TAMPEREEN TEKNILLINEN YLIOPISTO

JONI RÄSÄNEN

Pythonin käyttö satunnaisen L-järjestelmän toteuttamisessa

Kandidaatintyö

Tarkastaja: Outi Sievi-Korte
Jätetty tarkastettavaksi: 20.6.2013

TIIVISTELMÄ

TAMPEREEN TEKNILLINEN YLIOPISTO
Tietotekniikan koulutusohjelma
TEKIJÄN NIMI: Joni Räsänen
Kandidaatintyö, 16 sivua, 0 liitesivua
Kesäkuu 2013
Pääaine: Ohjelmistotekniikka
Tarkastaja: Outi Sievi-Korte
Avainsanat: C++, L-järjestelmä, L-systeemi, Python

L-järjestelmä on proseduraalisen generoinnin tekniikka. Tässä työssä tarkastellaan Pythonin käyttöä L-järjestelmän toteutukseen nopeuden kannalta. Työssä toteutetaan L-järjestelmän mukaisia merkkijonoja generoiva testiohjelma Pythonilla ja verrataan sen nopeutta vastaavaan C++:lla toteutettuun testiohjelmaan. Testejä on kolme kappaletta ja ajan mittaaminen suoritettiin käyttäen time-komentoa.

Python osoittautui generoitavasta merkkijonosta riippuen 10-80 kertaa hitaammaksi. L-järjestelmän satunnaisuudella ei ollut vaikutusta Pythonin suhteelliseen nopeuteen, mutta käsitellessä suurempia L-järjestelmän generoimia merkkijonoja Python osoittautui huonoksi valinnaksi. Tästä huolimatta Python sopii käytettäväksi pelien prototyyppejä ohjelmoitaessa, vaikka nämä sisältäisivät L-järjestelmiä.

SISÄLLYS

1. Johdanto 1
2. Proseduraalinen sisällön generointi 2
2.1 Proseduraalisen sisällön generoinnin käyttö 2
2.2 Satunnainen L-järjestelmä tekniikka 3
2.2.1 Formaali määritelmä 3
2.2.2 Geometrinen tulkinta 5
3. Pythonin soveltuvuus L-järjestelmän toteuttamiseen 7
3.1 Generoitavat merkkijonot 7
3.2 Pythonin ja C++:n tehokkuus 10
3.3 Testiohjelmat 11
3.3.1 C++-versio 11
3.3.2 Python-versio 12
4. Suorituskykymittaukset 14
5. Yhteenveto 16
Lähteet 17

TERMIT JA LYHENTEET

ω	Lindenmayer-järjestelmän aksiooma eli merkkijono, josta generointi lähtee liikkeelle.
n	Lindenmayer-järjestelmän syvyys eli montako kertaa tulosääntöjä sovelletaan.
L-järjestelmä	Lindemayer-järjestelmä
D0L-järjestelmä	Deterministinen kontekstivapaa L-järjestelmä.
time-komento	Linux-käyttöjärjestelmän kernelin komento, joka antaa prosessin käyttäjämoodissa ja etuoikeutetussa moodissa käyttämän ajan.

1. JOHDANTO

Proseduraalinen generointi on kasvattamassa suosiotaan niin tutkimuksen parissa kuin videopeleissä ja elokuvissa. Proseduraalisella generoinnilla tarkoitetaan pelin tai jonkin median sisällön luomista käyttäen algoritmia. Sisällön generointia on mahdollista suorittaa sekä etukäteen (offline) että ajonaikaisesti (online) ja molemmissa tapauksissa generoinnin nopeudella voi olla suuri merkitys. Yksi tekniikoista luoda sisältöä on Lindenmayer-järjestelmä (L-järjestelmä), joka soveltuu hyvin erilaisten fraktaalisten kohteiden muodostamiseen. Tämä tekniikka voidaan toteuttaa eri ohjelmointikielillä ja eri kielet soveltuvat paremmin erilaisiin tilanteisiin, joissa L-järjestelmää tarvitaan.

Tavoitteena on saada selville, mitä vaikutuksia Pythonin valinnalla on kuvien generointiin kuluneeseen aikaan verrattuna C++:lla toteutettuun generointiin. Työ auttaa ratkaisemaan millaisissa tilanteissa kannattaa käyttää Pythonia tässä ja vastaavanlaisissa merkkijonojen generointiin liittyvissä tehtävissä. Vertailtu tekniikka on nimeltään L-järjestelmä. L-järjestelmä generoi merkkijonoja, joille voidaan antaa graafinen tulkinta, jolloin muodostuu kuva tai malli, jota voidaan sitten käyttää pelien ympäristöissä tai taustoissa.

Työssä keskitytään ainoastaan L-järjestelmän merkkijonojen generoinnin nopeuteen ja sen geometrisen tulkinnan nopeus jää työn ulkopuolelle. Testattavat ohjelmat ottavat kuvauksen L-järjestelmästä ja tuottavat merkkijonon, joka vastaa jotain kaksiulotteista kuvaa. Testiohjelma toteuttaa niin sanotun satunnaisen (stochastic) L-järjestelmän. Työssä tutkitaan myös, onko generoitavan merkkijonon pituudella vaikutusta generointiin kuluneeseen aikaan. Tutkimus eroaa Fourmentin ja Gillingsin tutkimuksesta [5] keskittymällä L-järjestelmään ja sitä kautta saatavien esimerkkien avulla antaa tarkemman kuvan millaisia seurauksia Pythonin valinnalla on nimenomaan L-järjestelmän toteutuksessa.

Tutkimus koostuu kolmesta osasta. Ensiksi kasvien generointi toteutetaan sekä C++:aa, että Pythonia käyttäen ja sen jälkeen mitataan, kuinka pitkään kuvien generointiin kuluu aikaa. Lopuksi testien tuloksia arvioidaan ja niistä tehdään johtopäätöksiä tutkimuskysymykseen liittyen. L-järjestelmä ja proseduraalinen generointi on kuvattuna luvussa 2. Luvussa 3 on kuvattu generoinnin nopeuden mittaamisessa käytetyt menetelmät. Luvussa 4 on mittauksen tulokset sekä tuloksista tehtävät johtopäätökset. Yhteenveto työn tuloksista on luvussa 5.

2. PROSEDURAALINEN SISÄLLÖN GENEROINTI

Proseduraalinen sisällön generointi ei sovellu kaikkiin tilanteisiin, mutta oikein sovellettuna sitä voidaan käyttää luomaan yksityiskohtaista sisältöä halvemmalla verrattuna siihen mikäli se tehtäisiin käsin ihmisen toimesta. Fraktaalit ovat yksi esimerkki proseduraalisesta generoinnista ja L-järjestelmä on keino niiden luomiseen. Fraktaalilla tarkoitetaan itsesimilaarista kuviota eli kuviota, joka on sama huolimatta siitä millaisella tarkkuudella sitä tarkastellaan. Fraktaalien avulla voidaan kuvata monia luonnon muotoja joita ei pystytä kuvaamaan euklidisella geometrialla. [8]

2.1 Proseduraalisen sisällön generoinnin käyttö

Proseduraalinen sisällön generoinnissa sisältö luodaan käyttäen algoritmia sen sijaan, että se luotaisiin käsin. Proseduraalista generointia käytetään lähinnä pelien sisällön tuottamiseen, graafisiin demoihin ja elokuvissa taustojen tuottamiseen. Pelien sisällöllä tarkoitetaan tässä yhteydessä yleensä musiikkia, grafiikkaa, tarinaa tai kenttiä, muttei esimerkiksi tekoälyn toimia, joka on oma tutkimuksen alansa.

Roden [25] kertoo käsintehdyn sisällön haitoista. Kehittyneempi teknologia mahdollistaa parempi laatuisen sisällön, mutta samalla se lisää taiteilijoiden sisällön tuottamiseen käyttämää aikaa. Toiseksi käsintehtyä sisältöä ei ole yhtä helppo muokata kuin proseduraalisesti generoitua. Sisällön vaatimukset voivat muuttua jälkikäteen mikä usein estää siihen mennessä tehdyn sisällön käyttämisen lopullisessa tuotteessa. Kolmantena haittapuolena on se, että tuotantoon käytettävien työkalujen tuottama sisältö on eri muodossa kuin mitä pelimoottorin käyttää. Viimeisenä haittana hän listaa mahdollisuuden, että käsintehtävän sisällön tuottamisesta voi tulevaisuudessa tulla liian kallista. [25] Videopeli Elite oli ensimmäinen, joka käytti proseduraalista generointia maailmansa luomiseen [29]. Syynä proseduraalisen generoinnin käyttöön oli tuon aikaisten pelijärjestelmien muistin vähyys, joka esti suurien maailmojen luomisen, mutta käyttämällä proseduraalista generointia saatiin luotua suurempi maailma kuin missään pelissä siihen mennessä [1].

Tekniikat ovat kehittyneet, ja tämän myötä on suositeltavaa korvata mahdollisimman paljon kertakäyttöistä sisältöä joko uudelleenkäytettävällä tai proseduraalisesti generoidulla sisällöllä, mikäli laadun ei tarvitse olla paras mahdollinen [2]. Proseduraalinen generointi ei paranna laatua, mutta sen avulla voi saada samasta määrästä

sisältöä enemmän pelattavaa [6; 7]. Nykyään jo elokuvissa on alettu käyttämään proseduraalista generointia esimerkiksi vaikuttavien taustojen ja ympäristöjen luomiseen. [4]

Proseduraalisen generoinnin tekniikoita voi myös jakaa sen perusteella kuinka monta parametria ne ottavat vai käyttääkö se ainoastaan satunnaislukuja generoinnissaan. Jako voidaan myös suorittaa lopputuloksen deterministisyyden perusteella, eli tuleeko generoinnista aina sama sisältö riippumatta satunnaissiemenestä (random seed). On myös olemassa generointitapoja, joissa lopputuloksen sopivuus testataan jälkikäteen ja jatketaan generointia kunnes sopiva on löytynyt. Proseduraalinen generointi voidaan jakaa etukäteen tehtävään ja ajonaikaiseen generointiin. Etukäteen tehdyssä generoinnissa voidaan tulosta vielä parannella käsin generoinnin jälkeen taiteilijoiden toimesta. Siihen perustuu esimerkiksi väliohjelmisto (middleware) nimeltä Speedtree. [28] Etukäteen tehtävässä generoinnissa generoinnin nopeus on tärkeää, koska generointiin kuluva aika on poissa generointia suorittavan työntekijän tehokkaasta työajasta. Ajonaikana tehtävässä generoinnissa generointi voidaan suorittaa kenttien välillä tai sitä mukaan, kun generoitava sisältö tulee pelaajan havaintoalueelle. Kenttien välillä tehtävän generoinnin tulee olla nopeaa, jotta kenttien latausajat eivät kasva sietämättömän pitkiksi. Reaaliajassa tehtävä generointi taas ei saa viedä liikaa suoritusaikaa pois tekoälyltä ja muulta pelilogiikalta.

Proseduraaliset generointimenetelmät ovat kasvattamassa suosiotaan niiden vaivuttua taka-alalle tietokoneiden resurssien kasvettua tasolle, jossa muistia ei enää tarvinnut säästää joka vaiheessa. Erityisesti viime vuosina kasvaneet tekniset mahdollisuudet ovat nostaneet vaatimuksia pelien sisällön tasolle, joka on johtamassa siihen, että ainoastaan käsintehdyn sisällön käyttäminen on tulossa liian kalliiksi.

2.2 Satunnainen L-järjestelmä tekniikka

Proseduraalisen generoinnin tekniikka nimeltään L-järjestelmä soveltuu realististen kasvien tai levien mallintamiseen [21]. L-järjestelmiä on käytetty muun muassa David Cameronin elokuva Avatarin viidakkojen generoinnissa [4]. L-järjestelmän käyttöä musiikin [22; 17], salamoiden [13; 30] ja kaupunkien [18] generointiin on tutkittu. L-järjestelmän monipuolisuus on peräisin sen kyvystä tuottaa monipuolisia fraktaaleja L-järjestelmien ollessa vielä melko yksinkertaisia.

2.2.1 Formaali määritelmä

L-järjestelmä on formaali kielioppi, jossa merkkijonoista muodostetaan uusia merkkijonoja sääntöjen avulla. L-järjestelmiä voidaan jakaa kontekstivapaisiin ja kontekstiherkkiin järjestelmiin. Ne voivat olla satunnaisia (stochastic) tai säätöarvollisia (parametric). L-järjestelmät kehitettiin matemaattisena teoriana kasvien so-

Kuva 2.1: Samoilla säännöillä generoituja kasveja käyttäen satunnaistamista.

lujen sekä niiden joukkojen kehitykselle [14, katso 20]. Sittemmin L-järjestelmälle on kehitetty useita geometrisia tulkintoja tehden niistä monipuolisia työkaluja. L-järjestelmää voidaan käyttää generoimaan erilaisia fraktaaleihin perustuvia kuvioita ja tämä tekee siitä erinomaisen työkalun kuvaamaan luonnon erilaisia fraktaaleja kuvioita. L-järjestelmien määrittämiseen tarvitaan aakkosto, aksiooma sekä tuotantosäännöt ja mikäli kyseessä on satunnainen L-järjestelmä, todennäköisyys jokaiselle tuotantosäännölle.

L-järjestelmät ovat merkkijonoja, joille annetaan geometrinen tulkinta. L-järjestelmän generointi tapahtuu ylikirjoitusjärjestelmällä, jossa osa vanhasta merkkijonosta korvataan uudella merkkijonolla käyttäen joukkoa ylikirjoittamis- tai tuotantosääntöjä. [20] Ylikirjoitusjärjestelmät teki tunnetuksi Noam Chomsky [3] käyttämällä sellaista määrittelemään luonnollisen kielen piirteitä. Määrittelystä syntyi muodollinen (formal) kielioppi. L-järjestelmän ja Chomskyn kielioppien erona on se, että L-järjestelmässä tuotantosäännöt sovelletaan samanaikaisesti, kun taas Chomskyn kieliopeissa soveltaminen tapahtuu peräkkäin. [20]

Deterministisen, kontekstivapaiden L-järjestelmien (D0L-järjestelmä) formaali määritelmä on G = <V, ω, P>, missä V on aakkosto, ω on aksiooma eli lähtömerkkijono ja P on tuotantosääntöjen joukko, jossa säännöissä merkki korvataan merkkijonolla. Deterministisellä L-järjestelmällä generoitua merkkijonoa vastaavat kuviot ovat kaikki samanlaisia. [20] [26, sivut 10–11]

Samankaltaisuuden poistaminen lisää L-järjestelmän mahdollisia käyttökohteita.

Samankaltaisuudesta päästään eroon lisäämällä L-järjestelmään satunnaisuutta. Itse L-järjestelmään kohdistuva satunnaistaminen mahdollistaa sekä topologian, että geometrian satunnaistamisen. Satunnainen OL-järjestelmä on nelikkö G = <V, ω, P, π>, jossa V on aakkosto, ω aksiooma ja P tuotantosääntöjen joukko. Funktio π on tuotantosääntöjen todennäköisyysjakauma. Satunnaistaminen tekee L järjestelmästä epädeterministisen. Käytettäessä satunnaista L-järjestelmää kasvien luomiseen, saadaan luotua kuva, joka vaikuttaa saman lajin eri yksilöltä. [20] Kuvassa 2.1 on generoitu 4 kasvia käyttäen samaa satunnaista L-järjestelmää ja antamalle tälle kohdassa 2.2.2 esitetty geometrinen tulkinta. Katsomalla oksien muotoja ja esiintymistiheyttä kuva luo vaikutelman, että kasvit ovat saman lajin edustajia, mutta eri yksilöitä.

L-järjestelmät voivat olla myös kontekstiherkkiä, jolloin tuotantosääntöjen soveltaminen riippuu vaihdettavan merkin ympärillä olevista merkeistä. Säätöarvollisessa L-järjestelmässä jokaiseen merkkiin liitetään numeroarvo, joka vaikuttaa siihen milloin tulosääntöä sovelletaan. Nämä ominaisuudet on rajattu tutkimuksen ulkopuolelle niiden monimutkaisuuden takia.

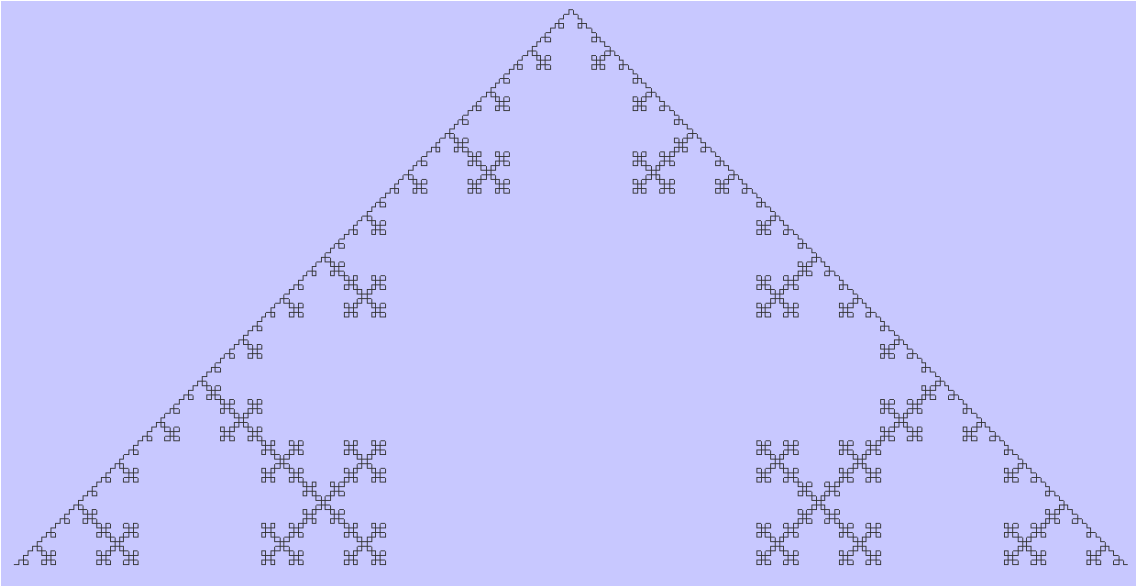
2.2.2 Geometrinen tulkinta

Geometrinen tulkinta on tapa muodostaa L-järjestelmän merkkijonoista kuvia. Jälkikäteen kehitetty ns. kilpikonnatulkinta antaa L-järjestelmälle geometrisen tulkinnan. Toiminta perustuu pisteeseen, jota kutsutaan kilpikonnaksi. Se liikkuu tai sen liikkumasuunta kääntyy merkille ennalta sovitulla tavalla. Näin muodostuu kuvia. Tässä luvussa kuvataan kaksiulotteisten kasvien generointiin soveltuva aakkostosta V.

Kilpikonnatulkinnassa kilpikonnan sijainti on kolmikko (x,y,α), jossa x ja y ovat kilpikonnan sijainti ja α on suunta, johon se on matkalla. Kun annetaan askeleen pituus d ja kulman lisäys δ, kilpikonna ymmärtää komennot 'F','+' sekä '-'. 'F' tarkoittaa liikkumista d verran eteenpäin piirtäen viivaa, '+' tarkoittaa kääntymistä δ verran vasemmalle ja '-' δ verran oikealle. Käyttämällä näitä komentoja aakkostona voidaan L-järjestelmää käyttää kuvien piirtämiseen. [23]

Yritettäessä kehittää sopivia sääntöjä D0L-järjestelmässä voidaan käyttää reunojen tai solmukohtien ylikirjoittamista tuottamaan halutunlaisia rakenteita. Reunojen ylikirjoittamisessa tuotantosäännöt kohdistuvat kuvion reunoihin, kun taas solmukohtien ylikirjoittamisessa operoidaan monikulmion kärjillä. Molemmissa yritetään mallintaa rekursiivista kuviota tasoon. Käytännössä valinta reunojen ja solmukohtien ylikirjoittamisen välillä on kiinni siitä, kumpi on kätevämpää. Niiden tunteminen kuitenkin auttaa ymmärtämään L-järjestelmän sisäistä toimintaa. [9; 10; 15]

Ylikirjoitusmekanismilla täytyy voida operoida suoraan akselipuita, jotta kilpikonnatulkinnalla voidaan kuvata muitakin kuin pelkkiä käyriä. Akselipuussa reunat



Kuva 2.2: Kochin pyramidi [8, s. 139].

ovat suunnattuja ja nimettyjä ja puu kulkee perustasta päätesolmuihin. Akselipuu on erikoistapaus juuripuusta (rooted tree). Siinä vain yksi osista on menossa suoraan ulospäin, muut ovat sivuttaissuuntaisia. Puun osaa kutsutaan akseliksi, jos sen ensimmäinen osa on juuressa tai sivuttaissuuntaisena. Jokainen sen jälkeen tuleva osa on suora ja viimeistä osaa ei seuraa muita osia. Akseli jälkeläisineen on oksa.[16, katso 20] [21]

Akselipuulla operoiminen onnistuu puu-OL-järjestelmässä. Siinä puuntuotantosääntö ylikirjoittaa reunan akselipuulla. Puu-OL-järjestelmän kolme komponenttia ovat nimetyt reunat, alkupuu ja puutuotantosäännöt. Jotta L-järjestelmällä voidaan mallintaa akselipuita, täytyy käyttää joko listaa tai hakasulkeista merkkijonoa. Sulutetussa OL-järjestelmässä ’[’ tarkoittaa, että nykyinen tila laitetaan pinoon ja ’]’ tarkoittaa, että nykyisen kilpikonnan tilaksi asetetaan pinon päällimmäisenä oleva tila. [23; 24]

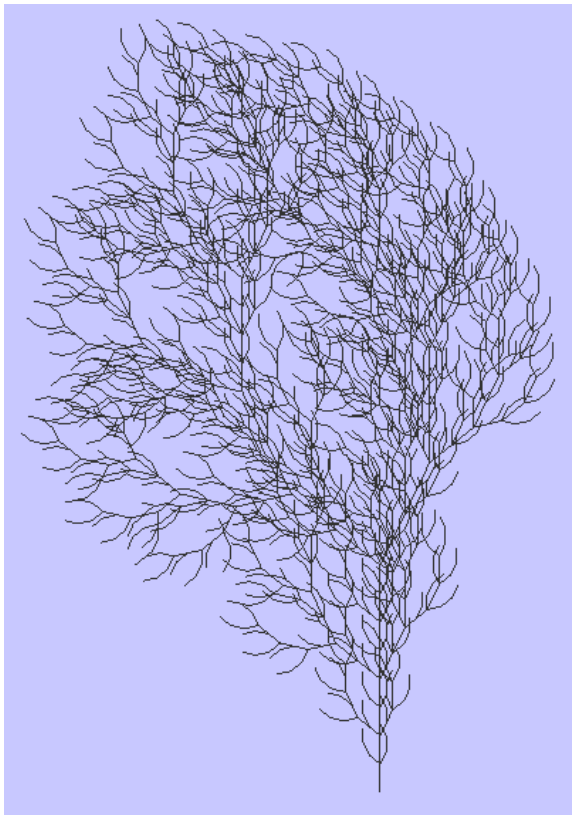
Esimerkiksi kuvassa 2.2 näkyvä Kochin pyramidi on generoitu käyttäen L-järjestelmää säännöillä alku -F, kulma 90°, syvyys n on 5,eli tuotantosääntöjä sovelletaan 5 kertaa ja tuotantosääntöjoukko P koostuu säännöstä $F \rightarrow F+F-F-F+F$, joka tarkoittaa, että merkki F korvataan merkkijonolla “F+F-F-F+F”. Kahden kerran jälkeen merkkijono on “-F+F-F-F+F+F+F-F-F+F-F+F-F-F+F-F+F-F-F+F+F+F-F-F+F”. Merkkijonoa luetaan alusta. Kilpikonnaa käännetään + ja - merkeillä sekä liikutetaan F merkillä. Näiden lisäksi on myös mahdollista tehdä haaroja ’[’ ’]’ merkeillä. Haaroittumista tarvitaan, jotta voidaan muodostaa akselipuita, jotka ovat kasveille ominainen rakenne. Ylikirjoittaminen on yksi tapa kehittää L-järjestelmistä halutunlaisia.

3. PYTHONIN SOVELTUVUUS L-JÄRJESTELMÄN TOTEUTTAMISEEN

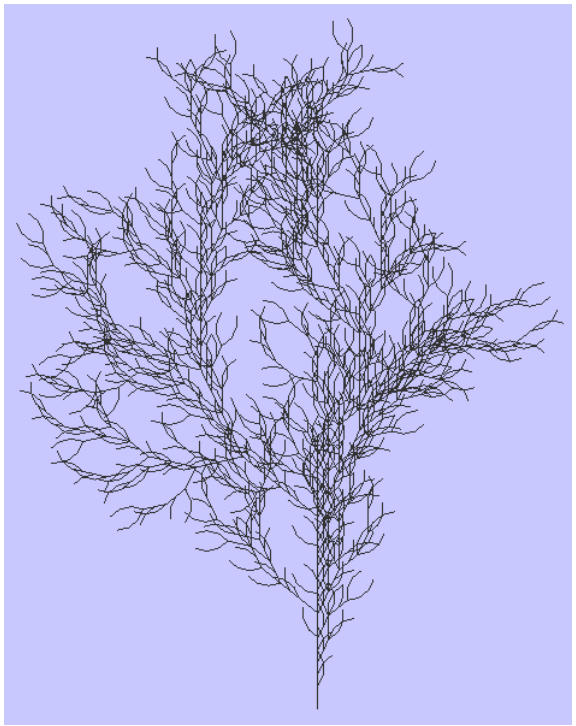
On tilanteita, joissa L-järjestelmiä halutaan luoda ajonaikaisesti ja tällöin on tärkeää, että se tapahtuu niin nopeasti kuin mahdollista, ettei kokemukseen synny katkoksia. Nopeus on myös tärkeää etukäteen generoitavissa kohteissa, jotta L-järjestelmiä voidaan muokata ulkonäön perusteella. Kaikilla ohjelmointikielillä on omat vahvuutensa ja heikkoutensa ja ainoa keino saada täysi hyöty irti käytettävistä kielistä on tutkia mikä kieli soveltuu mihinkin tilanteeseen.

3.1 Generoitavat merkkijonot

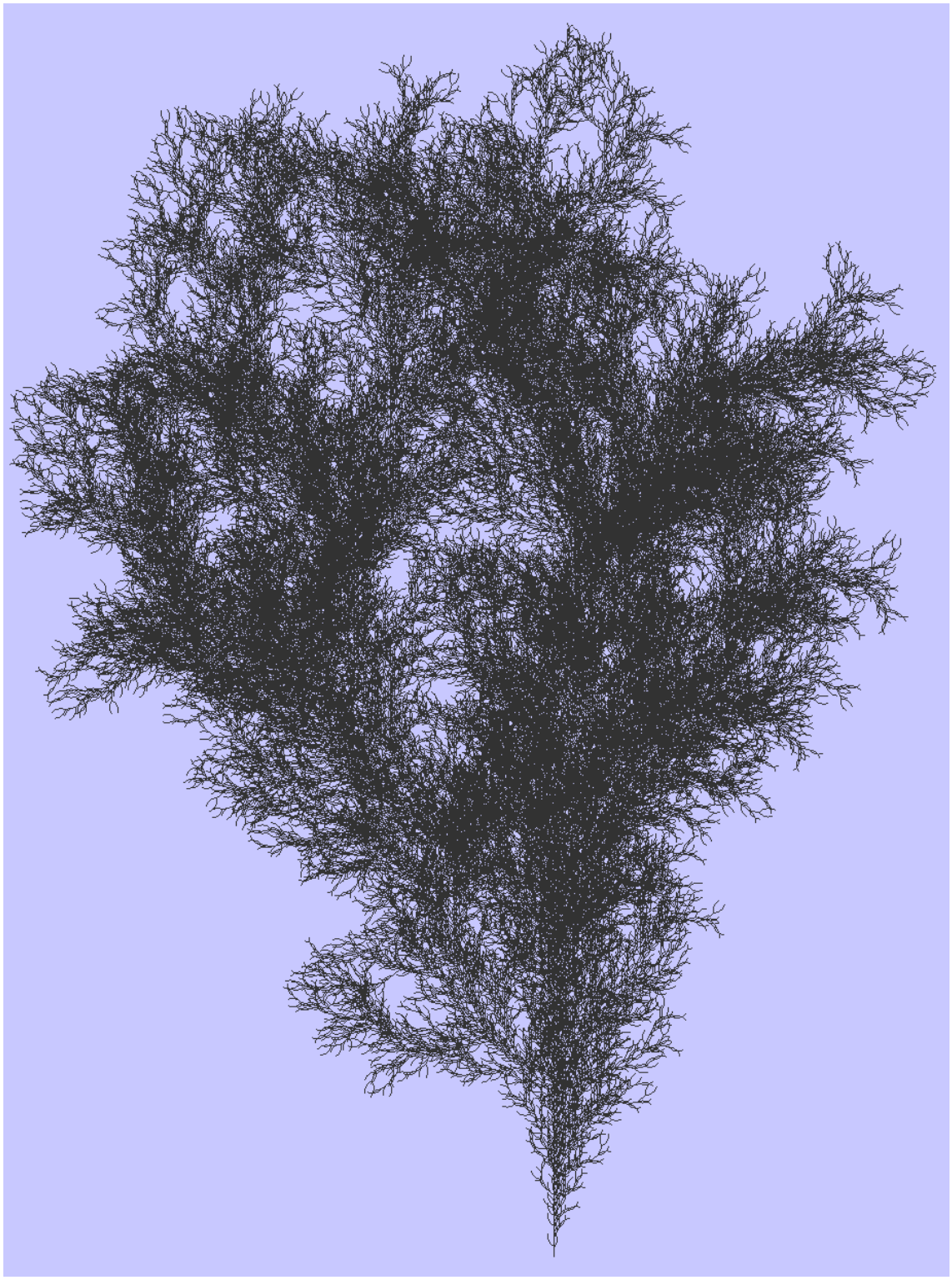
Soveltuvuutta nopeutta vaativiin tehtäviin testataan ajamalla testejä testiohjelmilla. Testeillä pyritään saamaan selville satunnaistamisen käyttäminen ja generoitavan L-järjestelmän koko vaikuttaa kielen valintaan. Ensimmäisessä testitapauksessa generoidaan merkkijono, joka vastaa kuvan 3.1 kasvia. L-järjestelmän syvyys n on 4, aksiooma ω on F ja säännöt ovat $F \rightarrow FF - [-F + F + F] + [+F - F - F]$. Tällä kuvalla mitataan miten testiohjelmat suoriutuvat ilman satunnaistamista. Toisessa testitapauksessa generoidaan merkkijono, joka vastaa kuvaa 3.2. Sen syvyys n on 4, aksiooma ω on F ja säännöt ovat $F \rightarrow FF - [-F + F + F] + [+F - F - F]$ ja $F \rightarrow FF + [-F + F + F] - [+F - F - F]$ yhtä suurella todennäköisyydellä. Tämä kuva on valittu, jotta sitä voi verrata ensimmäiseen testiin ja etenkin satunnaisuuden toteutuksen vaikutusta kielien suhteeseen.



Kuva 3.1: Pieni satunnaistamaton L-järjestelmä.



Kuva 3.2: Esimerkki pienestä satunnaisesta L-järjestelmästä.



Kuva 3.3: Esimerkki suuresta satunnaisesta L-järjestelmästä.

Kolmannessa testissä generoidaan merkkijono, josta syntyy kuva 3.3. Se on käyttää samaa L-järjestelmää kuin edellisessä testissä, mutta syvyys n on kasvatettu kuuteen. Tällä tavoin voidaan verrata, mitä vaikutusta on sillä, että sääntöjä sovel-

letaan useampaan kertaan ja tuloksena syntyy huomattavasti pidempi merkkijono.

3.2 Pythonin ja C++:n tehokkuus

Tavoitteena on selvittää, mitä vaikutuksia on Pythonin valinnalla L-järjestelmän merkkijonojen generointiin kuluneeseen aikaan. Python on valittu, koska sillä ohjelmoinnin on tuotteliasta [19; 27]. Täten Pythonia voidaan pitää hyvänä valintana proseduraalista generointia sisältävien tehtävien ratkaisemiseen. Mikäli halutaan mahdollistaa peleissä ajonaikana tapahtuva generointi ei se saa viedä liikaa aikaa muilta toiminnoilta. Vertailukielenä on C++, koska se on tehokas yleisessä käytössä oleva kieli, jota on perinteisesti käytetty monissa varsinkin suuremman luokan peleissä. Proseduraalisen generoinnin tekniikoista L-järjestelmä valittiin sen joustavuuden takia. Tässä työssä käsitellään kontekstivapaita ja satunnaisia L-järjestelmiä, koska niissä pääsi testaamaan vähän perusjärjestelmää monimutkaisempaa konseptia, eikä se ollut liian monimutkainen toteutettavaksi kohtalaisella vaivalla.

Mitattavat testiohjelmat lukevat syötteen, joka sisältää tiedon montako kertaa sääntöjä sovelletaan, mikä on aksiooma ja mitkä ovat tuotantosäännöt. Testiohjelmat tulostavat tuotetun merkkijonon. Mittauksen testiohjelmissa ei ole mukana tämän merkkijonon muuttamista kuvaksi, ainoastaan sen generointi, koska nämä muodostivat selkeästi kaksi erillistä vaihetta ja kuvan piirtämiseen on olemassa useita varteenotettavia vaihtoehtoja molemmissa kielissä.

Testit suoritetaan Linux-käyttöjärjestelmässä. Testeissä tarkastelussa on prosessiaika. Prosessiaika on prosessin luomishetkestä kulunut aika. Kerneli erottelee sen käyttäjän moodissa (user mode) aikaan user ja etuoikeutetussa moodissa (kernel mode) käytettyyn aikaan. [11, s. 20] [12, s. 206] Mittauksissa käytettiin time-komentoa, koska sen käyttö oli mittauksen suorittajalle ennestään tuttu ja koska kernel-operaationa se vaikuttaa luotettavalta. Tulosteet syötetään kohteeseen /dev/null, jottei tulosteiden esittäminen hidasta ohjelmaa ja näin aiheuta häiriötä mittaukseen. Mittaus suoritettiin AMD Athlon II prosessorilla. Tarkkuuden parantamiseksi olisi mahdollista antaa mittauksen prosessille korkein prioriteetti, jolloin ulkoiset keskeytykset eivät vaikuttaisi tulokseen. Näin ei toimittu, koska tuloksista etsitään vain suuria poikkeamia, eivätkä keskeytysten aiheuttamat viivästykset todennäköisesti vaikuta niiden esiintymiseen. Mittaukset suoritetaan useaan kertaan time-komennon epätarkkuuden vaikutuksen minimoimiseksi. Mittaukset suoritettiin 50 kertaa ja seuraavassa on esimerkkinä se osa testiskriptiä, jolla saadaan mitattua yhden testitapauksen suorittaminen Pythonilla:

```
time(
for i in `seq 1 50`;
do
python PlantGeneration.py < inputs/hexgoscur.txt > /dev/null
done
)
```

3.3 Testiohjelmat

3.3.1 C++-versio

C++-testiohjelmassa säännöt ovat tallessa standardikirjaston map-tietorakenteessa nimeltä *rules*. *rules*:ssa on oma vector-tietorakenne jokaiselle merkille, jolle on olemassa sääntöjä. Sääntöä sovellettaessa vector:in sisältä arvotaan yksi korvaava merkkijono. Ohjelmassa 3.1 on se ohjelman osa, jossa merkkijonon generointi tapahtuu. Siinä käydään sen hetkinen merkkijono läpi merkki kerrallaan ja korvataan merkki, mikäli sen korvaamiseen löytyy sääntö. Lopuksi sijoitetaan tuotettu merkkijono alkuperäisen tilalle. Tämä toistetaan syötteenä saadun syvyyden verran. Kääntäjänä toimi GCC versio 4.8.0 käyttäen -O2 vipua, joka optimoi sekä nopeutta, että binäärin-kokoa.

```
srand (time(NULL))
for(unsigned int i = 0; i < depth; ++i)
{
    std::string newseq = ""
    for(unsigned int i = 0; i < seq.size(); ++i)
    {
        if(rules->find(seq[i]) != rules->end())
        {
            std::vector<std::string>* alternatives
              = (*rules)[seq[i]]
            unsigned int r = rand()%alternatives->size()
            newseq += (*alternatives)[r]
        }
        else
        {
            newseq += seq[i]
        }
    }
    seq = newseq
}
std::cout << seq << std::endl
```

Ohjelma 3.1: L-järjestelmän generointi C++:lla

3.3.2 Python-versio

Python-testiohjelmassa säännöt talletetaan defaultdict-tietorakenteeseen nimeltä *replacementRules*, jossa on oma lista jokaiselle merkille, jolle on olemassa sääntöjä. Listassa on kaikki säännöt, joista yksi arvotaan, kun merkkiä korvataan. Ohjelmassa 3.2 korvataan merkkijonoa sääntöjen mukaisesti annettu määrä kertoja. Tulkkina toimi Python 3.3.2.

```
import random
random.seed()
for i in range(depth):
    newseq = ""
    for element in seq:
        alternatives =
            replacementRules.get(element,element)
        index = random.randint(0, len(alternatives) - 1)
        newseq = newseq + alternatives[index]
    seq = newseq
print(seq)
```

Listing 3.2: L-järjestelmän generointi Pythonilla

4. SUORITUSKYKYMITTAUKSET

Testeissä suoritetaan sama ohjelma eri syötteillä. Ensimmäisen ja toisen testin tuloksista saadaan tietoa satunnaisuuden merkityksestä kielen valintaan Pythonin tapauksessa ja toisen sekä kolmannen testin tuloksista saadaan selville miten generoitavan merkkijonon pituus vaikuttaa kielen valintaan. Järjestelmän keskeytykset vaikuttavat testeissä käytettävän time-komennon tulokseen. Epätarkkuudesta johtuen ajat on ilmoitettu vain sadasosa sekuntien tarkkuudella ja suhde yhden numeron tarkkuudella. On myös hyvä huomioida, että Fourmentin ja Gillingsin [5] mukaan Python on Linux-käyttöjärjestelmässä suhteessa hieman hitaampi kuin C++ verrattuna Windows-käyttöjärjestelmään.

Taulukosta 4.1 näkyy, että satunnaistamattoman L-järjestelmän merkkijonon generointiin 50 kertaa kului Pythonilla 11 kertaa pidempään kuin C++:lla. Tämän suhteen laskemiseen käytettiin user ja sys kohtien summaa, joka kertoo prosessin käyttämän kokonaissuoritusajan. Seuraavan testin tulokset näkyvät taulukosta 4.2. Tuloksena on, että pienen satunnaistetun L-järjestelmän generointiin kului myös 11 kertaa enemmän aikaa Pythonilla kuin C++:lla. Samasta kertoimesta voi päätellä, että satunnaisuuden käyttämisellä ei ollut merkittävää vaikutusta kuluneeseen aikaan. Satunnaistamisesta ei tulosten perusteella kannata luopua mikäli on päätynyt käyttämään Pythonia, eikä Pythonin käyttöä ole syytä hylätä sillä perusteella, että aikoo käyttää satunnaistettua L-järjestelmää.

Taulukko 4.1: Ensimmäisen testin ajamiseen kuluneet ajat.

	Python	C++
real	3.3s	0.2s
user	2.9s	0.1s
sys	0.3s	0.1s
Suhde	10x	

Taulukosta 4.3 näkyy, että kolmannessa testissä Pythonilla kului merkkijonon generointiin 83 kertaa enemmän aikaa kuin C++:lla. Jos tätä verrataan taulukon 4.2 tuloksiin huomataan, että kielten nopeudessa tapahtuu merkittävä muutos. Koska ainut ero testien välillä on koodin tasolla merkkijonon pituus, vaikuttaisi, että pitkien merkkijonojen käsittely Pythonissa on huomattavasti hitaampaa kuin C++:ssa.

Taulukko 4.2: Toisen testin ajamiseen kuluneet ajat.

	Python	C++
real	3.2s	0.2s
user	2.9s	0.1s
sys	0.3s	0.1s
Suhde	10x	

Python on tulkattava kieli, joka luultavasti on keskeisessä osassa sen huonossa suoriutumisessa pitkillä merkkijonoilla.

Taulukko 4.3: Kolmannen testin ajamiseen kuluneet ajat.

	Python	C++
real	25.0s	0.3s
user	24.6s	0.2s
sys	0.3s	0.1s
Suhde	80x	

Lähes 100 kertainen ero kielten välillä tarkoittaa, ettei Pythonia kannata käyttää varsinkaan suurien L-järjestelmien generointiin käyttäen tässä työssä esitettyä ohjelmaa. Sen sijaan Pythonia voi harkiten käyttää ajonaikanakin tapahtuvaan generointiin, kunhan kyseessä on pienehkö L-järjestelmä. Pelejä tehdessä on usein tarve tehdä prototyyppi pelistä ennen sen toteutusta ja tässä vaiheessa Python voi soveltua kieleksi, mikäli korvaa suuret L-järjestelmät pienemmillä. Tällä tavoin saadaan jonkinlainen käsitys miltä lopullinen peli tulisi näyttämään ottaen samalla irti hyödyn Pythonin hyvästä tuottavuudesta.

Tulokset ovat samansuuntaisia aiempien tutkimusten kanssa [5]. Tuloksissa ei tullut vastaan mitään erityisen yllättävää. Pythonin soveltuvuudesta erilaisiin satunnaista L-järjestelmää hyödyntäviin proseduraalisen generoinnin ongelmiin saatiin tietoa, mutta testejä olisi voinut tehdä monipuolisemmin ja testiohjelmia useampia. Testien perusteella voidaan sanoa, että satunnaistamisen vähäinen vaikutus ja suurten L-järjestelmien hidas generointi Pythonilla. Pythonin hitaus johtuu pää asiassa sen tulkattavuudesta. Tästä aiheutuva hitaus on mahdollista kiertää käyttämällä ohjelman kriittisissä osissa ulkoista komponenttia, joka on kirjoitettu tehokkuuteen tähtäävällä ohjelmointikielellä tai joka muuten mahdollistaa tehokkaalla kielellä kirjoitetun koodin liittämisen Python ohjelman osaksi.

5. YHTEENVETO

Työssä pyrittiin selvittämään Pythonin soveltuvuutta proseduraalisen generoinnin tekniikan nimeltä L-järjestelmä toteuttamiseen. Työ koostui testiohjelmien ohjelmoinnista Pythonilla ja vertailukielellä C++ sekä testien ajamisesta ja tulosten analysoinnista sekä kirjaamisesta.

Työ onnistui kohtalaisesti. L-järjestelmä osoittautui odotettua helpommaksi ohjelmoida, josta seurasi että testattavasta algoritmista tuli yksinkertaisempi kuin alussa odotin. Ohjelmointi vaihe osio meni hyvin. Yllättäen C++-versio ohjelmointi tuotti suurempia ongelmia kuin Python vaikka ei ollut aikaisempaa kokemusta Python-ohjelmoinnista.

Satunnaistamisen käyttäminen L-järjestelmässä ei huononna Pythonin suorituskykyä suhteessa muuhun koodiin. Tuloksissa kävi ilmi,että mikäli käsittelyssä on suuria L-järjestelmiä tai pitkiä merkkijonoja, kannattaa Pythonin käyttöä välttää. Täten kannattaa pyrkiä rajoittamaan Pythonin käyttö ainoastaan pieniä L-järjestelmiä vaativiin tehtäviin, kuten esimerkiksi pelien prototyypeissä.

Tämän saman tutkimuksen voisi toistaa pyrkimällä tarkempiin tuloksiin asettamalla testiprosessin prioriteetin korkeaksi ja tutkimalla, millaisia toimintoja koodi suorittaa ollessaan etuoikeutetussa-moodissa ja selvittämällä mitkä näistä johtuvat ohjelmointi kielestä. Tällä tavoin saataisiin tarkempi kuva siitä mitä vaikutuksia Pythonin käytöstä on nopeuden kannalta. Tutkimusta kannattaisi tehdä myös muihin proseduraalisen generoinnin tekniikoihin, kuten esimerkiksi soluautomaatteihin ja muihin kieliin kuten esimerkiksi skriptikieli Luaan. Jatkotutkimuksen voisi tehdä myös siitä, miksi Python on niin hidas suurilla L-järjestelmillä ja miten kehitteillä olevat tekniikat kuten juuri oikeaan tarpeeseen kääntäminen (just in time compilation) vaikuttavat tulokseen.

LÄHTEET

[1] Boyes, E. Elite cowriter talks about his new project, The Outsider; promises Elite 4 is still on its way. [WWW]. Gamespot. Julkaistu 22.11.2006 [Viitattu 07.06.2013]. Saatavilla: http://uk.gamespot.com//news/qanda-david-braben-from-elite-to-today-6162140?part=rss&tag=gs_news&subj=6162140

[2] Cook, D. "Content is bad". [WWW]. Lost Garden. Julkaistu 10.2.2007 [Viitattu 07.06.2013]. Saatavilla: http://www.lostgarden.com/2007/02/content-is-bad.html

[3] Chomsky, N. Three models for the description of language. IRE Trans. on Information Theory, 2(1956)3 sivut 113–124.

[4] Dunlop, R. Avatar. [WWW]. CGSociety. Julkaistu 14.1.2010 [Viitattu 08.06.2013]. Saatavilla: http://www.cgsociety.org/index.php/CGSFeatures/CGSFeatureSpecial/avatar

[5] Fourment, M., Gillingsin, M. A comparison of common programming languages used in bioinformatics [www]. Department of Biological Sciences, Macquarie University, Sydney, NSW 2109, Australia . 2008. [Viitattu 8.4.2013]. Saatavilla: http://www.biomedcentral.com/1471-2105/9/82/

[6] Gibbs, N. Procedural Content Generation [WWW]. Julkaistu 29.3.2004 [Viitattu 07.06.2013]. Saatavilla: http://www.skotos.net/articles/neo6.phtml

[7] Procedural Content Generation [WWW]. Introversion Software. Julkaistu 2.6.2007 [Viitattu 07.06.2013]. Saatavilla: http://www.gamecareerguide.com/features/336/procedural_content_.php

[8] Mandelbrot, B. The fractal geometry of nature. W. H. Freeman, San Francisco, 1982. 460 s.

[9] Habel, A., Kreowski, H.-J. On context-free graph languages generated by edge replacement. Graph grammars and their application to computer science; Second International Workshop, Lecture Notes in Computer Science 153, Haus Ohrbeck, Saksa 1982. Berliini 1983, Springer-Verlag. Sivut 143–158.

[10] Habel, A., Kreowski, H.-J. May we introduce to you: Hyperedge replacement. Graph grammars and their application to computer science; Third International Workshop, Lecture Notes in Computer Science 291, Warrenton, USA 1986. Bremen 1987, Springer-Verlag. Sivut 15–26.

[11] Haikala, I., Järvinen H. Käyttöjärjestelmät. Toinen painos. Jyväskylä 2004, Talentum media oy ja tekijät. 246 s.

[12] Kerrisk, M. Linux Programming Interface : A Linux and UNIX System Programming Handbook. San Francisco 2009, No Starch Press. 1556 s

[13] LaPointe, C., Stiert, D. Kurssiprojekti. Volume Lightning Rendering and Generation Using L-Systems. Hague, 2009. Rensselaer Polytechnic Institute, Advanced Computer Graphics 2009. 7 s.

[14] Lindenmayer, L. Mathematical models for cellular interaction in development, Parts I and II. Journal of Theoretical Biology, 18(1968)3 280–315.

[15] Lindenmayer, A. An introduction to parallel map generating systems. Graph grammars and their application to computer science; Third International Workshop, Lecture Notes in Computer Science 291, Warrenton, USA 1986. Utrecht 1987, Springer-Verlag. Sivut 27–40.

[16] Lindenmayer, A., Prusinkiewicz, P. Developmental models of multicellular organisms: A computer graphics perspective. Artificial Life: Proceedings of an Interdisciplinary Workshop on the Synthesis and Simulation of Living Systems, syyskuu 1987, Los Alamos, New Mexico. Addison-Wesley, Redwood City, 1989. Sivut 221–249.

[17] Manousakis, S. Pro gradu. Musical L-Systems. Hague, 2006. The Royal Conservatory. 133 s.

[18] Parish, Y., Müller, P. Procedural Modeling of Cities. SIGGRAPH '01 Proceedings of the 28th annual conference on Computer graphics and interactive techniques. New York, 2001. Sivut 301–308.

[19] Prechelt, L. An empirical comparison of C, C++, Java, Perl, Python, Rexx, and Tcl for a search/string-processing program. Technical report.Universität Karlsruhe, Karlsruhe, Germany, 2000. Saatavissa: http://page.mi.fu-berlin.de/prechelt/Biblio/jccpprtTR.pdflatex

[20] Prusinkiewicz, P., Lindenmayer, A. The algorithmic beauty of plants [Verkkodokumentti]. New York, Springer-Verlag. 1990. 240 s. [Viitattu 24.3.2013]. Saatavilla: http://algorithmicbotany.org/papers/abop/abop.pdf

[21] Prusinkiewicz, P., Lindenmayer, A. and Hanan, J. Developmental models of herbaceous plants for computer imagery purposes. SIGGRAPH '88 Proceedings of the 15th annual conference on Computer graphics and interactive techniques, Atlanta, Georgia, 1-5 elokuuta, 1988. New York 1988, ACM. Sivut 141–150.

[22] Prusinkiewicz, P. Score generation with L-systems. Proceedings of the International Computer Music Conference ’86. Regina, Saskatchewan, Kanada 1986. Sivut 455–457.

[23] Prusinkiewicz, P. Graphical applications of L-systems. Pro- ceedings of Graphics Interface ’86 – Vision Interface ’86, Regina, Saskatchewan, Kanada, 1986, CIPS. Sivut 247–253.

[24] Prusinkiewicz, P. Applications of L-systems to computer imagery. Graph grammars and their application to computer science; Third International Workshop, Lecture Notes in Computer Science 291, Warrenton, USA 1986. Regina, Saskatchewan, Kanada 1987, Springer-Verlag. Sivut 534–548.

[25] Roden, T., Parberry, I.From Artistry to Automation: A Structured Methodology for Procedural Content Creation. Entertainment Computing – ICEC 2004, Lecture Notes in Computer Science Volume 3166, Cambridge, UK 2004. Denton, Texas, USA 2004, Springer-Verlag. Sivut 151-156.

[26] Rozenberg, G., Salomaa, A. The mathematical theory of L-systems. Academic Press, New York, 1980.

[27] Seberino, C. Python: faster and easier software development. Pythonsoft, San Diego, California, 2000. Saatavissa: http://tracyreed.org/pythonpaper.pdf

[28] Togelius, J., Yannakakis, G., Stanley, K., Browne, C. Search-Based Procedural Content Generation. Applications of Evolutionary Computation, Lecture Notes in Computer Science Volume 6024. Istanbul, Turkki, 7-9 huhtikuu, 2010. Kööpenhamina 2010, Springer-Verlag. Sivut 141–150.

[29] First use of procedural generation in a video game [WWW]. Guiness World Records [Viitattu 07.06.2013]. Saatavilla: http://www.guinnessworldrecords.com/records-6000/first-use-of-procedural-generation-in-a-video-game/

[30] Lightning Bolts. [WWW]. Julkaistu 25.2.2009 [Viitattu 09.06.2013]. http://drilian.com/2009/02/25/lightning-bolts/