Tampereen ammattikorkeakoulu

Huonekaluja koodaamalla

Parametristen 3D-mallien luonti skriptipohjaisilla CAD-työkaluilla

Jaakko Mäntylä

Opinnäytetyö
Syyskuu 2021

Tietojenkäsittely
Ohjelmistotuotanto

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietojenkäsittely
Ohjelmistotuotanto

JAAKKO MÄNTYLÄ:
Huonekaluja koodaamalla
Parametristen 3D-mallien luonti skriptipohjaisilla CAD-työkaluilla

Opinnäytetyö 40 sivua, joista liitteitä 0 sivua
Syyskuu 2021

Tässä opinnäytetyössä perehdyttiin 3D-mallien luomiseen ohjelmoimalla sekä tarkasteltiin menetelmän mahdollisuuksia ja rajoitteita. Tavoitteena oli selvittää, miten perinteisten 3D-mallinnustyökalujen haasteita ja tiedostoformaattien rajoitteita pystyttäisiin ratkaisemaan hyödyntämällä ohjelmistokehityksessä vakiintuneita työkaluja ja menetelmiä.

Työssä suunniteltiin ja toteutettiin käyttäjän kustomoitavissa oleva parametrinen 3D-malli huonekalusta käyttäen CadQuery-kirjastoa. Lisäksi luotiin yksinkertainen verkkosivu, joka mahdollistaa tuotetun 3D-mallin esikatselun, haluttujen parametrien muuttamisen ja tiedostojen lataamisen web-käyttöliittymästä.

Työssä selvisi, että 3D-mallien luomisessa ohjelmoimalla on etuja graafisten työkalujen käyttöön verrattuna. Parametrisen 3D-mallin luomisessa erityisen hyödylliseksi osoittautui ohjelmointikielen tarjoama täsmällisyys ja mahdollisuus välittää paremmin mallin haluttu käyttäytyminen parametreja muokattaessa. Selkeitä hyötyjä ovat myös mahdollisuus käyttää ohjelmointikehityksessä vakiintunutta versiohallintatyökalua ja luotujen 3D-mallien helpompi uudelleen käyttö ja jakaminen.

Asiasanat: CAD, ohjelmointi, ohjelmistokirjastot

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree Programme in Business Information Systems

Software Development

Jaakko Mäntylä:
Programmatic 3D-modeling

Bachelor's thesis 40 pages, appendices 0 pages
September 2021

This thesis examined the possibilities and limitations of creating 3D models by programming. The aim was to investigate how the challenges in traditional 3D modeling tools and file formats could be mitigated with use of tools and methods used in software development.

The work was done by examining the current research regarding 3D modeling tools, specifically script-based 3D modeling tools, and by creating a customizable parametric 3D-model. The model was created with CadQuery Python library. In addition, a web interface was created for customizing the furniture model created.

The findings indicate that creating 3D models by programming has significant advantages over the traditional 3D modeling tools with graphical user interfaces. The exact nature of programming languages turned out to be especially useful for creating parametric 3D models and for determining the behavior of the model when parameters are modified. Other distinct benefits are the possibility of using proven version control tools that are already used in software development, and the ease of reusing and sharing the created 3D models.

Key words: CAD, programming

SISÄLLYS

1 JOHDANTO

Mitä jos ohjelmoija, muuttaakseen yhden muuttujan arvoa, joutuisi uudelleen kirjoittamaan koko tiedoston? Entä millaisia olisivat ohjelmistoprojektit, jos versiohallintatyökaluja ei olisi ja projektien jakamiseen käytettäisiin häviöllisiä binääritiedostoja? Silloin oltaisiin lähellä nykyistä tilannetta 3D-mallien luonnissa hallinnassa ja jakamisessa.

Pohdintani siitä, miten ohjelmistoprojekteissa käytettäviä työkaluja ja toimintamalleja pystyisi hyödyntämään saman kaltaisten ongelmien ratkaisuun 3D-mallinnusprojekteissa johtivat minut perehtymään työkaluihin, joiden avulla voi luoda 3D-malleja ohjelmoimalla ja tämän opinnäytetyön syntyyn.

Tämän opinnäytetyön tavoite on perehtyä vallitseviin ongelmiin 3D-mallien luonnissa ja siihen, miten 3D-mallien luominen ohjelmoimalla voisi tarjota ratkaisuja niihin. Tarkoituksena on kartoittaa, millaisia työkaluja 3D-mallien luomiseen ohjelmoimalla on tällä hetkellä tarjolla ja niiden tämänhetkistä tilaa, heikkouksia ja vahvuuksia. Tämä tapahtuu kirjallisuuskatsauksella ja luomalla loppukäyttäjän kustomoitavissa oleva 3D-malli ohjelmoimalla sekä web-käyttöliittymä mallin esikatseluun ja muokkaamiseen.

Seuraavassa luvussa perehdytään aiempaan tutkimukseen liittyen 3D-mallien luomiseen ohjelmoimalla ja siihen mitä hyötyjä tästä lähestymistavasta voisi olla. Kolmannessa luvussa käydään läpi tarkemmin vaatimukset toteutettavalle 3D-mallille ja web-käyttöliittymälle. Tämän jälkeen luodun 3D-mallin, web-käyttöliittymän ja näiden pohjalta luodun huonekalun toteutuksesta kerrotaan erikseen omissa luvuissaan.

2 3D-MALLINTAMINEN SCRIPTIPOHJAISILLA CAD-TYÖKALUILLA

2.1 3D-mallintaminen

3D-mallintamisella tarkoitan tässä opinnäytetyössä tietokoneella esikatseltavissa olevan kolmiulotteisen representaation luomista objektista, käyttäen tähän tarkoitukseen erityisesti luotua tietokoneohjelmaa tai ohjelmistokirjastoa. Tällä tavoin luotua representaatiota kutsutaan 3D-malliksi. 3D-malleja ja -mallintamista hyödynnetään esimerkiksi peleissä ja animaatioissa, mutta tämän työn kontekstissa keskitymme 3D-mallintamiseen suunnittelun apuvälineenä. Työssä käsitellään 3D-mallintamista ja -mallinnustyökaluja CAD (computer aided design) -kontekstissa. Ajatuksena on, että luotujen 3D-mallien tarkoitus on auttaa tuotteen suunnittelussa ja valmistamisessa.

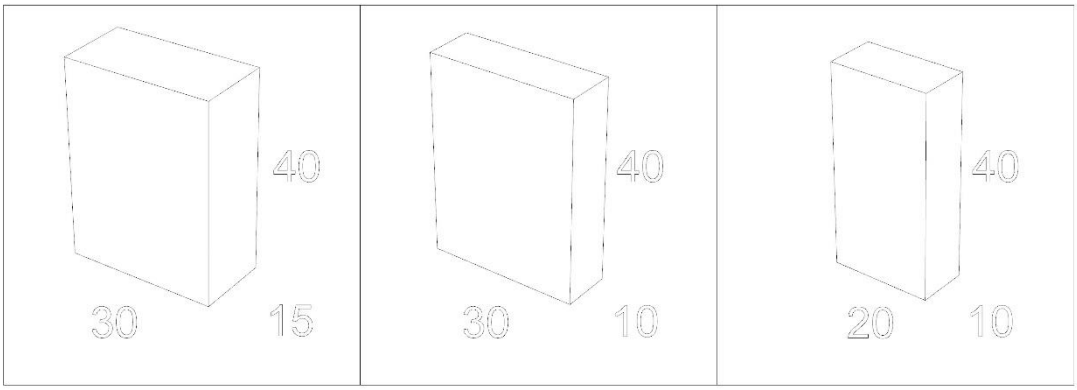
2.2 Skriptipohjaiset CAD-työkalut

Työkaluille, jotka mahdollistavat 3D-mallien luomisen ohjelmoimalla, ei ole vakiintunutta termiä. Tässä työssä käytän niistä termiä skriptipohjaiset CAD-työkalut. Tarkemmin sanottuna skriptipohjaisella CAD-työkalulla tarkoitan siis ohjelmistoja tai ohjelmistokirjastoja, joiden avulla käyttäjä pystyy kirjoittamalla komentosarjoja määrittelemään kolmiulotteisen kappaleen ja esikatselemaan määriteltyä kappaletta visuaalisena representaationa. Esimerkkejä tällaisista työkaluista ovat muun muassa: OpenScad, CadQuery ja CascadeStudio.

2.3 Suunnitteluaikeen välittyminen 3D-mallista

Design intent -termille ei ole vakiintunutta määritelmää (Otey, Company, Contero & Camba 2018, 50). Tietääkseni termille ei ole vakiintunutta suomennostakaan. Käytän tässä työssä käännöstä suunnitteluaie.

Suunnitteluaikeen määritellään useimmiten, joskaan ei aina, kuvaavan mallin haluttua käyttäytymistä, kun sitä muokataan (Otey ym. 2018, 50). Tätä määritelmää käytän myös tässä opinnäytetyössä. Seuraavassa kappaleessa avaan hieman suunnitteluaie termiä ja sen välittämiseen liittyviä haasteita esimerkin avulla.



Kuva 1. Suorakulmaiset särmiöt eri mitoilla

Oletetaan, että olemme mallintaneet kuvassa 1 vasemmalla näkyvän kaltaisen suorakulmaisen särmiön, jonka leveys on 30 cm, syvyys 15 cm ja korkeus 40 cm. Hypoteettinen mallimme on parametrinen, joten voimme muuttaa näitä mittoja halutessamme. Tulemmekin toisiin aatoksiin ja muutamme syvyydeksi 10 cm. Suunnittelu aie määrittää miten muiden parametrien pitäisi reagoida tähän. Tulisiko niiden pysyä samoina, kuten kuvan 1 keskimmäisessä kappaleessa, jossa leveys on edelleen 30 cm ja korkeus 40 cm. Vai oliko ajatuksena, että syvyyden tulee olla puolet leveydestä, jolloin uuden leveyden tulisi olla 20 cm, kuten vasemmanpuolimmaisessa kappaleessa? Halutun käytöksen määrittelee suunnitteluaie.

Vaikka suunnitteluaikeen tarkasta määritelmästä ei ole yhteisymmärrystä tieteellisessä yhteisössä, sen selkeän esityksen tärkeydestä ja sen selkeän esityksen hankaluudesta CAD-malleissa vallitsee konsensus (Otey ym. 2018, 50-52). Olen myös itse kokenut tämän ongelman, jouduttuani käyttämään huonekalumalliston uudelleen mallintamiseen kymmeniä tunteja, koska joidenkin osien mitat ovat muuttuneet ja käyttämäni CAD-työkalu ei ole mahdollistanut suunnitteluaikeen tallentamista riittävän tarkasti, jotta 3D-mallin muut osat olisivat muokkaantuneet oikein yhden osan mittojen muuttuessa.

Esimerkiksi Autodesk Fusion CAD-ohjelmistossa suunnitteluaikeen parempaan välittämiseen on käytettävissä parametrisia työkaluja. Fusion käyttää kaksiulotteisia luonnoksia (engl. sketch), joihin pystyy määrittelemään rajoitteita (engl. constraits) sen osien mitoille tai ominaisuuksille. Näiden luonnosten on tarkoitus

toimia kolmeulotteisten osien pohjana ja rajoitteiden välittää suunnitteluaie. Muillakin moderneilla parametrisilla CAD-työkaluilla on samankaltaisia ominaisuuksia. Aiemmin käytetyssä esimerkissä tämä voisi tarkoittaa syvyysparametrin luomista ja sen rajoittamista olemaan puolet leveydestä. Näistä huolimatta graafiset työkalut usein epäonnistuvat suunnitteluaikeen välittämisessä (Mathur, Pirron & Zufferey 2020, 409).

Mathurin, Pirron ja Zuffereyn mukaan, merkittävä syy tähän on, että graafisissa CAD-työkaluissa ei pystytä täysin yksiselitteisesti kuvaamaan, mitä käsiteltävälle mallille halutaan tehdä. Tämä johtaa tarpeeseen käyttää jonkinlaista heuristikkaa näiden epäselvyyksien ratkaisuun. Näiden heuristiikkojen toimintaperiaatetta ei yleensä etukäteen tiedetä ja ne johtavat usein erialisiin ja yllättäviin tuloksiin. 3D-mallin luominen ohjelmoimalla poistaa tämän ongelman, koska ohjelmoitaessa mallille suoritettavat välivaiheet listataan eksplisiittisesti, jolloin tarvetta ongelmallisille heuristiikoille ei ole. (Mathur, Pirron & Zufferey 2020, 409)

Sen lisäksi että 3D-mallien luominen ohjelmoimalla mahdollistaa tarkemman ja yksiselitteisen ilmaisun mallin käyttäytymisestä. Tarjoaa ohjelmointi toisenkin suunnitteluaietta selkeyttävän edun: koodiin pystyy kirjoittamaan kommentteja ja hyvin nimetyt muuttujat ja funktiot avaavat jo itsessään niiden toimintaa ja tarkoitusta. Tämä tarjoaa mahdollisuuden välittää paitsi tiedon siitä, miten mallin halutaan käyttäytyvän, mutta myös miksi mallin halutaan käyttäytyvän niin.

Mielestäni merkittävä etu on myös, että ohjelmoimalla pystyy määrittelemään suunnitteluaikeen monipuolisemmin kuin perinteisillä graafisilla CAD-työkaluilla hyödyntämällä ehtolauseita. Kiinnikkeeseen tulevan pyöreän reiän halkaisijan muuttaminen pitäisi olla melko helppoa nykyaikaisien graafistenkin CAD-työkalujen parametrisilla ominaisuuksilla, mutta entä jos reikään tulevan osan profiili saattaisi olla neliö tai ympyrä ja neliö profiili vaatii hieman erilaisen pyöristyksen reiän viereen? Tällaisen parametrisen mallin luomien perinteisillä graafisilla työkaluilla olisi mahdotonta, mutta erittäin helppoa ohjelmoimalla parilla if-lauseella.

2.4 3D-mallin integrointi osaksi laajempaa projektia

Käytettäessä CAD-työkalua, joka hyödyntää vakiintunutta ohjelmointikieltä, kuten esimerkiksi CadQuerya, joka on Python-kirjasto, mahdollistaa se 3D-mallin helpomman integraation osaksi suurempaa kokonaisuutta. Tässä projektissa esimerkiksi toteutan mallille web-käyttöliittymän, josta sitä on mahdollista esikatsella, muokata parametreja ja ladata mallista vietyjä tiedostoja. Mielestäni kiinnostavia käyttökohteita olisi myös automaatioputken luominen, jossa luoduille 3D-malleihin ajettaisiin erilaisia rakenteellisia testejä. Toisaalta taas myös ohjelmistokirjaston ominaisuuksien laajentaminen kokonaan uusilla omiin tarpeisiin sopivilla ominaisuuksilla on myös mahdollista ja verrattain helppoa.

2.5 Versiohallinta

Tuotemuotoiluprosessi on luonteeltaan iteratiivinen (Milton ja Rodgers 2011, 18). Toimiessani huonekalumuotoilijana huomasin, että tämä johtaa myös tarpeeseen tehdä muutoksia 3D-malliin projektin edetessä. Mielestäni tarve on hyvin saman kaltainen kuin mitä ohjelmoitaessakin. Asiakkaan tarpeet tarkentuvat, tai oma näkemys parhaasta tavasta toteuttaa jokin asia muuttuu, jolloin syntyy tarve tehdä muutoksia koodiin tai 3D-malliin.

Tämä luo tarpeen projektin aiempien versioiden tallentamiselle ja hallitsemiselle sekä mahdollisuudelle katselmoida tehtyjä muutoksia. Entistä tärkeämmäksi nämä ominaisuudet muodostuvat, jos projektissa on useampi osallistuja. Ohjelmistokehityksessä tähän tarkoitukseen on kehitetty erilaisia versiohallintatyökaluja, kuten Git, SVN ja Mercurial.

3D-mallien kanssa työskentelyyn ei vastaavia vakiintuneita versionhallintatyökaluja ole, vaikka joitain tämän suuntaisia työkaluja onkin kehitteillä. Esimerkiksi GitHub pystyy näyttämään muutoksen kahden STL-tiedoston välillä (Skalnik 2013). Se ei kuitenkaan mahdollista muutoshistorian säilyttämistä vain tiedostojen väliset erot tallentamalla, vaan jokainen versio on oma kokonainen tiedostonsa ja työkalu tukee vain STL-tiedostoja (GitHub Docs. n.d.). STL-tiedostoformaatti on itsessään myös erittäin rajoittunut. STL tiedosto on lista kolmikulmaisia

pintoja (engl. facet), jotka muodostavat yhdessä kappaleen pinnan (The STL Format. n.d.). Näin ollen se ei mahdollista edes oikeasti pyöreiden muotojen, saati sitten minkäänlaisen suunnitteluaikeen tallentamista.

Koska skriptipohjaiset työkalut mahdollistaisivat 3D-mallien tallentamisen koodina tekstimuodossa, mahdollistavat ne ohjelmistokehityksessä käytettyjen ja hyväksi todettujen versiohallintatyökalujen käyttämisen myös 3D-malleille.

2.6 3D-mallien jakaminen

Graafiset CAD-työkalut käyttävät yleensä omaa kyseiselle ohjelmistolle luotua tiedostotyyppiään. Tämä on ongelmallista tiedostojen jakamisen kannalta. Pystyäkseen katselemaan ja muokkaamaan tiedostoa siten että suunnitteluaie olisi säilynyt mallissa, täytyisi vastaanottajalla olla lisenssi ohjelmistoon, jolla tiedosto on luotu.

Ongelman pystyy osittain kiertämään viemällä tiedoston johonkin yleisesti käytettyyn muotoon kuten STEP. Monet CAD-ohjelmistot pystyvät myös muuntamaan toisten yleisimmin käytettyjen CAD-työkalujen tiedostoja kyseiselle alustalle. Näiden molempien tapojen ongelmana kuitenkin on, että useimmiten tiedostoa muuntaessa kaikki tieto suunnitteluaikeesta menetetään. Vapaan lähdekoodin skriptipohjaisen työkalun hyödyntäminen mahdollistaa kaiken 3D-malliin liittyvän tiedon helpon jakamisen.

3 TOTEUTUKSEN VAATIMUKSET

Tässä luvussa kuvailen tarkemmin tavoitteitani ja niistä syntyviä vaatimuksia tämän työn lopputuotteelle. Tavoitteita käsittelen ensimmäisessä alaluvussa ja seuraavissa alaluvuissa erittelen vaatimukset, jotka nämä tavoitteet asettavat luotavalle 3D-mallille ja web-käyttöliittymälle.

3.1 Tavoitteet lopputuotteelle

Tavoitteenani on luoda web-työkalu, jossa käyttäjä pystyy kustomoimaan huonekalujalan kiinnikkeen, esikatselemaan kiinnikettä esimerkkihuonekalun osana ja lataamaan tarvittavan 3D-mallin kiinnikkeiden tulostamiseksi. Syy juuri tällaisen 3D-mallin ja web-työkalun valintaan esimerkkiprojektiksi skriptipohjaisiin CAD-työkaluihin tutustumiseen on puhtaasti oma mielenkiintoni huonekalujen valmistusta kohtaan.

Työkalu on tarkoitettu tee-se-itse harrastajille. Tarkoituksena on mahdollistaa huonekalun valmistaminen vähillä työkaluilla ja nopeasti hyödyntämällä 3D-tulostusteknologiaa. Työkalun ydin ovat CadQueryn tuoma mahdollisuudet 3D-mallin kustomointiin ja 3D-tulostettavajalan kiinnike, jonka ansiosta jalan kiinnitys tukevasi haluttuun kulmaan onnistuu helposti. Näistä tavoitteista syntyvät vaatimukset itse 3D-mallille ja web-käyttöliittymälle, joista kerrotaan seuraavissa alaluvuissa.

3.2 Vaatimukset 3D-mallille

Malli koostuu 3D-tulostettavaksi tarkoitetusta jalan kiinnikkeestä, kansiosasta ja jaloista. 3D-mallin tulee olla loppukäyttäjän kustomoitavissa. Malli esittää käyttäjän valitsemista mitoista riippuen yksinkertaista istuinta tai pöytää. Jotta 3D-mallin kustomointi olisi parametreja muuttamalla loppukäyttäjälle mahdollista, täytyy mallin vastata hyvin suunnitteluaietta.

3D-mallissa tulisi pystyä parametrisesti muuttamaan:

- huonekalun korkeutta jalan pituutta muuttamalla
- jalan paksuutta

- jalan kiinnityskulmaa
- pöytälevyn tai istuimen leveyttä, pituutta ja korkeutta

3.3 Vaatimukset web-käyttöliittymälle

Web-käyttöliittymän tulee mahdollistaa 3D-mallin muokkaaminen ja esikatselu, sekä piirustusten ja 3D-mallien lataaminen omalle koneelle. 3D-mallinmuokkaaminen tapahtuu muuttamalla mallin parametreja. Web-käyttöliittymän tulisi mahdollistaa tämä tarjoamalla liukusäätimiä ja tekstikenttiä, joilla parametreja pystyy muuttamaan.

Tavoitteenani on saada 3D-mallin esikatselusta havainnollisen lisäksi visuaalisesti miellyttävä. Haluaisin saada esikatseluun tekstuurit 3D-mallille, esimerkiksi puun syykuvion puuosiin, ja muuten pelkistetyn ilmeen. Esikatselulle tulevat silti olemassa olevat työkalut asettamaan reunaehdot, sillä opinnäytetyö puitteissa ei ole mahdollista alkaa kehittämään omaa erillistä työkalua tähän tarkoitukseen. 3D-mallin esikatselun tueksi on tarkoituksenani liittää työkaluun muutama esimerkkivalokuva valmistetuista huonekaluista.

Web-työkalusta tulisi pystyä lataamaan käyttäjän omalle tietokoneelle 3D-malli jalan kiinnikkeestä STL-tiedostoformaatissa ja piirustukset dxf, svg tai muussa yleisesti käytetyssä muodossa. Käyttäjälle tulisi jäädä myös tieto käytetyistä asetuksista. Tämä voitaisiin toteuttaa esimerkiksi Json-tiedostolla tai generoimalla erityinen linkki käytetyistä asetuksista.

4 PARAMETRISEN 3D-MALLIN TOTEUTUS

Tässä luvussa käydään ensin läpi tärkeimmät työkalut, joita käytin 3D-mallin luomiseksi ja perustelen hieman miksi päädyin juuri niihin. Toisessa alaluvussa kuvailen vaihevaiheelta, miten ohjelmoin 3D-mallin kustomoitavalle huonekalujalan kiinnikkeelle.

4.1 Työkalut

CadQuery

Projektin tärkein työkalu on ohjelmistokirjasto, jolla itse 3D-malli luodaan. Siksi avaan tässä alaluvussa hieman tarkemmin, kuin muiden työkalujen kohdalla, miksi päädyin juuri CadQueryyn.

Aiemmin esimerkkeinä mainitsemistani skriptipohjaisista CAD-työkaluista OpenScadista, CadQuerysta ja CascadeStudiosta, OpenScad vaikuttaa ylivoimaisesti eniten käytetyltä hakukoneilla löytyvien tulosten perusteella. Tarkkoja tietoja käyttäjämääristä ei ole saatavilla, mutta suuntaa antanevat GitHubissa repositorioille annetut tähdet. OpenScadin repositoriolla kirjoitushetkellä niitä on noin neljätuhatta, kun CadQuerylla ja CascadeStudiolla niitä on lähempänä neljää sataa. Tähtiä käytetään GitHubissa mielenkiintoisten repositorioiden merkkaamiseen (GitHub Docs. n.d.). OpenScadin suosiosta huolimatta, päädyin valitsemaan työkaluksi tähän projektiin CadQueryn.

CadQuery on Python kirjasto parametristen 3D-mallien luontiin. Dokumentaatiossa kirjaston tavoitteiksi mainitaan:

- mallien rakentaminen vakiintuneella ohjelmointikielellä mahdollisimman samankaltaisesti kuin ihminen kuvailisi mallia
- parametristen loppukäyttäjän helposti kustomoitavien mallien luominen
- laadukkaiden CAD-tiedostoformaattien kuten esimerkiksi STEP ja AFM tukeminen
- vapaan lähdekoodin tekstiformaatin tarjoaminen 3D-malleille

Taustalla CadQuery käyttää OpenCascade mallinnus kerneliä. (CadQuery n.d.)

Päädyin CadQueryyn paljon yleisemmin käytetyn OpenScadin sijaan, koska CadQuery käyttää Python ohjelmointikieltä, kun taas OpenScadissa on oma erityinen kielensä (CadQuery n.d.; OpenScad n.d.). Koin että Mahdollisuus käyttää vakiintunutta ohjelmointikieltä on erittäin tärkeää, sillä se mahdollistaa olemassa olevien työkalujen paremman hyödyntämisen koodin käsittelyssä, kuten esimerkiksi PyCharm-editorin käytön kaikkien sen ohjelmoijalle tarjoamien apuvälineiden kera. Luonnollisesti Laajassa käytössä oleva vakiintunut kieli on myös paljon monipuolisempi, ja hiotumpi kuin yksittäistä käyttötarkoitusta varten luotu erillinen kieli. Python mahdollistaa myös kaikkien muiden olemassa olevien Python-kirjastojen saumattoman hyödyntämisen CadQuery malleissa.

CadQueryn taustalla toimiva Open Cascade 3D-mallinnus kerneli käyttää boundary representation (B-rep) -mallia kappaleiden määrittelyyn (Open CASCADE Technology. Introduction. n.d.). Tämä on toinen merkittävä CadQueryn puolesta puhuva tekijä, koska B-rep-malli mahdollistaa huomattavasti monipuolisemman tavan 3D-mallien luomiseen verrattuna OpenScadin käyttämään constructive solid geometry -malliin.

Myös Machado, Malpica ja Borromeo (2019, 25-26) päätyvät vertailussaan siihen lopputulokseen, että vaikka CadQuerylla mallintamisen aloittaminen on hieman haastavampaa ovat hyödyt kuitenkin OpenScadiin verrattuna niin merkittävät, että he suosittelevat CadQueryn käyttöä vapaan lähdekoodin tieteellisten välineiden suunnitteluun. Joskin he vertailivat CadQuery 1:stä, joka toimi FreeCad ohjelmiston mukana. Heidän mainitsemistaan hyvistä puolista, vakiintuneen ohjelmointikielen hyödyntäminen ja mahdollisuus viedä mallit parametrisiin standardi CAD-formaatteihin kuten esimerkiksi STEP, ovat edelleen paikkansa pitäviä projektissa käyttämäni CadQuery 2:n kohdalla (Machado, Malpica ja Borromeo 2019, 25-26; CadQuery n.d.).

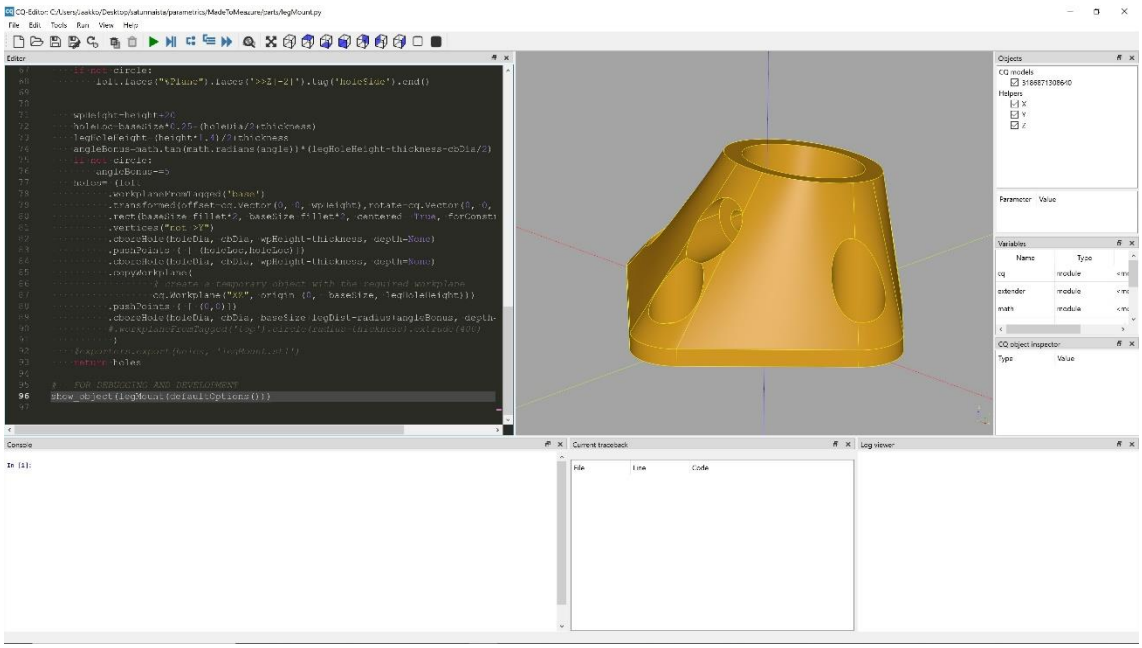
PyCharm

Päädyin käyttämään projektin koodin kirjoittamisessa JetBrainsin Pycharm-ohjelmointiympäristöä. Valitsin sen, koska kyseinen ohjelmisto on minulle entuudestaan tuttu. Tämä jätti vielä tarpeen työkalulle 3D-mallin esikatseluun.

CQ-editor

CadQuery tarjoaa, oman CQ-editor nimisen työpöytäsovelluksen skriptien muokkaamiseen ja 3D-mallin esikatseluun. Kuten aiemmin mainittiin CadQuery 2 on itsessään puhdas Python kirjasto, joten CQ-editor on erillinen projekti. (CadQuery n.d.)

Vaikka päädyin itse koodin editoinnissa käyttämään PyCharmia hyödynsin työskentelyssäni CQ-editoria 3D-mallin yksittäisten osien esikatseluun ja virheiden etsimiseen.



Kuva 2. CQ-editorin näkymä

CQ-editorissa oletuksena vasemmalle avautuu koodieditori ja sen viereen oikealle puolelle näkymä mallin esikatseluun. Alareunassa ja laitimmaisena oikealla, on työkaluja virheiden etsimiseen ja selvittämiseen, kuten esimerkiksi Python konsoli ja lokien katselija (engl. log viewer) sekä työkaluja mallin tarkempaan tutkimiseen kuten esimerkiksi objektilistaus, josta voi valita ja piilottaa haluamansa osat.

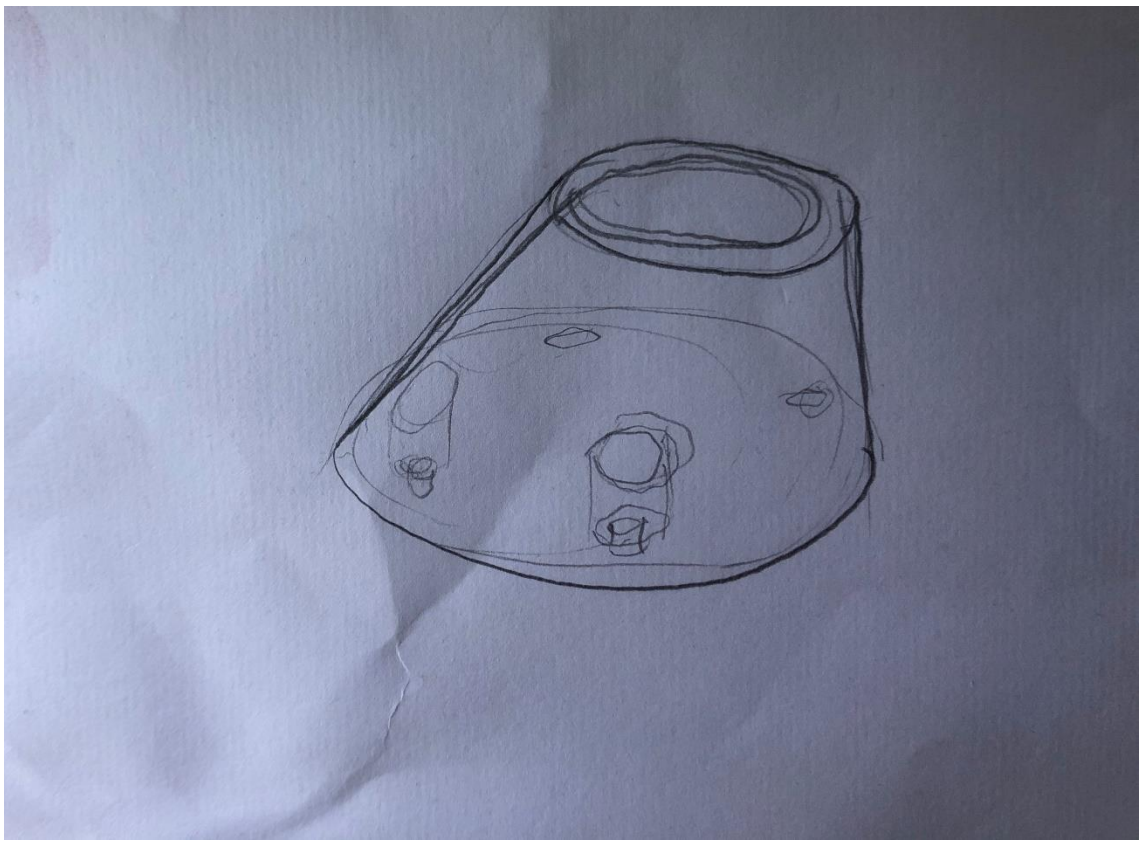
Jupyter Cadquery

Käytin 3D-mallin esikatseluun CQ-editorin lisäksi myös Jupyter Labia sekä Jupyter Cadquery - ja Voilá-lisäosia. Näistä työkaluista kerrotaan enemmän web-käyttöliitttymän toteutusta käsittelevässä osassa, koska työkalut liittyvät Python koodin jakamiseen ja datan esittämiseen. Nämä työkalut mahdollistivat 3D-mallin ja webkäyttöliittymän esikatselun internetselaimessa ja front end -ohjelmistokehitystä muistuttavan työskentelytavan.
Tein haluamani muutokset johonkin projektin Python-tiedostoista, minkä jälkeen päivitin selaimessani näkyvän Voilán avulla julkaistun Jupyter Notebook -sivun, jolloin muutokset päivittyivät näkyvään 3D-malliin.

4.2 Workflow ohjelmoitaessa 3D-mallia

Tässä kappaleessa käyn läpi, miten 3D-mallin luominen ohjelmoimalla käyttäen CadQuery-kirjastoa käytännössä tapahtuu. Esimerkiksi valitsin projektista 3D-mallin 3D-tulostettavasta huonekalujalankiinnikkeestä. Luotaessa 3D-mallia ohjelmoimalla, työskentely jakaantuu selkeästi kahteen osa-alueeseen: koodin muokkaamiseen ja 3D-mallin esikatseluun. Tässä tekstissä prosessiin perehdytään käymällä koodia pala palalta läpi ja kuvien avulla esitetään, miten nämä komennot vaikuttavat luotavaan 3D-malliin.

Ennen mallintamisen aloittamista minulla oli suunnilleen ajatus siitä, millainen kiinnikkeen tulisi olla.



Kuva 3. Karkea luonnos kiinnikkeestä.

Yllä olevasta luonnoksesta näkyy ajatukseni ruuveilla kiinnitettävästä kiinnikkeestä, joka kapenee jalkaa kohti. Luonnos on piirretty asennossa, jossa kiinnike tulostettaisiin. Käytössä kiinnike asennettaisiin pohjassa näkyvistä rei'istä vaikkapa jakkaran istuinosan pohjaan ja jalka kiinnitettäisiin päällä näkyvään suureen pyöreään reikään, eli käytössä kiinnike on 180 käännettynä astetta ympäri niin että päällä näkyvä iso reikä on alaspäin.

Kuvan 3 luonnoksen pohjalta aloin mallintamaan kiinnikettä. CadQuery-kirjastolla mallintaminen tapahtuu suurimmaksi osaksi CadQuery-objekteja ketjuttamalla. Objekti sulkee sisäänsä informaation luodusta geometriasta ja jokainen uusikomento palauttaa uuden CadQuery-objektin, joka sisältää viittauksen edelliseen objektiin (CadQuery n.d.).

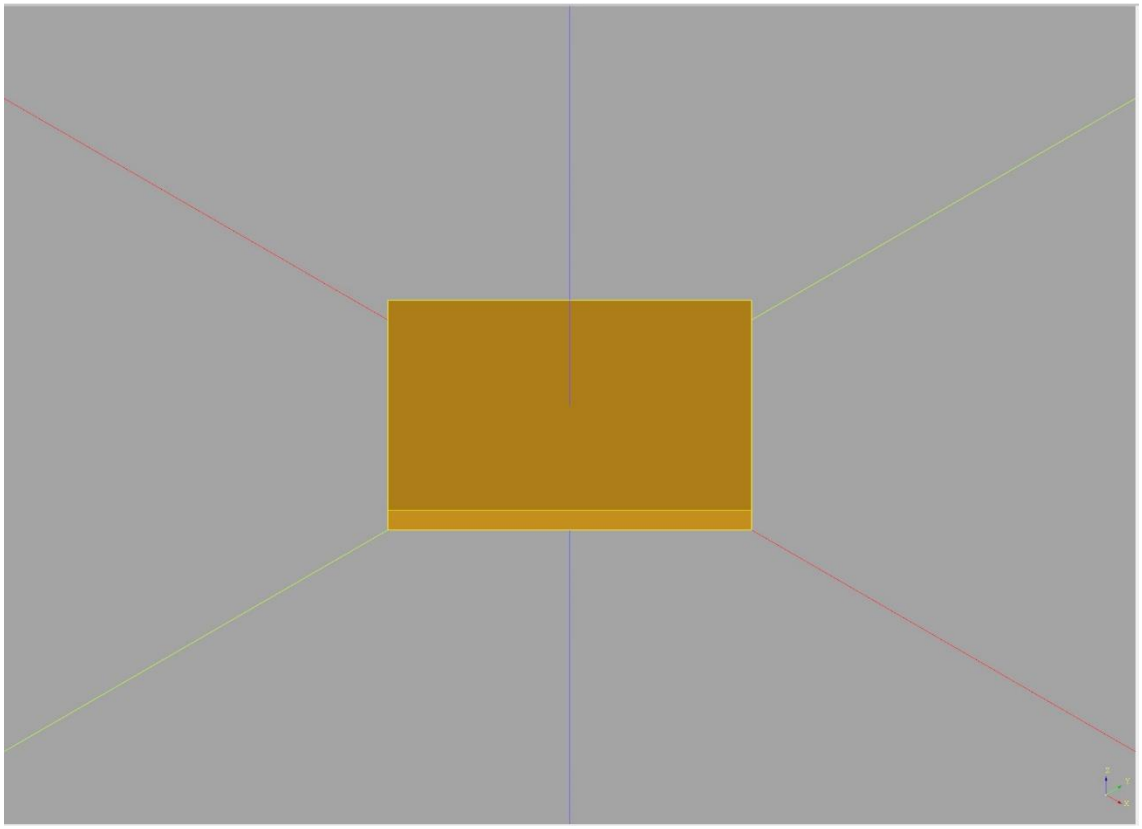
Ensin luodaan pohja osalle:

```
base = (extender.Workplane("XY").tag('base')
          .rect(baseSize, baseSize, centered=True)
          .extrude(thickness)                                    (1)
          .rotate((0, 0, 0), (0, 0, 1), 45)
          )
```

Koodissa muuttuja *extender* on itseluomani Python moduuli, joka lisää CadQueryyn muutaman tarvitsemani funktion. Palaan näihin itse luomiini funktioihin vielä myöhemmin, mutta jos toisin ei mainita, kaikki esimerkissä käytetyt komennot ovat CadQuery-kirjastoon sisältyviä.

Ensin luon koodissa työskentelytason (engl. workplane). Työskentelytasokonseptia käytetään monissa CAD-ohjelmistoissa. Siinä määritellään kaksiulotteinen taso kolmiulotteisessa tilassa, jolle muotoja aletaan luomaan. Seuraava komento *tag* mahdollistaa työskentelytason nimeämisen, jotta minun on helppo viitata siihen myöhemmässä koodissa. Seuraavalla komennolla *rect* luon suorakaiteen ja pursotan (engl. extrude) sen ylöspäin, eli luon kolmiulotteisen kappaleen, jolla on luodun suorakaiteen profiili ja pursotuksessa määritelty korkeus. Lopuksi käännän luotua kappaletta vielä 45 astetta z-akselin suhteen.

Näillä komennoilla loin yksinkertaisen kuvassa 4 näkyvän levyn. Kuvassa sininen viiva osoittaa z-akselin, vihreä x-akselin ja punainen y-akselin suunnan.



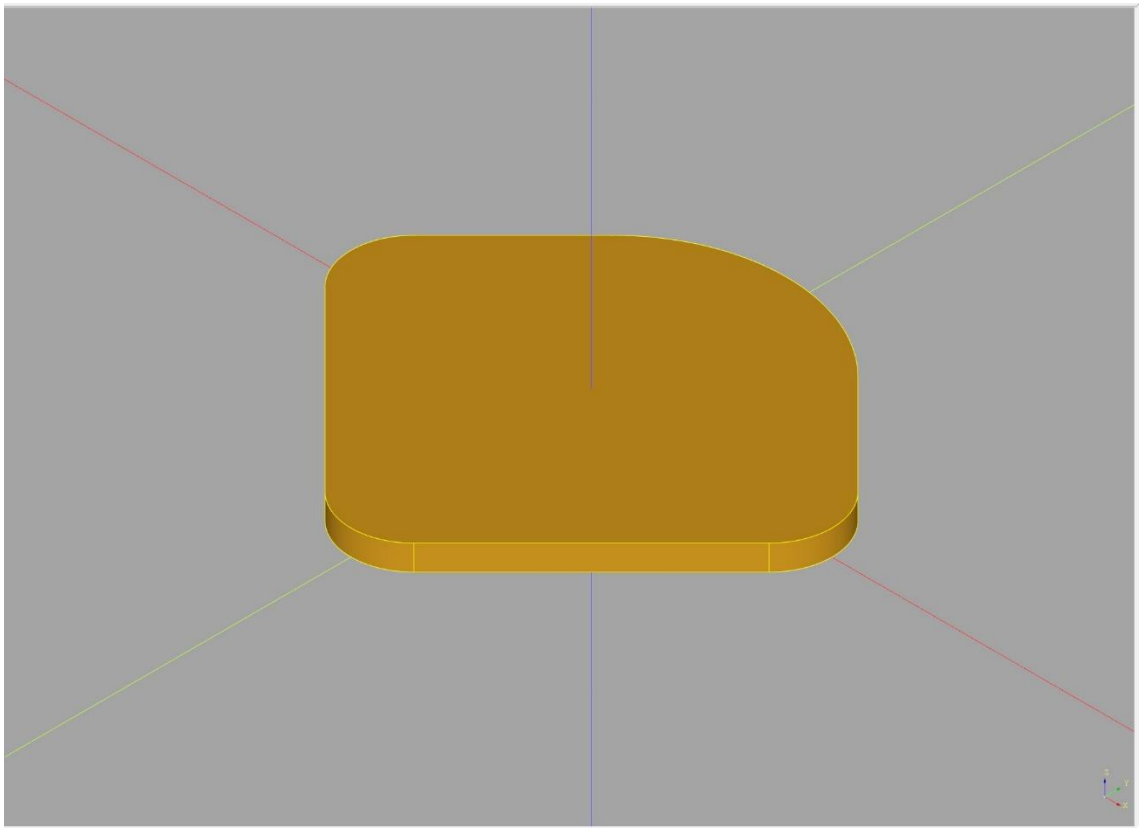
Kuva 4. Esimerkki koodilla luotu kolmiulotteinen kappale.

Esimerkki koodi luo siis yksinkertaisen suorakulmaisen särmiön. Koodissa käytetyt operaatiot, suorakulmion luonti työtasolle, pursotus ja kappaleen kääntäminen ovat CAD-ohjelmistojen perustoimintoja, jotka löytyvät lähes kaikista 3D-mallinnusohjelmistosta. Tässä ne vain suoritetaan kirjoittamalla komentoja graafisen käyttöliittymän klikkailun sijaan. Seuraavassa vaiheessa kappaleen kulmat pyöristetään:

```
filletedBase = (base
        .edges("(|Z and (not >Y))")
        .fillet(fillet)                                    (2)
        .edges("(|Z and (>Y))")
        .fillet(radius+10 if circle else fillet)
        )
```

Aiemmin luodusta objektista *base* valitaan särmät, jotka ovat z-akselin suuntaiset ja eivät kauimpana y-akselin suuntaan. Tämä tapahtuu reunojen valinta komennolla *edges*.

Kun halutut särmät on valittu kyselyllä, ne pyöristetään säteellä, joka on määritelty muuttujassa *fillet*. Tämän jälkeen valitaan vielä kyselyllä "(|Z and (>Y))" edellisestä valinnasta pois jätetty särmä, joka on kauimmaisena y-akselin suuntaan. Viimeisen särmän kohdalla pyöristyssäde määritellään if-lauseessa. Jos *circle* on totta, pyöristä säteellä *radius*+10 mm muuten käytä muuttujassa *fillet* määriteltyä arvoa.



Kuva 5. Kappale pyöristyksien jälkeen

Kuvan 5 3D-malli esittää pyöreän jalan pidikkeen pohjaa, eli tilannetta jossa muuttuja *circle* on tosi. Koodikatkelmassa (2) oikeiden särmien valintaan käytettin CadQuery-kirjastolle uniikkeja selektoreita kuten esimerkiksi: *"(|Z and (not >Y))".* Näillä selektoreilla pystytään valitsemaan tiettyjä osia kappaleen geometriasta, jotta niitä voidaan käsitellä koodissa.

Selektori koostuu geometrian ominaisuuksia kuvaavista ilmaisuista ja propositiologiikasta. Geometriaa kuvaavia ilmaisuja esimerkki-ilmaisussa *"(|Z and (not >Y))"* ovat: z-akselin suuntaista tarkoittava |Z ja kauimpana y-akselilla tarkoittava >Y. Propositiologiikkaa sen sijaan ovat konjunktio *and* ja negaatio *not*.

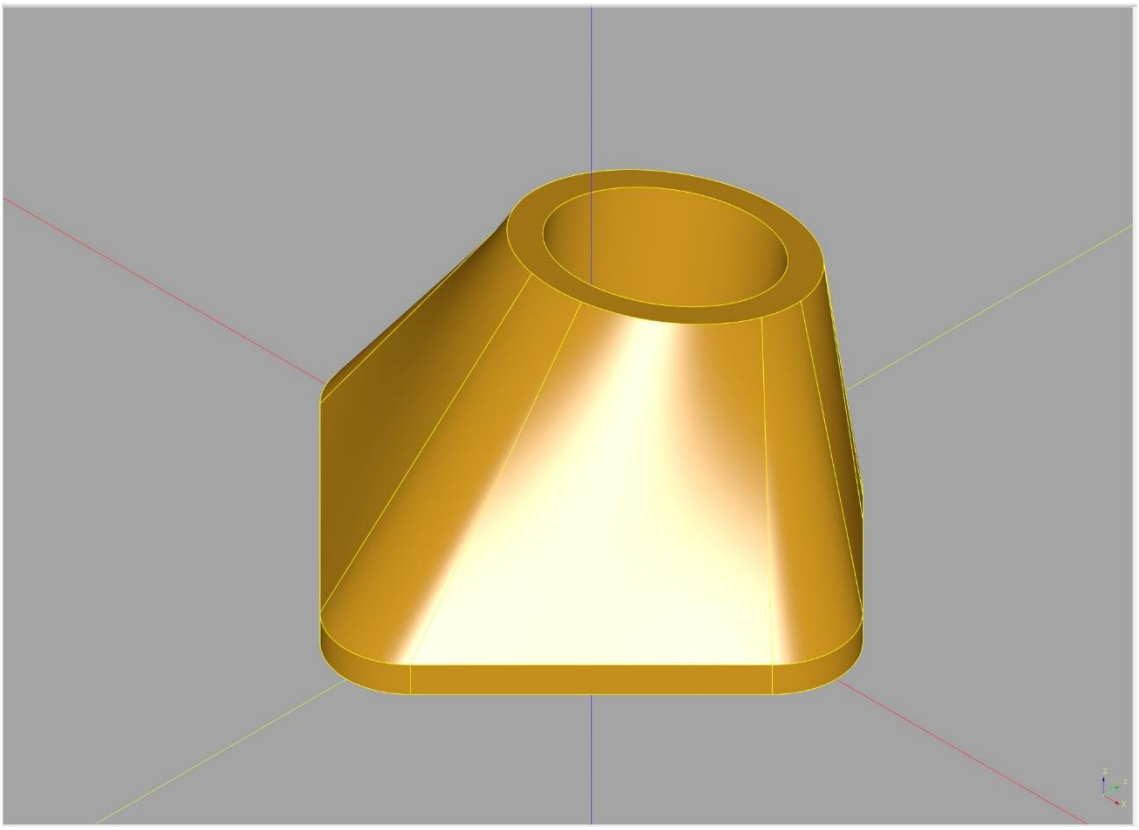
Mainittu esimerkki selektori siis valitsee särmät, jotka ovat z-akselin suuntaiset ja eivät kauimpana y-akselin suuntaan. Graafisissa käyttöliittymissä ei tällaisia selektoreja tarvita, koska käyttäjä pystyy tyypillisesti valitsemaan muokattavan osan esikatseltavasta 3D-mallista klikkaamalla. Osien valitseminen graafisesta käyttöliittymästä on huomattavasti nopeampaa ja intuitiivisempaa. Vaatimus propositiologiikan ymmärtämisestä ja selektorien käyttämien ilmaisujen osaamisesta kasvattaa myös kynnystä työkalun käyttöön. Toisaalta taas selektorin käyttäminen tekee valinnasta eksplisiittisen ja sitä kautta mahdollistaa paremman suunnitteluaikeen esittämisen luodussa mallissa ja poistaa aiemmin tässä työssä mainitun epämääräisten ratkaisuheuristiikkojen aiheuttamat ongelmat.

Lopussa käytettiin pyöristyssäteen määrittelyyn ehtolausetta, mikä ei olisi perinteisillä CAD-työkaluilla mahdollista ja on hyvä esimerkki monipuolisemmista mahdollisuuksista suunnitteluaikeen märittelyyn 3D-mallissa, jotka skriptipohjaiset CAD-työkalut avaavat.

Kun pyöristykset on tehty, seuraavassa alla näkyvässä koodissa kiinnikkeen pohjan päälle luodaan muoto, joka kapenee ylöspäin kohti huonekalujalan läpileikkauksen muotoa. Tavoitteena on tukea jalkaa ja luoda vakaa kiinnike.

```
loft = (filletedBase
          .faces(">Z")
          .wires()
          .toPending()
          .workplane()
          .transformed(offset=cq.Vector(0, legDist, 0))
          .transformed(rotate=cq.Vector(-angle, 0, 45))
          .transformed(offset=cq.Vector(0, 0, height))
          )
if circle:                                                    (3)
          loft = (loft.circle(radius).tag('top')
                   .loft(combine=True)
                   .faces(">Z")
                   .hole(diameter=legDiameter,
                    depth=legHoleDepth))
else:
            loft = (loft.rounded_rect(radius*2, radius*2,
                   fillet/4)
                   .loft(combine=True)
                   .faces(">Z")
                   .rect(legDiameter, legDiameter)
                   .cutBlind(-legHoleDepth, True)
                   )
```

Koodissa (3) ensin valitaan luodun kappaleen *filletedBase*, ylin siis korkeimpana z-akselilla oleva tahko ja tämän jälkeen valitaan vielä valitun tahkon reunat komennolla *wires*. Seuraavaksi luodaan uusi työskentelytaso, joka siirretään luodun kappaleen yläpuolelle kohtaan, johon kiinnikkeen yläreuna halutaan ja käännetään jalalle haluttuun kiinnityskulmaan. Tälle työtasolle luodaan muoto, johon luotu pohja halutaan yhdistää. If-lauseella tarkastetaan jälleen, ollaanko luomassa kiinnikettä pyöreälle vai neliskanttiselle jalalle ja työtasolle luodaan muoto sen mukaan. Komennolla *loft* luodaan kiinteä kappale, joka kapenee tasaisesti pohjanylimmän tahkon reunoista työtasolle luodun muodon reunoihin. Tämän jälkeen kappaleeseen luodaan vielä joko pyöreä tai neliskanttinen reikä johon jalka kiinnitetään.



Kuva 6. Kappale johon on lisätty ylöspäin kapeneva tukirakenne ja jalan kiinnitysreikä.

Kuvan mallissa kiinnitysreikä on luotu pyöreälle jalalle. Koodissa (3) hankaluuksia tuotti pohjan ylimmän tahkon reunojen valinta *loft*-komentoa varten. Loft on yleisesti CAD-työkaluissa käytetty komento ja se luo pinnan tai kiinteän kappaleen kahden reunan välille. Kuvan 6 tapauksessa yläosan ympyrän muotoisen tason

ulkoreunan ja aiemmin luodun pohjalevyn yläpinnan reunojen välille. Komentoa varten tulisi siis pystyä valitsemaan reunat kahdesta eri kappaleesta. Yksittäisen reunan valitseminen on melko suoraviivaista aiemmin mainituilla selektoreilla, mutta dokumentaatiosta ja sen tarjoamista esimerkeistä oli hankala selvittää, miten kaksien erillisten reunojen valitseminen yhtä komentoa varten onnistuu. Sain koodin toimimaan käyttämällä komentoa *toPending.*

Tämän kaltaiset hankaluudet ovat mielestäni osa suurempaa ongelmaa CadQueryssa. Se, mitä osaa ollaan muokkaamassa, mitä sille voi tehdä ja miten pitäisi olla havainnollisempaa. Tätä voitaisiin parantaa dokumentaatiota parantamalla, informatiivisemmilla virheviesteillä ja mahdollisesti parantamalla CQ-editorin ohjelmoijalle tarjoamia työkaluja.

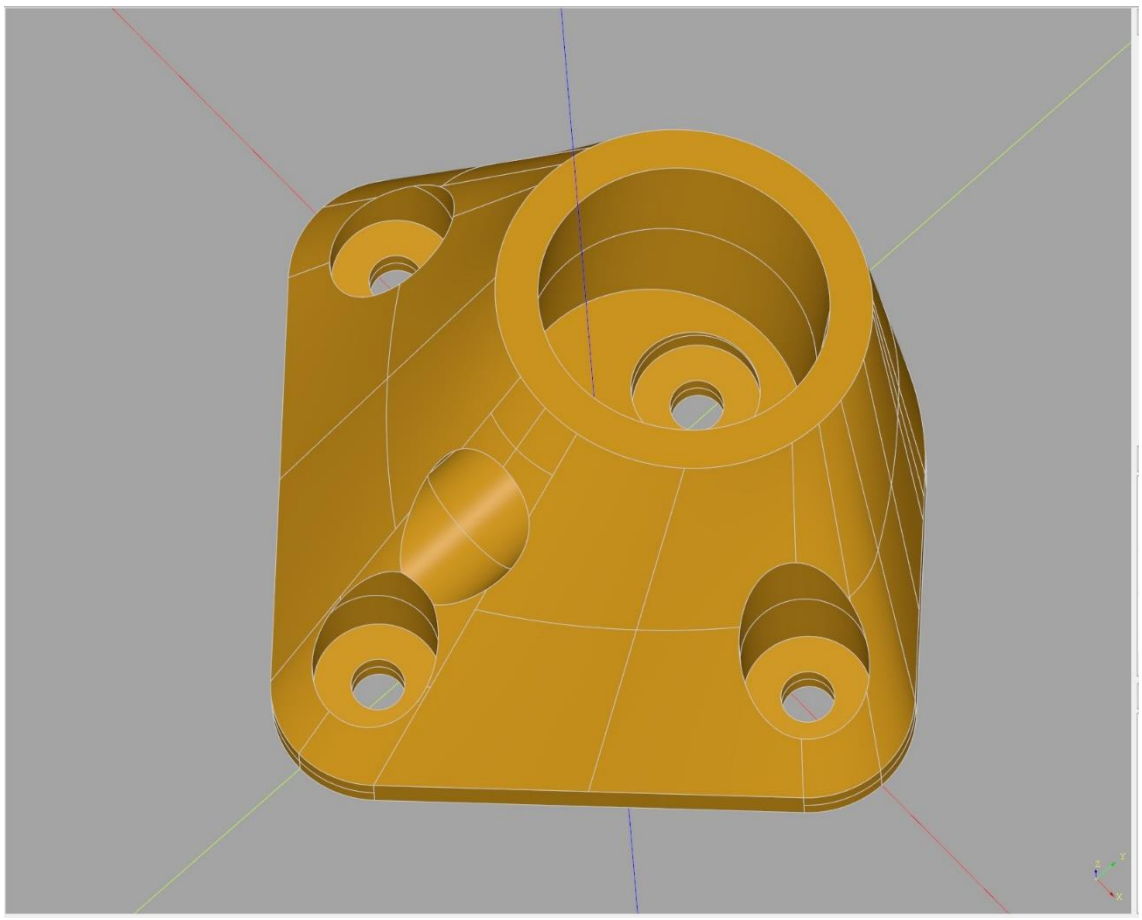
Toinen huomion arvoinen asia koodissa (3) on funktio *rounded_rect,* jolla luon suorakaiteen pyöristetyillä kulmilla tapauksessa, jossa luodaan kiinnike neliskanttiselle jalalle rivillä 17. Lisäsin tämän komennon, tai tarkemmin sanottuna funktion, luomalla oman moduulin, joka periytyy CadQuerysta. Mahdollisuus helposti laajentaa kirjaston toiminnallisuutta luomalla omia funktioita on ominaisuus, josta uskon lähes kaikkien CAD-ohjelmistoja ammattimaisesti käyttävien tahojen hyötyvän. Toisaalta taas se, että CadQueryssa ei valmiiksi ole funktiota suorakulmion luomiseen pyöristetyillä kulmilla kertoo myös siitä, että työkalusta puuttuu vielä monia CAD-työkaluissa tavallisia ominaisuuksia.

Kun olin saanut kiinnikkeen muodon mallinnettua, tuli siihen lisätä vielä reiät ruuveille. Loin malliin neljä reikää kiinnikkeen kiinnittämiseksi jakkaran tai pöydän kanteen ja yhden, jolla pystytään varmistamaan jalan kiinnitys ruuvilla:

```
holes= (loft
        .workplaneFromTagged('base')
        .transformed(offset=cq.Vector(0, 0, wpHeight),
                rotate=cq.Vector(0, 0, 45))
        .rect(baseSize-fillet*2, baseSize-fillet*2,
                centered= True, forConstruction=True)
        .vertices("not >Y")
        .cboreHole(holeDia, cbDia, wpHeight-thickness,
                depth=None)
        .pushPoints ( [ (holeLoc,holeLoc)])
        .cboreHole(holeDia, cbDia, wpHeight-thickness,
                depth=None)
        .copyWorkplane(                                    (4)
    # create a temporary object with the required workplane
                cq.Workplane("XZ", origin=(0, -baseSize,
                        legHoleHeight))
                )
        .pushPoints ( [ (0,0)])
        .cboreHole(holeDia, cbDia,
                baseSize+legDist-radius+angleBonus,
                depth=baseSize+legDist)
        )
```

Koodissa (4) jatketaan työskentelyä aiemmin luodun kappaleen parissa. Ensin rivillä 2 valitaan koko kappaleen pohja työtasoksi. Se on aiemmassa koodissa (1) ensimmäisellä rivillä merkattu tagilla ”base”. Tämän jälkeen työtasoa nostetaan rivillä 3, koska ruuvien kannoille luodaan ylhäältä päin isompi upotusreikä, joka viistää ylöspäin nousevaa seinämää, on reikä helpoin luoda siten että se aloitetaan reilusti koko kappaleen yläpuolelta. Noston jälkeen työtasoa käännetään vielä 45 astetta, z-akselin suhteen rivillä 4. Tämän jälkeen luodaan neliö, joka on hieman kiinnikkeen pohjaa pienempi. Rivillä 7 tästä neliöstä valitaan muut kulmat, paitsi kauimmaisena y-akselilla oleva käyttäen komentoa *vertices* ja "not >Y" selektoria. Näin saadaan määriteltyä pisteet kolmelle kiinnitysruuvin paikalle kiinnikkeen kulmiin, kuten kuvassa 7 näkyy. Valittuihin pisteisiin tehdään reiät, joissa on upotus ruuvin kannalle komennolla *cBoreHole* rivillä 8. Tämän jälkeen määritetään erikseen sijainti neljännelle huonekalun kannen pohjaan menevälle reiälle rivillä 10, joka sijoitetaan siten että ruuvin pystyy ruuvaamaan jalan kiinnitysreiästä.

Lopuksi koodissa (4) luodaan vielä reikä, kiinnikkeen sivuun, josta on tarkoitus pystyä varmistamaan ruuvilla huonekalu jalan kiinnitys kiinnikkeeseen. Tämä tapahtuu luomalla x- ja z-akseleiden suuntainen työtaso rivillä 15 ja siirtämällä se halutulle korkeudelle ja lopuksi rivillä 19 luomalla reikä ruuvikannan upotuksineen tason keskelle samalla *cBoreHole*-komennolla kuin aiemminkin.



Kuva 7. Valmis kiinnike ruuvireikineen

Lopputuloksena syntyy kuvassa 7 näkyvä valmis huonekalujalan kiinnike, jonka kaikkia dimensioita on helppo muokata muuttujien arvoja vaihtamalla. koko mallin geometria ja suunnitteluaie on kuvattu muutamassa kymmenessä rivissä koodia.

5 3D-MALLIN ESIKATSELU -WEB-KÄYTTÖLIITTYMÄN TOTEUTUS

Ensimmäisessä alaluvussa käyn läpi tärkeimmät työkalut, joita käytin web-käyttöliittymän luomiseksi ja perustelen hieman miksi päädyin juuri niihin. Toisessa alaluvussa esittelen luomani web-sovelluksen, kuvailen sen toimintaa ja arvioin toteutuksen onnistumista.

5.1 Työkalut

Jupyter Notebook

Käytän web-työkalun pohjana Jupyter Notebook -dokumenttia. Jupyter Notebook muodosuu kahdesta osasta: web-applikaatiosta dokumenttien luontiin ja Notebook dokumenteista (The Jupyter Notebook. n.d.). Notebook dokumentti yhdistää koodin, kuvien, tekstin, ja erilaisen datavisualisointien esittämisen (The Jupyter Notebook. n.d.).

Notebook dokumenttien muokkaaminen tapahtuu JupyterLab web-käyttöliittymässä. Loin projektille kuitenkin sellaisen arkkitehtuurin, että suurin osa varsinaisesta ohjelmoinnista tapahtuu Notebook dokumenttien ulkopuolella Python-tiedostoissa, jolloin pystyin hyödyntämään ominaisuuksiltaan kattavampaa Pycharm-ohjelmointiympäristöä.

Käytänkin Jupyer Notebookia projektissa lähinnä se tarjoaman web-applikaation julkaisun helppouden vuoksi. Cadquery 3D-mallin pystyy esitämään jupyter-cadquery lisäosalla, tarvittavat käyttöliittymäkomponentit kuten liukusäätimet ja tekstikentät voi luoda Ipywidget-nimisillä valmiilla ui-komponenteilla.

Jupyter-cadquery

Jupyter-cadquery on laajennus, joka mahdollistaa CadQuery-objektien renderoinnin JupyerLabissa (jupyter-cdquery GitHub repositorio. n.d.). Käytännösä jupyter-cadquery madollistaa sen missä tahansa Jupyter Notebook -dokumentissa. Hyödynnänkin jupyter-cadquerya web-köyttöliittymän toteutuksessa 3D-

mallin esikatselukomponenttina. Kuvassa 8 näkyy jakkaran 3D-malli jupyter-cadqueryn renderoimana.

Ipywidgets

Ipywidgetit ovat Python-objekteja, jotka mahdollistavat graafisten interaktiivisten käyttöliittymien luomisen Jupyter Notebook -dokumentteihin (Ipywidgets. n.d.). Käytän ipywidgetteja web-käyttöliitymässä 3D-mallin parametrien säätöön. Kuvassa 8 näkyvät käyttöliittymän kentät jalan halkaisijan, kulman, korkeuden ja kannen halkaisijan säätöön. Päädyin käyttämään tekstikenttiä liukusäätimien sijaan, koska ne tarjoavat tarkemman kontrollin.

Voila

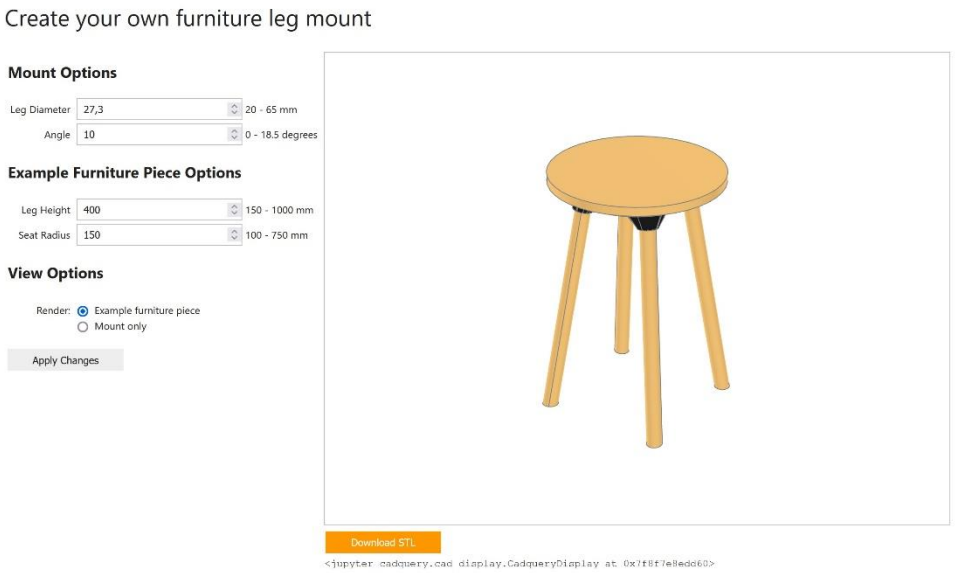
Voilá mahdollistaa Jupyter Notebook -dokumenttien julkaisun itsenäisinä web-applikaatioina (Using Voilá. n.d.). Työn kannalta olennaista on myös, että Voilá-lisäosa mahdollista julkaisun siten, että koodia ei näytetä loppukäyttäjälle, eikä satunnaisen koodin suorittaminen ole mahdollista.

Heroku

Heroku on kontteihin perustuva pilvipalvelualusta web-applikaatioiden julkaisuun (Heroku. n.d.). Julkaisin applikaationi Herokun alustalle. Päädyin käyttämään Herokua siksi, että se oli minulle entuudestaan tuttu.

5.2 3D-mallin esittäminen selaimessa

Kuva 8 on kuvakaappaus luomastani web-applikaatiosta toteuttamani parametrisen 3D-mallin muokkaamiseen. Opinnäytetyön kirjoitushetkellä applikaatio on myös kokeiltavissa osoitteessa: https://made-to-meazure.herokuapp.com/. Applikaatio muodostuu vasemmalla olevista säädöistä, oikealla olevasta mallin esikatselunäkymästä ja esikatselunäkymän alla olevasta latausnapista, josta käyttäjä pystyy lataamaan jalan kiinnikkeen STL-tiedostona omalle koneelleen 3D-tulostettavaksi.



Kuva 8. Käyttöliittymä 3D-mallin muokkaukseen

Halusin pitää säädöt mahdollisimman yksinkertaisina ensimmäiseen versioon käyttöliittymästä. Säädöt on jaettu kolmeen osaan: kiinnikkeeseen liittyviin säätöihin, esimerkkihuonekaluun liittyviin säätöihin ja käyttöliittymän näkymään liittyviin säätöihin.

Kiinnikkeestä käyttäjä pystyy määrittelemään halutun jalan paksuuden ja kiinnityskulman. Nämä ovat mielestäni kaksi ylivoimaisesti tärkeintä säätömahdollisuutta kiinnikkeelle ja siksi mielestäni on luonnollista aloittaa niillä. Kiinnikkeen koko säätyy automaattisesti jalan paksuuden mukaan.

Esimerkki huonekalun, siis huonekalun, joka näkyy käyttöliittymän esikatselunäkymässä, säätöihin kuuluvat jalan korkeus ja kannen halkaisija. Halusin sisällyttää nämä säädöt käyttöliittymän ensimmäiseen versioon, jotta käyttäjä pystyisi helposti arvioimaan mittasuhteita tietyn paksuisella ja levyisellä jalalla.

Näkymään liittyvissä säädöissä on yksinkertaisesti mahdollista valita, haluaako esikatsella pelkkää 3D-tulostettavaa kiinnikettä vai koko huonekalua. Alimpana säätöjen alla käyttöliittymässä on nappi säätöihin tehtyjen muutosten päivittämiseen.

Tiesin jo etukäteen, että opinnäytetyön aikataulurajoitteiden puitteissa, minulla ei olisi resursseja luoda loistavaa käyttöliittymää 3D-mallien esikatseluun internetissä ja tutustua 3D-mallien luomiseen ohjelmoimalla. Olen kuitenkin silti hieman pettynyt lopputulokseen. Toiveenani oli luoda toimiva käyttöliittymä luomani mallin esittelyyn ja palautteen keräämiseen. Käyttöliittymä, jonka sain aikaan, osoittautui kuitenkin käytettävyydeltään sen verran heikoksi, että epäilen sen hankaloittavan palautteen saamista itse 3D-mallista ja ideasta kokonaisuutena, koska huomio kiinnittyy liikaa käyttöliittymän puutteisiin.

Käyttöliittymän suurin heikkous on sen hitaus. Käyttöliittymän latautuminen kestää paikallisesti omalla tietokoneellani noin 15 sekuntia. Muutosten päivittyminen esikatselunäkymään noin kuusi sekuntia. Herokussa pyörivän testiversion latautuminen vie suunnilleen minuutin. Muutosten päivittyminen esikatseluun kestää pidempään kuin paikallisesti noin 20 sekuntia. Oletan tämän johtuvan siitä, että Herokun palvelimilla ilmaisversiona toimivalle sovellukselle on allokoitu vähemmän laskentatehoa, kuin mitä kannettavallani on tarjota sovellukselle.

Nämä odotusajat voisivat olla siedettäviä, jos kyseessä olisi palvelu, joka olisi tehty mittatilauksena yksittäiselle toimijalle esimerkiksi automatisoimaan jokin 3D-mallinnuksen työvaihe. Tällöin asiakas voisi halutessaan maksaa palvelimesta, jossa olisi enemmän laskentatehoa ja olisi luultavammin valmis hyväksymään hieman kankean käytettävyyden, jos palvelu tuo heille muuten lisäarvoa automatisoimalla jonkin aiemmin käsin tehdyn mallinnusvaiheen.

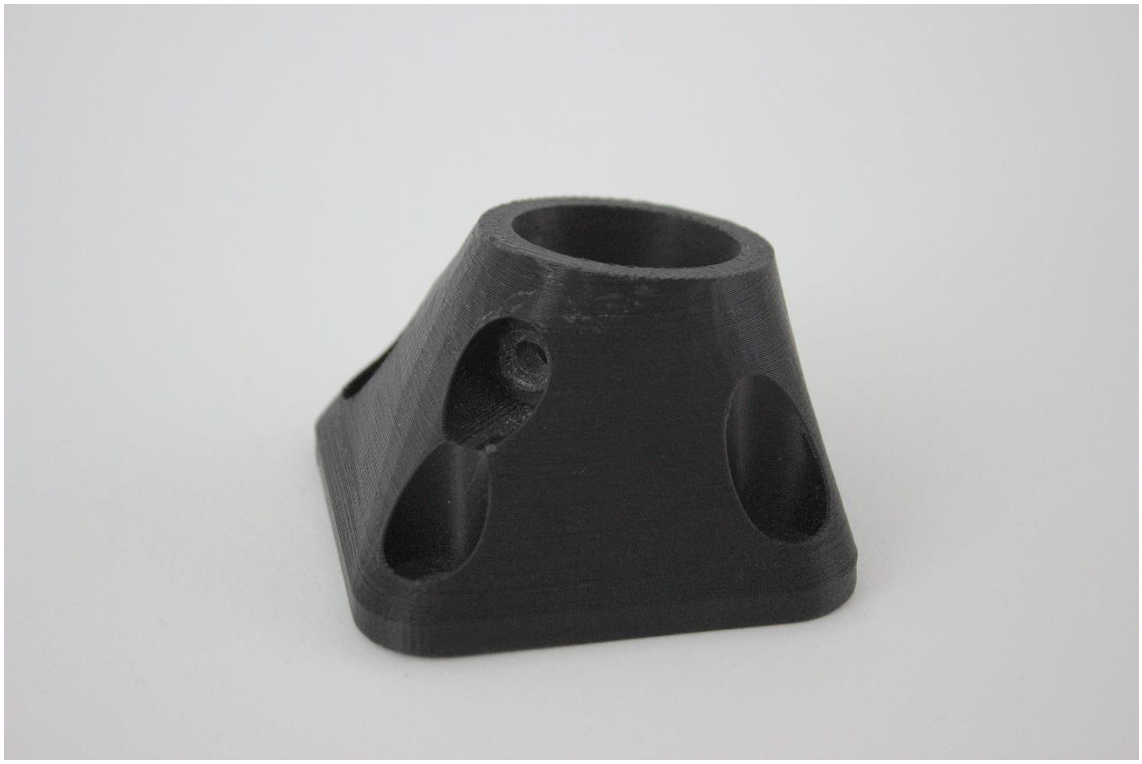
Koska käyttöliittymä on hitaan puoleinen jo yhdellä käyttäjällä, arvioisin että ei vaadittaisi kovinkaan montaa yhtäaikaista käyttäjää ja työkalusta tulisi jo täysin käyttökelvoton. Palvelimen laskentatehon kasvattaminen taas tulisi erittäin kalliiksi, jos palvelu onnistuisi keräämään kohtuullisen määrän käyttäjiä. Mielestäni, jotta tällainen palvelu voisi toimia jouhevasti julkisena yleisesti käytössä olevana työkaluna tulisi latausajat saada merkittävästi lyhyemmiksi.

Käyttöliittymän toteutuksesta jäi myös puuttumaan muutamia ominaisuuksia, jotka olisin halunnut sisällyttää siihen. Minulla ei ollut aikaa implementoida ladatun 3D-mallin luomiseen käytettyjen parametrien välittämistä käyttäjälle 3D-mallin

mukana. Jouduin luopumaan myös kaksiulotteisten rakennepiirustusten generoinnista, kansiosan paksuuden säätömahdollisuudesta ja esikatselun tekstuurien toteutuksesta aikataulullisten haasteiden vuoksi. Pyrin lisäämään näitä ominaisuuksia käyttöliittymän seuraavaan versioon.

6 HUONEKALUN TOTEUTUS

3D-tulostetun osan ja luodun huonekalun ominaisuudet ovat sivuseikkoja tämän opinnäytetyön kontekstissa. Mielestäni luomani kustomoitavan 3D-mallin toimivuutta on kuitenkin hankala arvioida näkemättä varsinaisia lopputuotteita, joita sen avulla on luotu. Siksi esittelen tässä luvussa esimerkkinä jakkaran, joka on luotu käyttäen luomaani 3D-mallia.



Kuva 9. 3D-tulostettu huonekalujalan kiinnike

Kuvan kiinnike on tulostettu Prusa i3 MK3S mallisella FDM-tulostimella. FDM-tyyppiset 3D-tulostimet luovat halutun muotoisen kappaleen sulattamalla filamentiksi kutsuttua muovinauhaa suuttimessa ja muodostamalla kerroskerrokselta halutun muotoisen kappaleen. Suunnittelin kiinnikkeen muodon siten, että se olisi helppo tulostaa.

Kiinnikkeen pohja eli osa, joka tulee huonekalun kantta vasten, muodostaa laajan tasaisen pinnan, jonka ansiosta tulosteen ensimmäinen kerros kiinnittyy hyvin tulostusalustaan. Kiinnikkeessä ei myöskään ole ulokkeita, jotka riippuvat tyhjän päällä, minkä ansiosta kiinnike pystytään tulostamaan ilman tukirakenteita, jotka jouduttaisiin leikkaamaan irti jälkikäteen.

Tulostin kuvan kiinnikkeen PLA-filamentista, jota valmistetaan kasvi tärkkelyksestä. Kiinnike ei ole läpikotaisin täyttä muovia, vaan noin 1.5 mm paksuisen ulkokerroksen sisällä kiinnike muodostuu verkkomaisesta tukirakenteesta, joka täyttää vain noin 15 % kappaleen tilavuudesta. Tämän rakenne on tavallinen 3D-tulostetuissa kappaleissa ja sen luo ohjelmisto, joka muuntaa 3D-mallin tulostimen lukemaksi gcode-tiedostoksi. Kiinnikkeitä tulostaessa käytin Pruca Slicer -nimistä ohjelmistoa.

Kuva 10. 3D-tulostetut kiinnikkeet kiinnitettyinä ruuveilla jakkaran pohjaan

Kiinnitin tulostetut kiinnikkeet jakkaran pohjaan tavallisilla puuruuveilla. Kiinnikkeissä on sivussa paikka myös yhdelle ruuville jalan kiinnityksen varmistamiseksi. Tiivis juuri jalan halkaisijan kokoinen reikä on silti olennaista, jotta liitoksesta tulee riittävän tukeva. Siksi hankinkin ensin pyöreän mäntyriman, josta leikkasin jalat ja mittasin sen tarkan halkaisijan ennen kiinnikkeiden tulostusta, sillä vain puolimilliä liian iso halkaisija kiinnikkeen jalan reiässä tekee jalan kiinnityksestä huteran tuntuisen.

Kuva 11. Jakkara, joka on tehty käyttäen 3D-tulostettuja kiinnikkeitä

Valmiista jakkarasta tuli mielestäni varsin onnistunut. Kiinnikkeet tuntuvat tukevilta ja ovat kestäneet kaksi kuukautta koekäyttöä. 3D-tulostettujen kiinnikkeiden ansiosta jakkaran tekeminen vaati vain kannen osien leikkaamisen vanerista ja jalkojen katkaisun pyöreästä rimasta. Nämä pystytään tekemään yksinkertaisilla työkaluilla ja nopeasti, siksi uskon, että nämä kiinnikkeet voisivat olla hyödylliset tee-se-itse- harrastajille.

7 POHDINTA

Tämän työn lopputuloksena syntyi käyttökelpoinen kustomoitava 3D-malli huonekalujalan kiinnikkeelle ja web-käyttöliittymän sen kustomoimiseksi. Käyttöliittymän käytettävyys jäi kuitenkin heikommaksi, mitä olisin toivonut hitaan päivittymisen vuoksi.

Tässä opinnäytetyössä selvisi, että 3D-mallintaminen ohjelmoimalla voi tarjota monia huomattavia parannuksia tämänhetkisiin graafisten ohjelmistojen tarjoamiin tapoihin luoda 3D-malleja. Merkittävimpinä parempi suunnitteluaikeen välittyminen, luotujen parametristen mallien helpompi kustomointi ja uudelleen käytettävyys sekä mahdollisuus hyödyntää olemassa olevia versiohallintatyökaluja. Toki siinä on myös haasteita. Suurimpana näistä on koodaustaidon vaatimuksen luoma lisäeste skriptipohjaisten työkalujen käyttöön.

Mielestäni täyden hyödyn saamiseksi skriptipohjaisten työkalujen tarjoamista mahdollisuuksista 3D-mallien paremman suunnitteluaikeen säilyttämisessä ja siitä että niiden jakaminen on helpompaa, tulisi tarjolla olla myös työkalu 3D-mallien haluttujen parametrien muokkaamiseen ilman ohjelmointi- tai mallinnusosaamista. Tällöin myös henkilöt, joilla ei ole minkäänlaista ohjelmointiosaamista pystyisivät hyödyntämään muiden luomia malleja. Tämä mielipide perustuu kuitenkin vain omiin kokemuksiini huonekalumuotoilijana ja tee-se-itse-harrastajana. Mielenkiintoinen jatkotutkimuskohde olisi kerätä palautetta luomastani web-käyttöliittymästä ja koittaa selvittää, olisiko kyseisen kaltaiselle työkalulle kysyntää markkinoilla.

Skriptipohjaisella työkalulla työskenneltäessä muokattavien geometrian osien valinta on mielestäni huomattavasti hankalampaa kuin graafisella käyttöliittymällä, vaikka skriptipohjaisilla työkaluilla valinta onkin yksiselitteisempi ja siten suunnitteluaikeen välittämisen kannalta parempi. Pitäisin Mathur, Pirron ja Zuffereyn ehdottamaa ohjelmoinnin rinnalla toimivaa, elementtivalinta kyselyiden syntetisointiin käytettävää graafista työkalua mielenkiintoisena apuvälineenä (2020, 418).

Jotta skriptipohjaiset työkalut voisivat saavuttaa merkittävää suosiota 3D-mallinnustyökaluina, tulisi niiden käytöstä kiinnostua muutkin kuin ohjelmistokehityksen ammattilaiset. Siksi helppokäyttöisyys, hyvä dokumentaatio ja havainnolliset esimerkit ovat mielestäni vielä suuremmassa roolissa, kuin ohjelmistoprojekteissa normaalisti. Kiinnostava tutkimusaihe voisikin olla, mitkä ovat suurimmat käytettävyysongelmat skriptipohjaisissa 3D-mallinnustyökaluissa, sellaisten käyttäjien kannalta, joilla ei juurikaan ole aiempaa ohjelmointikokemusta.

CadQuery vaikutti minun arvioni perusteella tällä hetkellä käyttökelpoisimmalta skriptipohjaiselta työkalulta 3D-mallien luomiseen. CadQueryssakin oli silti vielä puutteita nykyaikaisten Cad-työkalujen mittapuulla perustoiminnallisuudessa. Toisaalta koska kyseessä on vapaan lähdekoodin ohjelmistoprojekti, pystyy jokainen lisäämään ohjelmoimalla omaan projektiinsa haluamansa toiminnallisuudet tai osallistumaan itse CadQueryn kehitykseen. CadQuery ja sen liitännäinen Jupyter Cadquery tuntuvatkin olevan erittäin aktiivisen kehityksen kohteena. Molemmat kirjastot kokivat merkittäviä päivityksiä niinä kuukausina, joina tein opinnäytetyötäni.

Käyttöliittymän toteutus nosti esiin epäilyksen ongelmista laskentatehon suhteen. 3D-mallien näytöllä näkyvän geometrian luominen CadQuery Python -skriptistä vaatii laskentatehoa ja se tulisi todennäköisesti muodostumaan ongelmaksi, mikäli palvelulla olisi paljon käyttäjiä.

Selvitys siitä pitävätkö epäilykseni laskentatehon muodostumisesta haasteeksi paikkansa, ja jos näin on, miten ongelma pystyttäisiin ratkaisemaan esimerkiksi 3D-mallin määrittelevien Python-skriptien käsittelyn siirtämisellä front endiin, olisikin kiinnostava jatkokehityskohde.

Mielenkiintoisen oloinen vaihtoehto laskennan siirtämiseksi front endiin on WebAssembly. Se mahdollistaa ohjelmien kääntämisen WebAssemblyn tehokkaaseen binääriformaattiin, jota tukevat kaikki modernit selaimet (WebAssembly. n.d.). Python ohjelmia ei pysty kuitenkaan suoraan vielä kääntämään.

Työ oli luonteeltaan tutustuminen skriptipohjaisiin työkaluihin ja niiden mahdollisuuksiin. Tämän työn puitteissa en pystynyt vertailemaan aikaa, joka 3D-mallin luomiseen menee graafisella työkalulla verrattuna skriptipohjaiseen, tai toisaalta aikaa, mikä menee oppia hallitsemaan graafinen CAD-työkalu verrattuna skriptipohjaiseen. Nämä ovat olennaisia kysymyksiä, koska ollakseen varteenotettava vaihtoehto, skriptipohjaisilla työkaluilla pystyä työskentelemään yhtä nopeasti kuin graafisilla vastineilla, ja opetteluaikakin pitäisi olla kohtuullinen. Nämä olisivat mielenkiintoisia jatkotutkimusaiheita, joskin niiden tutkiminen vaikuttaa melko haastavalta.

Lopputuloksessa olennaisin osa on luotu 3D-malli ja web-käyttöliittymä. Henkilökohtaisista kokemuksistani skriptipohjaisten 3D-mallinnustyökalujen käytöstä ei voi suoraa vetää johtopäätöksiä skriptipohjaisten työkalujen hyödyllisyydestä laajemmassa ammattimaisessa käytössä, mutta se nosti esiin yksittäisiä selkeitä etuja ja haasteita, joita lähestymistavassa on, sekä potentiaalisia jatkokehitys ja -tutkimuskohteita.

LÄHTEET

CadQuery. n.d. Dokumentaatio. Luettu 30.4.2021.
https://cadquery.readthedocs.io/en/latest/intro.html

Cambda, J.D., Contero, M. & Company, P., 2016. Parametric CAD modeling: An analysis of strategies for design reusability. Computer-Aided Design 74, 18-31.

Fabbers. n.d. The STL Format. Luettu 30.7.2021.
https://www.fabbers.com/tech/STL_Format#Sct_binary

GitHub Docs. n.d. 3D File Viewer. Luettu 30.4.2021.
https://docs.github.com/en/github/managing-files-in-a-repository/working-with-non-code-files/3d-file-viewer

GitHub Docs. n.d. Saving repositories with stars. Luettu 30.4.2021.
https://docs.github.com/en/github/getting-started-with-github/saving-repositories-with-stars

Heroku. n.d. What is Heroku. Luettu 30.6.2021.
https://www.heroku.com/about

Ipywidgets. n.d. Simple Widget Introduction. Luettu 3.5.2021.
https://ipywidgets.readthedocs.io/en/latest/examples/Widget%20Basics.html

jupyter-cadquery GitHub repositorio. n.d. README.md. Luettu 3.5.2021.
https://github.com/bernhard-42/jupyter-cadquery

Machado, F., Malpica, N. & Borromeo, S. 2019. Parametric CAD modeling for open source scientific hardware: Comparing OpenSCAD and FreeCAD Python scripts. PLOS ONE14 (12), e0225795. https://doi.org/10.1371/journal.pone.0225795

Mathur, A., Pirron, M. & Zufferey, D. 2020. Interactive Programming for Parametric CAD. Computer Graphics Forum 39 (6), 408-425.

Milton, A. & Rodgers, P. 2011. Product Design. Lontoo: Laurence King Publishing.

Open CASCADE Technology. n.d. Brep Format. Luettu 30.4.2021.
https://dev.opencascade.org/doc/overview/html/specification__brep_format.html

Open CASCADE Technology. n.d. Introduction. Luettu 30.4.2021.
https://dev.opencascade.org/doc/overview/html/

OpenScad. n.d. About. Luettu 30.4.2021.
https://openscad.org/about.html

Otey, J., Company, P., Contero, M. & Camba, J.D. 2018. Revisiting the design intent concept in the context of mechanical CAD education. Computer-aided design and applications 15 (1), 47-60.

Skalnik, M. 2013. 3D File Diffs. The GitHub Blog. Luettu 8.5.2021. https://github.blog/2013-09-17-3d-file-diffs/

The Jupyter Notebook. n.d. Introduction. Luettu 3.5.2021. https://jupyter-notebook.readthedocs.io/en/stable/notebook.html

Voilá. n.d. Using Voilá. Luettu 3.5.2021. https://voila.readthedocs.io/en/stable/using.html

WebAssembly. n.d. Overview. Luettu 15.6.2021. https://webassembly.org/