälle näkyviin.

```
                #Move object to the current cell
                new_module.location = active_cell

                #Check for rotation
                if nID == "100000":
                    #Positive X, need to rotate 90 degrees on Z axis
                    old_euler = new_module.rotation_euler
                    old_euler[2] += r90
                    new_module.rotation_euler = old_euler
                if nID == "010000":
                    #Negative X, need to rotate -90 degrees on Z axis
                    old_euler = new_module.rotation_euler
                    old_euler[2] -= r90
                    new_module.rotation_euler = old_euler
                if nID == "001000":
                    #Positive YNeed to rotate 180 degrees on Z axis
                    old_euler = new_module.rotation_euler
                    old_euler[2] += r180
                    new_module.rotation_euler = old_euler
                if nID == "000100":
                    #Negative Y, don't need to to anything.
                    old_euler = new_module.rotation_euler
                    old_euler[2] += 0
                    new_module.rotation_euler = old_euler
                if nID == "000010":
                    #Positive Z, need to rotate -90 degrees on X axis
                    old_euler = new_module.rotation_euler
                    old_euler[0] -= r90
                    new_module.rotation_euler = old_euler
                if nID == "000001":
                    #Negative Z, need to rotate 90 degrees on X axis
                    old_euler = new_module.rotation_euler
                    old_euler[0] += r90
                    new_module.rotation_euler = old_euler

            #Update the view layer
            bpy.context.view_layer.update()

        #-----------------------------------------------------
        # END OF SPAWN LOGIC
        #-----------------------------------------------------
```

Kuvio 38. Module_End-kokoelman kääntämislogiikka

4.6 Lopputulos

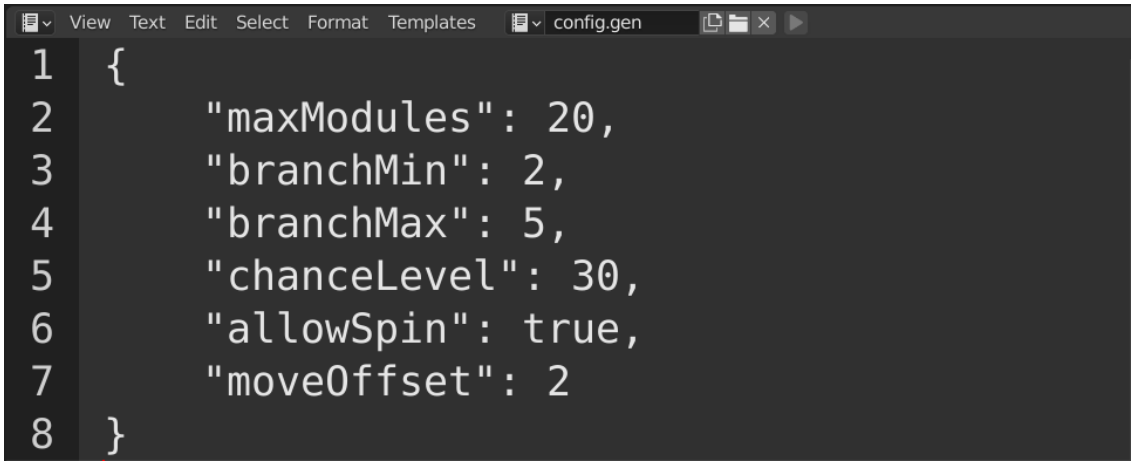
Kuviossa 39 nähdään O-Genilla generoitu 3D-malli, joka on yksittäisistä, modulaarisista 3D-malleista rakennettu lopputulos. Kentän origosta lähtee ensimmäinen haara. Tässä haarassa on kaksi solua, jossa tapahtuu haarautuminen. Haarautuvat solut kasvattavat uuden haaran satunnaiseen suuntaan ja jatkavat generointia, kunnes maxModules-arvo saavutetaan ja generointi loppuu. Kuvion 39 generoidussa mallissa on yhteensä viisi haaraa, ja haarautuminen tapahtuu kolmessa solussa.



Kuvio 39. Generoitu 3D-malli

Kuvion 39 generoituun malliin on vaikuttanut käyttäjän määrittämät asetukset, jotka ovat kirjattu JSON-tiedostoon. Kuviossa 40 nähdään JSON-tiedosto. Tämän mukaan generointiprosessi loppuu, kun 20 solua on generoitu. Haaran pituus arvotaan väliltä 2–5. Jokaisella solulla on 30 % mahdollisuus haarautua. AllowSpin antaa luvan Module_Straight-moduuleille pyöriä lokaalin eli oman Y-akselinsa mukaan. MoveOffset säätää solujen välisen etäisyyden. Tämän avulla voidaan tehdä 3D-malleja, jotka ovat suurempia kuin kaksi kuutiometriä, mutta käyttävät silti samaa kuutioverkoston resoluutiota. Esimerkiksi käyttäjä voi luoda

kokoelmiin 3D-malleja, jotka käyttävät kahden kuutiometrin sijaan 18m^3 tilavuuden. Tässä tapauksessa käyttäjän tulee vaihtaa moveOffsetin arvo lukuun 18.

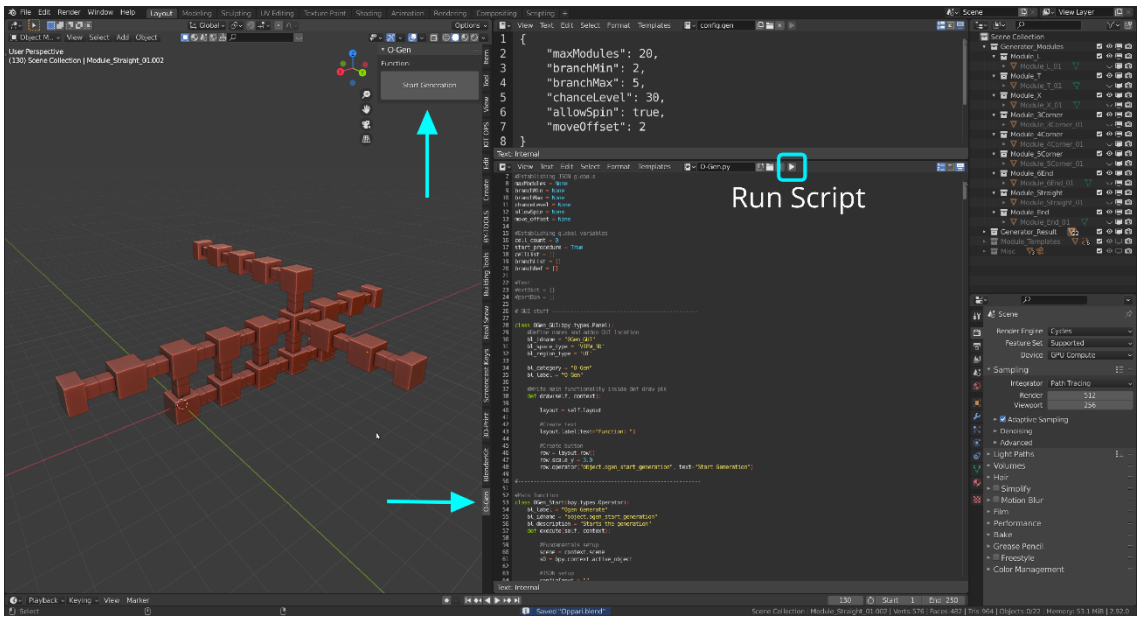


Kuvio 40. JSON-tiedoston asetukset

4.7 Laajennuksen käyttöönotto

Prototyypin voi ottaa käyttöön lataamalla .blend-tallennustiedoston, jossa laajennus sijaitsee. Blender aukeaa valitsemalla .blend-tiedoston ja käyttäjä pääsee käsiksi näkymään, jossa on viisi eri Blenderin moduulia. Nämä ovat vasemmalta oikealle ja ylhäältä alaspäin luettuna 3D-viewport, Text editor JSON-asetuksille, toinen Text editor laajennuksen rekisteröimistä varten, Outliner sekä Properties. (Kuvio 41.)

JSON-asetuksien muokkaamisen jälkeen käyttäjän tulee painaa Run script -painiketta, joka sijaitsee alemman Text editorin yläpalkissa play-painikkeen näköisenä. Vaihtoehtoisesti voi käyttää ALT+P-pikanäppäinyhdistelmää. Skriptin ajamisen jälkeen 3D-ikkunan sivupaneeliin esiintyy uusi kategoria nimeltään O-Gen. Käytävägeneraattori luo uuden käytävän joka kerta, kun käyttäjä painaa Start Generation -painiketta. JSON-tiedostojen muokkaamisen tai kokoelmien 3D-mallien muokkaamisen jälkeen käyttäjän tulee ajaa skripti uudestaan, jotta muutokset rekisteröityvät.



Kuvio 41. Laajennuksen käyttöönotto

Omien modulaaristen osien luominen onnistuu helposti. Outliner-moduulissa näkyy kokoelma nimeltään Module_Templates, joka on asetettu näkymättömäksi. Tämän kokoelman näkyviin tuominen paljastaa solun runkorakenteet ja akselimerkinnät, joita voi hyödyntää omien 3D-mallien luomisessa. Haluttujen toimenpiteiden jälkeen oma 3D-objekti tulee siirtää sille sopivaan kokoelmaan nimetä tunnistettavaan muotoon. Laajennus ei käytä alkuperäisiä 3D-malleja, jos niiden nimet vaihdetaan joksikin muuksi. Skripti tulee ajaa uudestaan, jotta muutokset rekisteröityvät.

5 POHDINTA

Aloitin 3D-mallintamisen toukokuussa 2017 ja opettelin käyttämään Blenderiä kolmen kuukauden ajan. Olin vakuuttunut, miten ilmaisella ohjelmalla pystyi luomaan täysin realistisen näköisiä ympäristöjä ja esineitä. Kokemukseni kasvaessa huomasin, että kaikista työvaiheista itse 3D-mallinnus vei eniten aikaa. Koko 3D-prosessi ei etene ilman, että kentässä on jonkinlainen 3D-malli. Muut työvaiheet, kuten materiaalikehitys, teksturointi, valaistus, kamera-ajon suunnittelu, animointi, fysiikan simulointi sekä jälkiprosessointi toteutetaan vasta, kun on olemassa jonkinlainen 3D-malli manipuloitavana. Tästä syntyi ajatus, olisiko minun mahdollista luoda oma laajennus, joka auttaisi aloittelija- ja keskitason mallintajaa.

5.1 O-Genin kehittäminen

Modulaarisuuden takia laajennusta ei tehty suoraan Unityyn tai Unrealiin. Käyttäjä pystyy tällä tavalla mallintamaan omat osat ja generoimaan käytäväkompleksin omaan käyttötarkoitukseensa. Tämä on mielestäni hyödyllinen ominaisuus, koska laajennusta voi käyttää moneen eri projektiin ilman visuaalista toistuvuutta.

3D-mallin muodon lisäksi tekstuureilla on suuri merkitys siinä, miltä generoitu kompleksi näyttää. Laajennuksen modulaarisia 3D-malleja pystyy teksturoimaan helposti esimerkiksi Quixelilla tai Substance Painterilla, mikä lisää joustavuutta. Tekstu rointiprosessi myös muuttuu renderointimoottorin muuttuessa, sillä kaikki renderointimoottorit eivät käytä PBR metallic roughness -teksturointiprosessia. Alustaan lukittautuminen on tästä syystä myös joustavuudelle haitallista, sillä kaikki eivät käytä Unityä, eivätkä kaikki käytä Unrealia. Moni kuitenkin käyttää Blenderiä 3D-mallinnustyökaluna sekä Quixelia tai Substance Painteria -teksturointiohjelmana näille alustoille.

Oma laajennus on järkevää tehdä, jos on tiedossa tietynlainen mallinnusprosessi tai käyttökohde. Laajennus pystyy säästämään potentiaalisesti paljon aikaa jatkossa. Esimerkiksi laajennus, joka tallentaa, käsittelee ja järjestää kaikki aikaisemmin käytetyt assetit koheesiin asset-pankkiin olisi oivallinen resurssinhallin-

tatyökalu pelikehittäjille, jotka työskentelevät paljon pelimoottorien parissa. Laajennuksien kehittämisen kustannustehokkuus on kuitenkin hyvin tapauskohtaista ja niillä on aina riski jäädä turhaksi projektin loputtua. Tämän takia laajennuksia kannattaisi tehdä toistuville prosesseille. Usein yksinkertaisiin, yleistyneisiin ja toistuviin prosesseihin kuluu pitkällä tähtäimellä paljon ylimääräistä aikaa. Yksinkertaisten prosessien automatisointi laajennuksella tai työkalulla olisi myös helppo toteuttaa, mikä taas parantaa laajennuksen kehittämisen kustannustehokkuutta.

Laajennuksen alkuperäinen idea on antaa valmiita rakennuspalasia aloittelevalle mallintajalle tarjoamalla työkalun, jolla voi nopeasti luoda jonkinlaisen kentän. Prototyypin kulmikkaat ja yksinkertaiset 3D-mallit ovat kuitenkin kyseenalaisia tämän tavoitteen toteuttamisessa. Aloittelevalle 3D-mallintajalle olisi mukavampaa, jos generoidut 3D-kentät olisivat yksityiskohtaisempia tai persoonallisempia, minkä takia niitä pitäisi vielä muuttaa lopullista julkaisuversiota varten. Jopa ikkunareikien lisääminen kulmikkaisiin 3D-malleihin tuo paljon monimutkaisuutta etenkin valaistuksessa. Käyttäjä voisi kokeilla esimerkiksi, miten erilaisesti valaistus toimii rasterointi- ja säteenseurantamoottorilla.

5.2 Huomioitavia asioita

Laajennuksen rakentaminen Blenderin sisällä oli mielestäni hyvä idea, sillä monet toiminnot vaativat Blenderin omien moduuleiden käyttämistä. Tämän lisäksi Blenderissä on Info-moduuli, jonne kirjataan kaikki tapahtumat aina hiiren klikkauksesta koodin syntaksivihreisiin. Info-moduuli toimi tästä syystä oivallisena debuggerina. Debuggeri osoitti visuaalisesti monella eri tavalla erilaisista vihreistä. Yksi näistä oli ponnahdusikkunan tyylinen tekstilaatikko, jossa kerrottiin millä rivillä virhe oli ja mikä on virheen tyyppi. Samassa tekstilaatikossa oli myös virhekoodi, jolla voi etsiä Blenderin dokumentaatiosta tarkempaa informaatiota. Ponnahdusikkunan tiedot löytyvät myös Info-moduulista.

Ohjelmointi Text editorissa oli selkeää, kun Blender ymmärsi syntaksit ja värikoodasi tekstin asianmukaisesti. Ohjelmoinnin perustoiminnot toimivat melko samalla tavalla kuin esimerkiksi Visual Studio tai Code. Text editorissa on erilaisia asetuksia, jotka vaikuttavat tekstin esittämiseen. Käyttäjä voi ottaa käyttöönsä

esimerkiksi asetuksen, joka näyttää rivien numerot tai rivityksen. Käyttäjä voi myös manuaalisesti muokata, minkä värisiä tunnistetut tekstit ovat. Visuaalisen muokkauksen lisäksi kaikki perustoiminnot, kuten tekstin etsiminen, kopiointi ja liittäminen sekä rivien vaihtaminen ovat mahdollisia.

Blender tarjoaa valmiita koodimalleja, joissa on yksinkertaisia toimintoja valmiiksi ohjelmoituina. Käyttäjä pääsee muokkaamaan näitä ja tekemään yksinkertaisia toimintoja. 3D-mallintaminen ja Python-ohjelmointi vaativat kuitenkin aikaisempaa kokemusta, jotta käyttäjä pystyy luomaan monimutkaisemman laajennuksen.

Blenderistä löytyy hyviä opetusvideoita, artikkeleita ja kursseja. Jokaisesta työnkulusta tai moduulista löytyy kattava kokoelma erilaisia oppitunteja, joita käytin opinnäytetyötä tehdessä. Tämän lisäksi on olemassa erilaisia Blender-yhteisöjä, joissa voi esittää kysymyksiä. Keskustelufoorumeiden lisäksi Blender Discord oli oivallinen tapa esittää ja saada nopeasti palautetta kysymyksiin.

Suurin osa ongelmista sijaitsi muuttujien arvotyypeissä, silmukoissa ja skriptin rekursiivisessa rakenteessa. Skriptistä huomaa, että moni toiminnallisuus toteutettiin siten, että käytössä oli useampi muuttuja yhtä muokkausta varten ja seuraavaan toimenpiteeseen luotiin aina uusia muuttujia. Tämä johtui siitä, että skripti ei toiminut, kun muuttujia referoitiin moneen kertaan tai kun referoitiin muuttujia, joilla oli eri arvotyyppejä. Skripti tarvitsee uudelleenjärjestelyn, jos parempi ratkaisu keksitään. Laajennus on kokonaisuudessaan yli tuhat riviä koodia, joista suurin osa sijaitsee module_conditions-prosessissa. Jokaiselle binäärimerkkijonolle kirjoitettiin oma kääntölogiikka erikseen. Module_conditions-prosessi, jossa muutettiin yksittäisiä arvoja ja osia, oli loppujen lopuksi kuitenkin erittäin toistuva. Module_conditions-prosessin uudelleenjärjestely on myös tästä syystä tarpeen.

5.3 Jatkokehitysideoita

Tällä hetkellä prototyypissä on selkeitä rajoitteita, joihin en ole keksinyt vielä ratkaisuja. Suurin rajoitus on tämänhetkisen laajennuksen muoto. Laajennus pitää käynnistää omana Blender-instanssina, eikä sitä voi asentaa .py-tiedostona ja käyttää generaattoria missä tahansa projektissa. Rajoitus johtuu laajennuksen muokattavuudesta. Oikeanlaisena asennettavana lisäosana käyttäjä ei pystyisi

muokkaamaan modulaaristen 3D-mallien ulkonäköä tai lisätä omia vaihtoehtoja kokoelmiin. Laajennuksella on paljon kehitettävää edessä, esimerkiksi JSON-tiedoston asetuksia voisi implementoida graafiseen käyttöliittymään.

Toinen kehitysajatus olisi käyttää enemmän aikaa 3D-mallintamisessa ja mallintaa visuaalisesti eri teemaisia malleja. Laajennus voisi esimerkiksi generoida luolan tai tunnelin pätkiä tai isompia, yksityiskohtaisempia huoneita. 3D-mallit teksturoitaisiin PBR-tekniikalla ja kaikki paketoitaisiin zip-tiedostoon. Tämän avulla käyttäjä voisi asentaa Blenderissä zip-tiedoston ja pääsisi käyttämään renderointivalmiita assetteja. Käyttäjä voisi sitten valita graafisen käyttöliittymän pudotusvalikosta, minkä teemaisen kompleksin haluaa generoitavaksi. Tästä paketoidusta versiosta voisi muuntaa laajennuksen Unityyn tai Unrealiin. Muunnetun version laajentaminen tai parantaminen tapahtuisi kuitenkin Blenderissä, ettei muutoksia tarvitsisi tehdä molemmille alustoille erikseen. Unityn ja Unrealin laajennuksien käyttöönotto olisi todennäköisesti kuitenkin hyvin erilaista ja vaatisi kahden eri version erillisen ylläpidon.

Laajennuksen proseduraalisessa generoinnissa on myös rajoitteita. Tileset-menetelmästä johdatetun kuutiojärjestelmän takia laajennus pystyy generoimaan 3D-malleja ainoastaan kuuteen eri suuntaan: ylös, alas, oikealle, vasemmalle, eteenpäin tai taaksepäin. Tämä tarkoittaa sitä, että esimerkiksi kahdeksankulmaiset yhtenevät rakenteet eivät ole mahdollisia, ellei kuutiojärjestelmän resoluutiota muuteta. Monimutkaisempi versio toteutettaisiin esimerkiksi niin, että yksi 3D-malli voi ottaa käyttöönsä neljän solun tilavuuden. Tällöin naapurisolujen määrä kasvaa sekä mahdolliset suunnat, joihin haarautuminen tapahtuu, lisääntyvät. Tämä voi myös vaikuttaa haitallisesti generoituun tulokseen olemalla visuaalisesti liian toistuva, jos modulaarisia 3D-malleja ei ole tarpeeksi monenlaisia. Generointityylin takia tämä vaatisi myös monimutkaisemman logiikan sekä enemmän kokoelmia erityyppisille muodoille. Monimutkaisuuden ja konkreettisen kehitysaikataulun välillä tulee löytää tasapaino. Tästä syystä päädyttiin lopputulokseen, jossa opinnäytetyöhön soveltuvin monimutkaisuus olisi 3D-malli per solu, jossa jokaisella solulla on mahdollisuus haarautua vain kuuteen eri suuntaan.

Laajennukselle luotiin solugenerointilogiikka myös Z-akselissa. Laajennukselle voisi myös kehittää toiminnon, jolla käyttäjä voisi lukita tiettyjä akseleita pois käytöstä. Tällä tavalla käyttäjä voisi lukita esimerkiksi Z-akselin pois käytöstä niin, että laajennus generoisi vain X- ja Y-akselin suuntaan. Tämä poistaisi korkeuseron ja generoisi vain tasaisia käytäviä. Käyttäjä pystyisi generoimaan esimerkiksi suuria labyrinttejä nostamalla maxModules-arvoa suureen lukemaan.

Prototyypissä pystyy tällä hetkellä luomaan soluja suoraan ylöspäin tai alaspäin, mikä ei ole kovin käytännöllistä. Laajennukselle voisi jatkossa kehittää ylimääräisen logiikan, joka antaisi 3D-malleille mahdollisuuden asettua vinosti tai viistoon. Tämän ansiosta 3D-mallit, jotka yhdistävät diagonaalisesti yhtä solua edessä ja yhtä solua ylhäällä, olisivat mahdollisia. Tämä logiikka tekisi mahdolliseksi esimerkiksi rappukäytävät tai luonnollisemman tavan siirtyä Z-akselissa eteenpäin.

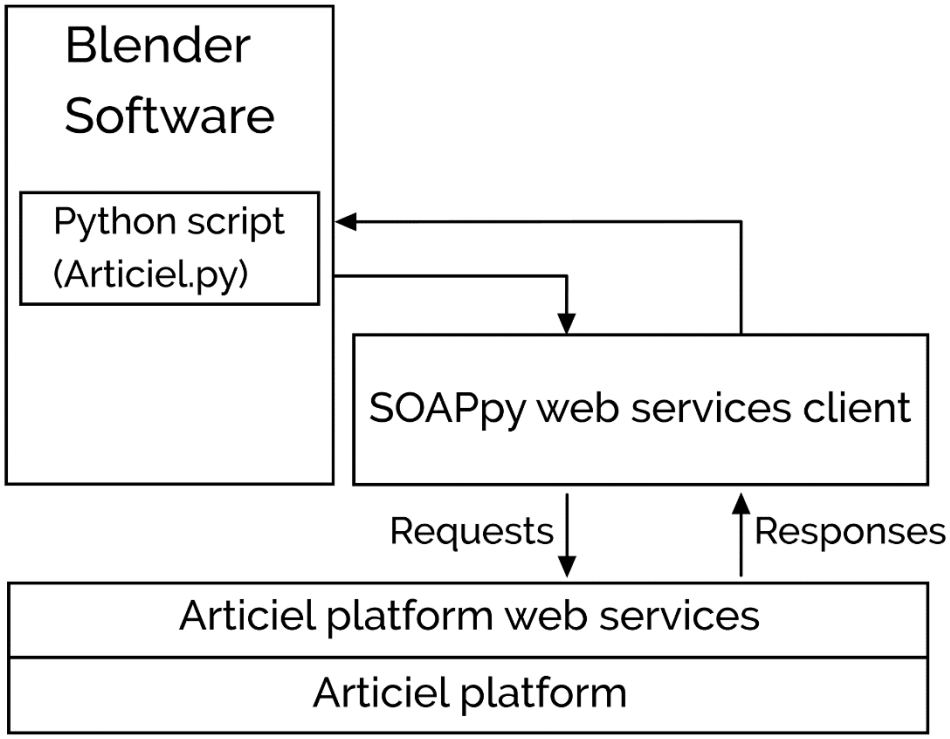
5.4 Lisäosien potentiaali

Blenderistä puuttuu vielä tärkeitä ominaisuuksia, kuten esimerkiksi fyysisten kameraominaisuuksien simulointi tai reaaliaikainen projektinhallinta. Blenderin virtuaalikamerasta löytyy esimerkiksi shutter speed eli valotusaika sekä aperture eli valoisuusaukon suuruus. Käyttäjä voi näillä asetuksilla muokata kameran fokusta tai motion blurin määrää. Yksittäisen asetuksen muuttaminen Blenderissä ei vaikuta lopullisen kuvan valoisuuteen tai kirkkauteen. Oikeassa kamerassa valotusaika, valoisuusaukko ja ISO-herkkyys ovat kaikki suhteessa toisiinsa ja vaikuttavat yhdessä valokuvan lopulliseen kirkkauteen.

Fabien Christin on luonut näiden puutteellisuuksien takia lisäosan nimeltään Photographer, joka lisää kamera-asetuksien määrää ja tuo enemmän muokattavuutta 3D-artisteille. Photographer-lisäosa tuo perustoimintojen lisäksi mahdollisuuden muokata esimerkiksi valkotasapainoa, valolähteiden värilämpötilaa, syväterävyyden määrää sekä sen anamorfista suhdetta. Asetuksien lisäksi Photographer muuntaa Blenderin omat numeeriset yksiköt oikeassa maailmassa käytettyihin yksikköihin. Esimerkiksi Blenderin valotusaukon yksikkö on numeroarvo väliltä 0,01–1,00. Photographer muuttaa sen esimerkiksi arvoon 1 / 100 s. Blenderin lamppujen valoisuusteho mitataan oletuksena watteina ja Photographer muuntaa ne Lumeneiksi, Candelaksi tai Luxeiksi. (Christin 2021.)

Toinen Blenderin puutteista on, että käyttäjä ei pysty tekemään reaalaikaista yhteistyötä toisten käyttäjien kanssa. Projektitiedostojen synkronoimiseen eri käyttäjien välillä tarvitaan erillinen Git-versionhallintatyökalu. Git ei ymmärrä Blenderin tallennustiedoston sisäistä dataa ja päivittää sen kokonaisena tiedostona toisin, kuin esimerkiksi Unityn Collab. Collab osaa yhdistellä älykkäästi käyttäjien tekemiä muutoksia saman projektitiedoston alla. Autodesk Maya on 3D-ohjelma, jota käytetään melko laajasti elokuvateollisuudessa. Mayaa on räätälöity eri yrityksille heidän käyttötarpeidensa mukaan. Mayalle on kehitetty ominaisuus, jolla pystyy synkronoimaan reaaliaikaisesti eri käyttäjien tekemiä muokkauksia riippumatta 3D-prosessin vaiheesta. Esimerkiksi mallintajan muokkaus 3D-hahmoon synkronoituu animoijalle ja hänen animoimansa 3D-hahmo päivittyy uudella versiolla.

Laboratoire de Téléinformatique de l’université du Québec à Montréal on Teleinformatiikan yliopisto Montrealissa, Quebecissä. Professori Omar Cherkaoui johti vuonna 2007 tutkimusprojektia, jonka tarkoituksena oli muuntaa Mayan reaaliaikainen projektinsynkronointiominaisuus Blenderille. Tutkimuksessa pohdittiin, miten samanlaisen toiminnallisuuden voi saavuttaa Blenderillä, ja he loivat lisäosan, jolla on samantapaisia ominaisuuksia. Lisäosan työnimikkeeksi tuli Articiel Collaborative Platform, joka toimi SOAPpy-arkkitehtuurilla. Lisäosa oli käytännössä verkkopalvelu, jossa eri käyttäjät kirjautuivat yhteiseen Blender-sessioon. Käyttäjien muokkaukset, kuten kuution liikuttaminen 3D-kentässä päivittyi saman session käyttäjille reaaliajassa. Lisäosan luonti toteutettiin muuntamalla Mayan C++ koodia Pythoniksi. Kuviossa 42 nähdään yksinkertainen selite, miten Articiel toimii. (Lesage ym. 2007, 2559.)



Kuvio 42. Articielin SOAPpy-arkkitehtuuri (Lesage ym. 2007, 2560)

Lisäosat ovat erittäin tehokas tapa lisätä moduuleita ja toiminnallisuuksia sovelluksiin. Blenderille löytyy kattava kokoelma erilaisia lisäosia Blendermarketista ja Gumroadista. Unitylle löytyy paljon ilmaisia tai maksullisia resursseja Unityn omasta asset storesta. Lisäosien rooli erilaisissa tuotanto-ohjelmissa on lisääntynyt huomattavasti, kun yksittäisille käyttäjille on julkaistu ohjelmia, jotka ovat mahdollistaneet amatööriprojektien kehittämisen. Opinnäytetyötä kirjoittaessa huomattiin, että lähes jokaista puuttuvaa tai puutteellista ominaisuutta kohden on olemassa jonkinlainen ammattilais- tai amatöörikäyttäjän luoma lisäosa, laajennus tai työkalu.

LÄHTEET

A46 Studio 2020. Artstation: Kolya the gun. Viitattu 30.3.2021 https://www.artstation.com/artwork/rR3Axe.

Advanced Micro Devices, Inc 2020. Using the Uber shader in Blender. Viitattu 9.12.2020 https://radeon-pro.github.io/RadeonProRenderDocs/en/plugins/blender/uber_shader.html.

Blaszczak, M. 2021. Artstation: Cyberpunk 2077 – Johnny Silverhand. Viitattu 30.3.2021 https://www.artstation.com/artwork/J9nVXn.

Blender Manual 2020a. Modeling, Meshes, Structure. Viitattu 5.12.2020 https://docs.blender.org/manual/en/latest/modeling/meshes/structure.html.

Blender Manual 2020b. Modifiers. Viitattu 5.1.2021 https://docs.blender.org/manual/en/latest/modeling/modifiers/introduction.html.

Blender Manual 2020c. Shader Editor. Viitattu 8.12.2020 https://docs.blender.org/manual/en/2.80/editors/shader_editor/index.html.

Blender Manual 2020d. Texture. Viitattu 9.2.2021 https://docs.blender.org/manual/en/latest/render/shader_nodes/textures/index.html.

Blender Manual 2020e. Node parts. Viitattu 9.2.2021 https://docs.blender.org/manual/en/latest/interface/controls/nodes/parts.html.

Blender Manual 2020f. Scripting and extending Blender. Viitattu 9.2.2021 https://docs.blender.org/manual/en/latest/advanced/scripting/introduction.html#general-information.

Blender Manual 2020g. Blener process/addons. Viitattu 15.2.2021 https://wiki.blender.org/wiki/Process/Addons.

Blender Manual 2020h. Blender Python API. Viitattu 9.2.2021 https://docs.blender.org/api/current/info_quickstart.html.

Blender Manual 2020i. Ambient Occlusion. Viitattu 7.5.2021 https://docs.blender.org/manual/en/2.79/render/blender_render/world/ambient_occlusion.html.

Blender Manual 2020j. Bump & Normal Maps. Viitattu 5.5.2021 https://docs.blender.org/manual/en/2.79/render/blender_render/textures/properties/influence/bump_normal.html.

Blender.org 2021. About: License. Viitattu 5.5.2021 https://www.blender.org/about/license/.

Bycer, J. 2015. Procedural vs Randomly generated content in game design. Viitattu 4.1.2021 https://www.gamasutra.com/blogs/JoshBycer/20150807/250760/Procedural_vs_Randomly_Generated_Content_in_Game_Design.php.

Carlson, W. E. 2017. Computer Graphics and Computer Animation: A Retrospective Overview. The Ohio State University. Viitattu 28.12.2020 https://ohiostate.pressbooks.pub/graphicshistory/.

Choi, R. 2020. Artstation: Destiny 2 Shadowkeep - Seventh Seraph Officer Revolver. Viitattu 30.3.2021 https://www.artstation.com/artwork/xJzqnO.

Christin, F. 2021. Blendermarket: Photographer. Viitattu 24.4.2021 https://blendermarket.com/products/photographer/?ref=222.

Denham, T. 2020. What is UV Mapping & Unwrapping? Viitattu 6.12.2020 https://conceptartempire.com/uv-mapping-unwrapping/.

Felinto D. 2020. Blender Developers Blog: Everything nodes and the scattered stone. Viitattu 19.4.2021 https://code.blender.org/2020/12/everything-nodes-and-the-scattered-stone/.

Herland, J. 2020. Introduction to HDRi. Viitattu 29.3.2021 https://www.jorgenhdri.com/explained.

Holt, C. 2020. By-Gen: Structured Generators. Viitattu 5.12.2020 https://www.notion.so/Structured-Generators-Documentation-3f1b5d2d053f47d4912f4389f29a0f4c.

Joensuu, J. 2016. 3D-alan sanasto, grafiikan termit suomeksi. Kajaanin ammattikorkeakoulu. Tradenomi, tietojenkäsittely. Opinnäytetyö. Viitattu 9.2.2021 http://urn.fi/URN:NBN:fi:amk-2016060612045.

Katsbits 2019a. Edit Mode Basics. Viitattu 5.12.2020 https://www.katsbits.com/codex/edit-mode-basics/.

Katsbits 2019b. Render Engines. Viitattu 9.12.2020 https://www.katsbits.com/codex/render-engine/.

Lesage, M., Cherkaoui, O., Abouzaid, F., Poirier, M., Raiche, G. & Riopel, M. 2007. Blender Plugin Implementations for 3D Collaborative Work. IEEE International Conference on Systems, Man and Cybernetics, 2557-2569. Viitattu 24.4.2021 http://dx.doi.org/10.1109/ICSMC.2007.4413770.

McCombs, S. 2010. Intro to procedural textures. Viitattu 8.2.2021 http://www.upvector.com/?section=Tutorials&subsection=Intro%20to%20Procedural%20Textures.

McDermott, W. 2018. Allegorithmic Substance: PBR Guide PDF. Viitattu 8.12.2020 https://academy-api.substance3d.com/courses/b6377358ad36c444f45e2deaa0626e65/attachments/2b57526e-4bf3-4fd6-ae88-e9a9313a35cc.

Petty, J. 2018. What is 3D modeling & what's it used for? Viitattu 5.12.2020 https://conceptartempire.com/what-is-3d-modeling/.

Price, A. 2019. How many people are using 3D design software worldwide? Viitattu 16.3.2021 https://www.quora.com/How-many-people-are-using-3D-design-software-worldwide-i-e-Maya-3DS-Max-Cinema-4D-and-etc/answer/Andrew-Price-23.

Purcell, T. J. 2004. Ray tracing on a stream processor. Stanford University. Department of computer science. Ph.D. dissertation. Viitattu 7.12.2020 http://graphics.stanford.edu/papers/tpurcell_thesis/.

Siddi, F. 2019. Blender by the numbers. Viitattu 16.3.2021 https://web.archive.org/web/20201101001725/https://www.blender.org/press/blender-by-the-numbers-2019/.

Tan, G. 2020. Artstation: Assassin's Creed Valhalla: Post Launch Trailer. Viitattu 30.3.2021 https://www.artstation.com/artwork/oA6r0O.

Thommes, S. 2020. Br'cks: procedural brick texture. Viitattu 5.12.2021 https://gumroad.com/l/brcks.

Unity Manual 2020. ShaderLab culling and depth testing. Viitattu 6.12.2020 https://docs.unity3d.com/2020.2/Documentation/Manual/SL-CullAndDepth.html.

Vivo, P. & Lowe, J. 2015a. The Book of Shaders: Noise. Viitattu 25.3.2021 https://thebookofshaders.com/11/.

Vivo, P. & Lowe, J. 2015b. The Book of Shaders: Shaders. Viitattu 29.3.2021 https://thebookofshaders.com/01/.

Wengenmayer, M. 2016. HDRi Light. Viitattu 28.12.2020 https://tgjp.github.io/Cheetah3D_jp/Cheetah3Dmanual/Objects/SceneObjects/HDRI-Light/HDRILight.html.

Zgraggen, D. 2019. How to use PBR texture in Blender. Viitattu 10.12.2020 https://www.cgbookcase.com/textures/how-to-use-pbr-textures-in-blender.