VIILUSORVIN SIMULOINTIMALLIN KEHITTÄMINEN

LAHDEN AMMATTIKORKEAKOULU
Tietotekniikan koulutusohjelma
Ohjelmistotekniikka
Opinnäytetyö
Kevät 2009
Aki Lietonen

Lahden ammattikorkeakoulu
Tietotekniikan koulutusohjelma

LIETONEN AKI: Viilusorvin simulointimallin kehittäminen

Ohjelmistotekniikan opinnäytetyö, 57 sivua

Kevät 2009

TIIVISTELMÄ

Tämä opinnäytetyö käsittelee simulointia ja siinä käytettäviä ohjelmistoja ja lähemmin viilusorvin simulointia Visual Components 3D Create- simulointiohjelmalla. Työ on tehty Raute Oyj:llä ja simulointi on tehty Rauten viilusorvista. Opinnäytetyön tavoitteena oli luoda viilusorvin toiminnasta simulaatiomalli, jonka avulla sorvausprosessin hahmotus olisi selkeää, ja jonka avulla voitaisiin edistää myyntiä sekä koulutusta.

Työssä selvitetään vanerin valmistusprosessia ja siihen kuuluvaa viilunsorvausta ja sitä edeltäviä työvaiheita. Siinä myös tutkitaan simuloinnin merkitystä järjestelmien suunnittelussa ja Visual Components 3D Create -simulointiohjelmiston soveltuvuutta simultointiin. Lisäksi työssä tutkitaan 3D Create -ohjelman sisältämää Python ohjelmointikieltä, jonka avulla onnistuu monipuolisten simulaatioiden luominen samassa kehitysympäristössä.

Viilunsorvaus on yksi tärkeimpiä vanerin valmistuksen työvaiheita, jonka onnistuminen on tärkeää muiden työvaiheiden sekä vanerin laadun kannalta. Sorvaukseen liittyy paljon vaatimuksia, jotta sen jälkeiset vaiheet onnistuisivat ja jotta vanerin laatu olisi mahdollisimman hyvä.

Simuloinnista on tullut yksi yleisimmistä tavoista tehdä tarkkoja suunnitelmia erilaisista järjestelmistä. Nykyiset simulointiohjelmat tarjoavat helpon ja nopean tavan kehittää simulaatioita yritysten tarpeisiin.

Python-ohjelmointikieli on tärkeä osa 3D Createn toimintoja. Sillä voidaan lisätä sovelluksen käyttötarkoituksia sekä luoda monimutkaisempia simulaatioita helposti. Python-ohjelmointikieli on dynaamisesti, mutta vahvasti tyypitetty olio-ohjelmointikieli, joka on hyvin siirrettävissä erilaisiin sovellutuksiin ja laitteisiin. Työssä käytetään Pythonia Excel-tiedostossa olevien laskukaavojen hakemiseksi 3D Createen.

Simulointityö saatiin valmiiksi onnistuneesti ja kaikki tärkeimmät toiminnot toimivat, kuten oli suunniteltukin. Simulaatiota voidaan alkaa käyttämään myynnin ja koulutuksen tukena, ja se tarjoaa hyvän pohjan jatkokehitykselle.

Avainsanat: Visual Components, 3D Create, simulointi, viilusorvi, vaneri

Lahti University of Applied Sciences
Faculty of Technology

LIETONEN AKI: Developing a veneer peeling simulation model

Bachelor´s Thesis in Software Engineering 57 pages

Spring 2009

ABSTRACT

This thesis deals with simulation and the software used in simulation, and more closely with simulation using Visual Components 3D Create simulation software. The practical part was carried out at Raute Oyj. The objective was to create a simulation model of the peeling operation of the machine. The simulation would be used in sales, training and to generally make the peeling process more understandable.

The veneer manufacturing process and processes before and after veneer peeling were studied. The significance of simulation in system designing and Visual Components 3D Create simulation software were also examined. The thesis also includes a study of the Python programming language, which is included as a module in 3D Create software. It allows the developing of complex simulations in the same simple simulation development environment.

Veneer peeling is one of the most important stages in veneer production, and its success is important for the working stages after peeling and for the quality of the veneer. Peeling has to meet a lot of requirements to guarantee a sufficiently high quality.

Simulation has become one of the most common ways to create accurate plans and designs of different systems. Today's simulations software offers an easy and fast way to develop simulations for the needs of companies. Simulation software programs have developed so much that they really offer an alternative for ordinary programming languages like C language.

The Python programming language is an important part of 3D Create functions. With Python one can easily add more complexity and use to the simulation. The Python programming language is dynamically but strongly typed object-oriented programming language, which is easily portable for many different solutions and devices. Python was used in this work to access the data in Excel files that contains existing Excel calculations. Parameters and movements were transferred from Excel to 3D Create.

The simulation was completed successfully and the most important functions work as intended. The simulation model is ready to be used in sales and training.

Key words: Visual Components, 3D, simulation, peeling, veneer

SISÄLLYS

1 JOHDANTO 1

2 VANERIN VALMISTUS 3
2.1 Sorvausta edeltävät vaiheet 3
2.2 Sorvauksen vaatimukset 4
2.3 Viilun sorvaus 6
2.3.1 Pöllin keskittäminen 7
2.3.2 Sorvauslaite 8

3 SIMULOINTI 11
3.1 Yleistä 11
3.2 Simuloinnin edut ja haitat 12
3.3 Järjestelmät ja mallit 14
3.4 Simulointi ja tuotantojärjestelmät 16

4 TYÖKALUT 17
4.1 Visual Components 17
4.2 3D Create 18
4.2.1 eCAT ja komponenttipaketit 19
4.2.2 Rajapinta ja simulaatioaika 20
4.2.3 Robotit 22
4.2.4 Komponenttien luominen 23
4.2.5 Komponenttiskriptaus 25
4.3 3D Video ja 3D Realize 26
4.4 Python ohjelmointikieli 28
4.4.1 Python-tulkki 29
4.4.2 Pythonilla työskentely 30
4.4.3 Pythonin ominaisuudet 32
4.4.4 Python 3D Create:ssa 33

5 SIMULAATION LUOMINEN 37
5.1 Suunnitelma 38
5.2 Sorvin mallinnus 39
5.3 Toimintojen toteutus 43
5.3.1 Excel-toiminnot 44

5.3.2 Python Script 47

6 YHTEENVETO 55

1 JOHDANTO

Simulointi on työprosessien ja järjestelmien eri vaiheiden matkimista ja tutkimista, joko ohjelmallisesti tai esimerkiksi pienoismallin avulla. Simulointi on jo pitkään ollut tärkeä osa uusien järjestelmien suunnittelua ja testausta sekä vanhojen järjestelmien päivityksiä ja optimointeja.

Ohjelmallisten simulointien luontiin on jo pitkään käytetty perinteisiä ohjelmointikieliä kuten C-ohjelmointikieltä, jolla voidaan luoda lähes minkälaisia simulaatioita tahansa. Perinteiset ohjelmointikielet ovat kuitenkin hankalia ja aikaa vieviä laajoissa ja monimutkaisissa simulaatioissa, jonka takia markkinoille on tullut monia yksinkertaisia simulointiohjelmistoja. Aiemmin simulointiohjelmat ovat olleet rajoittuneita ominaisuuksiltaan, mutta nykyiset ohjelmat pystyvät lähes samaan kuin ohjelmointikieletkin ja lisäksi ne ovat helppokäyttöisiä ja niillä on nopea kehittää simulointimalleja.

Tämä opinnäytetyö on tehty Raute Oyj:lle, joka on maailmalla hyvin tunnettu puunjalostamiseen tarkoitettujen tehtaiden sekä laitteiden toimittaja. Rauten päätuote on vanerin valmistamiseen tarkoitetut laitteet, linjat sekä tehdaskokonaisuudet. Raute on markkinajohtaja tehdaskokonaisuuksien toimittajana. Vuonna 2008 Rauten liikevaihto oli 98.5 miljoonaa euroa ja yrityksen palveluksessa työskenteli 570 henkilöä kahdeksassa maassa. Rauten päätoimipiste sijaitsee Nastolassa. Tässä opinnäytetyössä suunnitellaan ja toteutetaan Rauten viilusorvilaitteen simulaatio Visual Components 3D Create-ohjelmalla.

Opinnäytetyön tavoitteena oli tehdä liikkuva simulaatiomalli viilusorvilaitteesta, joka käyttäisi oikean laitteen mittoja ja liikeratoja. Simulaatiossa keskityttiin lähinnä vain sorvin tukilaitteen toimintaan ja liikkeisiin. Simulaatio-ohjelman toimintaa ohjaileva Python Script vastaa simulaation eri toiminnoista. Simulaatiomallin tavoite on helpottaa sorvausprosessin hahmottamista ja mahdollisesti olla apuna koulutuksessa sekä myynnissä.

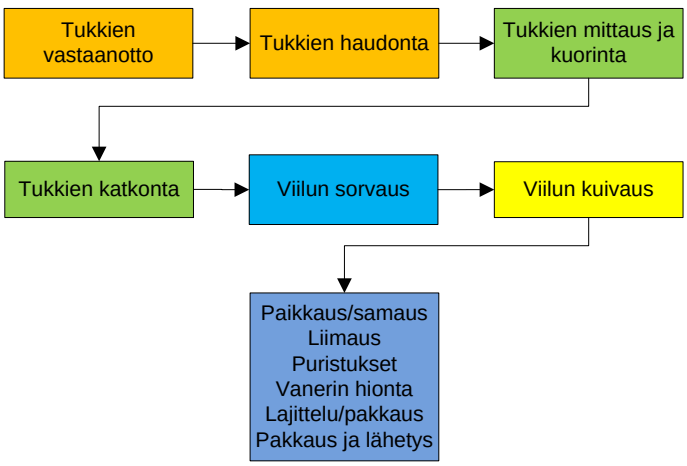
Ongelmana oli saada käytettyä simulaatiossa jo valmiiksi tehtyjä laskelmia sorvin liikeradoista. Laskelmat sijaitsivat Excel-taulukossa, josta niitä täytyi hakea ja päivittää. Työssä päätettiin tutkia Python Scriptin mahdollisuuksia tiedon hakemiseen ja siirtämiseen Excel-taulukosta.

2 VANERIN VALMISTUS

Viilun sorvaus on yksi tärkeimpiä vanerin valmistuksen vaiheita, mutta ennen kuin puutukkeja päästään sorvaamaan, täytyy niitä ensin hautoa ja sitten katkaista sopivan mittaisiksi ja muotoisiksi. Puutukkien mitalla ja muodolla on suuri merkitys viilun saantoon ja vanerin laatuun. Toinen tapa valmistaa viiluja on leikkaus, mutta leikkausta käytetään huomattavasti vähemmän kuin sorvausta. (Koponen 1995, 27–28.)

2.1 Sorvausta edeltävät vaiheet

Vanerin valmistusprosessi tapahtuu kuvion 1 mukaisesti, eli se aloitetaan tukkien vastaanotolla. Nykyään tehtaat vastaanottavat tukit lähes yksinomaan autoista. Vastaanoton jälkeen on vuorossa tukkien haudonta, jonka tarkoituksena on puun lämmön ja kosteuden nostaminen tasolle, jossa viilu leikkautuu pinnaltaan tasaisena, sileänä ja riittävän lujana. Tehtaalle tulleet tukkikuormat pudotetaan suoraan haudonta-altaaseen ja niitä pidetään siellä jokaiselle puulajille sopiva aika. Haudonnan jälkeen tukit kuljetetaan kuorintaan, jossa puutukin pintakerros ja epäpuhtaudet poistetaan. (Koponen 1995, 29–34.)



KUVIO 1. Vanerin valmistusprosessi

Suomessa kuorinta tehdään yleensä vasta haudonnan jälkeen, koska jäistä tukkia on hankala kuoria. Tukkien kuorinnan tarkoituksena on saada kuoren mukana tulleet epäpuhtaudet pois, jotta sorvin terä ei vaurioituisi sorvauksen aikana. Lisäksi ennen kuorintaa tukit kuljetetaan metallinilmaisimen läpi, jotta mahdolliset metallikappaleet tukin sisällä tai pinnalla voitaisiin havaita. Tukkien kuorinnassa on tärkeää, että tukin pinta ei vahingoitu kuorinnan aikana, jotta puuaineksen arvokkain pintaosa saataisiin hyödynnettyä. (Koponen 1995, 34.)

Viimeinen vaihe ennen sorvausta on tukkien katkaisu, jossa tukit katkaistaan viilun edellyttämiin pituuksiin. Katkaisun päätarkoitus on optimoida sorvauksesta saatavan viilun määrä ja laatu. Tukin läpimitta, pituus ja kartiomaisuus mitataan ennen katkaisua yleensä laserin käyttöön perustuvalla mittarilla. Tietokone käsittelee tiedot ja antaa jokaiselle tukille parhaimmat katkaisuvaihtoehdot. Katkaisussa on huomioitava esimerkiksi koivutukeissa yleinen mutkaisuus, jolloin katkaisu olisi hyvä tehdä mutkan kohdalta, jotta saataisiin mahdollisimman suoria tukkeja sorvaukseen. (Koponen 1995, 36.)

2.2 Sorvauksen vaatimukset

Viilu on vanerin perusosa, jota saadaan useimmiten sorvaamalla pölliä spiraalimaisesti vuosirenkaiden suunnassa. Sorvauksen sijasta voidaan käyttää viilun leikkausta, jossa pölliä leikataan kuvion 2 mukaisesti, yleensä vuosirenkaiden suuntaa vastaan. Leikkausmenetelmää käytetään huomattavasti vähemmän ja siitä saatavaa kuviollista viilua käytetään pääasiassa puutuotteiden pinnoitukseen. (Koponen 1995, 38.)

KUVIO 2. Viilun leikkaus (PuuProffa 2008)

Ennen viilun sorvausta on otettava huomioon, millaisia laatuvaatimuksia viilulle on asetettu ja millaiseen tarkoitukseen sorvattava erä on tarkoitettu. Vaatimukset on yleensä jaettu kahteen ryhmään: kaupallisen laadun vaatimuksiin ja sorvauksen jälkeen tapahtuvien työvaiheiden asettamiin teknisiin laatutasoihin. (Koponen 1995, 38.)

Kaupalliset laatuvaatimukset ovat monesti vanerin ulkonäköön, materiaalin laatuun ja massaan liittyviä. Esimerkiksi viilun pinnan sileys, halkeilu ja ulkonäkövirheet vaikuttavat pintaviilujen ulkonäköluokituksiin. Vanerilevyn mitoista tärkein on viilun paksuus, jonka on täytettävä vaneristandardien vaatimukset kuivauksen, liimauksen ja hionnan jälkeen. Viilun paksuus vaikuttaa suoraan raaka-aineen kulutukseen, joten sen optimoiminen viilun pituuden ja leveyden ohella on tärkeää. Viilujen on oltava myös riittävän vahvoja erityisesti poikittaisvetolujuudeltaan. (Koponen 1995, 38.)

Sorvauksen jälkeisten työvaiheiden asettamat tekniset laatutasot pitävät sisällään yleensä seuraavia vaatimuksia. Viilun leikkauksen saannon ja kuivauskoneiden toiminnan varmistamiseksi, sorvatun viilumaton on oltava ehjä ja mahdollisimman pitkä. Kuivauksessa tapahtuva viilun kutistuminen on otettava huomioon sorvin asetteessa, jonka säätäminen tapahtuu usein täysin kokemusperäisesti. Viiluissa ei saa olla liiallisia jännityksiä, jotta paikkaus ja saumaus onnistuisivat mahdollisimman hyvin. Vanerin liimauksen kannalta tärkeimpiä ominaisuuksia

ovat viilun sileys ja tasainen paksuus, jotka vaikuttavat siihen, että liiman kulutus on vähäisempää ja liimaus onnistuu paremmin. Levyn pinnan virheet tulevat viimeistään esiin hionnassa. Pintavirheitä ovat esimerkiksi liian karkea sorvausjälki, sorvinterien jättämät jäljet tai liian harvaksi sorvatun viilun aiheuttamat liiman läpilyönnit. Valmistusprosessin viimeisissä vaiheissa saattaa tulla suuriakin taloudellisia menetyksiä, jos alkuvaiheet eivät ole onnistuneet halutulla tavalla, joten esimerkiksi viilun lujuudella on suuri taloudellinen merkitys. Viilun on kestettävä kaikki sorvauksen jälkeiset vaiheet. (Koponen 1995, 38–39.)

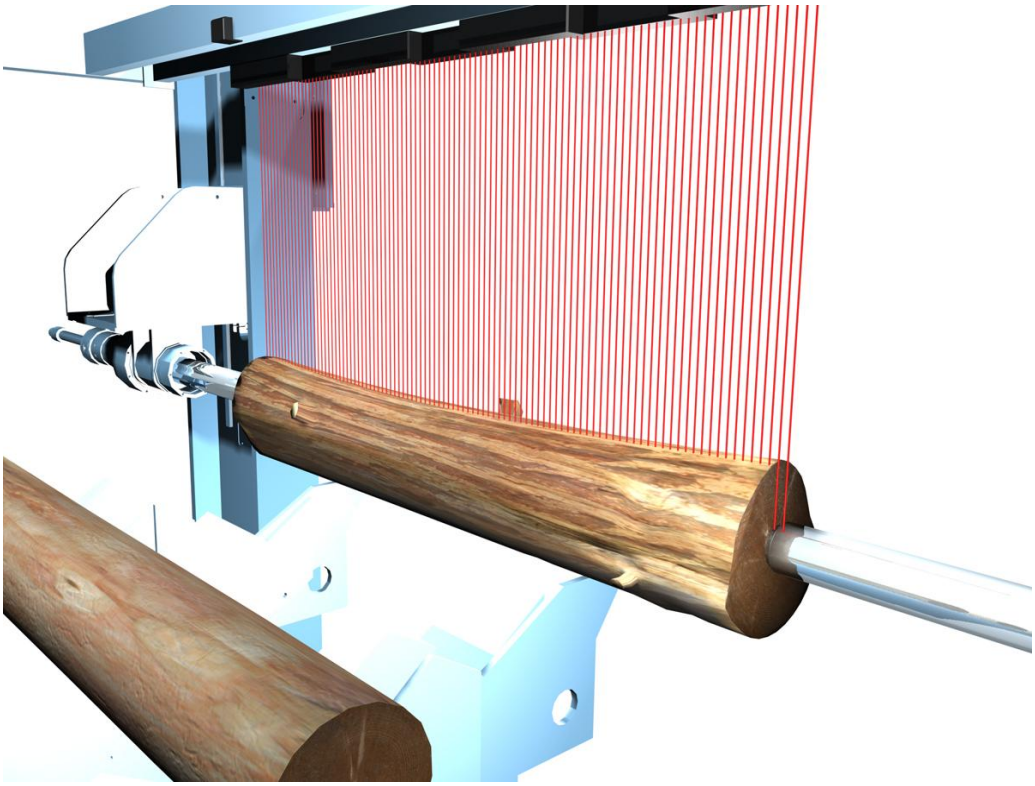
2.3 Viilun sorvaus

Sorvaus on yksi vanerintuotannon tärkeimpiä työvaiheita. Sorvauksessa viilu irrotetaan pyörivästä pöllistä spiraalimaisesti pöllin vuosirenkaiden mukaisesti, kuten kuviossa 3 ilmenee. Sorvauksen onnistuminen vaikuttaa kaikkiin sen jälkeisiin työvaiheisiin ja vanerin laatuun. Viilun sorvauksen työvaiheita ovat: pöllien siirto keskityslaitteeseen, pöllin keskitys, keskitetyn pöllin siirto sorvin karojen väliin, viilun sorvaus. Sorvaus tapahtuu pyörittämällä pölliä karojen välissä, tukilaitteiden tukemana ja samalla painamalla terää pöllin pintaa vasten.(Koponen 1995, 38–39.)

KUVIO 3. Viilun sorvaus (PuuProffa 2008)

2.3.1 Pöllin keskittäminen

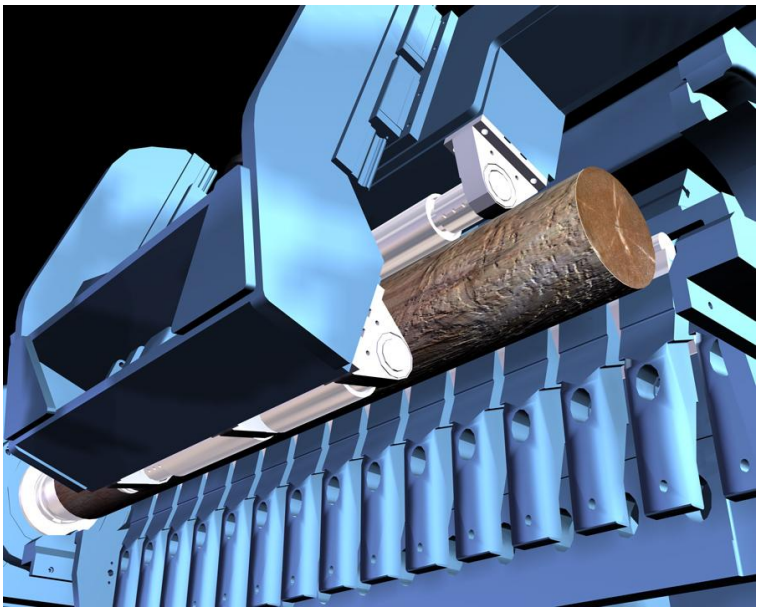
Sorvauksen ensimmäinen vaihe on tuoda pöllit keskittäjään, jonka tehtävänä on keskittää pöllit tarkasti karojen väliin. Pöllin keskittämisellä pyritään tuottavuuden maksimoimiseen asettamalla pölli asentoon, jossa saadaan hyödynnettyä mahdollisimman paljon pöllin laadukkainta pintaosaa. Pöllin keskittämiseen on kehitetty muutama eri menetelmä, joista vanhempaa eli mekaanista keskittämistä ei juurikaan enää käytetä uudemmissa laitoksissa. Mekaanisessa keskittämisessä pölli keskitetään molemmista päistä kolmella vipuvarrella. Kuviossa 4 näkyvä nykyaikaisempi vaihtoehto on käyttää tietokonetta keskittämisen apuna. Pöllin muotoa tarkkaillaan lasersäteiden muodostamassa verhossa ja pölliä pyöräytetään siinä kerran, jonka aikana tietokone tallentaa sen muodot ja laskee optimaalisen sorvausasennon. Keskittämisen jälkeen tarttuja siirtää pöllin sorvin karojen väliin oikeassa asennossa. (Koponen 1995, 40–41.)



KUVIO 4. XY-keskitys (Raute Oyj 2008)

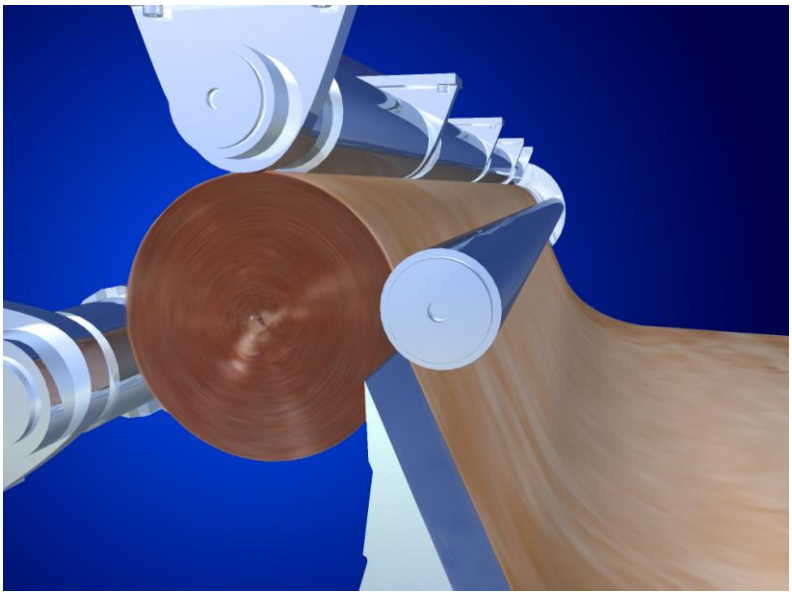
2.3.2 Sorvauslaite

Sorvi koostuu suuresta rungosta sekä liikkuvista mekaanisista osista, joista tärkeimpiä ovat kuviossa 5 näkyvät: teräpenkki, terä, vastaterä, karat ja tukilaite. Karoihin kiinnitetty pölli pyörii akselinsa ympäri jopa 1000 kierrosta minuutissa sorvauksen loppuvaiheessa, jonka takia pyörittämistä ei voida tehdä pelkillä karoilla. Pyörittämisessä auttavat moottoroitu vastaterä sekä moottoroidut tukilaitteen tukitelat. Tukilaite koostuu kahdesta tukirullasta, ylätuesta ja alatuesta. Tukirullilla pyritään pitämään sorvauksesta aiheutuvat taivuttavat voimat kurissa, pitämään pölli suorassa sekä auttamaan pöllin pyörittämisessä. (Koponen 1995, 42.)



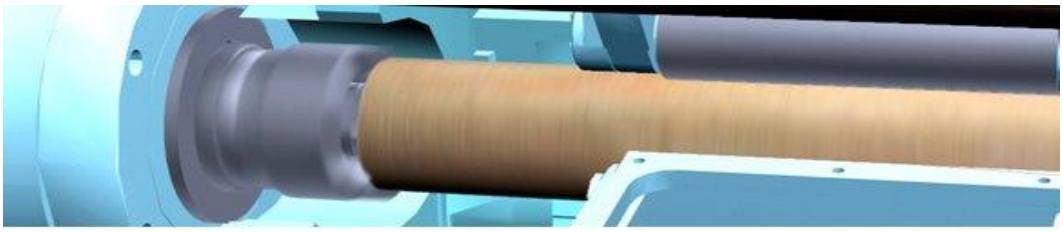
KUVIO 5. Teräpenkki, terä, vastaterä, pölli ja tukilaite (Raute Oyj 2008)

Terä on kiinnitetty teräpenkkiin, joka liikkuu horisontaalisesti pölliä vasten. Pöllin halkaisijan pienetessä myös terän on oltava lähempänä karoja. Vastaterän tehtävänä on luoda painetta pölliä vasten, jotta viilun irrotus pöllistä säilyisi leikkaavana. Jos painetta ei olisi, viilu alkaisi vuoleutumaan irti pöllistä, joka aiheuttaisi viilun pintaan halkeamia. Seuraava kuvio (kuvio 6) havainnollistaa viilun irtoamista pöllistä. (Koponen 1995, 42–44.)



KUVIO 6. Tukitelat, pölli, vastaterä ja terä (Raute Oyj 2008)

Viilun paksuus vaihtelee 2,5 – 4,8 millimetriin pehmeissä puissa, ja jopa niinkin ohutta kuin 0,7 millimetriä voidaan sorvata kovilla puilla kuten koivulla tai pähkinäpuilla (Sellers 1985, 62). Nykyiset viilusorvit pystyvät sorvaamaan pölliä yleensä noin 50 millimetrin halkaisijaan saakka, jonka jälkeen pölliä täytyy vaihtaa. Ennen 1980-luvun lopulla tehtyä kehitystyötä tämä ei ollut mahdollista. Sen jälkeisissä sorveissa pölliä ei pyöritetä pelkillä karoilla vaan suurimmaksi osaksi tukiteloilla. Tämän ansiosta karoista on voitu tehdä monivaiheisia, eli sorvauksen alussa karojen halkaisija voi olla esimerkiksi 80 millimetriä. Kun pöllin halkaisija pienenee, karasta vedetään ulko-osat pois seuraavan kuvion 6 mukaisesti, jonka jälkeen sorvausta voidaan vielä jatkaa. Tämä on erityisen tärkeää silloin kun sorvataan pöllejä, joissa on pieni halkaisija. (Bowyer, Shmulsky & Haygreen 2007, 360.)



KUVIO 7. Kara (Raute Oyj 2008)

Sorvauksen jälkeen kostea viilu kuljetetaan kuivaukseen, jossa viilun kosteus lasketaan liimauksen vaatimalle tasolle. Viilut on myös leikattava halutun kokoisiksi arkeiksi. Se voidaan tehdä joko ennen kuivausta tai sen jälkeen. Leikkausjärjestys on riippuvainen kuivauslaitteesta: jos kuivataan verkkokuivauskoneella, voidaan koko viilumatto kuivata suoraan sorvauksen jälkeen. Kuivauksen jälkeen ovat vuorossa loput työvaiheet: paikkaus, saumaus, liimaus, puristus, hionta, lajittelu, pakkaus, lähetys. (Koponen 1995, 51.)

3 SIMULOINTI

Tietokonesimulaatiossa on kyse tietokoneen apuna käyttämisestä erilaisten järjestelmien imitoinnissa. Kun tutkitaan jotain järjestelmää tieteellisestä näkökulmasta, täytyy tehdä joitakin oletuksia siitä, kuinka järjestelmä toimii. Näistä oletuksista muodostuu malli, jota käyttämällä voidaan ymmärtää, kuinka järjestelmä toimii. Jos järjestelmässä olevat suhteet, jotka luovat mallin, ovat yksinkertaisia, on mahdollista käyttää matemaattisia keinoja vastausten saamiseksi. Tällaista lähestymistapaa kutsutaan analyyttiseksi ratkaisuksi. Monet oikeat järjestelmät ovat liian monimutkaisia analyyttisille ratkaisuille, jolloin niiden tutkimiseen tarvitaan simulointia. (Averill, Law & Kelton 2007, 1.)

3.1 Yleistä

Simulointi on yksi eniten levinneistä tavoista tutkia tuotekehitystä ja yleistä liiketoiminnan hallintaa. Simulaatiossa on tarkoituksena kerätä tietoa simulaatiomallista, jonka avulla voidaan arvioida sen todelliset ominaisuudet. Nykyään simulaatiot tehdään tietokoneilla, joissa tietokoneen tehtävänä on hallita simulaatiomallia sekä esittää tulokset numeerisesti. Käytännön esimerkkinä voisi olla tehdas, jonka suunnitelmana on tehdä isompi laajennus. Simuloimalla laajennus etukäteen, voidaan välttyä riskeiltä, kuten mahdolliselta tuottavuuden väärin arvioinnilta. (Averill, Law & Kelton 2007, 1.)

Käyttökohteita simuloinnille on lähes rajattomasti. Melkein mitä tahansa tosimaailman tapahtumaa voi periaatteessa simuloida edes jollakin tasolla. On kuitenkin alueita, joissa simulointi on todettu erittäin tehokkaaksi työkaluksi. Sellaisia ovat esimerkiksi:

- teollisuusjärjestelmien suunnittelu ja analysointi
- asejärjestelmien ja niiden logistiikkatarpeiden arviointi
- viestintäverkkojen protokolla ja laitevaatimukset
- liikennejärjestelmien suunnittelu
- liiketoimintasuunnitelmien uudelleenrakennus.

(Averill, Law & Kelton 2007, 2.)

Erittäin laajat ja monimutkaiset järjestelmät eivät kuitenkaan aina tue simuloinnin käyttöä, sillä etenkin vanhemmilla simulointiohjelmilla tai simuloitaessa perinteisillä ohjelmointikielillä voi simulointi olla hidasta ja vaikeaa. Nykyiset kehittyneet simulointiohjelmat ovat tuoneet helpotusta hankalien ja laajojen järjestelmien simuloimiseen. Niissä on pyritty vähentämään käsin ohjelmointia ja lisätä valmiita ratkaisuja ja toimintoja, joita käyttäjä voi ottaa nopeasti käyttöön. Toinen ongelma laajojen järjestelmien simuloimisessa on ollut niiden tarvitsema suuri laskentateho, joka kuitenkin halpojen ja tehokkaiden suorittimien ansiosta on hävinnyt lähes kokonaan. (Averill, Law & Kelton 2007, 2–3.)

3.2 Simuloinnin edut ja haitat

Simuloinnissa on monia etuja, mutta huonosti suunniteltuina ne voivat olla myös haitallisia ja jopa antaa väärää tietoa. Yleisessä tiedossa on joitakin yleisiä ongelmia ja ansoja, joissa simulointiprojekti voi epäonnistua. Simulointiprojektia suunnitellessa on hyvä ottaa huomioon nämä yleiset ongelmat ja pohtia miten ne voidaan välttää. (Averill, Law & Kelton 2007, 76–77.)

Analysoinnilla ei aina voida korvata kaikkia simuloinnin tuomia etuja. Simuloinnilla voidaan arvioida jo olemassa olevan järjestelmän suorituskykyä ja toimintaa. Joissain tietyissä tilanteissa, tämä on tärkeää varsinkin kun etsitään järjestelmän mahdollisia hidastavia tekijöitä. Kun tutkitaan muita järjestelmäratkaisuja olemassa olevien rinnalle, simuloimalla erilaisia järjestelmäasetuksia, voidaan selvittää niiden toimintakykyä ja sopivuutta kuhunkin tilanteeseen. Simuloinnissa on helpompi säilyttää kokeelliset olosuhteet verrattuna oikean järjestelmän kanssa teh-

tyihin kokeisiin. Ehkä yksi parhaista eduista simuloinnissa on pitkän aikavälin testaukset, joissa voidaan simuloida järjestelmän tuottavuutta esimerkiksi vuoden aikana. Pitkän aikavälin testauksia on tehokasta tehdä simulointimalleilla, kun niissä olevaa aikaa voidaan nopeuttaa helposti. (Averill, Law & Kelton 2007, 76–77.)

Simulointi ei itsessään yleensä ole haitallinen ellei sen suunnittelu ole epäonnistunut pahasti. Simuloinnista aiheutuvat haitat, ongelmat tai väärät tulokset riippuvat monesti siitä, miten kriittisesti sen antamia tuloksia arvioidaan. Monesti simulaatioiden tulokset ja itse simulointi saattaa näyttää erittäin vakuuttavalta, vaikka sen suunnittelussa olisi jokin virhe, jonka johdosta tulokset eivät olisi käyttökelpoisia. Monimutkaisten simulointien kehittäminen on myös yleensä hidasta sekä kallista, joten siitä saadut tulokset pitäisi olla käytettävissä tehokkaasti. Jokainen simulaatio tuottaa vain arvioita järjestelmän oikeasta toiminnasta tietyillä syötteillä, joten yleensä tarvitaan monia yksittäisiä simulaatioajoja eri parametriarvoilla. Tämän takia simulaatiomallit eivät yleensä ole niin hyviä optimisoinnissa kuin vertaamassa vaihtoehtoisia järjestelmiä. (Averill, Law & Kelton 2007, 77.)

Päätettäessä simulaation käyttämisestä, on otettava huomioon kaikki hyödyt ja haitat sekä pohdittava tarkoin tehtävän käytettävyyttä ja kuinka realistisia arvoja siitä voidaan saada. Analyyttisten- ja matemaattisten mallien käyttöä ei tarvitse sulkea kokonaan pois, kun simulaatioita aletaan käyttää. Jos järjestelmästä on olemassa jo analyyttiset mallit, voidaan simulointia käyttää niiden oikeellisuuden varmistamiseen sekä mallien havainnollistamiseen. Yleisimpiä kompastuskiviä simulointikehityksen aikana ovat:

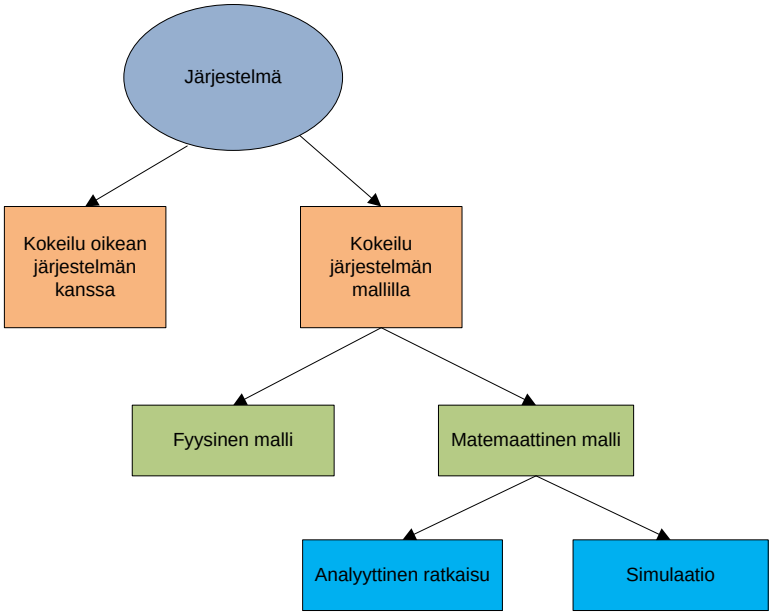
- vajaasti määritetyt tavoitteet
- yhteen järjestelmämalliin ja siitä saatuihin tuloksiin luottaminen
- satunnaisien tapahtumien liian vähäinen huomioiminen mallissa
- vääränlainen simulointiohjelmisto
- simulaatiomallin vajaa tarkkuus.

Huolellisella suunnittelulla on suuri merkitys simulaatiomallin onnistumisen kannalta. Hyvä suunnitelma ja kompastuskivien huomioiminen aikaisessa vaiheessa tuottavat yleensä halutun tuloksen. (Averill, Law & Kelton 2007, 77–78.)

3.3 Järjestelmät ja mallit

Järjestelmään kuuluu vaihteleva määrä laitteita, ihmisiä tai muita vastaavia yksittäisiä osia, joiden tarkoituksena on toimia yhdessä suorittaakseen jotain loogista tehtävää. Järjestelmän tila koostuu muuttujista, jotka määrittelevät sen tiettynä ajankohtana. Järjestelmät voi jakaa kahteen eri tyyppiin, tahdistettuun ja jatkuvaan järjestelmään. Tahdistetussa järjestelmässä siinä olevat muuttujat vaihtuvat samanaikaisesti eri ajankohdissa, jos niillä on tarvetta muuttua. Jatkuvan järjestelmän muuttujat vaihtuvat jatkuvasti suhteessa aikaan. Harvat järjestelmät ovat pelkästään tahdistettuja tai jatkuvia, yleensä niistä toinen on vallitsevampi, mutta useimmissa järjestelmissä tarvitaan molempia. (Averill, Law & Kelton 2007, 3.)

On olemassa erilaisia tapoja, joilla järjestelmää voidaan tutkia. Voidaan tehdä kokeiluja oikean järjestelmän kanssa ja kokeiluja järjestelmän mallilla. On aina hyvä, jos kokeiluja päästään tekemään oikean järjestelmän kanssa, esimerkiksi muuttamalla sitä tavalla, jolla voitaisiin tutkia esimerkiksi uusia säästömahdollisuuksia. On kuitenkin vaikeaa saada oikeaa järjestelmää muutettua edullisesti, ja siten, että järjestelmän toiminta ei häiriintyisi. Tilanteissa joissa järjestelmä on laaja tai muuten erittäin hankala kokeiltavaksi oikeasti, voidaan järjestelmää kokeilla mallilla, jonka tarkoituksena on olla oikean järjestelmän korvike. Mallia tutkittaessa ongelmana on sen oikeellisuus ja kyky kuvastaa oikeata järjestelmää. Seuraavassa kuviossa (kuvio 8) ilmenevät järjestelmän tutkimiseen liittyvät eri valinnat ja vaihtoehdot. (Averill, Law & Kelton 2007, 4.)



KUVIO 8. Järjestelmän tutkimismenetelmät

Järjestelmän mallin testaamisen voi jakaa kahteen eri osa-alueeseen, fyysiseen- ja matemaattiseen malliin. Fyysiset mallit ovat yleensä pienoismalleja oikeista laitteista, joita voidaan käyttää laitteen toiminnan testaamiseen. Fyysisiä malleja käytetään yleensä sen jälkeen, kun matemaattisella mallilla on todettu järjestelmä toimivaksi. Laajempien järjestelmien monimutkaisia tapahtumia ei yleensä voida kuvata fyysisillä malleilla, jonka takia fyysisiä malleja yleensä käytetäänkin isoissa yksittäisissä laitteissa toteamassa niiden toimivuutta, kuten esimerkiksi laivan pienoismalli kellumassa uima-altaassa. Hyvä esimerkki matemaattisesta mallista on d = rt, jossa d on matkan pituus, r on nopeus ja t on aika. Tällainen matemaattinen malli voi toimia hyvin avaruudessa, mutta huonosti esimerkiksi ruuhkaliikenteessä, jossa erilaisten muuttujien määrä on suuri. (Averill, Law & Kelton 2007, 3–4.)

Kun matemaattinen malli on tehty, täytyy tarkastella, miten se sopii järjestelmään, johon sitä on tarkoitus käyttää. Jos malli on tarpeeksi yksinkertainen, voidaan sitä työstää pelkillä suhteilla ja määrillä, ja saada tarkka analyyttinen ratkaisu. Jotkut analyyttiset ratkaisut voivat paisua erittäin monimutkaisiksi ja suorituskykyä vaativiksi, jolloin on tarvetta käyttää simulaatiota, eli numeerisesti työstää mallia kyseessä olevilla syötteillä ja tutkia minkälaista tulostetta se antaa. Simulointia on

usein sanottu viimeiseksi metodiksi mitä käytetään, mutta laajat ja monimutkaiset järjestelmät pakottavat käyttämään sitä monissa tilanteissa. (Averill, Law & Kelton 2007, 5.)

3.4 Simulointi ja tuotantojärjestelmät

Simuloinnin käyttö tuotantojärjestelmissä suunnitteluun ja optimointiin on ollut suosittua viime aikoina. Sen käyttöön ovat johtaneet useat tekijät, joita yleensä ilmenee monissa tuotantojärjestelmissä. Automaation lisäys järjestelmissä on johtanut monimutkaisimpiin järjestelmiin, joiden tutkimiseen tarvitaan yleensä simulointia. Simulointiohjelmien kehitys on johtanut niiden kasvuun rajusti. Tänä päivänä on saatavilla edullisia ja kaikkeen kykeneviä ohjelmistoja, joita monet yritykset ottavat käyttöönsä. Graafiset käyttöliittymät sekä simulointianimaatiot selkeyttävät simuloinnin tuloksia niille, joilla ei ennestään ole tuntemusta simuloinnista. (Averill, Law & Kelton 2007, 669.)

Ehkäpä suurin hyöty simuloinnista on insinöörille tai johtajalle, joka pääsee näkemään koko järjestelmän muutokset, kun tehdään jotain paikallisia muutoksia. Tuotantojärjestelmät vaativat simulointiohjelmistolta muutamia ominaisuuksia, joista suurin osa liittyy materiaalinhallintaan. Muun muassa kuljettimet, trukit, varastot ja erilaiset robotit ovat tärkeitä ominaisuuksia. (Averill, Law & Kelton 2007, 670–672.)

4 TYÖKALUT

Simulointityökalut on yleensä jaettu kahteen eri alueeseen, ohjelmointikieliin sekä simulointiohjelmisto paketteihin. Ohjelmointikielet ovat yleispäteviä kieliä, joilla simulointi tehdään pääasiassa perinteisesti ohjelmoimalla. Ohjelmointikielet tarjoavat lähes rajattomat mahdollisuudet simuloinnille, mutta ne ovat yleensä hankalia käyttää. Simulointiohjelmistot ovat helpompia käyttää, mutta joissain ongelmissa niiden rajoittuneet toiminnot eivät välttämättä riitä. (Averill, Law & Kelton, 189.)

Viimeisten kymmenen vuoden aikana simulointiohjelmistojen kehittäjät ovat tuoneet markkinoille monia ohjelmia, joita on helpompi käyttää ja jotka tuovat joustavuutta simuloinnin kehittämisen. Lisäksi ohjelmistojen kehittäjät ovat yrittäneet tehdä ohjelmistaan laajempikäyttöisiä luomalla niihin sisäisen tuen jollekin tehokkaalle ja helppokäyttöiselle ohjelmointikielelle. Tällaisia kieliä ovat esimerkiksi Java ja Python. Lisäämällä ohjelmoitavuuden tavalliseen simulointiohjelmaan, on kahden erityyppisen simulointityökalun ero alkanut hävitä, ja tilalle on tullut helppokäyttöinen ja monipuolinen tapa kehittää simulaatioita. (Averill, Law & Kelton, 189.)

4.1 Visual Components

Visual Components on yritys, joka tarjoaa Visual Components nimellä varustettua simulaatio-ohjelmistoperhettä, ja se sisältää 3D Create-, 3D Realize- ja 3D Video–ohjelmistot. Ohjelmistot tarjoavat työkalut monimutkaisten visuaalisten tehdaskokonaisuuksien tai yksittäisten laitteiden luomiseen, kasaamiseen ja katseluun. Laitteet ovat komponentteja, jotka koostuvat CAD geometrioista, rajapinnoista ja ominaisuuksista. Komponentit matkivat oikeaa laitetta ja niitä voidaan käyttää uudelleen eri projekteissa. Ohjelma sisältää paljon valmiita komponentteja, jotka ovat mallinnuksia oikeista laitteista. Lisäksi Visual Components 3D Creatella voi luoda omia komponentteja. Usean komponentin kokonaisuuden voi esittää ja tal-

lentaa layouttina, jolloin yhdestä tiedostosta voi ladata kokonaisen tehtaan simulaation. (Visual Components 2008, 4.)

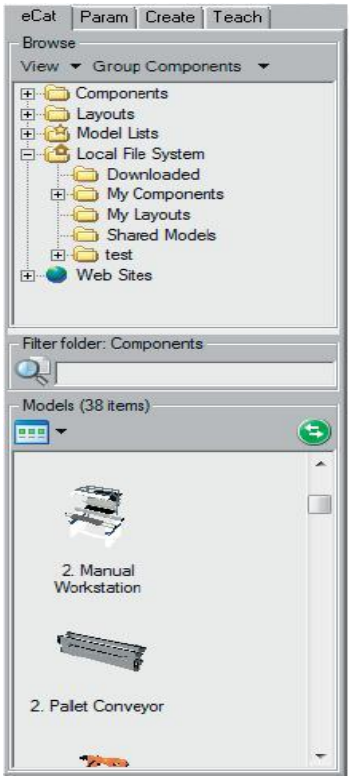
Visual Components perustettiin vuonna 1999, ja ryhmä simuloinnin ja visualisoinnin erikoisosaajia ryhtyi kehittämään uudelleenkäytettävää konseptia simulaatiomalleille. Ensimmäisenä tuotteena julkaistiin 3D Video, joka oli helppokäyttöinen simulaation katseluohjelma. Pian katseluohjelmaa laajennettiin 3D Realize-ohjelmalla, joka tarjosi yrityksille nopean, tehokkaan ja helppokäyttöisen myyntityökalun. Vuonna 2004 julkaistiin simulointikehitystyökalu 3D Create, ja ohjelmistoperhe oli valmis, kun simulointikehitystyökalu 3D Create julkaistiin. 3D Create tarjoaa kattavat simulointityökalut ja kehitystyökalut vaativampaankin käyttöön. (Visual Components 2008 d)

4.2 3D Create

3D Create on komponenttien mallinnukseen ja ohjelmoimiseen tarkoitettu ohjelmisto, jonka avulla voi rakentaa ja julkaista 3D simulaatiokomponentteja, layouteja sekä uusia 3D simulaatioita. 3D Creatella voi tuoda 3D CAD kuvia ohjelmaan, jotka se sitten osaa optimoida toimimaan jouhevasti myös tavallisissa tietokoneissa. 3D Create sisältää kaiken toiminnallisuuden mitä 3D Realize ja 3D Videokin ja se on samalla myös ohjelmiston kehitysympäristö. Kehitysympäristö tarjoaa mahdollisuuden sisällyttää simulaatiomallit muihin ulkoisiin ohjelmiin tai kehittää simulaatiosovelluksia, joissa on paljon muutakin toiminnallisuutta. Python ohjelmointikieli ja Microsoftin COM rajapinta tarjoavat työkalut vaativimpiinkin sovelluskehitys tarpeisiin, joilla voi muokata simulaatiota moniin eri tarkoituksiin. (Visual Components 2008, 6.)

4.2.1 eCAT ja komponenttipaketit

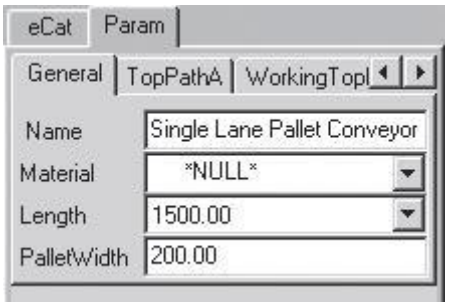
Komponenttien hallinta ja asentaminen 3D Createssa on tehty helpoksi. Komponenttipaketit ja eCat-toiminto helpottavat komponenttien asentamista ja niiden etsimistä koneelta. Komponenttipaketteihin voi koota kaikki simulaatiossa käytössä olevat komponentit yhteen pakettiin, jolloin voi varmistua siitä, että kaikki tarvittavat osat ovat aina mukana tiedostoja siirrettäessä. Komponenttipaketti on yksi tiedosto, jonka käyttäjä voi purkaa omaan eCat-hakemistoonsa. Kaikki paikallisetkomponentit ja etäkomponentit listataan yhteen paikkaan, seuraavan kuvion mukaisesti (kuvio 9), riippumatta niiden fyysisestä olinpaikasta. (Visual Components 2008, 14–17.)



KUVIO 9. eCat komponenttienhallinta

Käyttäjä voi aloittaa simulaatiokehityksen suoraan valitsemalla 3D Createn eCat kirjastosta valmiita komponentteja, ja asettamalla ne simulaatioympäristöön. Komponenteissa on yleensä valmiiksi määriteltyjä parametreja, joita käyttäjä voi muuttaa haluamallaan tavalla. Jokaisella komponentilla on ominaisuudet nimi

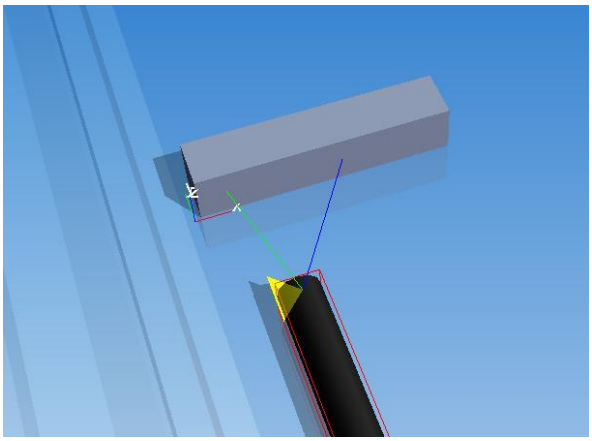
(Name) ja materiaali (Material). Nimi on komponentin nimi ja materiaali on sen väri tai pinta. Muut ominaisuudet ja parametrit, jotka näkyvät seuraavassa kuviossa (kuvio 10), ovat käyttäjän määrittelemiä. Niiden avulla käyttäjä voi esimerkiksi muuttaa komponentin leveyttä tai jotakin muuta ominaisuutta. Parametri tyyppejä on useita: kokonaisluku, reaaliluku, boolean, merkkijono ja URI. Näiden lisäksi on olemassa kokoelma (set) tyyppiset parametrit, kuten esimerkiksi kokonaisluku-kokoelma (integer set), johon voi määritellä useamman kokonaisluvun. Määritellyt kokonaisluvut näkyvät ominaisuuslistassa alasveto-valikkona. (Visual Components 2008, 22.)



KUVIO 10. Komponentin ominaisuuksien hallinta

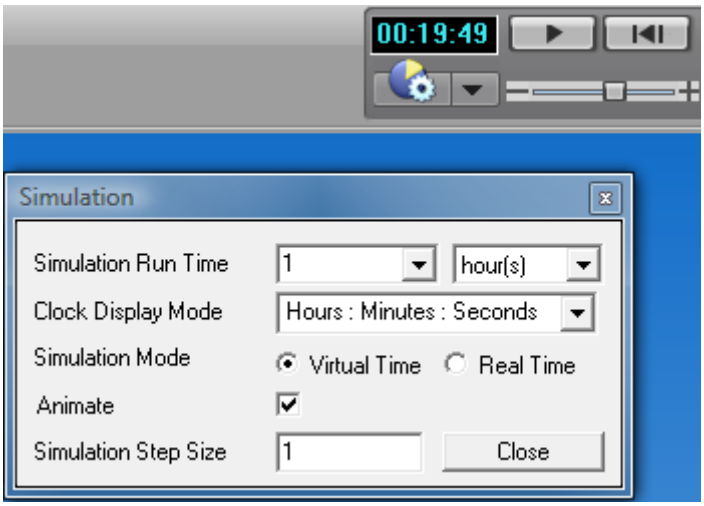
4.2.2 Rajapinta ja simulaatioaika

Kun komponentti on haettu eCat:sta simulaatioympäristöön, se menee automaattisesti plug and play tilaan. Tässä tilassa sitä voi liikuttaa ja yhdistää muihin laitteisiin, jos niissä on yhteensopiva rajapinta. Visual Components 3D Create:ssa komponenttien välinen kommunikointi tapahtuu yleisen rajapinnan välityksellä. Rajapinta on hieman kuin perinteinen kaapeli, joka kuljettaa tietoa laitteelta toiselle. Rajapinnassa on tietokenttiä, joita käytetään määriteltyihin informaatiovirtoihin. Tietokenttiä voi olla loputon määrä ja komponenttien välinen yhteys toimii vain, jos komponenteissa on yhteensopivat tietokentät. Jos tietokentät ovat yhteensopivat komponentteja siirrettäessä lähelle rajapintaliittymää tulee esille seuraavan kuvan mukainen viiva (kuvio 11), joka ohjaa käyttäjän oikeaan suuntaan. Rajapinnat pystyvät välittämään monenlaista tietoa, kuten materiaalivirtoja ja kommunikaatiosignaaleja. (Visual Components 2008, 23.)



KUVIO 11. Rajapinta

Simulaatioaika on 3D-maailman neljäs ulottuvuus. Visual Components:issa simulaatioaika on implementoitu tapahtumapohjaisesti, siten että simulaatioaika etenee epälineaarisina askelina, jotka määritellään fyysisinä tapahtumina. Komponenttien toiminnat käynnistävät tapahtumia, joiden implementointi ei ole samanlaista kuin animoinnissa, jossa aika on käyttäjän määrittelemä. Kaikissa Visual Components tuotteissa on kuvion 12 mallinen simulaationajanhallinta, jolla voi hallita esimerkiksi simulaation nopeutta, ajan näyttöä, toistoa ja keskeytystä.(Visual Components 2008, 19–20.)

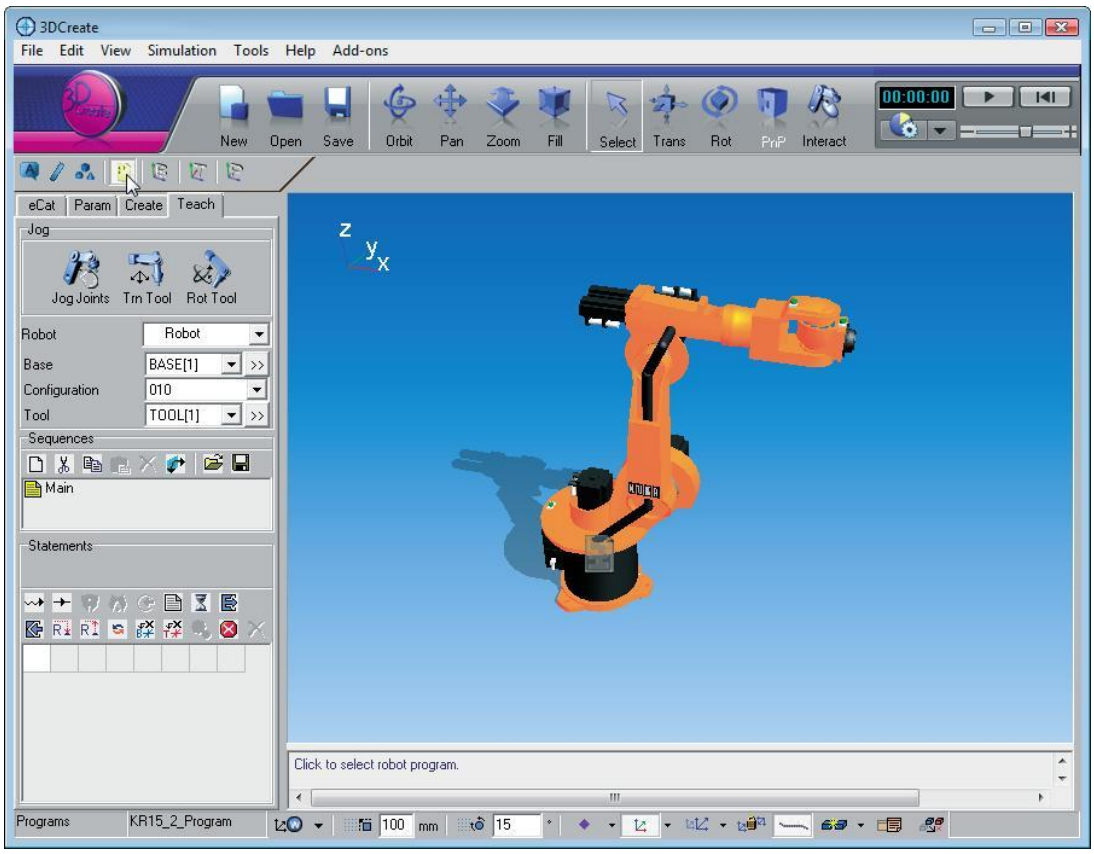


KUVIO 12. Simulaatioajan hallinta

4.2.3 Robotit

3D Creatella voi opettaa valmiiksi mallinnettuja robotteja tekemään erilaisia tehtäviä 3D ympäristössä. Ohjelmassa on kuviossa 13 näkyvä opetustoiminto, jonka avulla voi muokata ja ohjelmoida oikeista roboteista tehtyjä malleja. Robotin niveliä voi siirtää ja tallentaa liikkeet muistiin ja sen varteen voi kiinnittää erilaisia tarttumatyökaluja, joiden liikkeet voi myös määritellä. (Visual Components 2008, 46–52.)

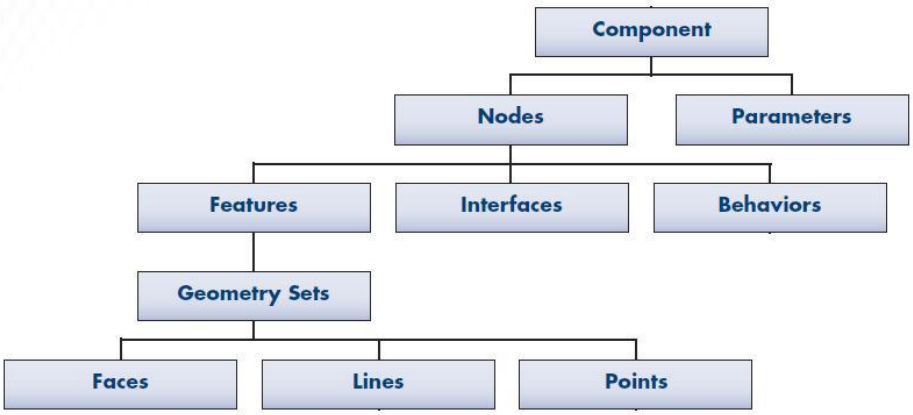
Robottien ohjelmoimiseen tarvitaan myös RSL-kielen (Robot Sequence Language) käyttöä, joka on helppokäyttöinen kieli robottien ohjelmoimiseen. Kieli ei sisällä minkäänlaisia ”if-else”-rakenteita tai ohjelmasilmukoita. RSL koostuu kolmesta tasosta: ohjelma, jakso, lause. Yhdessä RSL:n ja robotin nivelien liikuttamisen kanssa saadaan aikaiseksi robottiohjelma, joka suorittaa ohjelman sisältämät käskyt, kun simulaatiota ajetaan. (Visual Components 2008, 46–53.)



KUVIO 13. Robotin ohjelmointi

4.2.4 Komponenttien luominen

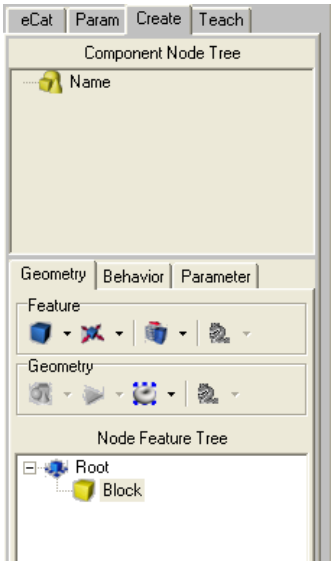
3D Createn yksi tärkeimmistä ominaisuuksista on komponenttien luominen tai muokkaaminen. Tämän ominaisuuden avulla käyttäjä voi muokata simulaatiostaan melkein millaisen tahansa. Komponentti on säiliö erilaisille simulaatio-objekteille, jotka noudattavat kuviossa 14 esiintyvää hierarkiaa. Simulaatio-objekteja ovat esimerkiksi: solmu (node), ominaisuus (feature), käyttäytyminen (behaviour) ja rajapinta (interface). Solmut määrittelevät komponentin rungon. Yksinkertaisissa komponenteissa on yleensä vain yksi solmu, mutta monimutkaisemmissa on useampia solmuja, jotka on järjestetty puurakenteeseen. Ominaisuudet ovat kaikkea, mitä voi nähdä simulaatioympäristössä. Ne ovat geometrisiä muotoja tai muokkaavat muita muotoja. Ominaisuudet on järjestetty omaan ominaisuus puurakenteeseen, jokaisella solmulla on oma ominaisuus rakenne. Käyttäytymiset määrittelevät, miten komponentteja simuloidaan ja rajapinnat määrittelevät kuinka komponentit kommunikoivat toistensa kanssa. (Visual Components 2008, 56.)



KUVIO 14. Komponentin rakenne (Visual Components 2008.)

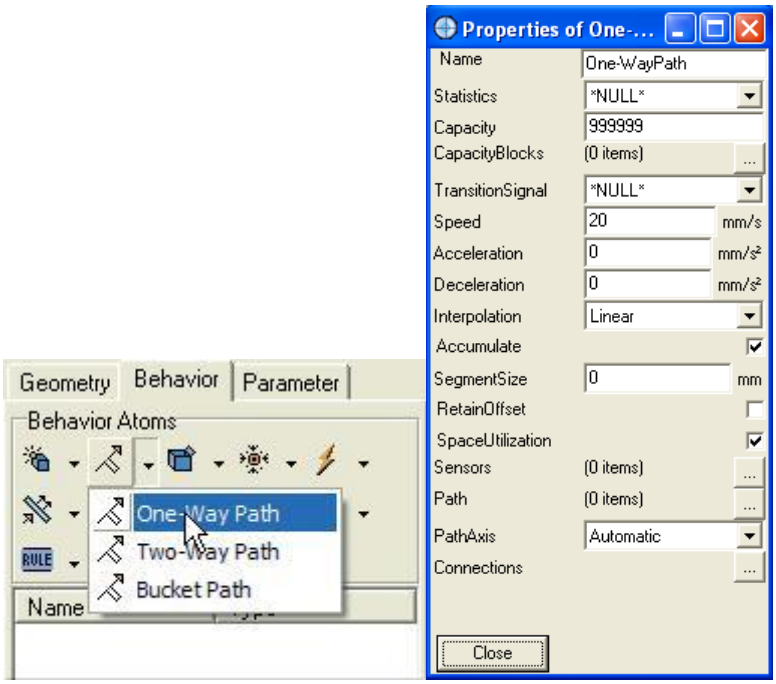
3D Create sisältää perustyökalut komponenttien mallintamiseen, mutta yleensä tarkempia geometrioita tarvittaessa voidaan käyttää eri sovelluksilla tehtyjä mallinnuksia. 3D Create tarjoaa tuen esimerkiksi 3D Studio objekteille ja materiaaleille sekä AutoCAD:in objektitiedostoille. 3D Createssa on kaksi tapaa luoda komponentteja: ne voidaan tuoda tiedostosta tai niistä voidaan tehdä oma komponentti. Oman komponentin luomisessa tarvitsee vain valita minkä mallinen kappale luodaan. Seuraavassa kuviossa (kuvio 15) näkyy kuinka ohjelma lisää uuden

solmun uudelle komponentille, jos yhtään solmua ei ole vielä luotu. Tämän jälkeen komponentin mittoja, sijaintia ja ulkonäköä voidaan muokata. (Visual Components 2008, 57–58.)



KUVIO 15. Solmu ja komponenttihierarkia

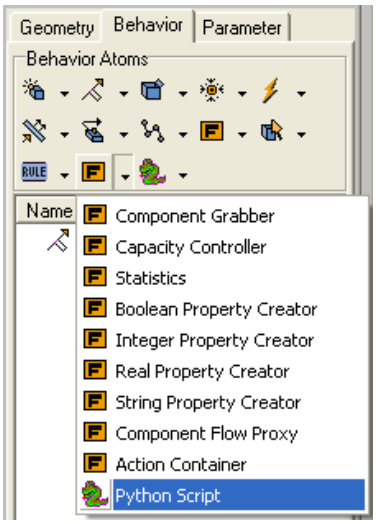
Käyttäytymisvälilehden alla on erilaisia työkaluja, joilla voi luoda komponenteille toimintoja. Esimerkiksi polku (path) toiminto on tehokas työkalu erilaisten kuljettimien kanssa. Polkuja on kolmenlaisia, joista kuviossa 16 on esillä yksisuuntainen polku ja sen ominaisuudet. Polun käyttöä varten tarvitsee luoda kehyskomponentteja (frame), joiden tehtävänä on olla lähtö- ja päätepisteenä. Kehyskomponentit lisätään polkutyökalun ominaisuuksiin, jolloin se tietää, mistä pisteestä mihin polku kulkee. Jokaisella työkalulla on ominaisuusikkuna, josta pääsee muuttamaan sen toimivuutta ja siihen liitettyjä komponentteja haluamakseen. (Visual Components 2008, 61–62.)



KUVIO 16. Polun luominen ja sen ominaisuudet

4.2.5 Komponenttiskriptaus

3D Createssa on kaksi tapaa luoda komponenteille toiminnallisuutta, valmiit käyttäytymiset sekä komponenttiskriptaus. Komponenttiskriptaus tehdään 3D Createssa käyttämällä Python ohjelmointikieltä, johon 3D Createssa on oma sisäinen moduuli nimeltä vcScript. Moduuli sisältää useita luokkia ja komentoja, joita ei ole normaalissa Python ohjelmointikielessä. Komponenttien ulkonäköä ja toimintaa voi muuttaa skriptien avulla. Eri ajoaikafunktiot tarjoavat lisää toiminnallisuutta ja mahdollisuuksia käyttäjälle. Skriptiä voi suorittaa simulaation aikana, sitä ennen ja sen jälkeen, muokatakseen komponentteja ja niiden toimintaa. Lisäksi jokaisella komponentilla voi olla useita skriptejä, jotka tekevät erilaisia tehtäviä. Seuraavassa kuviossa (kuvio 17) näkyy, kuinka Python Script komponentti luodaan. Luonnin jälkeen Python editorin voi avata käyttäytymislistasta. (Visual Components 2008, 76.)

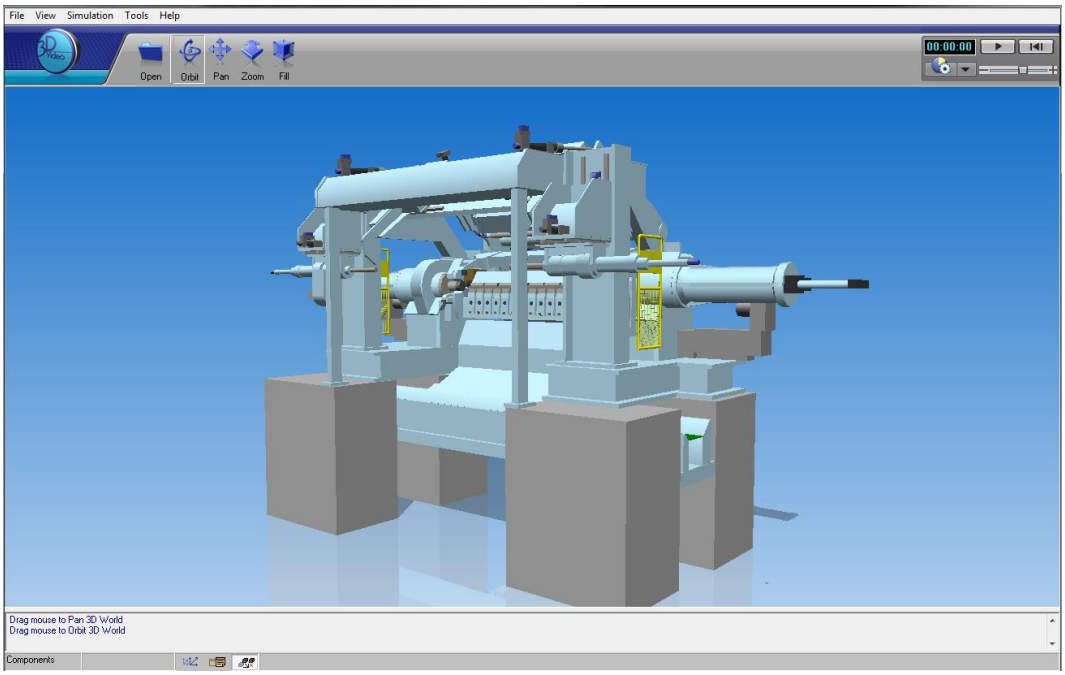


KUVIO 17. Python Script

3D Createssa on myös kaksi muutakin skriptaus tapaa: sovellusskriptit ja komentoskriptit. Sovellusskirptien avulla käyttäjä voi luoda pieniä lisäosia 3D Create-ohjelmaan, jotka latautuvat, kun ohjelma käynnistyy. Niiden avulla voi tehdä muun muassa uusia välilehtiä, tai jotain muita graafisia käyttöliittymäelementtejä. Komentoskripteillä voi luoda uusia komentoja, kuten erilaisia katselukulma-asetuksia, tiedonsiirtoa ja simulaatioympäristön manipulointia. Kun Python-komentoskripti ladataan, se määrittelee funktion ja rekisteröi sen komentona komentovalikkoon, ja laittaa sen käytettäväksi toiminnoksi käyttöliittymään. (Visual Components 2008, 76.)

4.3 3D Video ja 3D Realize

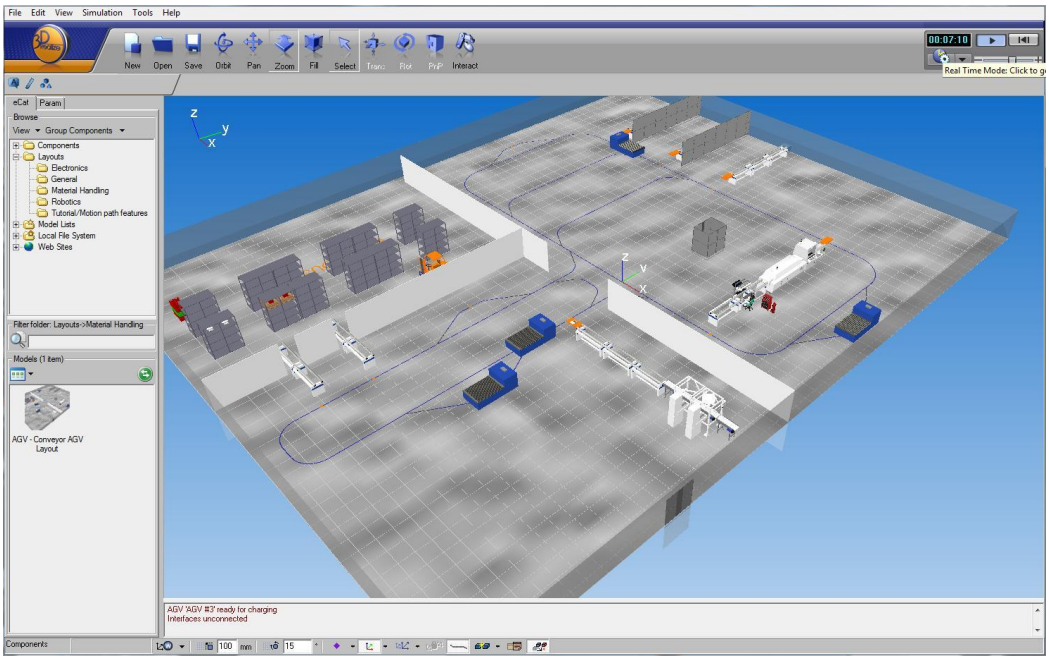
3D Video-ohjelma on ilmainen ja interaktiivinen simulaation katseluohjelma, jota voi käyttää toisilla Visual Component tuotteilla tehtyjen esirakennettujen pohjien katselemiseen. Ohjelma on suunniteltu yksinkertaiseksi ja kevyeksi, jotta sitä voi käyttää kuka tahansa ja myös vähemmän tehokkailla tietokoneilla. 3D videotiedostot ovat pieniä kooltaan, joten niiden lähettäminen asiakkaille tai työntekijöille ei vaadi nopeita verkkoyhteyksiä (Visual Components 2008b). Kuviossa 18 on näkyvissä 3D Video-ohjelma, johon on ladattu komponentti simuloinnin ajamista varten. (Visual Components 2008, 5.)



KUVIO 18. Viilusorvi 3D Videossa

3D Realize-ohjelma on layoutin esittämiseen tarkoitettu työkalu. Sillä voi luoda toimivia moduuleita tai tuotantojärjestelmiä komponentti kirjaston avulla. 3D Realizen käyttökohteita ovat erilaiset myynnin edistämistä helpottavat asiat, kuten materiaalien, prosessointiaikojen ja tilastoanalysointityökalujen tarjoamat kuvaajat. 3D Realizesta on hyötyä myös yrityksen sisäisessä tuotannon analysoinnissa. (Visual Components 2008, 5.)

3D Realize-ohjelmaa on helppo käyttää, sillä on nopea tehdä kuvion 19 kaltaisia tehdaskokonaisuuksia pelkästään vetämällä komponentit luettelosta ruudulle ja yhdistämällä niitä. 3D Realizen komponenttien kirjastot ovat eri automaatiotoimittajien ja Visual Components:n tekemiä, joten kaikkien laitteiden toiminnat ovat todenmukaisia. (Visual Components 2008c).



KUVIO 19. 3D Realize

4.4 Python ohjelmointikieli

Pythonin kehitti Guido van Rossum vuonna 1990. Python nimi on peräisin Monty Python's Flying Circus nimisestä komediasarjasta. Rossumin alkuperäinen idea oli kehittää edistyksellinen skriptikieli ABC-kirjanpitojärjestelmälle. Tuloksena oli kieli, jonka suunnittelu on niin yleinen, että se sopii lähes jokaiseen järjestelmään tai tehtävään. Sitä voidaan käyttää muun muassa: www-sivun luomiseen, C++:n ja Java:n kirjastojen muokkaukseen, elokuva-animaatiohin, simulointiin ja niin edelleen.

Pythonin käytettävyyden rajana ovat vain tietokonetyypit, koska se on kehitetty niin yleiseksi ohjelmointikieleksi, että se soveltuu lähes joka asiaan. Vuonna 1995 Python ohjelmointikielen suosio oli nousussa ja silloin perustettiin Pythonin kotisivu http://www.python.org, joka edelleen toimii virallisena internet sivustona. Pythonin organisaatio koostuu Python Software Foundationista, joka valvoo Pythonin lisenssiä vaikkakin Python on vapaan lähdekoodin ohjelmointikieli, joten se on vapaasti levitettävissä ja muokattavissa. (Lutz 2006, 8.)

Python on yleiskäyttöinen olio-ohjelmointikieli, jota yleensä käytetään skirptaukseen ja jota usein verrataan Perl:iin, Ruby:n tai Java:n. Sitä kutsutaankin yleensä oliopohjaiseksi skriptikieleksi. Skriptikieli viittaa yleensä yksinkertaiseen kieleen, jota voidaan käyttää nopeisiin ohjelmointitehtäviin. Vaikka Python onkin puhdas ohjelmointikieli, sopii termi skriptikieli hyvin kuvaamaan Pythonin olemusta. Pythonilla on paljon nopeampi kehittää sovelluksia, kuin käännettävillä kielillä kuten C++, mutta se ei tarkoita sitä, että Python olisi jotenkin rajallinen ohjelmointikieli. Python voi vaikuttaa rajalliselta yksinkertaisuutensa takia, mutta sen monimutkaisuus tulee esille vasta, kun kaikki sen ominaisuudet on otettu käyttöön. (Lutz 2007, 6–7.)

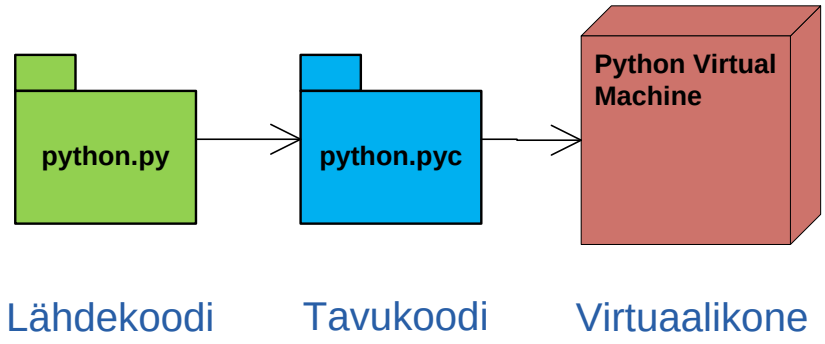
Yksi ainoista heikoimmista alueista Pythonissa on sen ajonopeus, joka ei välttämättä kaikissa sovelluksissa ole C/C++-ohjelmointikielien tasolla. Tämä johtuu pääasiassa siitä, että Python muuntaa lähdekoodin ensin väliformaattiin, joka tunnetaan tavukoodina (byte code), ja sen jälkeen tulkitsee sen. Tavukoodin ansiosta Python on helposti siirrettävissä muille alustoille, koska se on alustariippumaton formaatti. Mutta koska Pythonia ei käännetä binäärikonekoodiksi asti, se ei ole niin nopea kuin C. Monissa sovelluksissa nopeus ei tosin ole riippuvainen käännöksestä vaan esimerkiksi tietokantasovelluksissa pullonkaulana on usein verkkoyhteys. (Lutz 2007, 7.)

4.4.1 Python-tulkki

Tulkit ovat ohjelmia, jotka tulkitsevat muita ohjelmia sellaiseen muotoon, että tietokoneen laitteisto ymmärtää niitä. Python asennuspaketin mukana koneelle asentuu myös Python-tulkki ja kirjastot, joita ilman Python-ohjelmat eivät osaa toimia. Monet Linux jakelut sekä Mac OS X käyttöjärjestelmä sisältävät Pythonin jo valmiiksi. Windows käyttöjärjestelmälle se tarvitsee asentaa erikseen. (Lutz 2007, 22–23.)

Python lähdekooditiedostot ovat .py-päätteisiä ja kääntäjä muuntaa ne tavukoodiksi ja luo tiedoston, joka on .pyc-päätteinen. Python voi ajaa ohjelman pelkistä ta-

vukooditiedostoista vaikka lähdetiedostoja ei olisikaan. Tämä on yksi tapa jakaa Python-sovelluksia. Kun ohjelma on käännetty tavukoodiksi tai tavukooditiedosto on avattu jo olemassa olevasta käännöksestä, se lähetetään ajettavaksi Python-virtuaalikoneeseen (Python Virtual Machine), joka suorittaa tavukoodin ja käynnistää ohjelman. Python virtuaalikone ei ole erillinen ohjelma vaan se on oikeastaan vain yksi iso silmukka, joka iteroi läpi ohjelman tavukoodikäskyt ja ajaa ohjelmaa. Seuraavassa kuviossa (kuvio 20) on havainnollistettu Python sovelluksen kääntämisprosessia. (Lutz 2007, 26.)

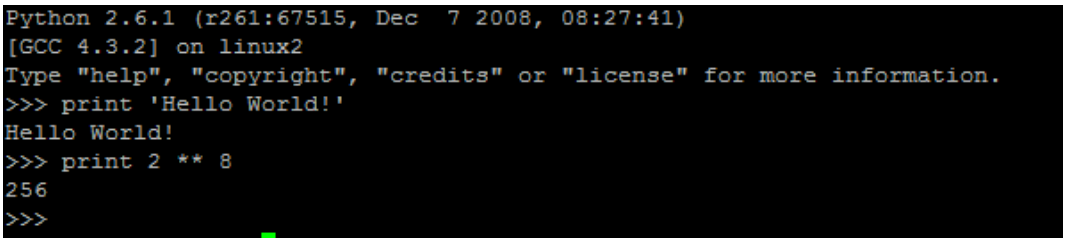


KUVIO 20. Python sovelluksen kääntö- ja käynnistysprosessi

4.4.2 Pythonilla työskentely

Pythonin asennuksessa tulee mukana IDLE kehitysympäristö, joka on yksinkertaisesti komentorivillä ajettava ohjelma, joka ottaa vastaan Python-ohjelmakoodia ja suorittaa sitä. Nopein tapa alkaa käyttää Pythonia on juuri IDLE. Unix ympäristössä se onnistuu kirjoittamalla komentoriville python, jolloin Python-istunto alkaa. Istuntoon voi suoraan kirjoittaa ohjelmakoodia, jonka Python käsittelee ja ajaa konsolinäkymään. Ohjelmassa näkyy >>> merkit kuvion 21 mukaisesti, jotka kertovat käyttäjälle, että Python odottaa seuraavaa komentoa. Kun komento on kirjoitettu ja painettu Enter, Python ajaa koodin, jonka jälkeen se jää odottamaan seuraavaa komentoa. Tämä on yksinkertainen tapa tutustua Pythoniin sekä suorittaa yksinkertaisia komentoja. Toinen tapa on kehittää ohjelma perinteisellä tavalla eli kirjoittamalla koko ohjelmakoodi, tallentaa se tiedostoon ja kääntää sekä ajaa ohjelma. Tämä onnistuu jollakin tekstieditorilla sekä monilla nykyisin suosituilla

ilmaisilla kehitysympäristöillä kuten NetBeans tai Eclipse, joihin kumpaankin on mahdollista saada Python plug-in. (Lutz 2007, 34-35.)



KUVIO 21. Python-istunto Linux-ympäristössä

Python-istunnossa onnistuvat monimutkaisemmatkin ohjelmointitehtävät, sillä vaikka se ajaa käyttäjän kirjoittaman koodirivin heti Enterin painalluksen jälkeen, se silti muistaa kaikki edellä tapahtuneet syötteet, vaikka käyttäjä jatkaa ohjelmointia. Yhdellä rivillä voi esimerkiksi alustaa muuttujan numerolla 1 ja sitten myöhemmin muuttaa sitä. Tätä seuraavassa kuviossa 22 näkyvää tapaa kutsutaan interaktiiviseksi ohjelmoimiseksi. Kuitenkin suuremmat pythonit projektit ja ohjelmat tehdään lähes poikkeuksetta jossakin graafisessa kehitysympäristössä. Niiden tarjoamat debug- ja editointiominaisuudet ja tietenkin tiedostopohjainen kehitys auttavat paljon, kun projekti on laajempi. (Lutz 2007, 34–36.)

```
>>> import random
>>> lottorivi = random.sample(range(1, 40), 7)
>>> lottorivi
[11, 1, 30, 21, 23, 32, 16]
>>> lottorivi.sort()
>>> print 'Voittava rivi: ' + ' '.join(str(x) for x in lottorivi)
Voittava rivi: 1 11 16 21 23 30 32
```

KUVIO 22. Monimutkaisempi ohjelma Python IDLE-ympäristössä

4.4.3 Pythonin ominaisuudet

Python tarjoaa monia sisäänrakennettuja työkaluja tiedon hallintaan. Näitä ovat esimerkiksi listat (list) ja etsintätaulut (dictionaries). Nämä työkalut helpottavat ja nopeuttavat ohjelman kehittämistä huomattavasti, kun aikaa ei tarvitse käyttää monimutkaisten tietorakenteiden suunniteluun ja luomiseen. Esimerkiksi pinon (stack) luominen voidaan Pythonissa toteuttaa tekemällä luokka, joka käyttää hyväkseen sisäänrakennettua lista tietorakennetta. Python käyttää myös valmiiksi optimoituja algoritmeja nopeuttaakseen tiedonhallintaa. Monesti nämä ovat tehokkaampia kuin itse toteutetut versiot ja heti käytettävissä. Pythonissa on myös perustietotyypit, jotka käsittelevät ja säilövät esimerkiksi numeroita, merkkijonoja, tiedostoja. Pythonissa ei käytetä tietotyyppien esittelyjä (esimerkiksi int i), vaan siinä voidaan suoraan määritellä uusi muuttuja tyyliin i = 1, joka luo i:stä automaattisesti integer-tyyppisen muuttujan. Hakasulkeiden lisääminen 1:n ympärille tekisi siitä listan, jossa on yksi alkio. Python on siis dynaamisesti, mutta vahvasti tyypitetty ohjelmointikieli. Sen tietotyypit todetaan ajon aikana, ja ne ovat pakotettuja eli kokonaisluku muuttujaa ei voida käsitellä merkkijonon tavoin. (Lutz 2007, 65–67.)

Pythonin funktioiden määrittelyssä ei tarvita esittelyitä, prototyyppejä tai otsikkotiedostoja (header). Kun käyttäjä tarvitsee funktiota, voi sen helposti esitellä seuraavasti: def esimFunktio(param1, param2):, jossa def aloittaa funktion esittelyn, sulkeiden sisälle tulevat argumentit ja esittely päättyy kaksoispisteeseen. Funktioissa ei määritellä paluuarvon datatyyppiä, eikä Python funktiot palauta mitään muuta kuin Python null arvon ellei return-lausetta ole funktiossa. (Dive Into Python 2004, 9–10.)

Python funktioilla ei ole varsinaisia alkuja tai loppuja, kuten aaltosulkeet. Ainoa erottaja on funktion jälkeinen kaksoispiste, jonka jälkeiset rivit sisennetään funktion alle. Python:issa sisennys toimii erottajana eri funktiolle ja ehtolauseiden sisällöille, kuten seuraavan kuvion 23 funktiosta voi nähdä. (Dive Into Python 2004, 13.)

```
def multiprint( n, txt ):
    i = 0
    while i < n:
        print txt
```

KUVIO 23. Python funktio

Python on täysiverinen olio-ohjelmointikieli: siinä voi määritellä omia luokkia, periä omista tai sisäänrakennetuista luokista ja toteuttaa luokat. Luokkien määrittely Pythonissa on yhtä helppoa kuin funktioidenkin. Luokat aloitetaan class sanalla ja päätetään kaksoispisteeseen aivan kuin funktioissakin. Luokan toteuttaminen tapahtuu yksinkertaisesti kutsumalla sitä kuin funktiota, kuten kuviossa 24 on esitetty. Kun oliota ei enää tarvita, Python siivoaa roskat pois automaattisesti. (Dive Into Python 2004, 50–53.)

```
class foo: #luokan luominen
    a, b, c, = 0, "bar", (1,2) # luokan ominaisuudet

i = foo() #luokan toteutus
print i.a, i.b, i.c #tulostetaan luokan ominaisuudet
```

KUVIO 24. Python luokan luonti ja toteutus

Pythonin vahvuuksia ovat oliopohjaisuus, helppokäyttöisyys, ilmaisuus, siirrettävyys, tehokkuus ja toiminta muiden kielien ja ohjelmien seassa. Pythonia kehitetään jatkuvasti ja sen käyttäjäkunta kasvaa koko ajan, joten se on nykyaikana ja tulevaisuudessa erittäin varteenotettava ohjelmointikieli. (Lutz 2007, 12–19.)

4.4.4 Python 3D Createssa

Pythonin tehtävänä 3D Createssa on tuoda ohjelmaan lisää mahdollisuuksia tehdä erikoisia simulaatioita ja toimintoja. Sitä voi käyttää komponenttien ominaisuuksien lisäämiseen tai itse 3D Createn toimintojen lisäämiseen. Komponenttiskriptit voivat suorittaa tehtävät ennen simulaatiota, sen aikana ja sen jälkeen. Skriptin metodit suoritetaan tapahtumissa, joita on kahdenlaisia, simulaatio ajallisia ja -ajattomia. Seuraavassa kuviossa (kuvio 25) on havainnollistettu tapahtumafunkti-

oiden toimintaa sekvenssikaavion avulla. Ylempi osuus käsittelee simulaation latausta ja alemmassa osiossa simulaation käynnistystä ja sammutusta. (3D Create 2009.)

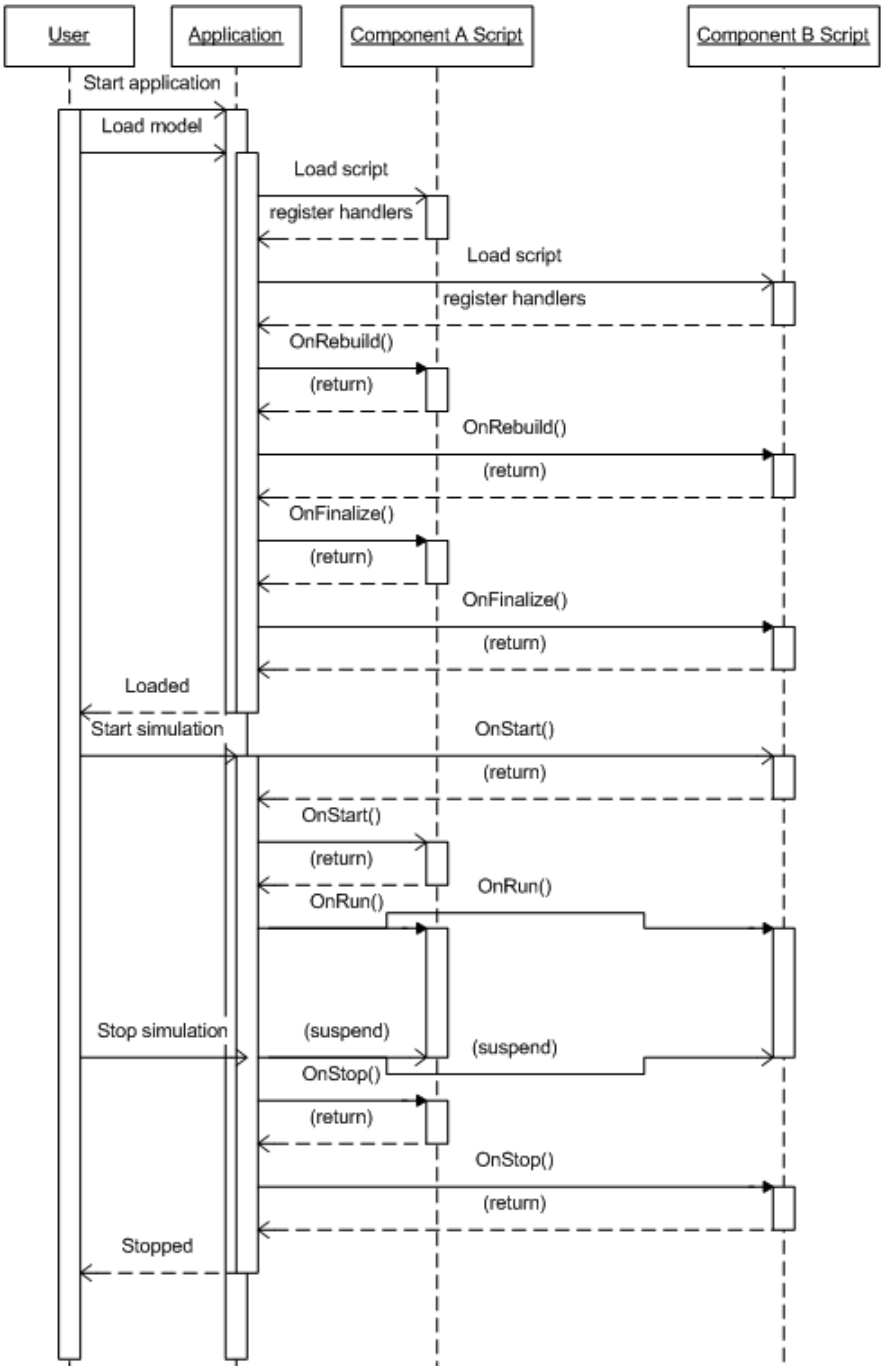
Simulaatioajallisia tapahtumia:

- OnStart() - Ensimmäinen metodi, jota kutsutaan, kun simulaatio käynnistetään. Tässä metodissa olisi hyvä alustaa simulaatioon liittyvät muuttujat.
- OnRun() - Pääsilmukka, jonka suoritus alkaa, kun simulaatio alkaa ja OnStart() on suoritettu. Tapahtuman suoritus tapahtuu vaiheittain simulaation käydessä.
- OnStop() - Kutsutaan, kun simulaatio pysäytetään.
- OnContinue() - Kutsutaan, kun simulaatiota jatketaan.
- OnReset() - Kutsutaan, kun simulaatio resetoidaan.
- OnSignal() - Signaalinkäsittelijä, jota kutsutaan, kun signaali yhdistetään vastaanottavaan skriptiin.
- OnSimulationUpdate() - Kutsutaan, kun kutsuttua simulaatiota päivitetään, eli kun siinä tapahtuu jotain.
- OnDestroy() - Kutsutaan kun skripti tuhoutuu.

Ei simulaatioajalliset tapahtumat:

- OnFinalize() - Kutsutaan, kun kaikki komponentit on ladattu ja kopioitu.
- OnRebuild() - Kutsutaan, kun jonkun komponentin geometriaa rakennetaan uudelleen, eli kun se muuttuu jotenkin.

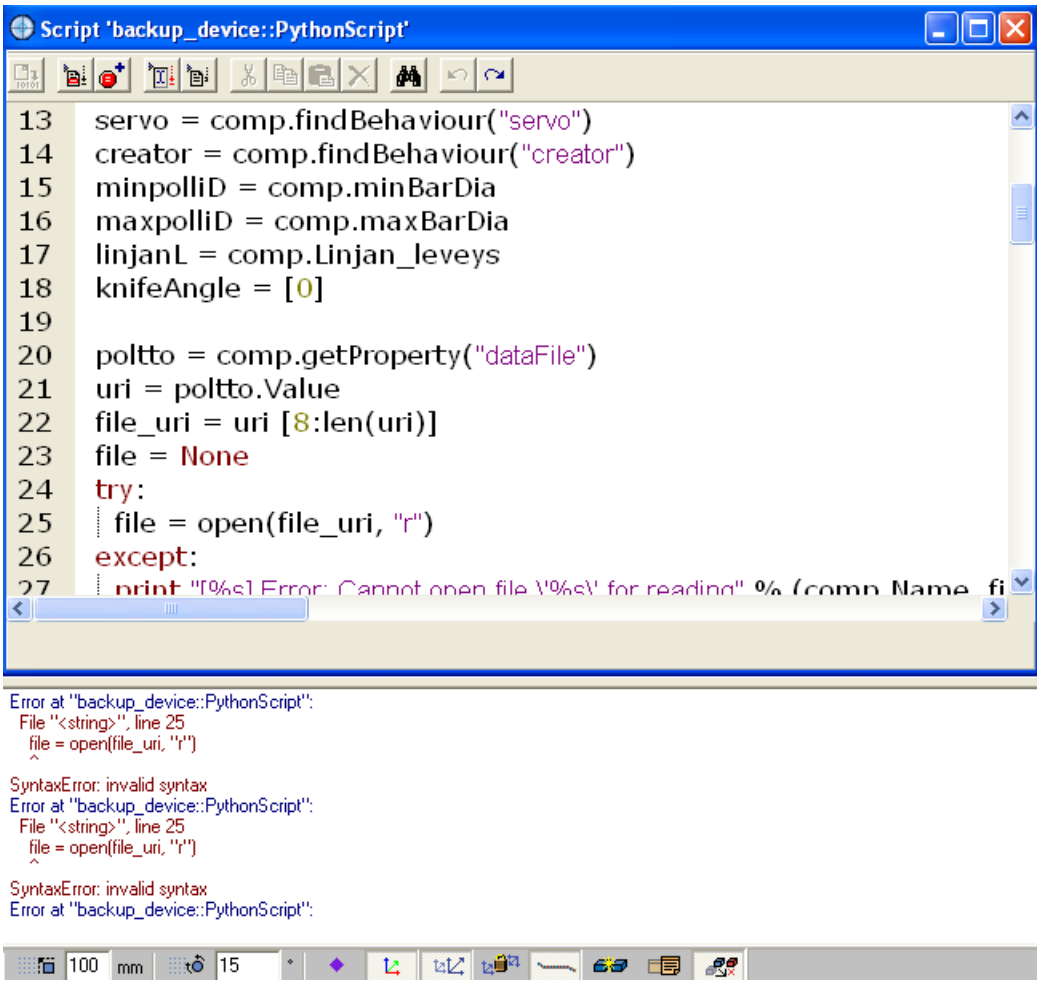
Näiden tapahtumafunktioiden lisäksi Pythonilla on mahdollista tehdä myös omia funktioita, joita voidaan kutsua esimerkiksi tapahtumafunktiossa. Koska kaikkien komponenttien skriptit toimivat itsenäisesti, ei voida tietää etukäteen, kutsutaanko komponentti A:n tapahtumakäsittelijää ennen komponentti B:n tapahtumakäsittelijää tai päinvastoin. Tämän takia komponenttien välinen kommunikaatio hoidetaan signaaleilla, kuten komponentit, jotka kommunikoivat rajapintojen kanssa keskenään. (3D Create 2009.)



KUVIO 25. Tapahtumafunktioiden toiminta (3D Create 2009)

Python Script käyttäytyminen (behaviour) luodaan Behaviour-välilehdessä, ja sen sijainnilla solmussa (node) ei ole suurta merkitystä. 3D Createssa on Python Script:ille oma editori, jossa ohjelmakoodia voi työstää. Editori tarjoaa perus tekstinkäsittely toiminnot, tiedostoa ei tarvitse tallentaa, mutta sen voi tallentaa erilliseen tiedostoon. Editorista löytyvät myös tiedoston avausmahdollisuus, ohjelmakoodin kääntäminen, kursoriin asti suoritus, trace ja breakpoint. Python Script luo

automaattisesti muutaman koodirivin, kun käyttäytyminen luodaan. Siellä on pääohjelmasilmukka OnRun()-tapahtumafunktiossa, OnSignal(signal)-tapahtumafunktio sekä aina ensimmäisellä rivillä tarvittava from vcScript import* lause, jotka ovat näkyvillä seuraavassa kuviossa. Import-lauseella importoidaan Visual Components script moduuli, jonka avulla käyttäjä pääsee käsiksi vcScript-objektin sisältämiin metodeihin. Python Script:issä olevat virheet tulostuvat ohjelman alalaidassa olevaan ikkunaan, seuraavan kuvion 26 mukaisesti. (3D Create 2009.)

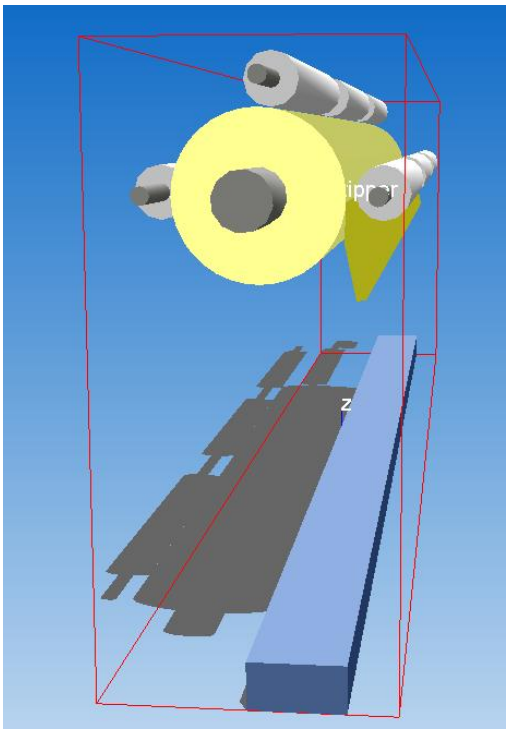


KUVIO 26. Python Script editori sekä debug-näkymä

5 SIMULAATION LUOMINEN

Tehtävänä oli perehtyä Rauten viilusorvin toimintaan ja tehdä siitä simulaatiomalli Visual Components 3D Create- simulointiohjelmalla. Tehtävä aloitettiin perehtymällä sorvin toimintaan ja hankkimalla tietoa sen geometrioista ja liikkeiden matemaattisista kaavoista. Tehtävän lopullinen tavoite ei ollut aluksi vielä selvillä, joten sitäkin vielä suunniteltiin ja päädyttiin tulokseen, jossa sorvin simulaation tulisi palvella pääasiassa koulutustarkoituksissa ja toisena tavoitteena palvella myyntitarkoituksia.

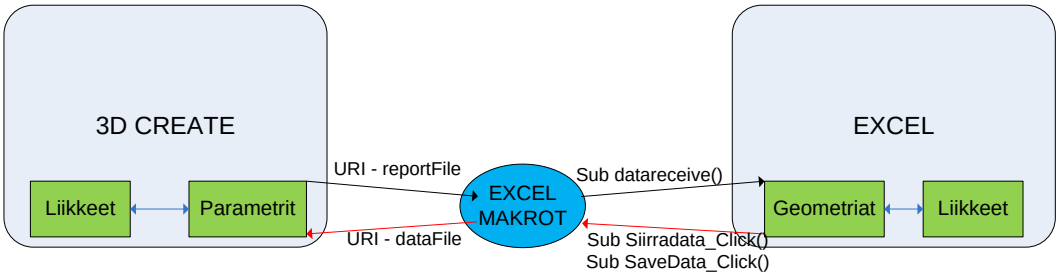
Ennen varsinaista simulaation kehittämistä tehtiin seuraavassa kuviossa 27 näkyvä yksinkertainen simulaatiomalli, jonka tarkoituksena oli hahmottaa sorvin toimintaa simulaatioympäristössä. Tarkoituksena oli mallintaa tukitelat, vastaterä, terä sekä pyörivä tukki, jonka halkaisija pienenee ajan kuluessa. Yksinkertaisen mallin jälkeen ryhdyttiin suunnittelemaan varsinaista työtä.



KUVIO 27. Yksinkertainen sorvin simulaatiomalli

5.1 Suunnitelma

Tavoitteena oli tehdä parametrisoitu simulaatio, joka käyttäisi jo olemassaolevia oikean sorvin geometrioita ja liikkeitä. Simulaatiosta pitäisi saada selkeä kuva sorvin toiminnasta ja siitä pitäisi olla kaksi näkymää selvyyden vuoksi, luuranko-malli sekä täydellinen graafinen malli, joita voi vaihdella nappia painamalla. Sorvin parametrit ja liikeradat ladataan 3D Createen Microsoft Office Excel:stä, johon tehtiin pieniä muutoksia, jotta se toimi 3D Createn kanssa. Excel-tiedostoon tehtiin makroja, jotka synkronoivat parametrit ja geometriat 3D Createn kanssa, kuten kuviossa 29 on esitetty. Jos 3D Createssa muutetaan jotain parametriä, ne voidaan ladata Excel-tiedostoon, joka laskee uudet liikeradat ja lähettää uudet arvot takaisin 3D Create-ohjelmaan.



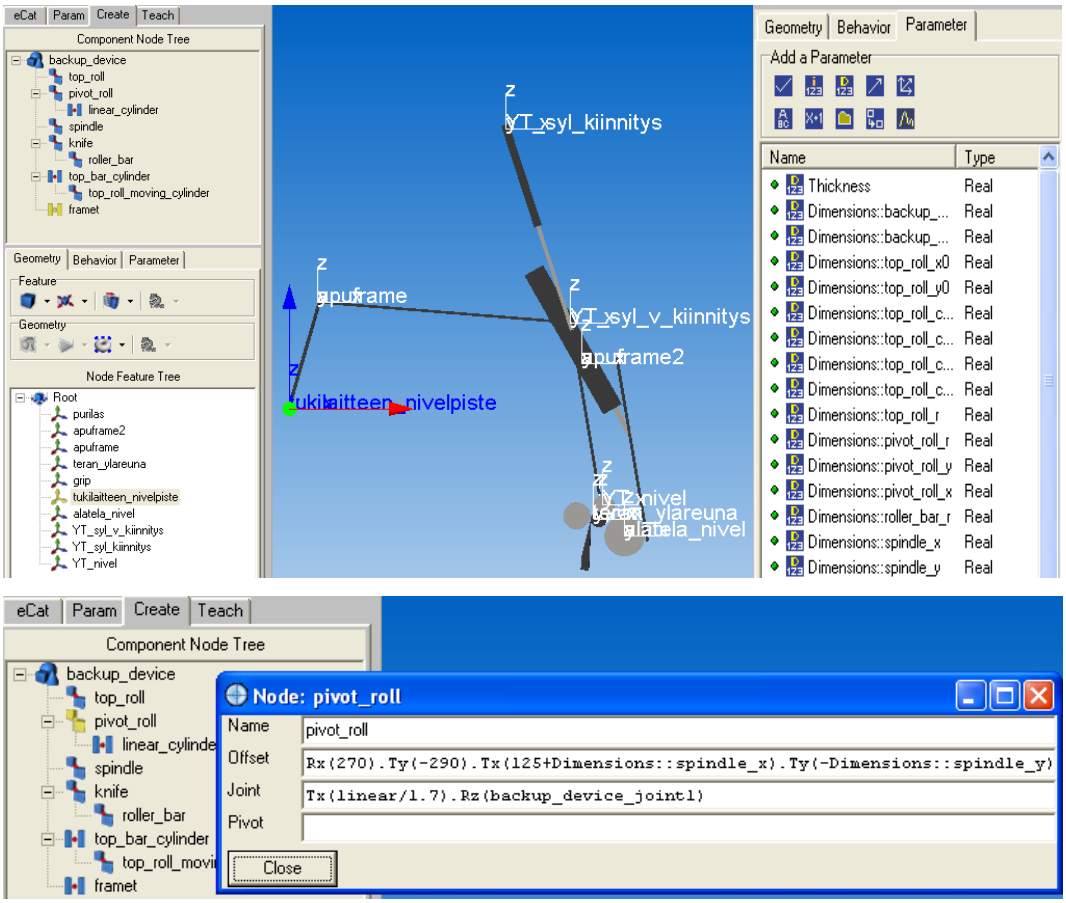
KUVIO 28. Simulaatiosuunnitelma

3D Create-ohjelman puolella luodaan edellisen kuvion mukaiset URI-tyyppiset parametrit, joihin voi arvoiksi liittää tiedostoja. Simulaatiossa tarvitaan muutamaa eri tekstitiedostoa, joiden välityksellä Excelin tietoja voidaan välittää. Pythoniin on olemassa Excel-moduuli, jonka avulla voidaan tietoja siirtää suoraan, ilman että niitä tarvitsee siirtää tekstitiedoston kautta, mutta sen toimivuutta 3D Create-ympäristössä ei lähdetty kokeilemaan. Simulaatiokoodin täytyy huomata muutokset parametreissa, jolloin se aina päivittää tekstitiedostot, kun parametrit vaihtuvat.

5.2 Sorvin mallinnus

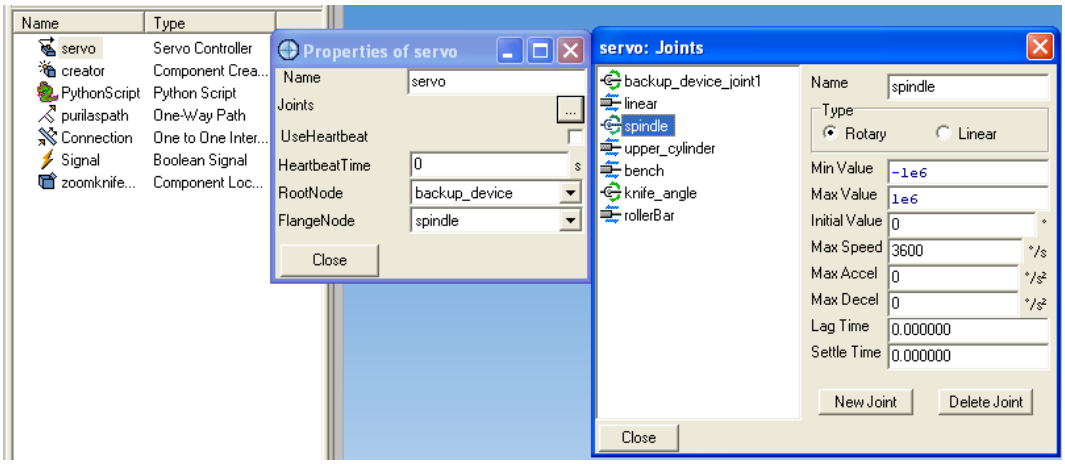
Kun tietoa sorvin liikkeistä ja geometrioista oli kerätty tarpeeksi, voitiin mallin hahmottelu aloittaa 3D Creatella. Tarkoitus oli ensin tehdä luurankomalli ja saada se toimimaan jouhevasti sorvin oikeilla parametreilla, jonka jälkeen graafinen malli kohdistettaisiin luurankomallin päälle, oikeisiin kohtiin. Graafinen malli oli jo olemassa ja se vaatisi vain hieman muutoksia, jotta se saataisiin samaan kokoon kuin luurankomalli.

Luurankomallin suunnittelu aloitettiin asettamalla kehys (frame) komponentteja oikeille paikoilleen seuraavan kuvion 29 mukaisesti, jonka jälkeen niiden avulla oli helppo asetella luuranko-komponentit oikeille kohdille. Komponenttien luomisessa oli otettava huomioon nivelten (joint) käyttäminen. Silloin kun käytetään niveltoimintoa, jonka avulla komponentteja voidaan liikutella lineaarisesti tai pyörivällä liikkellä, täytyy komponentit tehdä aina origossa ja siirtäminen tapahtuu offsetin avulla. Jos komponentteja siirretään siirroksella (transform) eli perinteisesti vetämällä, aiheutuu siitä ongelmia simulaatiossa, kun nivelet alkavat liikkua, koska nivelet liikuttavat komponentteja aina origon suhteen. Offset-arvoina käytettiin seuraavassa kuviossa 29 näkyviä parametreja, joihin sorvin oikeat arvot tallenettiin ja joita voidaan muuttaa Param-välilehdestä ja siirtää ne Exceliin.



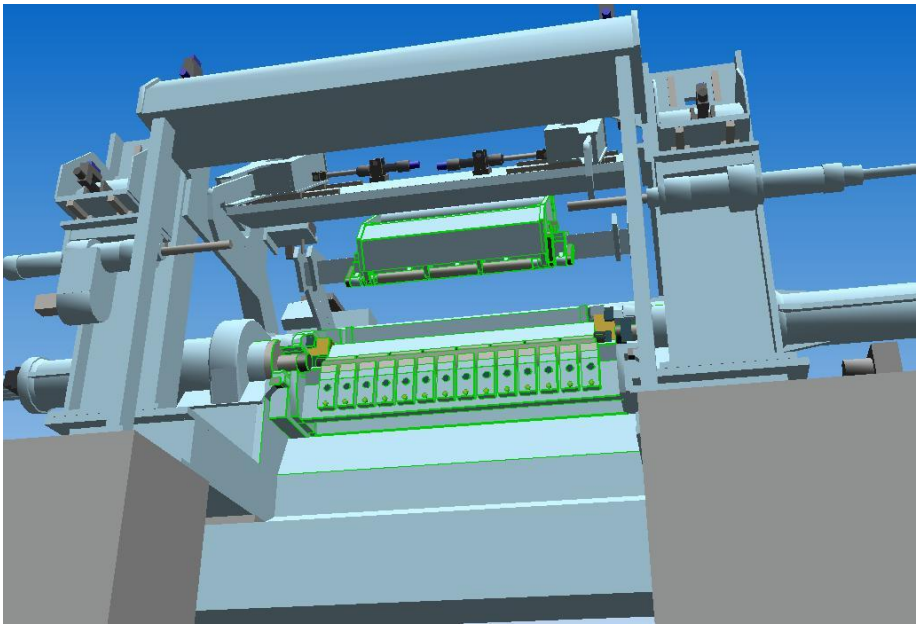
KUVIO 29. Framet, luurankomalli, parametrit, nivelet (joint) sekä offset

Nivelet toimivat simulaatiomallin komponenttihierarkiassa, eli solmuissa ja ne määritellään Servo käyttäytymisessä seuraavan kuvion 30 tavalla, jossa niille määritellään liikkeen tyyppi sekä muitakin ehtoja ja parametrejä. Mallia suunniteltaessa täytyy miettiä, minkälaisia liikkeitä kukin komponentti tulee tarvitsemaan. Solmulle määritellään offsetit ja jointit, joita solmuun liitetyt alasolmut ja komponentit perivät. Joint-kenttään määritellään liikkeen suunta sekä arvo. Laajemmissa liikkuvissa komponenttikokonaisuuksissa, joissa on monenlaista eri liikettä, komponentit on hyvä ryhmitellä eri solmuihin ja luoda niille omat nivelet.



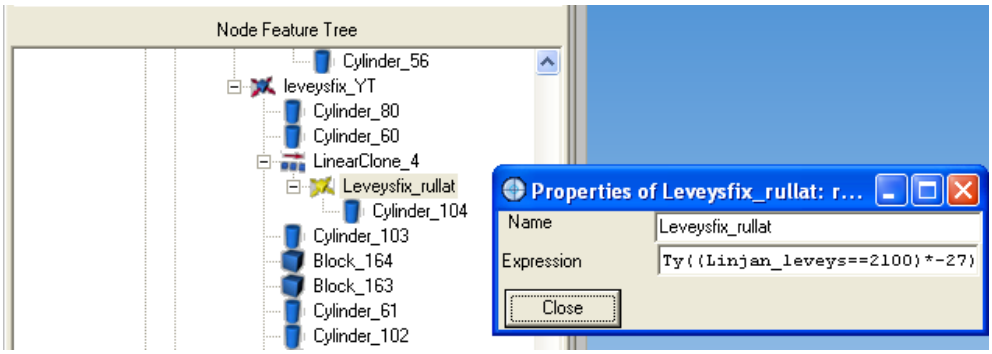
KUVIO 30. Servo ja siihen liittyvät jointit

Kun luurankomalli oli saatu toimimaan halutulla tavalla, alettiin työstämään valmiista graafisesta mallista sopivaa versiota luurankomallin päälle. Ensimmäisenä tehtävänä oli karsia siitä sorvilaitteen runkoon liittyvät komponentit pois ja jättää jäljelle vain tarvittavat tukilaitteet ja teräpenkki. Seuraavasta kuviosta 31 voi nähdä, kuinka siitä on valittu vain tukilaitteen ja terän komponentit. Graafinen malli oli tehty hieman eri tavalla ja eri koordinaatteihin, joten sitä piti käännellä ja siirrellä sopivaan kohtaan. Tämän lisäksi sitä täytyi skaalata, jotta se olisi yhtenevä oikeiden mittojen kanssa.



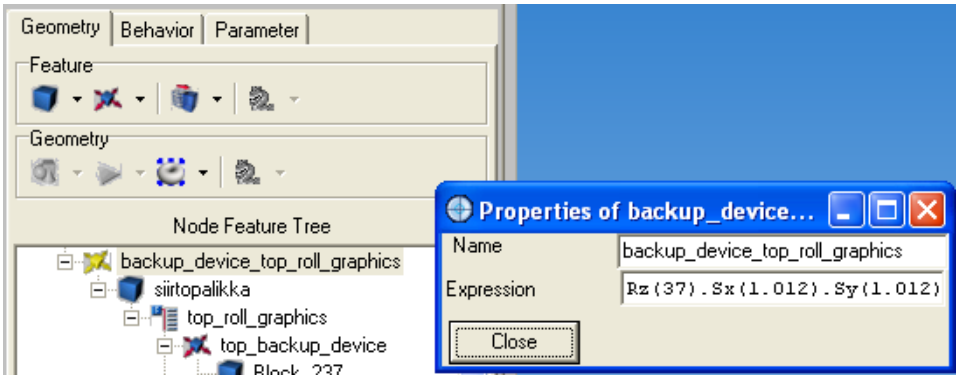
KUVIO 31. Alkuperäinen sorvimalli, jossa tukilaite ja teräpenkki valittuna

Alkuperäisessä graafisessa mallissa ollut linjan leveydenmuunnos ominaisuuteen täytyi tehdä pieni korjaus, koska se ei suoraan toiminut halutulla tavalla, kun mallia oli käännetty. Leveyden muuntaminen aiheutti sen. että osa komponenteista liikkui väärään suuntaan koordinaatistossa, joten tämän korjaamiseksi täytyi tehdä siirros (transform) komponentti, jonka ominaisuusvalikon ilmaisu (expression) kentässä määriteltiin ehtolauseella, milloin komponenttia siirrettiin koordinaatistossa. Määrittely tapahtuu Tx, Ty ja Tz syötteillä, seuraavan kuvion 32 mukaisesti.



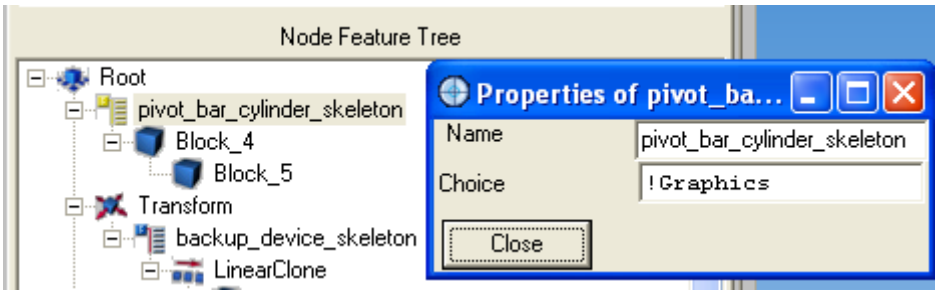
KUVIO 32. Linjan leveyden muuttamisen korjaus ehtolauseella

Lisäksi graafista mallia oli skaalattava hieman, jota varten luotiin myös transform-komponentti, jossa skaalaus suoritettiin. Skaalaus suoritetaan transform-komponentin properties-valikon expression kentässä, jossa syötteellä Sx, Sy ja Sz voidaan komponenttia skaalata haluttuun suuntaan xyz-koordinaatistossa. Seuraavassa kuviossa 33 näkyy x- ja y-akselin suuntaisen skaalauksen määrä.



KUVIO 33. Graafisen mallin skaalaus

Kun luurankomalli ja graafinen malli oli onnistuneesti yhdistetty päällekkäin, täytyi enää luoda boolean tyyppinen parametri ja switch-komponentteja, jotka huolehtivat siitä, mitkä komponentit näkyisivät. Käyttäjä voi muuttaa boolean parametria Param-välilehdestä check box-napilla arvoihin true ja false. Switch-komponentin alle asetetut komponentit näkyvät vain, kun switch-komponentin Choice-kentän ehto on tosi. Seuraavassa kuvassa (kuvio 34) näkyy switch-komponentti, jonka alla olevat komponentit näkyvät vain, kun Graphics parametri on epätosi.



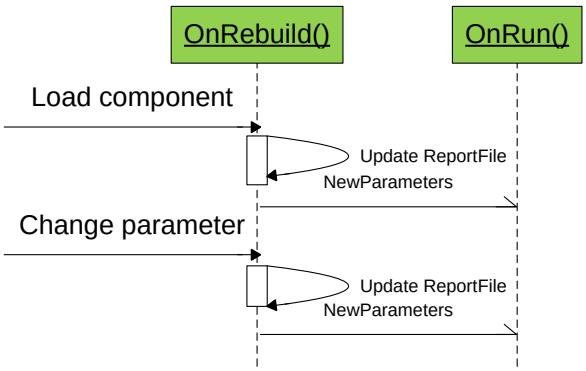
KUVIO 34. Switch-komponentti ja boolean parametri Graphics

Tämän jälkeen kaikki oli valmista toiminnan toteutukselle, eli liikkeiden ohjelmoimiselle, toimintojen suunnittelulle, Excel-tiedoston toimintojen suunnittelulle sekä Python ohjelmakoodin luomiselle.

5.3 Toimintojen toteutus

Tässä simulaatiossa kaiken toiminnan ydin oli Python Script behaviour, joka kontrolloi kaikkia servoja, pölli-komponentin luontia, parametrien sekä liikeratojen siirtämistä Excel-tiedostosta 3D Create-ohjelmaan ja takaisin Excel-tiedostoon. Lisäksi sen tehtäviin kuuluu useiden parametrien kontrollointi. Monesti simulaatioita voidaan tehdä kokonaan ilman- tai vähäisellä Python ohjelmoinnilla, mutta tässä simulaatiossa, jossa parametreja haetaan, luetaan ja päivitellään ulkoisesta tiedostosta, sen rooli oli merkittävä.

Python Scriptissä on kaksi funktiota, OnRun(), jonka 3D Create luo aina automaattisesti Python Script-komponenttia luodessa sekä OnRebuild(), joka on Visual Components Python moduulin yksi funktio. OnRun()-funktiossa luetaan tiedostot, jotka Excel on luonut ja siellä tapahtuu kaikki servojen liike. Funktio käynnistyy, kun simulaatio käynnistetään start-napista. OnRebuild()-funktion tehtävä on päivittää parametrit tekstitiedostoon, joiden avulla Excel voi laskea arvot uudelleen liikeradoille. OnRebuild() funktio toteutuu kuvion 35 sekvenssikaavion mukaisesti aina ladattaessa simulaatiomallia tai jos simulaatiomallissa tapahtuu joku uudelleenrakennus tapahtuma, esimerkiksi parametrin arvon muuttaminen. Jos parametrin ominaisuuksiin ei ole määritelty Simulating, niin sitä ei kesken simuloinnin voida vaihtaa.

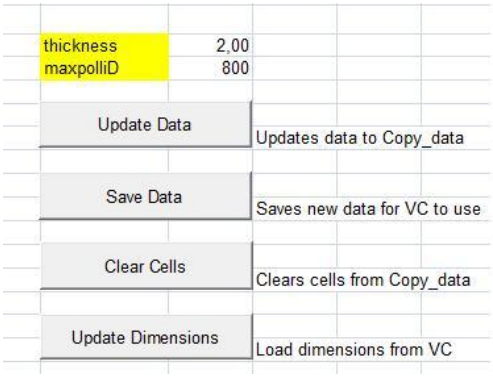


KUVIO 35. OnRebuild()-funktion toimintaa

5.3.1 Excel-toiminnot

Excel-tiedoston tarkoituksena on toimia pääasiassa liikeratojen muuttajana. Simulaatiomallissa olevaan URI-parametriin määritellään tekstitiedosto, joka sitten tuodaan Excel-tiedoston johonkin määrättyyn tyhjään taulukkoon, jonka jälkeen Excel:issä olevat liikeratojen laskukaavat laskevat liikeradat uusilla arvoilla. Excel-tiedoston toiminta toteutettiin neljällä yksinkertaisella makrolla, jotka on liitetty kuviossa 36 näkyviin Excel:in forms-komentonappeihin. Nappia painamalla

Excel ajaa makron, joka on tehty joko nauhoittamalla tai kirjoitettu Excel:in sisältämällä Visual Basic-ohjelmointikielellä.



KUVIO 36. Excel-tiedoston käyttöliittymä

Update Data-napin toiminto on yksinkertaisesti siirtää liikeradat tyhjään taulukkosivuun. Nappia painamalla ohjelmakoodi valitsee alueen, jossa liikeradat sijaitsevat ja kopioi sen, jonka jälkeen siirtyy toiselle taulukkosivulle ja liittää kopioidut solut sinne. Seuraavassa kuviossa 37 on esiteltynä koko vaadittava koodi solujen siirtämiseen taulukosta toiseen.

```
Sub Siirradata_Click()

Sheets("Data_Handling").Select  'Select Data_Handling sheet
Range("B5:E805").Select 'Select a range of cell:cell
Selection.Copy 'Copy the selection
Sheets("Copy_data").Select 'Select first cell of Copy_data sheet
bottomcel = Range("A65536").End(xlUp)
ActiveCell.Offset(0, 0).Select  'Make sure first cell is selected
Selection.PasteSpecial Paste:=xlValues, Operation:=xlNone,
SkipBlanks:=False, Transpose:=False 'Paste
Module3.knifedata 'Run Module 3
ActiveCell.Offset(0, -4).Select 'Set active cell back to first
Sheets("Data_Handling").Select 'Go back to Data_Handling sheet

End Sub
```

KUVIO 37. Update Data-napin toiminta

Save Data -napin toimintona on palata tyhjään taulukkosivuun ja muuttaa se tekstitiedostoksi, jotta simulaatiomallin Python ohjelmakoodi osaisi tulkita sitä. Tekstitiedostoksi muuntaminen tapahtuu avaamalla käyttäjälle tallennusdialogi, jossa käyttäjä voi valita hakemiston ja tiedoston nimen, johon taulukkosivun tiedot tal-

lennetaan samassa muodossa kuin ne ovat taulukossakin. Clear Cells-napin toimintana on tyhjentää koko toinen taulukkosivu, jos siirto-operaatiossa on jossain vaiheessa jokin epäonnistunut. Seuraavan kuvion 38 neljäs koodirivi luo tallennusdialogin ja loppuosassa käsitellään taulukkosivun valinta ja taulukon tallennus tekstitiedostoksi.

```
Sub SaveData_Click()

Dim fn As Variant
'Set default name and type for file to save, exit if it fails
fn = Application.GetSaveAsFilename("Copy_data.txt", _
     "Excel files,*.txt", 1, "Select your folder and filename")
    If TypeName(fn) = "Boolean" Then Exit Sub

Sheets("Copy_data").Select 'Select Copy_Data sheet
Application.DisplayAlerts = False
ActiveWorkbook.SaveAs FileName:=fn, FileFormat:=xlText,
CreateBackup:=False 'Save the active work book
Application.DisplayAlerts = False
Sheets("Data_Handling").Select 'Go back to Data_Handling sheet

End Sub
```

KUVIO 38. Save Data-napin toiminta

Viimeisenä toimintana Excel-tiedostossa on Update Dimensions, jonka tehtävänä on hakea simulaatiomallin tallentamasta tekstitiedosta uudet geometriat sekä siirtää ne oikeisiin soluihin, josta kaavat osaavat suorittaa laskutoimitukset. Geometriat on tallennettu tekstitiedostoon, joka voidaan importoida Excel-tiedostoon helposti. Importointi suoritetaan napinpainallus skriptissä, jonka jälkeen vastaanotetuista parametreista valitaan halutut ja siirretään ne oikeille paikoilleen. Seuraavassa kuviossa 39 on näkyvillä viimeinen koodirivi, joka päivittää taulukon tekstitiedostosta.

```
Sub datareceive()

Col = -1

Sheets("Python_Data").Select 'Select Python_Data sheet
Range("A1").Select  'Select A1

Selection.QueryTable.Refresh BackgroundQuery:=False 'Update table from txt file

End Sub
```

KUVIO 39. Update Dimensions toiminta

5.3.2 Python Script

Python Script käyttäytyminen on luotu juuri (root) solmuun, sijainnilla ei ole juuri merkitystä, sillä käyttäytymisen nimen perusteella sen voi liittää muihin komponentteihin tai toimintoihin mistä vain. Python Scriptin tehtävänä on lukea tekstitiedostoista Excel-tiedoston tallentamat liikeratatiedot, käyttää niitä servojen arvoina ja viedä parametrien tiedot tekstitiedostoon.

Jos skriptissä tarvitaan muuttujia, jotka muuttuvat simulaation aikana, niin ne pitää yleensä nollata ajojen välissä tai aina jokaisen ajon alussa. Tässä työssä tällaiset muuttujat on määritelty globaaleiksi muuttujiksi, ja ne nollataan aina simulaation alussa. Kun simulaatio nollataan, komponentti palaa alkuasentoon. Globaaleita muuttujia voidaan kutsua funktioissa lisäämällä muuttujan eteen global määrittely, kuten seuraavassa kuviossa 40 on nähtävillä. Tähän simulaatiomalliin tarvittiin listamuuttujia sekä normaaleja liukuluku (float)- tai kokonaisluku (integer)-tyyppisiä muuttujia. Lisäksi getComponent()-metodille luotiin myös globaali muuttuja, koska sitä käytetään jokaisessa funktiossa, jossa pitää päästä kiinni johonkin käyttäytymiseen, komponenttiin tai parametriin. Listoihin säilötään terän ohjauskulman arvot, jotka on määritelty parametreihin, ja joita käyttäjä voi muuttaa.

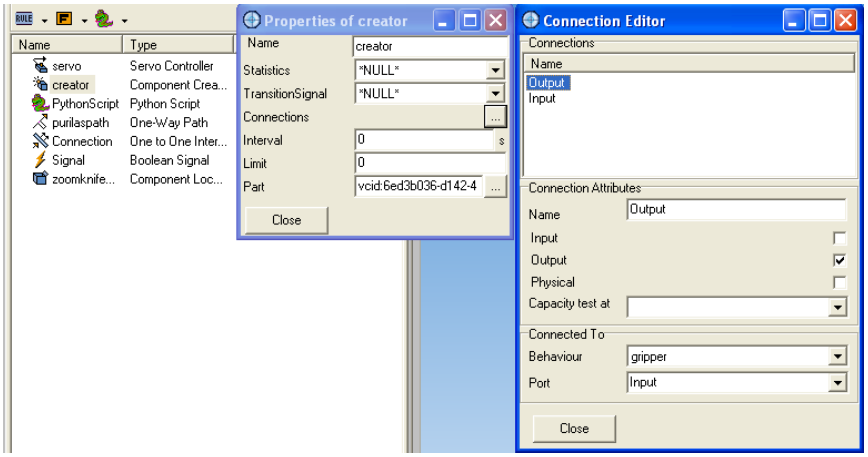
```
#global
counter = bar = ylat = ylat2 = alat = ylaisku = 0  #counter and movement
comp = getComponent()
thickness  = comp.Thickness #thickness of the veneer
knifeListlen = 0  # the lenght of the knife parameter list
knifeAngle = [] # knife angle parameters list
knifeAngleName = [] #knife angle name list
alatGraph = [] # data transferred to excel
diamGraph = [] # data transferred to excel

def OnRun():
  global counter, ylat, bar, alat, ylat2, ylaisku, comp, knifeAngle, knifeListlen, knifeAngleName
  global alatGraph, ylatGraph, diamGraph
```

KUVIO 40. Globaalit muuttujat

Simulaation liike perustuu pääasiassa pöllin halkaisijan muutokseen, kun terä leikkaa pölliä pienemmäksi joka kierroksella, tarvitsee muiden komponenttien liikkua mukana. Tässä työssä pöllin luonti tapahtuu creator-käyttäytymisellä, jolla

luodaan uusi komponentti kehyksen määrittämään paikkaan. Pölli komponentille on annettu parametri sen halkaisijalle ja pituudelle, joita muutetaan simulaation aikana Excel-taulukon mukaisesti riippuen määritellystä viilunpaksuudesta ja linjan pituudesta, jotka määritellään parametreista. Pölli komponentin polku määritellään creator-välilehteen kuvion 41 tapaisesti, josta creator-käyttäytyminen osaa avata komponentin sitä kutsuttaessa.



KUVIO 41. Creator Behaviour

Python Script:issä toteutetaan pöllin luonti, pyöritys, liikutus pois karoilta sekä tuhoaminen. Liikuttamiseen on käytetty polku käyttäytymistä (path behaviour), jolle määritellään kehys, kiihtyvyys ja nopeus, jossa kehys määrittää paikan johon komponentti liikkuu. Simulaatiossa on myös kaksi muuta parametria sorvauksen ja pöllin hallintaan, maxBarDia ja minBarDia (kuviossa 41 minpolliD ja maxpolliD), jotka ovat reaalilukutyyppisiä ja niiden tehtävänä on määritellä pöllin maksimi koko sekä minimikoko, joka määrittää sorvauksen loppumisen. Kun liikeratoja käydään läpi tekstitiedostosta niin servot alkavat liikkua vasta, kun on päästy määritellyn maksimihalkaisijan alle ja käsittely lopetetaan kun minimihalkaisija saavutetaan, jonka jälkeen pölli siirretään polkua pitkin pois karoilta ja tuhotaan. Tämä on nähtävissä kuvion 42 if-ehtolauseessa.

```
def OnRun():
  ...
  purilaspath = comp.findBehaviour("purilaspath")
  servo = comp.findBehaviour("servo")
  creator = comp.findBehaviour("creator")
  minpolliD = comp.minBarDia #the minimum bar diameter
  maxpolliD = comp.maxBarDia #the maximum bar diameter
  linjanL = comp.Linjan_leveys #the width of the line
  ...
  part = creator.create() #create the bar
  part.D = maxpolliD
  part.polliL =  linjanL-40 #lenght of bar
  ...
  if bar < minpolliD/2: #if bar is small enough
    purilaspath.grab(part) #drop it
    delay(2)
    counter = 0
```

KUVIO 42. Käyttäytymis- ja parametri määrittelyt, pöllin luonti ja pudotus

Liikeratojen haku tiedostosta ja uusien parametrien vienti tiedostoon on tässä työssä Python Script:in tärkeä käyttötarkoitus. Tekstitiedostoille luodut kaksi URI-parametria määrittelevät tiedostoille hakemistopolun, jonka avulla tiedostoon päästään käsiksi skriptin puolella. Ensiksi URI-parametrin arvo haetaan muuttujaan, jonka jälkeen käytetään kuviossa 43 näkyvää try-except rakennetta sen avaamiseen. Tämän jälkeen rivien määrä luetaan muuttujaan, jota voidaan käyttää myöhemmin for-lauseessa, kun tiedoston tietoja käydään läpi.

```
poltto = comp.getProperty("dataFile")  #file for fetching data from excel
uri = poltto.Value
file_uri = uri [8:len(uri)]
file = None

try:
  file = open(file_uri, "r") #open file for reading
except:
  print "[%s] Error: Cannot open file \'%s\' for reading" % (comp.Name, file_uri)
  return

lines = file.readlines()  #read the amount of lines in the file
```

KUVIO 43. Tiedoston lukeminen

Tiedostoon kirjoitus toteutetaan samantapaisesti, try-except rakenteella, vain open metodiin laitetaan w:n sijasta r argumentti. Tiedostoon kirjoitus on toteutettu On-

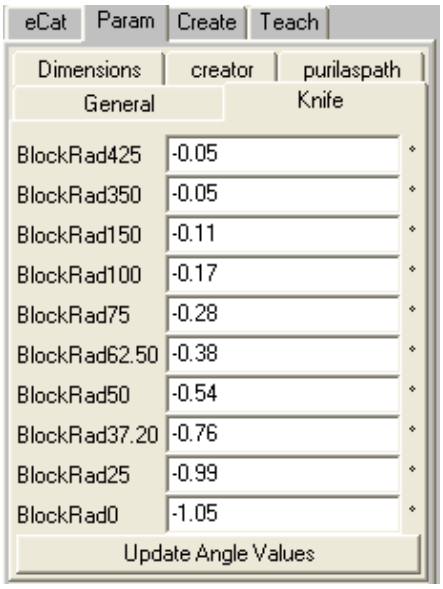
Rebuild()-funktiossa, joka ajetaan aina, kun komponentissa tai parametreissa tapahtuu joitakin muutoksia. Toinen URI-parametri pitää sisällään tekstitiedoston, jonne tallennetaan ne parametrit, jotka halutaan siirtää Exceliin. Parametrit saadaan haettua getComponents().Parameters- metodilla, josta suodatetaan pois kaikki parametrit, jotka eivät ole real-tyyppisiä. Tämän jälkeen halutut parametrit ja niiden arvot kirjoitetaan tiedostoon write()-metodilla. Parametrit haetaan ja kirjoitetaan tiedostoon for-lauseessa seuraavan kuvion 44 mukaisesti.

```
try:
  file = open(file_uri, 'w')
except:
  print "[%s] Error: Cannot open report file \'%s\' for writing" % (comp.Name, file_uri)
  return

for i in comp.Properties: #get all parameters
  if i.Type == VC_REAL: #from those parameters get all REAL parameters
    #write name and value to the file
    value = round(i.Value,2)
    file.write(str(value))
    file.write("\t")
    file.write(str(i.Name))
    file.write("\n")
print uri, "**UPDATED**"
```

KUVIO 44. Tiedostoon kirjoittaminen

Kaikki servojen liikkeet suoritetaan yhdessä while-silmukassa, joka pyörii niin kauan, kun simulaatio on päällä. Ennen while-silmukkaan etenemistä, terän ohjauskulman liikkeet otetaan listaan talteen. Teränohjauskulmalle on kymmenen parametria, joista jokainen muuttaa terän kulmaa. Kulman muutos tapahtuu tietyllä pöllin säteen arvolla, joka on nähtävillä seuraavan kuvion parametrien nimessä. Parametreja voidaan muuttaa aina jokaisen pöllin jälkeen, jos simulaatiota halutaan ajaa eri kulmalla. (Kuvio 45)



KUVIO 45. Terän parametrit

Parametrien nimen perään on lisätty se pöllin halkaisija, jonka arvo on kyseessä, jotta Python Script:issä voidaan hakea juuri oikea arvo, kun simulaatiota pyöritetään. Terän ohjauskulman parametrit lisätään listaan for-silmukassa, joka sisältää kuvion 46 mukaisen ehdon if i.Name[0:12] == ”Knife::Block”:, jonka avulla kaikista parametreista otetaan listaan vain ne, joiden nimi vastaa ehtoa. Lisäksi listan pituus ja parametrin nimi otetaan talteen, ja niitä käytetään yhtenä ehdoista terän liikutuksessa.

```
for i in comp.Properties: #reading the names and values of knife parameters to list
  if i.Name[0:12] == "Knife::Block": #from all parameters we choose only the ones with Knife::Block in the name
    knifeAngle.append(i.Value) #add to list the value
    knifeAngleName.append(i.Name[15:20]) #add to list the name
knifeListlen = len(knifeAngle)
```

KUVIO 46. Terän parametrien lisäys listaan

Koko simulaation keskeisin osa on Python Script:in OnRun()-funktiossa oleva pääsilmukka, jonka tehtävänä on hallita kaikkea liikettä, silloin kun simulaatio on käynnissä. Silmukka on toteutettu while True -ehdolla, jonka sisällä sijaitsee pöllikomponentin luominen, pöllin pudotus karoilta sekä kaikki servojen liikuttamiskäskyt. While -ehto pyörii niin kauan, kun OnRun()-funktio on aktiivinen eli kun simulaatio on päällä. Pääsilmukan sisällä on for-ehto, joka käy läpi aiemmin ava-

tun tekstitiedoston kaikki rivit ja jossa erotellaan Excel-tiedostosta tuodut liikeratojen rivit viiteen lohkoon. Lohkoista saadaan jokaiselle eri servolle oma parametri, joka muuttuu jokaisella for-silmukan kierroksella, parametrit tallennetaan eri muuttujiin joita sitten käytetään servon liikutus metodissa. Seuraavassa kuviossa 47 on nähtävillä while-silmukka, pölli-komponentin luonti sekä for-silmukka, jossa eri muuttujiin tallennetaan tekstitiedostossa olevia liikeratoja.

```
while True: #the main loop
  part = creator.create() #create the bar
  part.D = maxpolliD
  part.polliL =  linjanL-40 #lenght of bar

  for line in lines:
    p = line.split("\t",5) #split the lines to 5 cols
    bar = float(p[0])   #Reading values from the file
    alat = float(p[1])
    ylat = float(p[2])
    ylaisku = float(p[3])
    rollerBarDia= float(p[4])
    rollerBar = float(p[5])
```

KUVIO 47. While- ja for-silmukat sekä tekstitiedoston lohkojen talteenotto

Seuraavaksi pääsilmukassa käsitellään servojen liikutusta ja niiden ehtoja. Ensimmäinen ehto koskee määritettyä pöllin maksimihalkaisijaa, jota verrataan tekstitiedosta haettuun bar-muuttujaan. Jos bar-muuttuja on pienempi kuin määritetty pöllin halkaisija, aloitetaan servojen liikkeet ja pöllin halkaisijan pienentäminen. Servojen liikkeitä kontrolloidaan servo.setJointTarget(index, arvo)-metodilla, jossa index on nivelen (joint) järjestysnumero listassa ja arvoksi laitetaan haluttu liikemäärä. (Kuvio 48)

Terän liikkeiden suoritus koostuu seuraavassa kuviossa olevista neljästä if-ehdosta. Ensimmäisessä tarkastellaan, onko laskurimuuttujan arvo suurempi kuin teränarvojen listan pituus, jos ei ole, niin voidaan tarkastella, onko pöllin halkaisija pienempi kuin terän parametrinimen säde. Jos pölli on jo pienempi kuin terän säde laskurimuuttujan kohdalla, tallennetaan sen halkaisija-arvon mukainen teräkulman muutos muuttujaan, ja liikutetaan servoa sekä lisätään laskuria yhdellä. Tämä rutiini jatkuu niin kauan kunnes pöllin halkaisija on pienempi kuin määritel-

ty minimipaksuus, jonka jälkeen pölli pudotetaan ja poistutaan for-silmukasta ja while-silmukka aloittaa uuden kierroksen.

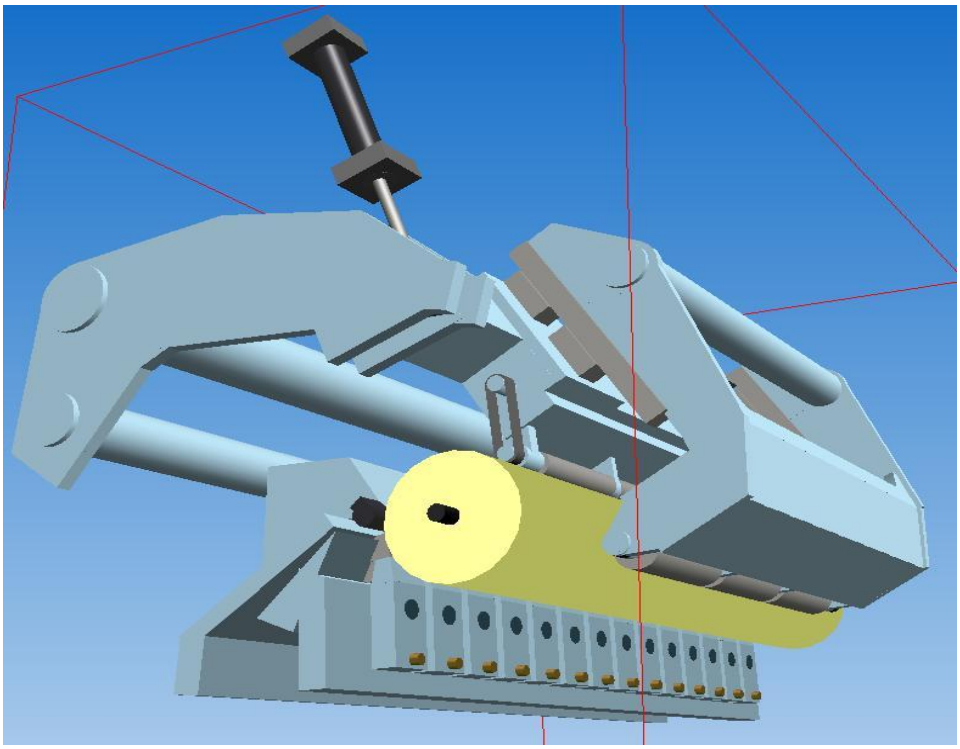
```
if bar < maxpolliD: #start moving the joints when bar diameter goes below the maximum.
  part.D = bar    #change bar diameter
  servo.setJointTarget(0, ylat-55) #backup device joint angle
  servo.setJointTarget(1, alat-50) #nosebar roll linear movement
  servo.setJointTarget(3, ylaisku) #upper cylinder movement
  servo.setJointTarget(2, -bar) #bar
  servo.setJointTarget(4, (bar/2)-35)  #bench
  servo.setJointTarget(6,  -rollerBar) #rollerbar

  if counter < knifeListlen: #knife movement
    if bar < (float(knifeAngleName[counter])*2): #if bar dimension has passed the knife angle point,
      kangle  = float(knifeAngle[counter]) #it moves the knife according to the parameter in that dimension value
      servo.setJointTarget(5, kangle) #move knife joint
      counter += 1
  servo.move()  #move servos
  servo.setJointTarget(2,0)
  servo.moveImmediate()

  if bar < minpolliD/2: #if bar is small enough
    purilaspath.grab(part) #drop it
    delay(2)
    counter = 0
```

KUVIO 48. Servojen liikuttaminen

Näiden vaiheiden jälkeen simulaatio on valmis toimimaan. Luodaan pölli-komponentti, haetaan tekstitiedostoista tarvittavat liikeradat sekä liikutetaan servoja. Pythonin käyttö Visual Components 3D Create simulaatiokehityksessä on varteenotettava vaihtoehto, kun tarvitaan erikoisempia ratkaisuja mallin kehittämiseen. Se on tehokas ohjelmointityökalu, jolla on nopea kehittää simulaatioita, ja joka sallii monimutkaisten liikkeiden kontrolloinnin suoraan skriptissä ilman ohjelman hidasteluita. Kuviossa 49 on esillä viilusorvin simulaation lopputulos, jossa on näkyvissä graafinen malli sekä pölli pyörimässä.



KUVIO 49. Viilusorvin simulaation lopputulos

6 YHTEENVETO

Tämän työn tarkoituksen oli tutkia vanerin tuotantoon oleellisesti liittyvän viilusorvin simulointia, Visual Components 3D Create -simulointiohjelmalla. Tavoitteena oli luoda simulaatiomalli sorvin tukilaitteen toiminnasta, jonka tarkoituksena on tukea puutukkia sen pyöriessä terää vasten. Simulaatiomallin parametrit noudattavat oikean laitteen mittoja ja sen liikeradat synkronoidaan Excel-taulukossa sijaitsevilla laskukaavoilla, jos parametreja muutetaan. Simulaation tarkoitus oli auttaa hahmottamaan tukilaitteiden ja sorvin toimintaa sekä tarjota apua myynnin edistämiseen ja koulutukseen.

Simuloinnissa oli tarkoitus käyttää jo aiemmin tehtyjä laskutoimituksia sorvin tukilaitteiden liikkeistä. Tämän takia Visual Components 3D Create -ohjelmassa olevalla Python Script:illä oli merkittävä osuus simuloinnin onnistumisessa. Python-ohjelmointikielellä simulaation kehittäminen oli nopeaa ja helppoa. Se sopii erittäin hyvin 3D Create-ohjelmaan tukemaan jo muita hyviä ominaisuuksia. Pythonilla voi paneutua suoraan itse asiaan, eikä syntaksiin tai muihin teknisiin asioihin tarvitse kiinnittää niin paljon huomiota.

Simulaation mallinnus onnistui toivotulla tavalla ja tärkeimmät toiminnot, kuten Excel-taulukon kanssa toiminta, parametrit ja liikkeet, toimivat kuten pitääkin. Simulaation käyttötarkoituksena oli olla myynnin ja koulutuksen apuvälineenä omissa tai asiakkaiden projekteissa ja laitostoimituksissa. Työn onnistumisen myötä simulaatiota voidaan alkaa käyttämään erilaisissa koulutus tarkoituksissa tai tukena erilaisissa myyntitilanteissa. Lisäksi simulaatio tarjoaa erinomaisen alustan jatkokehitykselle.

Jatkokehitystä tehtiinkin vielä opinnäyteosuuden jälkeen. Kehitettiin esimerkiksi komponentti, joka piirtää siirreltävän ruudun, suurentaen terän ja tukilaitteiden näkymää monta kertaa lähemmäksi. Komponentti piirtää myös erilaisia tietoja suurennosruutuun, joiden avulla terän ja tukilaitteiden toimintaa on helppo hahmottaa. Tällaiset komponentit toimivat hyvin, kun halutaan esitellä laitteen toi-

mintaa tarkemmin. Sorvauslaitteen liikkeet ovat yleensä hyvin pieniä, joten suurentaminen auttaa näkemään pintaa syvemmälle.

Toinen jatkokehitysidea oli tallentaa sorvauksessa tapahtuvia liikkeitä tiedostoon ja siirtää ne excel-taulukkoon, jossa sitten nappia painamalla tehtäisiin graafeja saaduista tuloksista. Eri parametreilla ajettaessa voitaisiin vertailla tukilaitteen toimintaa.

Lähteet

Koponen, H. 1995. Puulevytuotanto. Saarijärvi: Gummerus Oy Kirjapaino

Sellers,T. 1985. Plywood and Adhesive Technology. Illustrated. Marcel Dekker Inc

Bowyer, J., Shmulsky, R. & Haygreen, J. 2007. Forest Products and Wood Science. 5th Edition. Illustrated. Revised. Blackwell Publishing

Lutz, M. 2006. Programming Python. 3rd Edition. Illustrated. O'Reilly

Lutz, M. 2007. Learning Python. 3rd Edition. Revised. O'Reilly

Averill, M., Law, W. & Kelton D. 2007. Simulation Modeling and Analysis. 4th Edition. McGraw-Hill

Visual Components Oy. 2008a. Visual Components 3D Simulation Software, Quick Start Guide. Visual Components Oy [viitattu21.2.2009]. Saatavissa: http://download.visualcomponents.net/elib/2009/eCat/EquipmentLibrary/3%20Support/Tutorials/Quickstart/QuickStart.pdf.

Visual Components Oy. 2008b. Visual Components Products, 3D Video. Visual Components Oy [viitattu22.2.2009]. Saatavissa: http://www.visualcomponents.com/index.php?id=143.

Visual Components Oy. 2008c. Visual Components Products, 3D Realize. Visual Components Oy [viitattu22.2.2009]. Saatavissa: http://www.visualcomponents.com/index.php?id=142.

Visual Components Oy, 2008d. Visual Components Products, About Us. Visual Components Oy [viitattu24.2.2009]. Saatavissa: http://www.visualcomponents.com/index.php?id=147.

Dive Into Python. 2004. Dive Into Python Online Book. Revision 5.4. Mark Pilgrim [viitattu1.3.2009]. Saatavissa: http://www.diveintopython.org/download/diveintopython-pdf-5.4.zip.

Raute Oyj. 2008. Kuvia [viitattu9.3.2009]. Saatavissa: Raute Oyj:n sisäisestäverkosta.

PuuProffa. 2008. Puuviilut. Pro Puu ry [viitattu9.3.2009]. Saatavissa: http://www.puuproffa.fi/arkisto/viilut_ja_viilutus.php.

3D Create. 2009. Visual Components 3D Create User's manual. Visual Components Oy[viitattu9.3.2009]. Saatavissa: 3D Create asennushakemiston \Doc\en\3DCreate.chm tiedosto.