Joensuun Yliopisto / Tietojenkäsittelytieteen laitos / Syksy 1997

Kandidaatintutkielma

Algoritmien animointivälineistä

Markku Turunen <mturunen@cs.joensuu.fi> 5.11.1997

SISÄLLYSLUETTELO

1 JOHDANTO

Algoritmien animointi on saanut runsaasti huomioita osakseen viimeisimmän vuosikymmenen aikana. Aihetta on sekä tutkittu tieteellisesti menestyksellisesti [Brown, 1988] että saatu aikaan käyttökelpoisia sovelluksia [Stasko, 1992].

Huolimatta alaan kohdistetusta mielenkiinnosta ei tarpeellisia läpimurtosovelluksia kuitenkaan ole saatu aikaiseksi: animointiin tarkoitettujen ohjelmien käyttö sekä opetuksen että ohjelmoinnin apuna on valitettavan vähäistä ja alan tutkimus sekä hajanaista että selkiintymätöntä.

Luon tässä tutkielmassa katsauksen algoritmien havainnollistamiseksi tehtyjen välineiden nykytilanteeseen. Kuvaan joukon järjestelmiä yhteisten kriteerien pohjalta. Näin pyrin saamaan aikaiseksi yleiskuvan aihealueen konkreettisista sovelluksista ja niiden tarjoamista mahdollisuuksista. Toisena tavoitteenani on analysoida järjestelmissä esiintyviä piirteitä sekä suuntauksia ja näin koota yhteen näkemys kehittyneen animointijärjestelmän ominaisuuksista.

Toisessa luvussa lähestyn aihealuetta esittämällä algoritmien animoinnin suhteessa ohjelmien visualisoinnin kenttään. Luon myös lyhyen historiikin, tarkastelen soveltuvuusalueita ja käyttöä sekä määrittelen käytettävän terminologian. Kolmannessa luvussa esitän joukon jäsennyksiä sekä kriteerejä välineiden kategorioimiseksi ja arvioimiseksi. Esitän myös ne perusteet, joita käytän neljännessä luvussa järjestelmien evaluointiin.

Neljännen luvun katsaus ei pyri olemaan niinkään kattava vaan edustava: esittelen ainoastaan järjestelmät, jotka ovat joko teoreettisesti tai käytännöllisesti mielenkiintoisia. Luvun lopuksi esittelen myös oman järjestelmäni, J’anime*!*:n. Viidennessä luvussa kokoan yhteenvedon järjestelmien piirteistä ja niissä esiintyneistä suuntauksista. Nämä näkemykset syntetisoin malliksi kehittyneen algoritmien animointijärjestelmän ominaisuuksista. Lopuksi esitän kuudennessa luvussa johtopäätöksiä sekä yhteenvetoa aikaisemmista luvuista.

2 ALGORITMIEN ANIMOINTI

Esittelen tässä luvussa keskeisimpiä käsitteitä ja näkökohtia aihealueeseen. Ensimmäiseksi selvitän algoritmien animoinnin suhteen muuhun ohjelmien havainnollistamiseen. Esitän myös lyhyen katsauksen historiaan ja sovellusalueisiin. Viimeisessä aliluvussa selvitän tärkeimpien erikoistermien merkitykset.

2.1 Algoritmien animointi ohjelmien havainnollistamisena

Ohjelman visualisointi (program visualization) ja *visuaalinen ohjelmointi* (visual programming) käsittelevät molemmat tietojenkäsittelytieteen parissa tapahtuvaa ohjelmien havainnollistamista. Laajasti katsottuna molemmat kuuluvat tieteellisen visualisoinnin alueeseen.

Ohjelmien visualisoinnilla tarkoitan ohjelmien rakenteen ja toiminnan esittämistä visuaalisin keinoin. Visuaalisen ohjelmoinnin katson puolestaan käsittävän ne luonteeltaan vahvasti visuaaliset menetelmät ja apuvälineet, jotka on tarkoitettu ohjelmien konstruointiin.

Eroa ovat pyrkineet selventämään mm. Price, Baecker ja Small [1993], jotka esittävät, että ohjelmien visualisointi käsittää ohjelmien ymmärtämistä parantavat menetelmät. Visuaalisella ohjelmoinnilla he tarkoittavat ohjelmien konstruointiin käytettäviä visuaalisia menetelmiä.

Kohdistan tässä tarkastelussa huomion ohjelmien visualisointiin. Lienee kuitenkin syytä huomioida, etteivät alueet ole toisiaan poissulkevia: esimerkiksi integroitujen ohjelmienkehitysympäristöjen (IDE) debuggerit ja luokkaselaimet ovat yleisesti käytettyjä ohjelmien visualisoinnin menetelmiä (visuaalisen) ohjelmoinnin puolella.

Ohjelman visualisointi voi kohdistua joko sen pohjalla olevaan abstraktioon, algoritmiin tai konkreettiseen, implementoituun ohjelmaan. Jälkimmäinen voidaan laajentaa edelleen käsittämään useita ohjelmia, jolloin kysymyksessä on ohjelmiston visualisointi.

Ohjelmiston visualisointi on käsiteenä hyvin väljä: Price, Baecker ja Small [1993] näkevät sen yläkäsitteenä ohjelmien visualisoinnille. Hyrskykari [1994, 23] taas näkee

sen laajasti kuvallisten havainnollistuksien käyttämisenä koko ohjelmointiprosessin tukena.

Karakterisoivana piirteenä ohjelmiston visualisoinnissa voi pitää sitä, että siinä visualisoinnin kohteena voi olla useampia ohjelmia, päinvastoin kuin ohjelman tai algoritmin visualisoinnissa, missä kohteena on aina yksittäinen ohjelma tai sen osa. Pysyn tässä katsauksessa yksittäisten ohjelmien (ja ennen kaikkea niiden kiinnostavien osien) tarkastelutasolla.

Algoritmien visualisoinnilla tarkoitan tietokoneohjelmien pohjana olevien keskeisten periaatteiden mahdollisimman toteutusriippumatonta havainnollistamista. Ohjelman visualisoinnilla taas tarkoitan havainnollistamista, joka koskee algoritmin jotakin tiettyä ja nimenomaista toteutusta, mahdollisesti vieläpä tarkoin määrätyssä kontekstissa. Keskeisin ero näiden kahden välillä onkin juuri siinä, haluammeko tarkastella ohjelmaa vai sen pohjana olevaa ajatusmallia.

Näen ohjelman ja algoritmin havainnollistuksen eron todellisuuden jäljentämisen ja tarinan kertomisen kaltaisena erona: ohjelman havainnollistus pyrkii olemaan mahdollisimman uskollinen esittämänsä ohjelman toteutukselle. Vaikka sen ei tarvitsekaan esittää kaikkia yksityiskohtia, sen tulee kuitenkin pohjata kaikki esittämänsä asiat kuvaamansa ohjelman toteutukseen ja noudattaa tämän suorituksen kulkua.

Algoritmin havainnollistus sen sijaan voi poiketa huomattavastikin sen muodostavasta ohjelmasta: itse asiassa niillä ei tarvitse olla näennäisesti mitään tekemistä toistensa kanssa. Sekä koodi että tietorakenteet voivat olla pintarakenteeltaan täysin erilaisia algoritmissa ja tämän toteutuksessa.

Mielestäni hyvän kuvan aiheeseen ja sen kehitykseen saa seuraavien kolmen määritelmän pohjalta:

> *"Computer program visualization is the use of computer graphics to enhance the art of presenting and communicating programs and thereby to facilitate the understanding and effective use of computer programs by people".* [Baecker & Marcus, 1990]

Baecker ja Marcus määrittelevät osuvasti ohjelmien visualisoinnin toiminnaksi ohjelmien käyttäytymisen ymmärtämisen parantamiseksi. Brown [1992] kehittelee ajatusta edelleen kohti algoritmien animoinnin näkökulmaa:

> *"Algorithm animation is a form of program visualization that is concerned with dynamic and interactive graphical displays of a program's fundamental operations".*

Brownin määritelmä esittää dynaamisen visualisoinnin, animaation, kohteeksi ohjelman perustavanlaatuisat tapahtuvat. Hundhausen [1994] vie teemaa eteenpäin seuraavasti:

> *"Algorithm animation - the process of viewing the underlying logic of a computer algorithm through a series of pictures that are strategically chosen to illustrate the algorithm in execution".*

Hundhausenin määritelmä sisältää algoritmien havainnollistamisen tärkeimmät elementit, jotka eivät ole tulleet esille aikaisemmista määritelmissä: nimenomaan *algoritmin* logiikan esittämisen sekä esitettävien asioiden huolellisen valitsemisen. Mielestäni jälkimmäinen huomio on erityisen tärkeä, sillä kiinnostavien tapahtumien joukko ei välttämättä ole lainkaan sama kuin perustavanlaatuisten tapahtumien joukko.

Pyrin tässä tarkastelemaan pääasiassa algoritmien visualisointia. On kuitenkin mahdotonta erottaa algoritmin ja sen toteuttavan ohjelman havainnollistamista toisistaan, varsinkin näiden kahden ollessa yleensä keskinäisessä vuorovaikutussuhteessa. Uskon kuitenkin, että jako ohjelman ja algoritmin visualisointiin voidaan nähdä ratkaisevana ominaisuutena visualisointijärjestelmissä.

Visualisointi kohdistuu perinteisesti joko ohjelman koodiin tai tietorakenteisiin. Paitsi suoritettavissa olevaan ohjelmaan, sama pätee myös algoritmin esittämiseen: tällöin algoritmin pseudokoodiesitys vastaa ohjelman koodia.

Jako koodin ja tietorakenteiden esittämiseen pätee hyvin proseduraalisen ohjelmoinnin alueella; siirryttäessä olioparadigmaan, funktionaaliseen ohjelmointiin tai rinnakkaisohjelmiin nousevat esille olioiden, päätöspuiden ja viestien välityksen tapaiset näkökohdat.

Havainnollistaminen tapahtuu usein ainoastaan graafisesti, mikä näkyy myös useimmissa määritelmissä. Tämä on kuitenkin aivan liian rajoittava näkökulma, sillä esimerkiksi ääni on havaittu mielenkiintoiseksi välineeksi havainnollistamisessa ([Brown & Hershberger, 1991] sekä [DiGiano, 1992]). Tietojeni mukaan muiden kuin audiovisuaalisten keinojen käyttämisestä ei ole kokeiluja.

Itse koen äänen käyttämisen olennaiseksi osaksi ohjelmien havainnollistamista. Kuitenkin nykyisten järjestelmien nojautuessa pitkälti pelkkään kuvalliseen informaatioon, käytän jäsennyksessä ja analysoinnissa lähinnä kuvallisen havainnollistuksen ominaisuuksia.

Visualisointi voi olla luonteeltaan joko staattinen tai dynaaminen. Dynaamisissa visualisoinneissa löytyy aste-eroja, minimissään ne voivat olla staattisten kuvien vaihtumista väliajoin ja rikkaimmillaan sulavan animaation esittämistä.

Animaatio-termiä käytetään varsin mieluusti dynaamisten visualisointien yhteydessä. Hyrskykari [1994, 22-23] esittää, että dynaaminen visualisointi tulisi käsittää animaatioksi silloin, kun muutokset kuvataan tarpeeksi pehmeästi. En yhdy täysin hänen näkemykseensä, sillä näen mieluummin animaation Brownin [1988, 14] sekä Pricen, Baeckerin ja Smallin [1993] tavoin pikemminkin jatkuvana ominaisuutena. Jyrkkä erottelu ei mielestäni ole tarpeen tässä asiassa.

2.2 Historiaa

Ensimmäisiä askelia ohjelmien visualisoinnissa olivat staattisia ohjausvuokaavioita tuottavat järjestelmät ([Haibt, 1959] sekä [Knuth, 1963]). Dynaamista visualisointia esitettiin ensimmäistä kertaa filmille tallennettuna ([Knowlton, 1966]).

70-luku oli erilaisten ohjelmakoodin staattisiin esityksiin keskittyvien järjestelmien (pretty-printing) ja vuokaavioiden kulta-aikaa [Price *et al.*, 1993]. Kahdeksankymmentäluvulle siirryttäessä alkoivat tekniset edellytykset olla jo suotuisat monipuolisempiin esityksiin graafisten työasemien kehittyessä ja yleistyessä.

Nykyisen ohjelmien visualisoinnin liikkeellepanevana voimana pidetään lajittelualgoritmeja esittelevää filmiä *Sorting Out Sorting* [Baecker, 1981]. Tämä 30

minuutin mittainen elokuva esittelee yhdeksän eri lajittelualgoritmin toimintaa niiden tietorakenteiden käsittelyn näkökulmasta. Esitys sisältää myös algoritmien samanaikaisen suorituksen vertailun [Price *et al.*, 1993].

Ensimmäisenä nykyaikaisena algoritmien animointijärjestelmänä pidetään yleisesti Brownin yliopistossa kehitettyä BALSA-järjestelmää [Brown & Sedgewick, 1984]. Paitsi teknisesti edistyksellinen, on BALSA merkittävä myös siitä syystä, että se on ollut laajassa käytössä. Näin se poikkeaa useimmista muista järjestelemistä, jotka ovat tyypillisesti ainoastaan kehitysympäristössään käytettyjä prototyyppejä. Järjestelmästä kehitettiin edelleen BALSA-II [Brown, 1988].

Seuraava askel BALSA:n aloittamassa kehityskaaressa oli samanaikaisten algoritmien esitykseen soveltuva Zeus [Brown, 1992], josta kehitettiin myös kolmiulotteinen versio [Brown & Najork, 1993]. Viimeisimpänä etappina Brownin työssä on elektronisen luokkahuoneen metaforaan perustuva CAT-järjestelmä [Brown & Najork, 1996] sekä tämän Java-kielinen versio JCAT [Brown *et al.*, 1997].

TANGO [Stasko, 1990] ja sen seuraaja XTango [Stasko, 1992] olivat merkittäviä askeleita järjestelmien kehityksessä ja ennen kaikkea yleistymisessä. Graafisten elementtien pehmeiden siirtymien tuen ohella XTango on merkittävä ennen kaikkea siitä syystä, että se lienee vieläkin laajimmin levinnyt varsinainen algoritmien animointijärjestelmä. Stasko jatkoi kehitystyötä, jonka tuloksena syntyi POLKA [Stasko & Kraemer, 1993] ja tähän liitettävä korkeamman tason käyttöliittymä SAMBA [Stasko, 1996].

Rinnakkaislaskennan havainnollistuksen alueella merkittävin suunnannäyttäjä lienee kolmiulotteisia visualisointeja tuottava Pavane [Roman *et al.* 1992]. Pavane käyttää deklaratiivista animointien määrittelytapaa vastakohtana useimpien järjestelmien käyttämälle imperatiiviselle määrittelytavalle.

Visualisoijan työn helpottamista on lähestytty eri suunnilta. Graafiseen suorakäyttöön perustuvan ALADDIN-animaatiokehittimen [Hyrskykari, 1994] kantava idea on animoinnin helpottaminen editorin käytöllä. Samankaltaista lähestymistapaa käyttävät mm. Dance [Stasko, 1991] ja Opsis [Michail, 1996; 1997].

UWPI [Henry *et al.*, 1990] oli ensimmäinen merkittävä esimerkki automaattisesta ohjelmien visualisointijärjestelmästä. Tätä perinnettä ovat jatkaneet mm. VCC [Baeza-Yates *et al.*, 1992] sekä Eliot [Lahtinen *et al.*, 1996].

2.3 Sovellusalueista

Ohjelmien visualisointijärjestelmät on tarkoitettu yleensä joko opetuksen- tai ohjelmoinnin apuvälineiksi. Vaikka osa välineistä laajentaakin kenttää mm. dokumentoinnin [Bentley & Kernighan, 1987], tutkimuksen [Helttula, 1988] sekä simuloinnin [Brown, 1993] suuntaan, voidaan näitä kahta, erityisesti opetusta, pitää pääsovellusalueina.

Opetuksen apuvälineenä ohjelmien visualisointijärjestelmiä voidaan käyttää hyväksi sekä opetustilanteissa (luennoilla, harjoituksissa) että itsenäisen opiskelun välineenä. Konseptia voidaan laajentaa käsittämään myös esitystä, tietojen jakamista ja yhteistyönä tapahtuvaa opiskelua. Varsinkin www-pohjaisten välineiden yleistyessä on näihin alettu kiinnittää voimakkaasti huomioita (mm. [Brown & Najork, 1996] ja [McNally *et al.*, 1997]).

Ohjelmoijan apuvälineet on useimmiten tarkoitettu ohjelmakoodin havainnollistamiseen ja ohjelmien asteittaiseen suoritukseen sekä muuttujien ja tietorakenteiden automaattiseen esitykseen (mm. [Birch *et al.*, 1995] ja [Lahtinen *et al.*, 1996]). Raja monien integroitujen ohjelmointiympäristöjen tarjoamien ohjelmointityökalujen ja edellä kuvattuja piirteitä sisältävien ohjelmien visualisointijärjestelmien kesken on sumea.

En pyri tässä esityksessä ottamaan kantaa sen paremmin käytön kuin visualisoinninkaan näkökulmiin. Näitä molempia on käsitelty kirjallisuudessa jossain määrin (mm. [Stasko *et al.*, 1993], [Stasko, 1996], [Boroni *et al.*, 1996], [Meisalo *et al.*, 1997], [Mukherjea & Stasko, 1994] ja [Hundhausen, 1994]). Tyydyn tässä ainoastaan toteamaan, että sekä teoreettisen että empiirisen tutkimuksen piirissä on paljon tehtävää. Suurimpina puutteina näen järjestelmien evaluointien lähes täydellisen puuttumisen sekä havaintopsykologisten ja liikkuvan kuvan lukemisen teoriaa ja estetiikkaa koskevien näkökohtien unohtamisen.

2.4 Terminologia

Pyrin välttämään erikoisterminologian käyttöä mahdollisuuksien mukaan, mutta luonnollisesti tämä ei ole aina mahdollista. Esittelen alla luettelonomaisesti tärkeimmät esiintyvät erikoistermit ja niiden merkitykset.

Annotointi Visualisoinnin tuottamistapa, jossa animaatio määritellään erillisen editorin avulla muuttamatta lähdekoodia [Price *et al.*, 1993]. Joissain lähteissä käytetään myös *instrumentoinnin* synonyymina [Roman & Cox, 1993].

Deklaratiivinen Määrittely Visualisoinnin tuottamistapa, jossa animointi tuotetaan käyttäen korkean tason apuvälineitä tai kieltä [Price *et al.*, 1993] määrittelemään kuvaus ohjelman tilan ja tämän esityksen välille [Roman & Cox, 1993].

Instrumentointi Visualisoinnin tuottamistapa, jossa lähdekoodin lisätään kiinnostaviin kohtiin animointikomentoja [Price *et al.*, 1993].

Post-mortem tekniikka Visualisoinnin muodostamistapa, jossa tiedot visualisointia varten kerätään ohjelman suorituksen aikana, mutta itse animaatio näytetään suorituksen jälkeen [Price *et al.*, 1993].

3 JÄRJESTELMIEN LUOKITUKSESTA JA ARVIOIMISESTA

Tässä luvussa esittelen tunnetuimmat ohjelmien visualisointijärjestelmien luokitukset niiden syntyjärjestyksessä. Esitän näistä myös yhteenvedon sekä kritiikkiä. Lopuksi kokoan mallin, jonka avulla jäsennän ja arvioin järjestelmiä myöhemmissä luvuissa.

Esitetyt luokitukset käsittelevät koko ohjelmien visualisoinnin kenttää. Koska tarkastelen ensisijaisesti algoritmien animointijärjestelmiä, korostan luokituksien kuvauksissa erityisesti tätä tarkoitusta palvelevia osuuksia. Pyrin kuitenkin esittämään luokitukset pääpiirteissään täydellisinä.

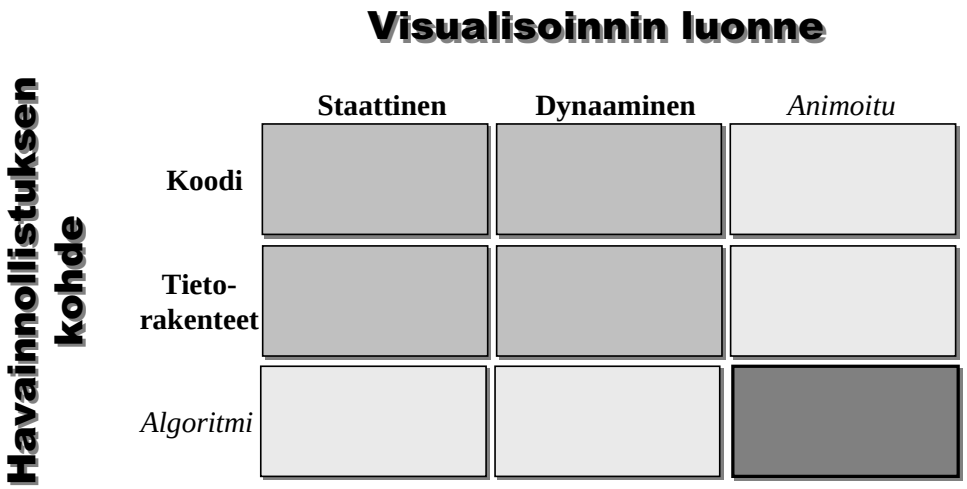
3.1 Ohjelmien visualisointijärjestelmien luokituksia

Järjestystä terminologiseen kaaokseen toi ensimmäisenä Myers [1986] erottelemalla visuaalisen ohjelmoinnin ja ohjelmien visualisoinnin toisistaan sekä esittämällä yksinkertaisen luokittelumallin. Tämän jälkeen seuraava esitys oli Brownin [1988] väitöskirjassaan esittämä jäsentely. Seuraavaksi esittivät omat näkemyksensä Stasko ja Patterson [1993], Roman ja Cox [1993] sekä Price, Baecker ja Small [1993].

3.1.1 Myersin luokitus

Myers [1986] jakoi visualisointijärjestelmät alunperin neljään luokkaan kahden ominaisuuden, havainnollistuksen kohteen ja visualisoinnin luonteen mukaisesti. Havainnollistuksen kohteena voi hänen mukaansa olla joko ohjelman koodi tai tietorakanteet. Visualisoinnin luonne voi olla puolestaan joko staattinen tai dynaaminen.

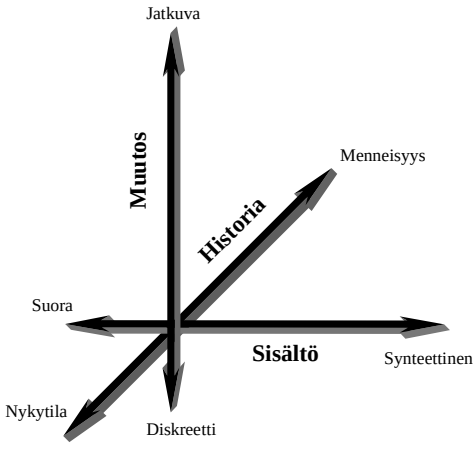
Myöhemmin Myers [1990] lisäsi havainnollistuksen kohteeksi myös algoritmin. Lisensiaatintyössään Hyrskykari [1994, 28] puolestaan lisäsi tähän malliin visualisoinnin kolmanneksi luonteeksi animaation. Myersin malli on esitetty taulukossa 1 Hyrskykarin lisäyksin varustettuna.



Taulukko 1: Myersin luokittelu (muokattu lähteestä [Hyrskykari, 1994] **).**

3.1.2 Brownin kriteerit

Brownin väitöskirjan [1988, 14-18] osana oleva esitys ei pyri kategorioimaan visualisointijärjestelmiä kokonaisuudessaan, vaan keskittyy jatkuvan kuvauksen ominaisuuksiin. Brown esittää ominaisuudet määriteltäväksi jatkuvina kolmen akselin, *sisällön*, *muutoksen* ja *historiatiedon*, suhteen. Kuva 1 havainnollistaa mallia koordinaatistossa.



Kuva 1: Brownin visualisoinnin ominaisuudet (alunperin [Brown, 1988, 14]).

Sisältö kuvaa havainnollistettavan ohjelman ja tästä muodostetun visualisoinnin suhdetta. *Suora kuvaus* ohjelman ja visualisoinnin välillä on isomorfinen, minkä

mukaisesti alkuperäinen tietosisältö pystytään tuottamaan havainnollistuksesta ja päinvastoin.

Synteettisessä kuvauksessa voidaan havainnollistaa ohjelman toimintojen ja tietosisällön abstraktioiden lisäksi sellaisia seikkoja, joita alkuperäisessä ohjelmassa ei varsinaisesti esiinny. Kuten Brown [1988, 16] huomauttaakin, usein havainnollistuksissa esiintyy useita eritasoisia sisältökuvauksia samanaikaisesti.

Muutoksen luonne voi vaihdella esityksessä diskreetistä sulavaan animaatioon. Brown [1988, 18] esittää tässä hyvän huomion erilaisten muutoskäytäntöjen tarpeellisuudesta: laajoissa visualisoinneissa jatkuva muutos voi toimia häiritsevänä tekijänä tarjoten liian matalatasoista informaatiota. Hän kiinnittää myös huomiota muutosten aiheuttamaan ajankäyttöön, mikä voi muodostua ongelmalliseksi, jos samaa visualisointia ajetaan sekä erittäin pienillä että suurilla syötteillä.

Historiatiedoilla Brown [1988, 17] ei tarkoita pelkkää listaa suoritetuista operaatioista ja objektien tiloista noina muutoshetkinä vaan näkymiin upotettua tietoisuutta aikaisemmasta etenemisestä. Pidän myös mahdollisena konseptin laajentamista toiseen suuntaan, tarjoamaan tietoa tulevasta suorituksen kulusta. Tämänkaltainen toiminto onkin toteutettu suppeassa mittakaavassa Dynalab-järjestelmässä [Birch *et al.*, 1995].

3.1.3 Staskon ja Pattersonin luokitus

Stasko ja Patterson [1993] esittävät neljään kriteeriin perustuvan luokituksen. Kriteerit järjestelmien karakterisoimiseksi ovat *näkökulma*, *abstraktiotaso*, *animaation luonne* ja *automaatiotaso*.

Näkökulma ilmaisee visualisoinnin tarkoitusta eli sitä havainnollistettavan ohjelman kohdealuetta, johon tarkastelu halutaan suunnata. Stasko ja Patterson esittävät neliportaisen mallin, jonka alimmalla tasolla on parannettu näkymä *ohjelmatekstistä*. Pelkkää ohjelmatekstiä havainnollistavia järjestelmiä askeleen kehittyneempiä ovat erilaiset *tietorakenteita* havainnollistavat järjestelmät. Tietorakenteiden lisäksi voidaan havainnollistaa suorituksen kulkua, aliohjelmien kutsugraafia, työpinoa jne. Tällaisia järjestelmiä Stasko ja Patterson nimittävät *ohjelman tilan*

visualisointijärjestelmiksi. Korkeimmalla tasolla järjestelmät havainnollistavat ohjelman pohjalla olevaa *algoritmia* tai korkean tason periaatteita.

Vasta viimeisellä tasolla voidaan puhua algoritmien visualisointijärjestelmistä. Stasko ja Patterson katsovat kolmea alempaa tasoa edustavat järjestelmät ohjelmien visualisointijärjestelmiksi. Tehtäessä eroa ohjelmien ja algoritmien visualisointijärjestelmien välille näen hyväksi ohjenuoraksi Staskon ja Pattersonin esittämän ajatuksen, jonka mukaisesti algoritmien animointijärjestelmän tulee sisältää mahdollisuus esittää korkean tason ohjelmointimetodeja.

Abstraktiotaso ilmaisee ohjelman ja animaation välisen suhteen tasoa. Se voi olla hyvinkin erilainen samaa näkökulmaa käyttävissä visualisoinneissa.

Yksinkertaisimmillaan animaatio on isomorfinen kuvaamiensa objektien kanssa (vrt. Brownin [1988] mallin sisältöön). Tällainen näkemys ei kuitenkaan ole riittävä, sillä usein algoritmien ymmärtäminen on helpompaa, kun abstraktiotasoa nostetaan sekä esitetään muitakin kuin siinä suoranaisesti esiintyviä asioita.

Stasko ja Patterson korostavat visualisoitaviin asioihin liittyvien merkityssisältöjen tärkeyttä. Tällä he tarkoittavat animoijalla olevaa tietämystä visualisoitavien asioiden tarkoituksesta. Tämä auttaa heidän mukaansa luomaan informaatiosisällöltään rikkaampia ja abstraktimpia kuvauksia. Uskon, että nimenomaan tämä asia erottaa algoritmien ja ohjelmien visualisointijärjestelmiä toisistaan.

Animaation luonne kuvaa visualisoinnin dynamiikkaa eli sitä, kuinka ohjelman tilojen kuvaukset vaihtuvat suorituksen edetessä. Mikäli muutokset ovat riittäviä tarjoamaan selkeän kuvan ohjelman toiminnasta voidaan visualisointia pitää animaationa.

Staskon ja Pattersonin mukaan algoritmien animointijärjestelmä havainnollistaa ohjelman suoritusta sekä esittämällä visualisointeja ohjelman sallituista tiloista että näiden välisistä siirtymistä. Sallitut tilat kuvaavat ohjelman suorituksen aikana esiintyviä tiloja, joilla on merkitystä ohjelman tarkoituksen ja toiminnan kannalta.

Sallittujen tilojen väliset muutoskohdat esittävät keinotekoisia tiloja, joita ei ohjelmassa esiinny, mutta joilla on merkitystä ohjelman toiminnan ymmärtämisen

kannalta. Huomasin itse asian tärkeyden seuraamalla suoria ohjelman tilan havainnollistuksia: muutosten kuvaamatta jättäminen tekee visualisoinnista huomattavan paljon vaikeaselkoisemman verrattuna animoituun versioon.

Stasko ja Patterson kuitenkin huomauttavat, Brownin [1988, 18] tapaan, etteivät tilojen väliset kuvaukset ole aina tarpeellisia, vaan joskus jopa haitallisiakin. Erityisesti tämä tulee ilmi havainnollistettaessa laajaa aineistoa, jolloin tarpeettomat siirtymät häiritsevät yleiskuvan saantia.

Automaatiotaso voi vaihdella täysin automaattisesti luoduista esityksistä aina animoijan kokonaan itse määrittelemiin visualisointeihin. Yleensä kysymyksessä on jonkin asteinen välimuoto, jolloin osa, esim. tietorakenteiden kuvaukset, muodostetaan automaattisesti, mutta animoijalla tai katsojalla on kontrolli näiden valintaan ja esitykseen.

Stasko ja Patterson näkevät abstraktiotason ja automaatiotason käänteisiksi tekijöiksi. Uskon tämän pitävänkin usein paikkansa: korkean automaatiotason omaavat järjestelmät eivät yleensä havainnollista algoritmeja, vaan tarjoavat lähinnä apua ohjelman konkreettisen havainnollistamiseen. Korkean abstraktiotason havainnollistukset sen sijaan tarvitsevat runsaasti tietoutta ohjelman tarkoituksesta ja ovat näin joko miltei mahdottomia automatisoida tai soveltuvat vain hyvin kapeille erityisaloille.

	Näkökulma	Abstraktiotaso	Animaation luonne	Automaatiotaso
Tietorakenteita havainnollistavat	**Tietorakenne**	**Matala**	**visualisointi**	**Korkea**
Ohjelman tiloja esittävät	**Ohjelma**	**Matala**	**visualisointi**	**Korkea**
Ohjelmaa animoivat	**Ohjelma**	**Keskitasoinen**	**hyvin dynaaminen visualisointi**	**Korkea**
Algoritmia visualisoivat	**Algoritmi**	**Korkea**	**visualisointi**	**Matala**
Algoritmia animoivat	**Algoritmi**	**Korkea**	**animaatio**	**Matala**

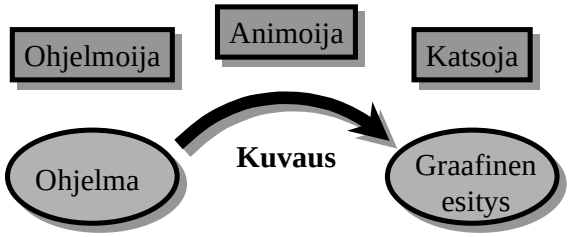
Taulukko 2: visualisointijärjestelmien ominaisuudet (muokattu lähteestä [Stasko & Patterson, 1993]).

Stasko ja Patterson jakavat järjestelmät viiteen luokkaan: tietorakenteita esittäviin, ohjelman tilaa esittäviin, ohjelmaa animoiviin, algoritmia visualisoiviin ja algoritmia animoiviin. Näiden suhteet heidän edellä esittämiinsä kriteereihin on kuvattu taulukossa 2.

3.1.4 Romanin ja Coxin taksonomia

Roman ja Cox esittivät ohjelmien visualisointijärjestelmien taksonomiansa ensin neljään kriteeriin perustuen [Roman & Cox, 1992] ja pian tämän jälkeen tarkennettuna yhden lisäkriteerin turvin [Roman & Cox, 1993]. Esitän tässä tarkennetun version.

Roman ja Cox pohjaavat jaottelunsa malliin, missä visualisointi nähdään kuvauksena ohjelmasta graafiseen esitykseen. Tämän prosessin osallistujiksi he näkevät havainnollistettavan ohjelman tekijän, kuvauksen suorittavan animaattorin, sekä kuvauksen katselijan. Kuva 2 esittää tätä perusasetelmaa.



Kuva 2: Visualisointi kuvauksena ohjelmasta esitykseen (alunperin [Roman & Cox, 1993, s. 2]).

Lähtökohtansa perusteella Roman ja Cox ehdottavat visualisointijärjestelmiä luokiteltavan viiden piirteen, *visualisoinnin alan*, *abstraktiotason*, *määrittelymenetelmän*, *liittymän* ja *esityksen* mukaisesti.

Visualisoinnin ala on samankaltainen Staskon ja Pattersonin [1993] luokituksen näkökulman kanssa. Roman ja Cox näkevät yhdeksi tärkeimmistä järjestelmiä kategorioivaksi piirteeksi sen, mihin ohjelman alueeseen havainnollistaminen kohdistuu. Kuten Stasko ja Patterson omassa taksonomiassaan, hekin löytävät neljä näkökulmaa, joihin visualisointi voi kohdistua.

Yksinkertaisimmillaan visualisoinnin kohteena on *ohjelmakoodi*. Paitsi suoranaisesti tähän keskittyvissä ohjelmissa (pretty-printers), koodin esitystä käytetään apuna

yhdessä kehittyneempien menetelmien kanssa. Useissa algoritmien animointivälineissä onkin yhtenä osana ohjelman koodin esitys.

Miltei kaikki järjestelmät esittävät informaatiota ohjelman *tietorakenteista* muodossa tai toisessa, abstraktiotason vaihdellessa ”suorasta” (viime kädessä bittivirtaa) hyvinkin kontekstisidonnaisiin esityksiin.

Suorituksen kulkua voidaan havainnollistaa prosessorin ohjelmalaskurin tasolta aina ohjelman suurimpiin kokonaisuuksiin, kuten luokkiin ja moduleihin saakka. Yleensä seuranta tapahtuu ainoastaan yhdellä tasolla, joskin tätä on ajatusta sovellettu monimuotoisemminkin [Storey *et al*., 1994].

Roman ja Cox määrittelevät ohjelman *käyttäytymisen* tapahtumien aiheuttamina muutoksina ohjelman tilassa. Laskentaa seuraamalla saatavat tapahtumat voivat esittää yksittäisiä askeleita tai suurempia kokonaisuuksia. Tapahtumat voivat kohdistua esim. muuttujien arvoihin, kutsuihin sekä ohjelman osien väliseen kommunikaatioon.

Abstraktiotasoja Roman ja Cox erottelevat kolme: *suoran-*, *rakenteisen- ja synteettisen esityksen*. He kuitenkin lisäävät, etteivät näiden rajat ole tiukat ja että monet järjestelmät esittävät usean tasoisia havainnollistuksia. Kuten Brown [1988] sekä Stasko ja Patterson [1993] myös Roman ja Cox ottavat kantaa abstraktien visualisointien tarpeellisuudesta.

Suora esitys kuvaa ohjelmaa jostakin tietystä näkökulmasta, usein mekaanisesti ilman tietoutta ohjelman tarkoituksesta. Roman ja Cox, aivan kuten Stasko ja Pattersonkin [1993] huomauttavat tämän harvoin olevan riittävää vähänkään monimutkaisempien asioiden esittämiseen.

Rakenteinen esitys kiinnittää huomion joihinkin tiettyihin ohjelman tai sen suorituksen ominaisuuksiin. Tähän päästään Romanin ja Coxin mukaan joko informaation yksityiskohtien peittämisellä tai käyttämällä korostustekniikoita. Rakenteisen esityksen he sanovat välittävän informaatiota taloudellisemmin, sillä se poistaa katsojalle epärelevantit näkökohdat.

Synteettisessä esityksessä informaatio ei ole enää suoraan ohjelmasta saatua, vaikka se tästä pystytäänkin johtamaan. Tämä näkökulman vaihdos johtuu Romanin ja Coxin mielestä ohjelmoijan valintojen ja animaattorin tarpeiden yhteentörmäyksestä.

Määrittelymenetelmä sisältää ne keinot, joilla animoija määrittelee ohjelman visualisoitavat kohdat ja sen, kuinka nämä esitetään. Roman ja Cox näkevät määrittelyn helppouden ja tehokkuuden eräiksi ohjelmien visualisointijärjestelmien tärkeimmiksi piirteiksi. Itse toivoisin, että huomiota kiinnitettäisiin määrittelymenetelmän lisäksi enemmän myös muihin näkökohtiin. Korostamalla määrittelymenetelmää on vaara ajautua insinööritieteen näkökulmaan, jossa korostetaan tuotantoprosessia lopputuloksen kustannuksella.

Roman ja Cox jakavat järjestelmät neljään kategoriaan määrittelymenetelmien kannalta: *esimäärittelyä* käyttäviin, *annotointiin* perustuviin, *deklaratiivista* määrittelyä hyödyntäviin ja *manipulatiivista* lähestymistapaa edustaviin.

Esimääritelty kuvaus tuottaa kiinteän tai parametrisoidun visualisoinnin, missä animoijalla ei ole yleensä paljoakaan valtaa valita visualisoitavia asioita tai informaation esitystapaa. Tällaisten usein sovelluskohtaisten järjestelmien eduiksi Roman ja Cox mainitsevat nopeuden ja standardin esitystavan.

Näen esimäärittelyn ongelmaksi nimenomaan esitettävien asioiden vapaan valinnan puuttumisen. Vaikka järjestelmä olisi hyvin parametrisoitu, ovat sillä tuotetut animaatiot aina sidoksissa suoraan ohjelmaan ja sen käsittelemään tietoon. Tämä rajaakin käyttökelpoisuutta erityisesti algoritmeja havainnollistettaessa, missä on usein tarve luoda synteettisiä esityksiä.

Annotoinnissa graafinen esitys muodostetaan sijoittamalla ohjelmakoodiin kutsuja strategisiin kohtiin. Menetelmän etuna verrattuna useimpiin muihin tapoihin on sen vapaus: animoijalla on täysi määräysvalta esimerkiksi abstraktiotasojen määrittelyyn. Roman ja Cox pitävät menetelmän suurimpana haittapuolena tarvetta ohjelmakoodin manipulointiin.

Näen ohjelmakoodin manipuloinnin kieltämättä ongelmalliseksi, mutta enemmänkin ohjelmien kuin algoritmien visualisointijärjestelmiä koskevaksi. Varsinaisissa algoritmien visualisointijärjestelmissä havainnollistuksien ei tulisikaan perustua

ohjelmaan perinteisessä mielessä, vaan esim. algoritmien animointikieleen. Suorituksen simulointi auttaisi häivyttämään edellä kuvatun kaltaiset ongelmat myös ohjelmien havainnollistuksessa.

Deklaratiivisessa määrittelyssä animoija esittää kuvauksen ohjelman tilan ja visualisoinnin välille. Muutokset ohjelman tilassa heijastuvat muutoksina visualisoinnissa. Suora deklaratiivinen *assosiointi* tuottaa esimääritellyn tapaisen kuvauksen, missä graafinen objekti muuttaa arvoaan aina kun siihen sidotun muuttujan arvo vaihtuu. Kehittyneempänä versiona deklaratiivinen määrittely sallii joustavamman kuvauksen ohjelman ja graafisen esityksen välille.

Manipulaatio mahdollistaa animaatioiden konstruoinnin käyttäjän graafisiin objekteihin kohdistamien käsittelykomentojen pohjalta. Animoija luo kuvauksen ohjelman tapahtumien ja visuaalisten tapahtumien välille. Näen menetelmän ongelmaksi sen toistaiseksi heikon soveltuvuuden yleisluontoiseen käyttöön, sillä useimmiten menetelmää hyödyntävät järjestelmät on tarkoitettu ainoastaan spesifien alueiden algoritmien visualisointiin (esim. binääripuiden visualisointiin tarkoitettu Opsis [Michail, 1996; 1997]).

Liittymä käsittää Romanin ja Coxin taksonomiassa järjestelmän tarjoamat konkreettiset mahdollisuudet informaation esittämiseen sekä katsojan apuvälineet esitykseen vaikuttamiseen. Edelliset muodostavat *graafisen sanaston* ja jälkimmäiset *interaktiivisuuden*.

Graafinen sanasto muodostuu objekteista, joiden avulla animoija muodostaa esityksen. Roman ja Cox luettelevat viisi tyypillistä objektiluokkaa: *yksinkertaiset objektit*, *yhdistetyt objektit*, *visuaaliset tapahtumat*, *maailmat* ja *maailmojen kombinaatiot*. Mielenkiintoiseksi heidän esityksessään näen sen, että se sopii hyvin yhteen nykyisin yleisesti käytettävän VRML 2.0-kielen [VRML 2.0, 1996] kanssa. Uskon VRML-kielestä muodostuvan vahvan pohjan tulevaisuuden järjestelmien esitysmuodolle.

Roman ja Cox tiivistävät katsojan interaktiomahdollisuudet kahteen perustyyppiin: kontrollereiden kautta tapahtuvaan kommunikointiin ja graafisen esityksen suoramanipulointiin. Jälkimmäinen voi kohdistua joko esitykseen itseensä tai sen tulkitsemaan laskentaan.

Esityksellä Roman ja Cox viittaavat visualisoinnin semantiikkaan eli tapoihin, joilla informaatiota välitetään esityksessä. He luettelevat neljä merkittävää tapaa: *grafiikan tulkinnan*, *analyyttisen esityksen*, *selittävän esityksen* ja *esitysten yhdistämisen*.

Ihmisten *tavat tulkita grafiikkaa* on syytä ottaa huomioon esityksiä suunniteltaessa, vaikkei graafisella sanastolla olekaan valmiiksi määriteltyä semanttista merkitystä. Roman ja Cox huomauttavatkin, että useimmat visualisoinnit tarvitsevat graafisen esityksen tueksi selitystekstejä.

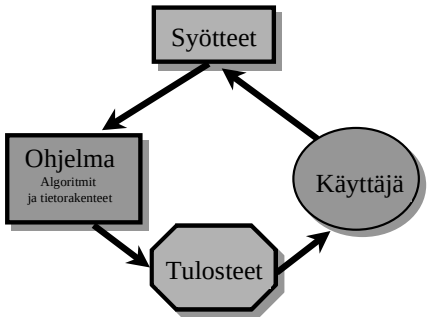
Analyyttinen esitys ohjelmasta kohdistaa huomion sellaisille alueille kuin oikeellisuuden todistamiseen. Tätä lähestymistapaa käyttävätkin mm. Pavane [Roman *et al*., 1992] ja Opsis [Michail, 1996; 1997].

Selittävä esitys käsittää visuaalisia elementtejä, jotka tuottavat animoitavasta ohjelmasta sellaista lisäinformaatiota, mitä siinä ei eksplisiittisesti ole esitetty. Tämän informaation tarkoituksena on tukea muuta esitystä esim. korostamalla esteettistä näkemystä tai kiinnittämällä katsojan huomiota.

Esitysten yhdistämisellä Roman ja Cox tarkoittavat sellaista esitysten joukkoa, jossa yksittäiset esitykset tukevat toisiaan yhteisten päämäärien saavuttamiseksi. Tämä voi tapahtua joko esittelemällä useita algoritmeja samasta aiheesta, tarjoamalla erilaisia näkymiä samaan algoritmiin tai valitsemalla syötteet mahdollisimman edustavasti.

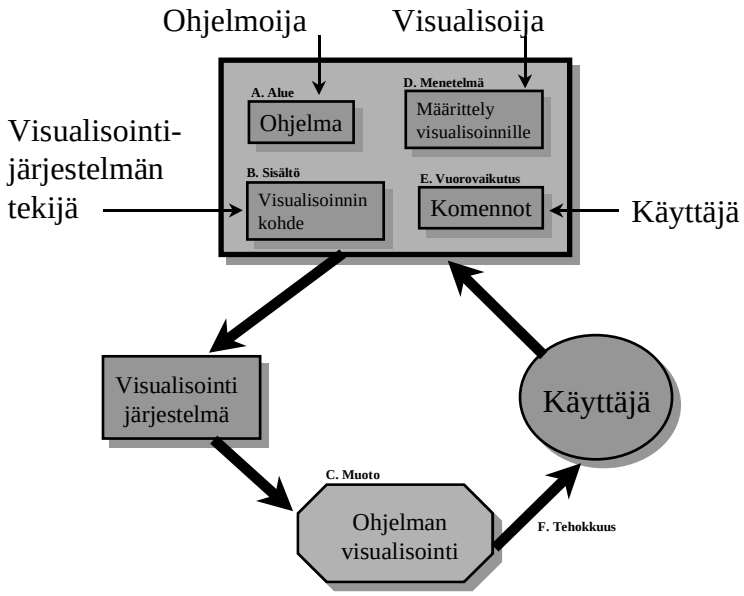
3.1.5 Pricen, Baeckerin ja Smallin taksonomia

Price, Baecker ja Small [1993] pohjaavat taksonomiansa malliin, jossa ohjelma nähdään annetuista syötteistä tulosteita tuottavana mustana laatikkona. Käyttäjällä on mahdollisuus vaikuttaa sekä syötteisiin että tulosteisiin. Kuva 3 esittää tätä mallia.



Kuva 3: Ohjelman malli (alunperin [Price et al., 1993]).

Price, Baecker ja Small määrittelevät ohjelman visualisointijärjestelmien mallinnuksen rakenteen edellä esitetyn yksinkertaisen mallin pohjalta. Heidän mukaansa visualisointiprosessiin osallistuu neljän ryhmän edustajia: *ohjelmoijan*, *visualisointijärjestelmän tekijän*, v*isualisoijan* sekä *katsojan* (vrt. Roman ja Cox [1993]). Heidän mallinsa on esitetty kuvassa 4.



Kuva 4: Yleinen visualisointijärjestelmien malli (alunperin [Price *et al.*, 1993]).

Kuvassa 4 esitetyn jäsennyksen mukaisesti Price ja kumppanit esittävät hierarkisen luokituksen, jonka ylätason kategorioina ovat *alue*, *sisältö*, *muoto*, *menetelmä*, *vuorovaikutus* ja *tehokkuus*.

Visualisoinnin alue käsittää ne ohjelmat, joita järjestelmä pystyy visualisoimaan eli käyttämään syötteinään. Alakohtia on kaksi: *yleisyys* ja *skaalautuvuus*.

Visualisoinnin yleisyys määräytyy *laitealustan*, *käyttöjärjestelmän*, käytettävän *ohjelmointikielen* sekä visualisoitavaksi soveltuvien *ohjelmien* perusteella. Lienee syytä huomata, että kysymyksessä ei useinkaan ole pelkkä soveltuvuus, vaan nimenomaan järjestelmän tarjoama tuki: esimerkiksi pelkästä mahdollisuudesta rinnakkaisten ohjelmien visualisointiin ei liene paljoakaan iloa ilman kunnollista tukea.

Järjestelmän skaalautuvuus suurille ohjelmille ja syötteille määrittää rajat, kuinka laajoja ajoja järjestelmällä pystytään suorittamaan. Mielestäni ohjelman koolla on enemmän merkitystä ohjelmien kuin algoritmien animointiin tarkoitetuissa järjestelmissä. Algoritmithan eivät ole tyypillisesti kovin suuria (korkeintaan muutamia kymmeniä pseudokoodirivejä), mutta saattavat kylläkin käsitellä valtavia syötemääriä.

Sisältö kertoo mitä osaa ohjelmasta havainnollistetaan. Perinteiseen tapaa Price, Baecker ja Small erottelevat *ohjelman* ja *algoritmin* visualisoinnin omiksi alueikseen. Kaksi muuta kategoriaa muodostuvat abstraktiotasoa määrittelevästä *uskollisuuden ja täydellisyyden* luokasta sekä *tiedon hankkimisen aikaa* määrittävästä alakohdasta.

Ohjelman havainnollistuksessa voidaan huomio kiinnittää *koodiin* ja *tietorakenteisiin*. Edelleen nämä sisältävät *ohjaus-* ja *tietovuon* määritykset. *Algoritmin havainnollistuksessa* tarkastelun kohteet ovat yhtenevät, koodin sijasta vain puhutaan *askeleista*.

Uskollisuus toteutukselle on tärkeää ohjelmien havainnollistuksessa. Algoritmien visualisoinnissa se voi muodostaa yhdessä *täydellisyyden* kanssa tarpeettoman painolastin. Rinnakkaisohjelmien tapauksessa nousee esille kysymys suorituksen häiritsemisestä, sillä visualisointi saattaa vaikuttaa aikariippuvaisen ohjelman suoritukseen. Näen suorituksen häiriintymisen lähinnä ohjelmia havainnollistavien järjestelmien ongelmaksi, sillä algoritmitasolla ongelmaa ei tulisi esiintyä.

Visualisointiin tarvittava tieto voidaan hankkia joko ajon- tai käännöksen aikana (käännös lienee syytä mieltää tässä laajassa merkityksessä). Ajonaikainen tiedonhankinta voidaan edelleen jakaa sen mukaan, kuinka ohjelman aika-avaruus suhteutuu visualisoinnin aikavirtaan sekä siihen, missä vaiheessa visualisointi generoidaan. Visualisoinnin muodostamisen yhteydessä Price, Baecker ja Small puhuvat reaaliaikaisesta tai *post-mortem*– tekniikasta sen mukaan, muodostetaanko havainnollistus ohjelman ajon aikana vai tämän jälkeen.

Muoto käsittelee järjestelmän mahdollistamia visualisoinnin ominaisuuksia. Näitä ovat *media, esitystyyli, esitystarkkuus, näkymät* sekä *synkronointi*.

Esitysmedia on tyypillisimmin graafinen työasema, nykyisin useimmiten väritoistoon kykenevä. Graafisen työaseman muuttuminen verkostoituneeksi multimediaympäristöksi nostaa edelleen tämän ja seuraavan kategorian rikkautta. Price, Baecker ja Small ottavat tässä kantaa virtuaaliympäristöjen puolesta, mikä on mielestäni erittäin tarkkanäköistä nykykehityksen valossa.

Esitystyyli koostuu Pricen, Baeckerin ja Smallin mukaan *käytetystä graafisesta sanastosta*, *animaatiosta* ja *äänistä*. Graafinen sanasto käsittää visualisointiin käytettävien peruselementtien ominaisuudet ja sisältää edelleen värin ja lisädimensioiden hyväksikäytön informaation välittäjinä. Animaatiota käytetään yleisimmin välittämään ohjelman suorituksen ajallista ulottuvuutta. Mielestäni kiinnostavaa on, mihin kaikkeen muuhun sitä voidaan käyttää. Esimerkiksi äänen käytön olen huomannut suhteellisen vähäiseksi ja suoraviivaiseksi työasemien lisääntyneistä mahdollisuuksista huolimatta.

Esitystarkkuus on yleensä kiinnitetty. Mahdollisuus suodattaa yksityiskohdat pois yleiskuvan saamiseksi olisi erittäin hyödyllinen ominaisuus; uskon, että tämän puuttuminen johtuu pitkälti visualisointien jäsentymättömyydestä. Kiinnostavan näkökulman jäsennykseen ja tämän etuihin kuten esitystarkkuuden valintaan tarjoavat Storey, Fracchia ja Carpendale [1994].

Näkymiä voidaan tarjota joko samasta ohjelmasta eri aspekteista tai vaihtoehtoisesti samanaikaisesti useasta eri ohjelmasta. Molemmissa tapauksissa näkymien tulee olla toisiinsa synkronoituja. Luonnollisesti tämä on monimutkaisempaa eri ohjelmien tapauksessa kuin saman ohjelman eri näkökulmien välillä.

Visualisointimenetelmä käsittelee esityksen muodostustapoja. Price, Baecker ja Small jakavat luokan kahtia: määrittelyn *tyyliä* kuvaavaan osaan ja ohjelman sekä visualisoinnin *kytkentää* kuvaavaan alaluokkaan.

Visualisoinnin määrittelytyyli voi vaihdella täysin käsityönä tehdystä aina täysin automaattiseen (ja pelkän animaation tapauksessa muuttumattomaan, kiinteään esitykseen). Käytännössä useimmat järjestelmät sijoittuvat välimaastoon tarjoten animoijalle apua esim. kirjastojen ja editorien muodossa.

Määrittelytyyli sisältää edelleen *räätälöitävyyden astetta* ja automaattisen visualisoinnin tapauksessa *tekoälyn tasoa* kuvaavat näkökulmat. Mikäli visualisointi on muunneltavissa (muutoinkin kuin antamalla erilaisia syötteitä), voidaan huomio kohdistaa tapaan, jolla tämä suoritetaan (suoramanipulointi, koodin modifiointi, deklaratiivinen määrittely).

Ohjelman ja sitä kuvaavan visualisoinnin *kytkentämekanismina* voi olla koodin *instrumentointi*, editorin välityksellä tai automaattisesti suoritettu *annotointi* tai *monitorointi*. Kaksi ensimmäiseksi mainittua tapaa muuntavat ohjelman lähdekoodia, erilaiset monitorointitavat taas joko lisäävät suoritusympäristössä ohjelmaan kiinnekohtia tai tarkkailevat ympäristön tilaa muuttamatta alkuperäistä ohjelmaa.

Kytkentämenetelmään liittyvät alaluokat käsittelevät riippuvuutta lähdekooditietoisuudesta sekä järjestelmän ja sillä tuotetun visualisoinnin yhteyttä. Automaattisesti havainnollistuksia tuottavissa järjestelmissä lähdekooditietoisuutta ei tarvita lainkaan, kun taas instrumentoimalla visualisoitavissa järjestelmissä animoijan tulee tuntea ohjelmakoodi melko tarkkaan.

Yleensä järjestelmät ja niillä tuotetut animaatiot ovat tiukasti toisiinsa sidottuja, mikä käytännössä tarkoittaa sitä, että visualisointi on näytettävissä ainoastaan joko järjestelmän itsensä avulla tai erikseen tätä varten tehdyllä ohjelmalla. Molemmat tavat sitovat esitykset tiettyyn laiteympäristöön tai kontekstiin. Näen tarvetta yleisesti hyväksyttyjen tiedostomuotojen käyttöönottamiseen. Yksi hyvä ehdokas on mielestäni VRML 2.0 [VRML 2.0, 1996].

Vuorovaikutusmenetelmiä ovat Pricen, Baeckerin ja Smallin mukaan *tyyliin*, *navigointiin* ja *ohjausmahdollisuuteen* liittyvät kohdat. Erilaisia vuorovaikutustyylejä ovat mm. painikkeet, valikot, komennot jne. eli tyypilliset käyttöliittymäelementit.

Navigoinnin merkitys kasvaa sitä mukaa mitä laajemmista visualisoinneista on kysymys. Alakohdat sisältävät informaation suodattamiseen ja suorituksen ajalliseen kontrollointiin sekä nopeuteen liittyviä asioita.

Ohjausmahdollisuus liittyy valmiiden esitysten kokoamiseen. Ohjausmahdollisuudella on mielestäni käyttöä sekä ohjelmien että algoritmien visualisoinnin alueilla,

ohjelmoinnin apuvälineenä mm. testauksessa ja algoritmien tapauksessa esitys- ja levitysvälineenä.

Tehokkuus muodostuu monista osatekijöistä, jotka Price, Baecker ja Small jakavat neljään luokkaan: järjestelmän *käyttötarkoitukseen, soveltuvuuteen ja selkeyteen, suoritettuihin empiirisiin tutkimuksiin* sekä *käyttökokemuksiin*.

Käyttötarkoitus voi vaihdella opetustilanteista tuotantoympäristöihin. *Soveltuvuus ja selkeys* määrittyvät sen mukaan, kuinka hyvin ja nopeasti käyttäjä ymmärtää tuotetun visualisoinnin merkityksen. *Empiiristen tutkimusten* merkitys on sekä konkreettisessa järjestelmien kehitystyössä että tieteellisessä tutkimuksessa. *Käyttökokemus* määräytyy sekä saatavuuden että saatujen kokemusten perusteella.

3.2 Luokituksista

Esitetyt luokitukset voidaan jakaa kolmeen osaan: alun yrityksiin jäsentää järjestelmiä tai niiden piirteitä (Myers [1986], Brown [1988]), pääpiirteisiin kohdistuviin malleihin (Stasko & Patterson [1993], Roman & Cox [1993]) sekä yksityiskohtaisuuteen ja kattavuuteen pyrkivään taksonomiaan (Price, Baecker & Small [1993]).

Myersin [1986] esittämä luokitus on toiminut pohjana muille malleille, ja se pureutuukin kahteen käytännössä tärkeimpään asiaan: visualisoinnin kohteeseen ja –luonteeseen. Kun luokitukseen lisätään määrittelytapaa kuvaava tekijä, saa siitä käyttökelpoisen ”quick and dirty”–työkalun yleisluontoiseen järjestelmien karakterisointiin.

Brownin [1988] esitys ei pyrikään olemaan kattava visualisointijärjestelmien luokittelu, vaan ainoastaan pyrkimys esittää kriteereitä animointijärjestelmien näyttöjen arviointiin. Brownin esityksessä on huomionarvoista ensinnäkin se, että hän kuvaa ominaisuudet jatkuvina ja toisekseen historiatietojen mukanaolo.

Staskon ja Pattersonin [1993] sekä Romanin ja Coxin [1993] taksonomiat ovat syntyneet samoihin aikoihin, ja sisältävätkin paljolti samoja elementtejä: abstraktiotason käsite esiintyy kummassakin luokituksessa, Staskon ja Pattersonin näkökulma vastaa liki täydellisesti Romanin ja Coxin havainnollistuksen alaa ja

Staskon ja Pattersonin automaatio käsittelee pitkälti samoja asioita kuin Romanin ja Coxin määrittelymenetelmä.

Suurimmat eroavaisuudet näissä kahdessa mallissa löytyvät jäljelle jäävistä luokista: Stasko ja Patterson pohtivat tarkkaan esitysten dynaamisuutta, kun taas Roman ja Cox keskittyvät kommunikointiin, esitysten tyyliin ja havainnollistuksien konstruointiin käytettyihin rakenneosiin.

Näen Staskon ja Pattersonin esityksen enemmän havainnollistuksen luonteeseen keskittyvänä, käyttäjäläheisempänä, kun taas Romanin ja Coxin visualisoinnin luomiseen fokusoivana, animoijan näkökulmaa lähempänä olevana.

Eniten kiinnostusta herätti Romanin ja Coxin luokituksessa ensinnäkin se seikka, etteivät he painota visualisoinnin dynaamisuutta, päinvastoin kuin muiden luokitusten tekijät. Toisekseen vaikkakin he korostavat synteettisten esitysten tarvetta, he eivät varsinaisesti puhu missään vaiheessa algoritmien havainnollistamisesta, vaan heillä visualisoinnin kohteena on aina ohjelma.

Pricen, Baeckerin ja Smallin [1993] taksonomia pyrkii kattavuuteen ja onnistuu siinä melko hyvin, vaikka heidän esityksestään puuttuvat esim. sellaiset käytännön kannalta erittäin tärkeät seikat, kuten saatavuus, vaatimukset, asennettavuus, ohjeistot ja mukana tulevat esimerkit. Täytyy kuitenkin muistaa, että Price, Baecker ja Small tarkoittivat luokituksensa laajennettavaksi.

Suurin ongelma Pricen ja kumppaneiden mallissa on sen yleisyys ja tästä seuraava raskaus. Luokitus yrittää kattaa koko ohjelmien visualisoinnin kentän, mikä käytännössä johtaa siihen, että samaa luokitusta yritetään soveltaa täysin erilaisiin ja eri tarkoituksiin suunniteltuihin järjestelmiin. Mielestäni esitys kadottaa yleiskuvan ja keskittyy sen kustannuksella liikaa yksityiskohtiin.

Ainoastaan Romanin ja Coxin [1993] sekä Pricen, Baeckerin ja Smallin [1993] taksonomiat pohjautuvat johonkin (eksplisiittiseen) malliin. Nämä mallit paitsi esittävät visualisointioperaation luonteen, myös tuovat esille prosessiin osallistujat. Kumpikaan luokittelu ei kuitenkaan käytä pohjalla olevaa mallia täysipainoisesti hyväkseen (esim. osallistujien näkökulmasta).

Yhteistä kaikille järjestelmille on se, että ne korostavat abstraktien esitysten tärkeyttä algoritmien visualisointiin tarkoitetuissa järjestelmissä. Tässä yhteydessä Staskon ja Pattersonin [1993] esittämä ajatus visualisoitavien asioiden merkityksen tärkeydestä on mielestäni erittäin onnistunut ja käyttökelpoinen.

Haluan korostaa peruskäsitteiden merkitysten tärkeyttä. Kaikki tässä esitellyt luokitukset puhuvat ohjelmien ja algoritmien suhteesta selkiintymättömästi. Tulevien esitysten tärkein tehtävä olisikin määritellä ohjelman ja algoritmin erilaiset käsitteet ja näiden suhteet. Uskoisin myös prosessiin osallistuvien henkilöiden näkökulmien mukaanottamisen tuovan selkeyttä esityksiin.

Eniten luokituksissa minua häiritsi, erityisesti Romanin ja Coxin [1993] tapauksessa, omien järjestelmien sisältämien piirteiden korostaminen, mikä on sinällään varsin inhimillistä ja ymmärrettävää. Mielenkiintoista olisi kuitenkin nähdä esitys sellaisen henkilön kirjoittamana, jolla ei olisi tällaista painolastia taakkanaan.

3.3 Käytetyt arviointiperusteet

Koska mikään kuvaamistani visualisointijärjestelmien luokituksista ei sellaisenaan ole sopiva tässä yhteydessä käytettäväksi, esitän tässä käytännönläheisen, tarpeisiini sovelletun mallin, jota käytän pohjana myöhemmin tapahtuvassa järjestelmien esittelyssä ja arvioinnissa.

Arviointiin käyttämäni kohdat ovat *käyttötarkoitus ja soveltuvuus*, *visualisoinnin kohde*, *visualisoinnin luonne*, *esitystekniikka*, *havainnollistuksien konstruoinnin välineet*, *määrittelymenetelmä*, *vuorovaikutus*, *mukana seuraavat havainnollistukset* sekä *saatavuus ja vaatimukset.*

3.3.1 Käyttötarkoitus ja soveltuvuus

Näen ohjelmien visualisointiin tarkoitettujen järjestelmien soveltuvan kolmelle alueelle: *opetuksen* oheisvälineiksi, *tutkimustyön* apuvälineiksi sekä *ohjelmoijan* työkaluiksi. Periaatteessa näiden kesken voidaan tehdä karkea jako, jossa algoritmeja havainnollistavat järjestelmät soveltuvat paremmin opetuksen ja tutkimuksen välineiksi, kun taas ohjelmia visualisoivat järjestelmät ovat enemmän ohjelmoijan

käyttöön tarkoitettuja. Missään nimessä tätä jakoa ei pidä ottaa jyrkkänä, ainoastaan suuntaa antavana.

Vaatimukset kaikilla kolmella alueella ovat hieman erilaiset: ohjelmoijan työkaluksi soveltuvilta järjestelmiltä vaaditaan joustavuutta soveltua erilaisiin tilanteisiin sekä uskollisuutta visualisoinnin ja ohjelman välisessä kuvauksessa. Opetuksen apuvälineiden tulee mahdollistaa abstraktit kuvaukset ja tarjota korkeatasoinen käyttöliittymä. Tutkimuksen apuvälineet voivat olla hyvinkin rajoittuneille alueille suunnattuja, mutta niiden tulisi mahdollistaa laajat ja monitasoiset tiedot ohjelmasta ja sen käsittelemistä tiedoista sekä mahdollisuus manipuloida näitä. Myös erilaiset vertailut ja mahdollisuus käyttää marginaalisia syötteitä ovat tärkeitä ominaisuuksia tutkimuksen apuvälineeksi soveltuvissa järjestelmissä.

Soveltuvuuden osalta on syytä selvittää kustakin järjestelmistä, *minkälaisten* ja *millä tavoin esitettyjen* ohjelmien havainnollistukseen ne soveltuvat. Ensimmäisellä tarkoitan niiden ohjelmien tyyppiä, joiden visualisointiin järjestelmä sopii. Havainnollistettavien ohjelmien tyyppiä voi rajoittaa esim. koko, käytetyt tietorakenteet, rinnakkaisuus jne. Esitysmuodolla tarkoitan käytännössä määrittelykieltä, joka on useimmiten jokin yleinen ohjelmointikieli.

3.3.2 Visualisoinnin kohde

Visualisoinnin kohde kertoo, mitä osaa ohjelmasta halutaan havainnollistaa. Teen tässä kahtiajaon: järjestelmä soveltuu pääasiallisesti joko *ohjelmien* tai *algoritmien* havainnollistamiseen.

Sekä ohjelmista että algoritmeista voidaan havainnollistaa *ohjelmakoodia*, *tietorakenteita* ja *suorituksen* kulkua. Kustakin voidaan arvioida, millä tavoin visualisointi on järjestelmässä toteutettu.

Liitän tähän yhteyteen myös Brownin [1988] esittämät *historiatiedot*. Visualisoinnin kohteena voi olla pelkän nykytilanteen lisäksi ajallisesti eteen- tai taaksepäin sijoittuvat tapahtumat ja tilat.

3.3.3 Visualisoinnin luonne

Visualisoinnin luonteen määrää pitkälle *abstraktiotaso*. Havainnollistus voi olla suora, ohjelman toimintaa heijastava tai jonkinasteinen yleistys toiminnasta ja tietorakenteista. Erityisen kiintoisaa tässä yhteydessä on, millainen *rakenne* visualisoinnille muodostuu.

Visualisoinnin luonteeseen vaikuttaa myös käytetty *tyyli*, joka voi olla *selittävä*, *esittävä*, *analyyttinen*, *interaktiivinen*, *yhteistyöhön tarkoitettu* jne. Käytännössä useimmat järjestelmät ovat joko ohjelman tapahtumia tai tietorakenteita esittäviä, vaikka esimerkkejä selittävistä, analyyttisistä, interaktiivisista ja yhteistyöhön tarkoitetuista järjestelmistä onkin.

3.3.4 Esitystekniikka

Esitystekniikasta kiinnostavin yksityiskohta lienee *dynaamisuuden luonne*, eli ovatko muutokset diskreettejä vai jatkuvia. Täytyy huomata, että saman järjestelmän sisällä voi olla mahdollista muodostaa hyvinkin eriluonteisia visualisointeja.

Koska rikkaassa esityksessä on lähes mahdotonta saada kaikkea informaatiota samaan esitysikkunaan, useimmat järjestelmät mahdollistavat usean erilaisen *näkymän* käytön. Nämä näkyvät voivat olla kaikki samasta ohjelmasta tai useista eri ohjelmista. Varsinkin jälkimmäisessä tapauksessa *synkronointi* on tärkeässä asemassa.

Esitysmuoto voi olla tarkemmin määrittelemätön, järjestelmän sisäinen (kuten useimmiten onkin) tai joku yleisesti tunnettu tiedostoformaatti. Jälkimmäisessä tapauksessa visualisointi voidaan esittää millä tahansa tiedostoformaattia tukevalla esitysohjelmalla, ensimmäisessä tapauksessa ainoastaan järjestelmän omilla ohjelmilla, jolloin vuorovaikutustekniikka on ratkaisevassa asemassa.

3.3.5 Havainnollistuksien konstruoinnin välineet

Esitysprimitiivit muodostavat perustan havainnollistuksien konstruointiin. Grafiikkaobjektit, äänet, efektit, puhe jne. sekä näiden muodostamat kirjastot ja rutiinit (makrot) luovat yhdessä visualisoinnin kuvailukielen. *Komentokieli* mahdollistaa esityksien tallentamisen, muokkaamisen ja levittämisen.

3.3.6 Määrittelymenetelmä

Havainnollistuksien määritykseen käytetty metodi voi olla koodin *instrumentointi*, jolloin ohjelmaa muokataan lähdekoodin tasolla, *annotointi*, missä kiinnostavat kohdat merkitään lähdekoodia suoraan muuttamatta tai *monitorointi*, missä tarkkaillaan ohjelman tilaa ja tietorakenteita.

Määrittelymenetelmän luonne voi olla *suora* tai *deklaratiivinen*, jolloin määritellään joukko tiloja ja näihin soveltuvia kuvauksia. Käytännössä useat eri määrittelymenetelmät voivat olla käytettävissä samanaikaisesti. Myös deklaratiivista ja suoraa menetelmää voidaan yhdistää.

3.3.7 Vuorovaikutus

Vuorovaikutustekniikka voi olla menuihin, komentoihin, painikkeisiin jne. pohjautuva. Jotkut järjestelmät tarjoavat myös objekteihin kohdistuvan suoramanipuloinnin. Vaikka kaikki nykyiset järjestelmät pohjautuvatkin näppäimistön ja hiiren käyttöön, on mahdollista että uudet järjestelmät tulevat tukemaan puhe-, katse-, kosketus- sekä muita vaihtoehtoisia kommunikointimenetelmiä.

Vuorovaikutusmahdollisuudet jakautuvat kahteen tapaukseen, animoijan sekä katselijan vuorovaikutusmahdollisuuksiin. Osassa järjestelmiä animoijalle ei tarjota mitään välineitä, vaan hänen oletetaan käyttävän ulkoisia ohjelmia (esim. tekstieditoria).

Animoijan näkökulmasta tärkein väline lienee määrittelyeditori, joka mahdollistaa *suoramanipulaation* hyväksikäytön. Tällainen editori voi toimia hyvin monella tasolla, määrittelymenetelmästä riippuen.

Tärkeän animoijan käyttöön tarkoitetun välineiden luokan muodostavat valmiiden esitysten luontiin tarkoitetut keinot. Nauhoitustoiminto voi primitiivisimmillään olla pelkkä ajon suora taltiointi, mutta voi myös tarjota mahdollisuuden kommentointiin jne.

Katsojan vuorovaikutusmahdollisuuksista tärkein lienee *mahdollisuus vaikuttaa esityksen kulkuun*. Katsojalle voidaan tarjota mahdollisuus esityksen pysäyttämiseen,

kelaamiseen tai siirtymiseen haluttuun kohtaan sekä nopeuden ja suunnan kontrollointiin.

Varsinkin suurilla syötteillä *navigointi* muodostaa tärkeän osan käyttöliittymää. Mahdollisuus näkymän *zoomaukseen* on usein tarjolla vektorigrafiikkaan pohjautuvissa esitysmuodoissa. Myös erilaisia *erikoisnäyttöjä*, kuten kalansilmänäkymää, voidaan käyttää hyväksi. Esitettävien osien *valinta* on tärkeä etenkin silloin, kun esitys koostuu lukuisista pienistä yksiköistä. Yhteistyöhön tarkoitetuissa välineissä *tietoisuus muiden toimista* on hyödyllistä.

3.3.8 Mukana seuraavat havainnollistukset ja ohjeet

Käytäntö on osoittanut, että järjestelmän ominaisuuksiakin tärkeämpää saattaa olla mukana seuraavien valmiiden visualisointien määrä ja taso. Myös riippumattomien, vapaasti saatavilla olevien havainnollistuksien saatavuus nostaa järjestelmän käyttökelpoisuutta.

Toinen huomionarvoinen seikka on mukana seuraava ohjeistus. Pahimmillaan järjestelmät tarjoavat ainoastaan lyhyen ’README’–tiedoston. Kunnollisen ohjeistuksen puuttuminen onkin silmiinpistävää näissä usein opetuksen ja opiskelun välineiksi tarkoitetuissa järjestelmissä.

3.3.9 Saatavuus ja vaatimukset

Osa esitellyistä järjestelmistä on vapaasti saatavilla, osa on tutkimusprototyyppejä, joita ei ole annettu julkisesti levitettäväksi. Tärkeää on myös *käyttöjärjestelmä- ja laitteistovaatimukset.* Osa järjestelmistä on tehty sellaisille laitteistoille, jotka eivät ole laajalti levinneitä tai ovat jäämässä syrjään.

Järjestelmillä saattaa olla lisäksi muita erityisvaatimuksia, kuten vaatimukset tietyistä apuohjelmista tai grafiikkakirjastoista jne. Myös asennus on tärkeä huomioitava seikka: seuraako mukana asennusohjelmaa, voiko järjestelmän asentaa tavallinen käyttäjä, onko järjestelmä käännettävä ennen asennusta jne.

4 ALGORITMIEN ANIMOINTIJÄRJESTELMÄT

Tämän luvun tarkoituksena on luoda katsaus merkittävimpiin järjestelmiin. Annan selkeän pääpainon nimenomaan algoritmien animointiin tarkoitetuille järjestelmille. Olen ottanut mukaan myös ohjelmien havainnollistukseen tarkoitettuja välineitä, koska ne sisältävät sellaisia ominaisuuksia, joita voidaan hyödyntää osana algoritmien animointijärjestelmiä.

Käytän tässä kohdassa 3.3 esitettyä kriteeristöä, en kuitenkaan kategorioimaan pyrkivässä mielessä vaan lähinnä karakterisoimaan suurien linjauksien osalta. En myöskään noudata esitystä pikkutarkasti kohta kohdalta, vaan esitän kustakin järjestelmästä lähinnä sen erikoispiirteet sekä merkittävimmät ominaisuudet

Niiden järjestelmien kohdalla, jotka ovat vapaasti saatavilla, esitän luonnehdinnan asennuksesta, vaatimuksista, ohjeistuksesta jne. Pyrin tällä arvioimaan sitä, millaisin ponnistuksin järjestelmä on otettavissa käyttöön. Samoin pyrin arvioimaan mahdollisia mukana tulevia esimerkkejä.

4.1 BALSA, ZEUS, CAT ja JCAT

Brownin yhteyteen liittyy joukko visualisointijärjestelmiä, joiden kaari alkaa Apollo–työasemille kehitetystä BALSA-I:stä [Brown & Sedgewick, 1984] ja jatkuu Macintosh-pohjaisessa BALSA-II:ssa, jonka Brown esittelee palkitussa väitöskirjassaan [1988].

BALSA onkin ollut ajankohdan huomioon ottaen erittäin edistyksellinen. Se sisälsi mahdollisuuden useisiin näkymiin, scriptien käyttöön, näyttöjen zoomaukseen ja algoritmien vertailuun. Kaikkein mielenkiintoisinta kuitenkin nykypäivän näkökulmasta on Brownin esimerkeissä historiatietojen läsnäolo. Tätä ominaisuutta ei tapaa vieläkään useimmista järjestelmistä.

BALSA pohjautuu instrumentointiin, Brownin termein kiinnostavien tapahtumien merkitsemiseen. Instrumentointi onkin toiminut pohjana paitsi Brownin omille järjestelmille myös suurimmalle osalle muita myöhemmin kehitettyjä järjestelmiä.

BALSA:n ongelmana on kuitenkin animaatioiden vaatima suuri työmäärä. Monet ominaisuudet, joiden tulisi kuulua suoraan järjestelmään, kuten pehmeät siirtymät ja animaation suorittaminen takaperin, jäävät animoijan harteille.

BALSA:n seuraaja Zeus [Brown, 1992] jatkoi kehitystä tarjoten käyttäjälle mahdollisuuden ajonaikaiseen suoramanipulointiin. Käyttäjällä on mahdollisuus tutkia ja muuttaa objektien arvoja, suorittaa animointikutsuja jne. Animoijalle Zeus tarjoaa graafisen ympäristön esitysten luontiin ja mahdollisuuden käyttää rinnakkaisuutta hyväksi. Zeuksen yhteydessä on tehty myös kokeita äänen ja värien suhteen [Brown & Hershberger, 1991] ja ympäristöä on laajennettu edelleen kolmiulotteiseksi [Brown & Najork, 1993].

Uusin askel Brownin järjestelmissä on CAT [Brown & Najork, 1996]. Algoritmien animaatiot eivät ole enää yksittäisiä visualisointeja, vaan ne liittyvät opetusympäristöön (kirjalliseen materiaaliin) ja tapahtuvat elektronisessa luokkahuoneessa. Järjestelmä on yleisessä tietoverkossa ajettavissa ja siihen voi osallistua samanaikaisesti useita henkilöitä, yksi opettajan ja muut oppilaan roolissa. CAT onkin ensimmäinen vakavasti yhteistyötä hyödyntämään pyrkivä järjestelmä. Obliq kielisestä CAT-järjestelmästä on tehty myös Java-kielinen versio JCAT [Brown *et al*., 1997].

Brownin järjestelmät ovat hyödyntäneet aina uutta teknologiaa, käyttäjän näkökulmasta ehkä liikaakin. BALSA oli viimeinen järjestelmä, joka oli rakennettu laajalti levinneelle alustalle. Brownin edistyksellisten järjestelmien kohtaloksi onkin muodostunut jääminen lähinnä tutkimusprototyypeiksi.

Paitsi käyttöympäristöjen osalta, on Brown käyttänyt myös ohjelmointikielten valinnoissa muita kuin laajalle levinneitä ratkaisuja. C- ja Pascal kielisiä ohjelmia visualisoimaan kykenevän BALSA:n jälkeen kielivalintoja ovat olleet Modula-3 ja Obliq, molemmat marginaalisia kieliä.

Kaiken kaikkiaan näen Brownin järjestelmät ehdottomasti kehittyneimmiksi ja innovatiivisimmiksi; pidän kuitenkin suurena takaiskuna niiden jäämistä pelkäksi suunnannäyttäjiksi ilman konkreettisia hyötynäkökohtia. Tulevaisuus näyttää kuitenkin valoisammalta Java-pohjaisen JCAT-järjestelmän ansiosta.

4.2 Tango, XTango, Dance, POLKA ja SAMBA

Staskon kehittämä algoritmien animointivälineiden sarja alkoi Tangosta [Stasko, 1990]. Koska kuitenkin tämän seuraaja, hieman karsittu versio XTango [Stasko, 1992] on

levinneempi ja suositumpi, tarkastelen tässä lähinnä sen ominaisuuksia. Käytän tässä kuitenkin yksinkertaisuuden vuoksi Tango–nimeä kuvaamaan järjestelmiä.

Tangossa animoija määrittelee BALSA:sta tuttuun tapaan visualisoinnin käyttäen koodin instrumentointia. Tässä suhteessa Tango ei tarjoakaan mitään uutta. Tangon voima piilee animoijan mahdollisuudessa käyttää pehmeitä siirtymiä, eli se sisältää yksinkertaisen animointikielen funktiokutsujen muodossa.

Toinen merkittävä piirre Tangossa on sen levinneisyys ja mukana seuraava esimerkkialgoritmien määrä. Nykyisessä distribuutiossa on mukana kymmeniä yleisimpiä algoritmeja, joita Stasko on saanut mm. opetuskäytön sivutuotteena. Koska Tango on vapaasti saatavilla eikä vaadi unix-käyttöjärjestelmältä mitään erityisvaatimuksia, lienee se eri versioissaan levinnein järjestelmä.

Dance–liittymän [Stasko, 1991] tarkoituksena on avustaa animoijan työtä tarjoamalla mahdollisuus määritellä visualisointeja interaktiivisesti. Tango, kuten sen seuraajatkin, on valitettavasti rajoittunut c-kielisten algoritmien animointiin. Animointirajapinta onkin Tangossa turhan matalatasoinen ja vaatii nykyajan mittapuulla liikaa järjestelmän toiminnan tuntemusta. Tango jääkin lähinnä opetuksen ja esityksen välineeksi, opiskelijakäyttöön se vaatii turhan paljon osaamista ja aikaa.

Tangon seuraaja Polka [Stasko & Kraemer, 1993] lisää tuen useille näkymille, mahdollistaa rinnakkaisalgoritmien käytön ja sisältää hieman rikkaamman suorituksen kontrolloinnin. Perusrakenne Polkassa on kuitenkin edelleen sama kuin Tangossakin.

Polkan SAMBA–liittymä [Stasko, 1996] tuo animaatioiden konstruoinnin huomattavasti käyttäjäläheisemmäksi. C-kielisten funktiokutsujen sijaan animaatio muodostetaan lukemalla animointikomentoja sisältävää ascii-tiedostoa. Animaatio on mahdollista muodostaa paitsi suoraan animointikomentoja tekstieditorilla kirjoittamalla, myös lisäämällä *millä tahansa kielellä* kirjoitettuun ohjelmaan tulostuskomentoja. Menetelmä sinällään ei ole uusi, sitä käytti jo viime vuosikymmenellä Anim [Bentley & Kernighan, 1987].

Käyttöliittymä on vielä Polkassakin melko alkeellinen. Pääosin se perustuu komentopainikkeisiin, joitakin syöteitä pystytään kuitenkin antamaan interaktiivisesti

hiirellä osoittaen, tosin hyvin kömpelösti. Käyttömukavuutta lisäävä piirre on mahdollisuus näytön zoomaukseen.

Huolimatta pitkästä kehityskaaresta ja suhteellisen laajasta levinnäisyydestä Staskon järjestelmät ovat jääneet hieman raakileiksi. Eniten hämmästyttää pysyttäytyminen pelkässä unix-ympäristössä opiskelijoiden siirtyessä jatkuvasti muiden ympäristöjen pariin. Siirtotyö sekä Windows että Java–ympäristöihin on kuitenkin Staskon [1996, 4] mukaan meneillään.

Toinen ongelma Staskon järjestelmissä liittyy puutteelliseen ohjeistukseen ja (sinällään runsaslukuisiin) esimerkkeihin. Ohjeistus on erittäin niukkaa, lähinnä teknistä ja luettelomaista. Esimerkit ovat lähinnä kokoelma samaan hakemistoon liitettyjä animaatioita – mitään yhteistä kontekstia tai selityksiä niihin ei ole liitetty, joten ne jäävätkin irrallisiksi esimerkeiksi järjestelmän mahdollisuuksista.

Staskon järjestelmät on käännettävä ennen käyttöönottoa. Kokeilemissani tapauksissa tämä sujui toisessa (Sun Solaris) suoraan, toisessa (Intel Linux) vasta kääntämisen ohjaustietoja muokkaamalla. Tämänkaltaiset operaatiot aiheuttavat sen, että jo käyttöönoton kynnys järjestelmissä on liian korkea.

4.3 PAVANE

Pavane [Roman *et al.*, 1992] lienee vakavin yritys rinnakkaisohjelmien visualisoinnissa. Merkittäviä piirteitä järjestelmässä on sen kolmiulotteisen grafiikan käyttö ja deklaratiivinen havainnollistuksien määrittelytapa.

Deklaratiivinen määrittelytapa on valittu järjestelmään siksi, että ohjelmaan upotetut animointikutsut saattaisivat muuttaa ohjelman suorituksen kulkua ja antaa virheellisen kuvan suorituksesta. Deklaratiivisessa määrittelyssä ohjelmakoodiin ei kajota, vaan visualisoinnit tuotetaan määrittelemällä ohjelman laskennan ja graafisten objektien välille vastaava kuvaus.

Erittäin mielenkiintoinen on Pavanen kehittelijöiden ajatus, jonka mukaan ohjelman oikeaksi todistamisen ominaisuudet ovat samalla niitä asioita, joita tulisi visualisoida. Mielestäni tämä sinänsä mielenkiintoinen väite on vaarallinen sokeasti noudatettuna –

algoritmin ymmärtämisessähän ei ole kyse sen yksityiskohtien ymmärtämisestä vaan nimenomaan kokonaisuuden hahmottamisesta.

4.4 ALADDIN

Aladdin [Hyrskykari, 1994] on Tampereen yliopistossa kehitetty tutkimusprototyyppi, jonka tarkoituksena on vähentää havainnollistuksien konstruointiin kuluvaa työmäärää. Tähän päämäärään pyritään käyttämällä animaatioeditoria havainnollistuksien määrittelyyn. Animointikomentoja ei instrumentoida suoraan algoritmiin, vaan ne muodostavat oman määrittelytiedostonsa, joka yhdistetään ohjelmakoodiin suoritusympäristössä. Menetelmästä Hyrskykari käyttää nimitystä annotointi.

Aladdin on kiintoisa yritys ensinnäkin sen tarjoaman korkean tason määrittelyrajapinnan vuoksi ja toisekseen siinä käytetty menetelmä auttaa jäsentämään ja rakenteistamaan määrittelyä. Ongelmana näen tämän kaltaisissa järjestelmissä niiden käyttämän määrityskielen: graafiset oliot ovat Aladdinissakin järjestelmän sisäisiä, mihinkään yleiseen formaattiin sitoutumattomia. Käytännössä tämä johtaa siihen, että niiden valikoima jää suppeaksi.

Aladdin on Macintosh-pohjainen ja Modula-2 kielellä kehitetty. Myös algoritmien kielenä käytetään Modula-2:sta. Järjestelmästä ei ole julkaistu vapaasti levitettävää versiota. Hyrskykarin lisensiaatintyössä on esimerkkinä ainoastaan yksi AVL-puun visualisointi, pelkästään sen perusteella on vaikea arvioida järjestelmän hyötyä ja soveltuvuutta. Järjestelmä näyttääkin jääneen tutkimuksen yhteydessä rakennetuksi prototyypiksi.

4.5 ELIOT ja JELIOT

Helsingin yliopistossa kehitetty Eliot [Lahtinen *et al.*, 1996] pyrkii samaan päämäärään kuin Aladdinkin, eli animoijan työn helpottamiseen vähentämällä esitysten konstruointiin kuluvaa työmäärää. Keskeisenä periaatteena tässä käytetään tietorakenteiden automaattista animointia.

Animoija kirjoittaa ohjelman muunnellulla c-kielellä, joka eroaa perus c-kielestä ainoastaan tietorakenteiden määritysten osalta. Animointikomentoja ei ohjelmaan

tarvitse lisätä. Tämän jälkeen ohjelmaan linkitetään animoidut tietorakenteet ja tuotetaan suorituskelpoinen ohjelma. Grafiikan tuottamiseen käytetään Polka-järjestelmää [Stasko & Kraemer, 1993]. Alkuperäinen Eliot on toteutettu unix-ympäristössä, nykyisin siitä on saatavilla Java-kielinen versio Jeliot [Haajanen *et al.*, 1997].

Automaattinen animaatio perustuu tässä muutamiin perustietotyyppeihin kohdistuvien sijoitus-, vertailu- ja laskuoperaatioiden monitorointiin. Nämä saavat aikaan muutoksia valmiiksi määritellyissä grafiikkaobjekteissa.

Eliotin käyttämään konseptiin liittyy perustavanlaatuinen ongelma: kuinka pystytään esittämään automaattisesti niitä piileviä korkean tason ajatusmalleja, joita algoritmeihin välttämättä sisältyy ei-triviaaleissa tapauksissa? Tekoälyn nykyisen kehityksen valossa on selvää, että algoritmeja aidosti ymmärtävän tekoälyn tuottaminen on mahdotonta vielä pitkään aikaan.

Näen automaatioon perustuvat järjestelmät tarpeellisina kartoitettaessa automaation mahdollisuuksia. Kuitenkaan en usko, että automaatio voi toimia *viime kädessä* algoritmien animoinnin perustana, sillä aina tarvitaan tietoutta visualisoitavien asioiden merkityksestä. Sen sijaan yhdistelemällä automaatiota sekä animoijan tietämystä ja kokemusta voimme luoda entistä helppokäyttöisempiä ja hyödyllisempiä järjestelmiä.

4.6 GAIGS

GAIGS [Naps, 1990] on Windows-pohjainen algoritmien visualisointijärjestelmä, josta on olemassa myös DOS- ja Macintosh versiot. Perusideana on lukea ascii-tiedostosta tietorakenteiden tilojen kuvauksia ja muuntaa nämä graafisiksi esityksiksi. Järjestelmään on määritelty valmiiksi kuvaukset perustietotyypeille, kuten pinoille, linkitetyille listoille, taulukoille, graafeille ja puille.

Tietorakenteiden kuvauksia pystytään luomaan paitsi suoraan tekstieditorilla myös instrumentoimalla ohjelmiin tulostuskomentoja tai käyttämällä järjestelmän kirjastoja osana ohjelmaa. On myös mahdollista suorittaa ulkoisia algoritmeja järjestelmän alaisuudessa ja konstruoida näin esityksiä.

Mukana seuraavat tietorakenteiden kuvaukset ovat selkeitä, mutta eivät mitenkään erityisen näyttäviä, värien käytön ollessa paikoin onnistunutta. Valmiita Pascal-kielisiä algoritmeja on viitisenkymmentä ja mukana seuraa kattava dokumentti järjestelmän käytöstä.

Ongelmaksi GAIGS:n kohdalla jää sen perusluonne: diskreetisti vaihtuvat tietorakenteiden kuvaukset eivät kerta kaikkiaan ole riittävä perusta algoritmien havainnollistamiseen. GAIGS soveltuu hyvin esimerkiksi tietorakenteita käsitteleville kursseille, varsinaiseen algoritmien animointiin siitä ei kuitenkaan ole.

4.7 Dynalab

Rossin kehittämä Dynalab [Birch *et al.*, 1995] on tekstipohjainen Pascal-ohjelmien suoristusta havainnollistava ohjelma. Itse asiassa Dynalab on lähinnä debuggeri – se ei tarjoa mitään tietorakenteiden visualisointeja vaan näyttää kaiken tiedon tekstipohjaisena listamuotoisesti.

Dynalabista löytyy kuitenkin muutama mielenkiintoinen seikka: käyttäjän kannalta se on perinteisiä debuggereita miellyttävämpi, koska se tarjoaa mm. mahdollisuuden ohjelman suorittamiseen takaperin, syötteiden lukemiseen levyltä, ohjelman suorituskustannusten laskemiseen ja vaihtoehtoisiin askelluksen suorittamistapoihin. Tekniseltä kannalta mielenkiintoista on pohjaratkaisu, virtuaalikoodin käyttäminen.

Dynalab on vapaasti levitettävä Windows-ohjelma, jonka mukana seuraa useita vaihtoehtoisia esimerkkipaketteja. Ohjelma on valmis ajettavaksi, eikä omaa mitään erityisvaatimuksia. Koska siinä käytetään ohjelmoinnin opiskeluun erinomaisesti soveltuvaa Pascal-kieltä, näen tämän järjestelmän kaikesta rajoittuneisuudestaan huolimatta hyväksi apuvälineeksi ohjelmoinnin peruskursseille.

4.8 Muu alueella tehty työ

Olen edellä pyrkinyt esittelemään ainoastaan kaikkein kiinnostavimmat ja kehityksen kannalta merkittävimmät järjestelmät. Esiteltyjen järjestelmien piirteet kattavatkin melko hyvin ohjelmien visualisoinnin parissa tehdyn työn olennaiset kohdat. BALSA:sta [Brown & Sedgewick, 1984] alkanut kehitys on johtanut paitsi

instrumentointiin perustuvan konseptin kehittämiseen myös vaihtoehtoisiin määrittelytapoihin perustuvien kokeilujen syntyyn.

Esitettyjen järjestelmien lisäksi on tehty lukuisa joukko erilaisia havainnollistuksia, joista suurin osa on yksittäisiä visualisointeja. Mielenkiintoisimpia löytämiäni esimerkkejä on binääripuita käsittelevä Opsis [Michail, 1996;1997], joka yhdistää visuaalista ohjelmointia ja oikeaksi todistamisen näkökulmaa. Toinen mielenkiintoinen idea on Excel-taulukkolaskentaohjelman hyödyntäminen animoinnissa [Rautama *et al*., 1997].

Myös erilaisia malleja sekä pohjaratkaisuja on tuotettu. Mielestäni erityisen tärkeää jäsennyksen näkökulmaa lähestytään Storeyn, Fracchian ja Carpendalen [1994] artikkelissa. Automaation mahdollisuutta ovat kartoittaneet Lin [1991], Ford [1993] sekä Manuel [1995]. Empiirisiä ja teoreettisia perusteita on tutkinut Hundhausen [1994].

Ibrahim [1994] esitti kenties ensimmäisen www:tä hyödyntävän mallin. Nykyään hänen esityksensä on kuitenkin jo auttamattomasti vanhentunut tarjoten lähinnä historiallista perspektiiviä. Mocha [Baker et al., 1996] on ajanmukaisempi malli, mutta painotuksissaan hieman omituinen. WWW:n roolia kokonaisuutena pohditaan McNallyn ja kumppaneiden [1997] artikkelissa.

4.9 J'anime!

J'anime*!* on Joensuun yliopistossa toteuttamani www-pohjainen algoritmien animointijärjestelmä. Järjestelmä pohjautuu Java-kielisten ohjelmien instrumentointiin. J'anime*!* ei ole tarkoitettu varsinaisesti itsenäiseksi animointivälineeksi (vaikka se siihen soveltuukin), vaan se on tarkoitettu kirjalliseen oppimateriaaliin liitettäväksi.

J'anime*!*:n merkittävin piirre liittyy algoritmien jäsennykseen. Metaforana käytetään aikavaativuutta, jonka mukaisesti algoritmit jaetaan perustavanlaatuisiin yksiköihin, askeleisiin, sekä edelleen merkityksellisiin tapahtumiin. Näin saadaan malli, jonka mukaisesti toteutetut algoritmit voidaan suorittaa rinnakkain semanttisesti synkronoidusti. Myös suorituksen kulkua pystytään havainnollistamaan tarkasti.

Toinen J’anime*!*:n yhteydessä kokeiltu piirre liittyy tietorakenteiden esitysmalleihin. Käytännössä tämä tarkoittaa vahvasti parametrisoituja, optimaaliseen objektien esitykseen pyrkiviä grafiikkaobjektien joukkoja.

J’anime*!*:n jatkokehitys on toistaiseksi jäissä. Suunnitelmia on kuitenkin olemassa mm. jäsennyksen edelleen kehittämiseksi sekä animaatiokielen luomiseksi.

5 KEHITTYNEEN JÄRJESTELMÄN OMINAISUUKSIA

Suhteutan tässä edellisessä luvussa esiteltyjen järjestelmien piirteitä ja kirjallisuudessa esiintyneitä ajatuksia kohdassa 3.3 esittämiini arviointiperusteisiin. Tällä pyrin kokoamaan yhtenäisen kuvan järjestelmistä sekä esittämään millainen kehittyneen algoritmien animointijärjestelmän tulisi olla.

5.1 Käyttötarkoitus ja soveltuvuus

Suurin osa järjestelmistä on tarkoitettu lähinnä opetuskäyttöön, erityisesti demonstraatiovälineiksi. Vaikka järjestelmiä on käytetty tutkimuksen välineenä jo BALSA:sta alkaen [Brown, 1988, 5], pääpaino näyttää kuitenkin olevan opetuksessa.

Nykysuuntauksena on interaktiivisten, oppilaslähtöisten järjestelmien kehittäminen. Täytyy kuitenkin muistaa, että pelkkä mahdollisuus visualisointien muodostamiseen ei tee järjestelmästä sen paremmin opetuksen kuin tutkimuksenkaan välinettä. Kehittyneen järjestelmän tulisikin erikoistua tai vaihtoehtoisesti tarjota kaikkiin käyttötarkoituksiin soveltuvat ratkaisut.

Useimmat järjestelmistä soveltuvat proseduraalisilla ohjelmointikielillä toteutettujen suhteellisten pienten ohjelmien visualisointiin. Poikkeuksena ovat ne järjestelmät, jotka soveltuvat millä tahansa kielellä toteutettujen ohjelmien havainnollistamiseen. Tällöin ne kuitenkin yleensä käyttävät post-mortem tekniikkaa, eivätkä näin ollen tarjoa mahdollisuutta ajonaikaisiin visualisointeihin.

Kielivalinta onkin yksi vaikeimpia käytännön ongelmia. Toimivimman tuntuiselta ratkaisulta vaikuttaa Dynalabin [Birch *et al.*, 1995] malli, missä toteutetaan virtuaalitulkki ja luodaan tälle useita erikielisiä lähdekoodiratkaisuja. Toisena mahdollisuutena näkisin, että nimenomaan algoritmien animoinnin tapauksessa käytettäisiin erityistä animaation määrittelykieltä.

5.2 Visualisoinnin kohde

Kaikki järjestelmät Dynalabia [Birch *et al.*, 1995] lukuunottamatta keskittyvät tietorakenteiden havainnollistamiseen. Useimmissa järjestelmissä on mukana lisäksi ohjelmakoodin esitys. Suorituksen kulkua sen sijaan esitetään huomattavasti harvemmin. Tämä tuntuukin jääneen case-välineiden alueeksi.

Mielenkiintoista on, että Brownin [1988] esittämiä historiatietoja ei järjestelmissä liiemmälti näy. Kuitenkin näen ominaisuuden erittäin hyödylliseksi, joskin vaikeasti hyödynnettäväksi.

5.3 Visualisoinnin luonne

Abstraktiotaso on järjestelmistä usein hyvin konkreettinen, käytännössä se on monesti pelkkää suorituksen monitorointia. Osa järjestelmistä tarjoaa kuitenkin monipuolisempia esityksiä, mielenkiintoisimpia ovat useita eritasoisia näkymiä tarjoavat visualisoinnit. Tällä saralla on kuitenkin vielä paljon tehtävää. Muokkaan tässä Peter Greenawayn sanomaa: ”*algoritmien animointi on aivan liian kiinnostava väline jätettäväksi pelkkään ohjelman visualisointiin*”.

Havainnollistuksen luonne on useimmiten esittävä, elokuvamainen. Uskon kuitenkin, että tulevaisuudessa interaktiiviset ja yhteistyöhön pohjautuvat järjestelmät tulevat yleistymään.

5.4 Esitystekniikka

Ainoastaan muutama järjestelmä käyttää diskreettejä siirtymiä tilojen välillä. Tässä suhteessa näytetään olevan yhtä mieltä sulavan animaation tarpeellisuudesta. Täytyy kuitenkin muistaa, että joskus on tarpeen esittää muutokset diskreetteinä informaatiotulvan välttämiseksi.

Monet järjestelmät tarjoavat useita synkronoituja näkymiä, mikä onkin sinällään ilahduttavaa. Jälleen täytyy kuitenkin muistaa, että pelkkä mahdollisuus näihin ei tee visualisoinneista sen selkeämpiä, vaan niiden käytön täytyy olla hallittua ja perusteltua.

Useiden eri algoritmien samanaikaiseen esitykseen liittyy joukko ongelmia, joita ei mikään esitellyistä järjestelmistä ratkaise tyydyttävästi. Tämän näen johtuvan pääasiassa siitä, ettei havainnollistuksille ole luotu selkeitä rakenteita.

Esitysmuoto on useimmiten ohjelman sisäinen, mikä käytännössä tarkoittaa sitä että esitykset ovat ainoastaan järjestelmän itsensä avulla katseltavissa. Olisi kuitenkin tärkeää tukeutua johonkin tunnettuun esitysmuotoon, joita sekä kaksi- että

kolmiulotteisen grafiikan alueelta löytyy enemmän kuin tarpeeksi. Itse näkisin VRML 2.0-kielen [VRML 2.0, 1996] erittäin käyttökelpoiseksi.

5.5 Havainnollistuksien konstruoinnin välineet

Osa järjestelmistä tarjoaa joukon grafiikkaobjekteja animaatioiden konstruointiin. Rikkaimmillaankin nämä ovat kuitenkin melko primitiivisiä. Tässä suhteessa olisikin järkevintä tukeutua johonkin lukuisista animointikielistä, jolloin saataisiin käyttöön heti sekä määrällisesti että laadullisesti rikas valikoima objekteja. Samalla voisi ratketa komentokielen ongelma.

5.6 Määrittelymenetelmä

Useimmat järjestelmät käyttävät instrumentointia sen eri muodoissa. Käytännön esitysten perusteella näyttääkin siltä, että instrumentointi tulee säilymään käyttökelpoisena välineenä vielä pitkään. Erilaiset monitoroinnin muodot ja deklaratiivinen määritystapa ovat kuitenkin varteenotettavia tekijöitä, varsinkin ohjelmien visualisointijärjestelmissä.

5.7 Vuorovaikutus

Vuorovaikutustekniikka perustuu järjestelmissä useimmiten valikoihin ja painikkeisiin. Ainoastaan muutama järjestelmä tarjoaa suoramanipulointimenetelmiä. Tämänkin ongelman katson ratkeavan ainakin osittain jonkin kehittyneen esitysformaatin kautta.

Esitysten kulun vaikuttamismahdollisuuksiin olisi syytä kiinnittää parempaa huomiota. Erityisesti hämmästyttää esitysten taaksepäin suorittamisen mahdollisuuden puuttuminen useista järjestelmistä. Tämä osoittaa mielestäni heikkoa perusratkaisua esitysten konstruoinnissa.

Navigoinnin mahdollisuudet oli lähes poikkeuksetta jätetty yksinkertaiseen, manuaaliseen zoomaustoimintoon. Esimerkit olivat kautta linjan sen verran suppeita, etteivät navigoinnin puutteet tulleet esille niiden pohjalta. Mikäli järjestelmä kuitenkin pyrkii skaalautumaan laajoille syötteille, tulee sen tarjota riittävät navigoinnin välineet, erityisesti kolmiulotteista grafiikkaa käytettäessä.

5.8 Mukana seuraavat havainnollistukset ja ohjeet

Ainoastaan Staskon järjestelmissä (luku 4.2) sekä Dynalabissa [Birch *et al.*, 1995] ja GAIG:ssa [Naps, 1990] oli kunnollinen valikoima esimerkkejä. Muiden esimerkkivalikoima oli pahimmillaan yhden havainnollistuksen varassa. Ohjeistuksen puolesta ainoastaan GAIGS oli riittävästi varusteltu. Tämä tuntuu aika paradoksaalisesta ottaen huomioon, että ollaan tekemisissä opetusohjelmien kanssa.

5.9 Saatavuus ja vaatimukset

Useat järjestelmistä ovat tutkimustyön ohessa syntyneitä prototyyppejä, eikä niitä ole annettu julkiseen levitykseen. Monet levitykseen tarkoitetuista järjestelmistä on tehty sellaisille laite- ja alustaratkaisuille, jotka ovat jarruttaneet käytännön saavutettavuutta. Ainoastaan muutamat järjestelmistä ovatkin laajalti levinneitä.

Pelastusta etsitään www-pohjaisista ratkaisuista. Tällä hetkellä siirretäänkin useita järjestelmiä joko Java-kielelle tai muulla tavoin tietoverkkoja hyödyntäväksi. Onkin mielenkiintoista nähdä, nostaako tämä järjestelmien todellista käyttöastetta.

6 LOPUKSI

Olen luonut tässä työssä katsauksen algoritmien visualisointiin tarkoitettujen järjestelmien nykytilaan sekä teorian että käytännön alueilla. Vertailemalla teorioita ja suhteuttamalla niitä joukkoon toteutettuja järjestelmiä olen koonnut yhteen näkemyksiä kehittyneen algoritmien animointijärjestelmän ominaisuuksista.

Eniten järjestelmien toteutuksissa näyttää olevan ongelmia kokonaisuuden hallinnan suhteen. Järjestelmiltä puuttuu miltei järjestään joko kunnollinen suunnittelu, toteutus, esimerkkiaineisto, ohjeistus, esillepano tai evaluointi. Mikään järjestelmistä ei toteuta näitä kaikkia tyydyttävästi.

Näen alan hyötyvän erityisen paljon multimediatyöasemien ja vaihtoehtoisten kommunikointitapojen nopeasta kehityksestä. Käytettävät tekniset välineet ovat kuitenkin jo nyt melko hyvät. Uskon, että tarvitaan ennen kaikkea kunnollista sisältötuotantoa ja tämän teoriaa. Yhdistämällä edistykselliset tekniset ratkaisut huolellisesti konstruoituihin esimerkkeihin ja opiskelijan tarpeita painottavaan työskentelyyn, voidaan luoda aidosti merkityksellisiä kokonaisuuksia.

Järjestelmien kehityksen kannalta näkisin tärkeimmäksi asiaksi huomion kohdistamisen jäsentyneempiin visualisointeihin. Epämääräisestä ohjelman käsitteestä on päästävä selkeään algoritmin malliin. Tähän uskon päästävän perusmerkitysten selkeyttämisellä, algoritmin jäsennyksellä sekä erityisen algoritmien animaatiokielen käyttöönotolla.

LÄHDELUETTELO

[Baecker, 1981] Baecker, R. *Sorting Out Sorting*. Narrated colour videotape, 30 minutes, presented at ACM SIGGRAPH '81 and excerpted in ACM SIGGRAPH Video Review #7, 1983. Los Altos, CA: Morgan Kaufmann.

[Baecker & Marcus, 1990] Baecker, R., Marcus, A. *Human Factors and Typography for More Readable Programs*. Addison-Wesley, 1990.

[Baeza-Yates et al., 1992] Baeza-Yates, R., Jara, L., Quezada, G. *VCC: Automatic animation of C programs*. COMPUGRAPHICS '92. Lisboa, Portugal, 1992: 389-397.

[Baker *et al.*, 1996] Baker, J., Cruz, I., Liotta, G., Tamassia, R. *The Mocha algorithm animation system*, In Proceedings of the International Workshop on Advanced Visual Interfaces. ACM Press, 1996.

[Bentley & Kernighan, 1987] Bentley, J., Kernighan, B. *A System for Algorithm Animation. Tutorial and User Manual*. Tech. Rep. 132, Computer Science, AT&Bell Laboratoties, Murray Hills, NJ, January 1987.

[Birch *et al.*, 1995] Birch, M., Boroni, C., Goosey, F., Patton, S., Poole, D., Pratt, C., Ross, R. *DYNALAB: A Dynamic Computer Science Laboratory Infrastructure Featuring Program Animation*, In Twenty-sixth SIGCSE Technical Symposium on Computer Science Education (SIGCSE Bulletin), volume 27: 29-33, March 1995.

[Boroni *et al.*, 1996] Boroni, C., Eneboe, T., Goosey, F., Ross, J., Ross, R. *Dancing with Dynalab. Endearing the Science of Computing to Students*. Proceedings of the Twenty-Seventh SIGCSE Technical Syposium on Computer Science Education Philadelphia, Pennsylvania February 15-18, 1996: 135-139.

[Brown, 1988] Brown, M. *Algorithm Animation*. The MIT Press, Cambridge, MA, 1988.

[Brown, 1992] Brown, M. *Zeus: A system for algorithm animation and multi-view editing (Research Report No.75)*. DEC Systems Research Center, Palo Alto, CA.

[Brown, 1993] Brown, M. *The 1992 SRC Algorithm Animation Festival*. In Proceedings of the 1993 IEEE Symposium on Visual Languages: 116-123, 1993.

[Brown & Hershberger, 1991] Brown, M., Hershberger, J. *Color and sound in algorithm animation*. Computer, 25(12): 52-63.

[Brown & Najork, 1993] Brown, M., Najork, M. *Algorithm animation using 3D interactive graphics* (Technical Report 110a). DEC Systems Research Center, Palo Alto, CA.

[Brown & Najork, 1996] Brown, M., Najork. M. *Collaborative Active Textbooks: A Web-Based Algorithm Animation System for an Electronic Classroom*. Proceedings of IEEE Symposium on Visual Languages, 1996: 266-275.

[Brown & Sedgewick, 1984] Brown, M., Sedgewick, R. *A System for Algorithm Animation*. Computer Graphics 18, 3(July 1984): 177-185.

[Brown *et al.*, 1997] Brown, M., Najork, M., Raisamo, R. *A Java-Based Implementation of Collaborative Active Textbooks*. To appear in 1997 IEEE Symposium on Visual Languages (VL'97), Capri, Italy, Sept. 23-26, 1997.

[DiGiano, 1992] DiGiano, C. *Visualizing program behavior using non-speech audio*. Master's thesis, University of Toronto, Department of Computer Science, 1992.

[Ford, 1993] Ford, L. *Automatic Software Visualization using Visual Arts techniques*. Research Report No. 279, Department of Computer Science, University of Exeter, September 1993.

[Haajanen *et al.*, 1997] Haajanen, J., Pesonius, M., Sutinen, E., Tarhio, J., Teräsvirta, T., Vanninen, P. *Animation of User Algorithms on the Web*. To appear in Proceedings of the IEEE Symposium on Visual Languages '97, IEEE, 1997.

[Haibt, 1959] Haibt, L. *A Program to Draw Multi-Level Flow Charts*. In Proceedings of The Western Joint Computer Conference, 15: 131-137. San Francisco, CA.

[Helttula, 1988] Helttula, E. *Erillisten joukkojen käsittelyalgoritmien analyysi ja havainnollistaminen*. Pro gradu –tutkielma, Tietojenkäsittelyopin laitos, Tampereen yliopisto, 1988.

[Henry *et al*., 1990] Henry, R., Whaley, K., Forstall, B. *The University of Washington Illustrating Compiler*. In Proceedings of The ACM SIGPLAN'90 Conference on Programming Language Design and Implementation: 223-233. New York. ACM.

[Hundhausen, 1994] Hundhausen, C. *The Search for an Empirical and Theoretical Foundation for Algorithm Visualization*. Unpublished technical report, Department of Computer & Information Science, University of Oregon, Eugene, OR.

[Hyrskykari, 1994] Hyrskykari, A. *Erään algoritmien animaatiokehittimen suunnittelu ja toteutus*. Raportti B-1994-5. Tietojenkäsittelyopin laitos, Tampereen Yliopisto, 1994.

[Ibrahim, 1994] Ibrahim, B. *World-wide algorithm animation*. In Proceedings of the First World-Wide Web Conference (WWW'94), Geneva, Switzerland, Computer Science Department, University of Geneva: 305-316, May 25-27, 1994.

[Knowlton, 1966] Knowlton, K. *L6: Bell Telephone Laboratories Low-Level Linked List Language*. 16 mm black and white sound film, 16 minutes. Murray Hill, NJ: Technical Information Libraries, Bell Laboratories, Inc.

[Knuth, 1963] Knuth, D. *Computer-Drawn Flowcharts*. Communications of the ACM, 6(9): 555-563.

[Lahtinen *et al*., 1996] Lahtinen, S.-P., Lamminjoki, T., Sutinen, E., Tarhio, J. Tuovinen, A.-P. *Towards automated animation of algorithms.* In Proceedings of Fourth International Conference in Central Europe on Computer Graphics and Visualization 96 (ed. N. Thalmann and V. Skala). University of West Bohemia, Department of Computer Science, 1996: 150-161.

[Lin, 1991] Lin, Y.-J., *A Framework for Automatic Algorithm Animation.* Technical Report No. CS-91-37, Brown University, Department of Computer Science, May 1991.

[Manuel, 1995] Manuel, D. *The Impossible Dream: Towards General and Automatic Visualizations*. Technical Report No. 322, Department of Computer Science, University of Exeter, February 1995.

[McNally et al., 1997] McNally, M., Jimenez-Peris, R., Patino-Martinez, M., Tarhio, J., Naps, T., Bergin, J., Proulx, V. *Using the WWW as the delivery mechanism for interactive, visualization-based instructional modules*. To appear in the working group report of ItiCSE '97, Integrating Technology into Computer Science Education, ACM, 1997.

[Meisalo *et al*., 1997] Meisalo, V., Rautama, E., Sutinen, E., Tarhio, J. *Teaching algorithms with animation – a case study using Eliot*. To apper in Proc. of LeTTET'96, Learning Technology and Telematics in Education and Training, Joensuu, Finland, 1997.

[Michail, 1996] Michail, A. *Teaching binary tree algorithms through visual programming*. In Symposium on Visual Languages: 38-45. IEEE, September 1996.

[Michail, 1997] Michael, A. *Visual Programming without Procedures*. Technical Report UW-CSE-97-05-02. Department of Computer Science and Engineering, University of Washington, Box 352350 Seatle, Washington, 98195.

[Myers, 1986] Myers, B. *Visual Programming, programming by example and program visualization: a taxonomy*. Proceedings of the CHI'86 Conference on Human Factors in Computer Systems, April 1986: 59-66.

[Myers, 1990] Myers, B. *Taxonomies of Visual Programming and Program visualization*. Journal of Visual Languages and Computing, 1(1): 97-123, 1990.

[Mukherjea & Stasko, 1994] Mukherjea, S., Stasko, J. *Applying Algorithm Animation Techniques for Program Tracing, Debugging, and Understanding*. Proceedings of the 15th International Conference on Software Engineering, Baltimore, MD, May 1993: 456-465.

[Naps, 1990] Naps, T. *GAIGS User's Manual*. Lawrence University P.O. Box 599 Appleton, WI 54912. 1990.

[Price *et al.*, 1993] Price, B., Baecker, R. Small, I. *A Principled Taxonomy of Software Visualization*. Journal of Visual Languages and computing, 4(3): 211-266, 1993.

[Rautama *et al.*, 1997] Rautama, E., Sutinen, E., Tarhio, J. *Excel as an algorithm animation environment*. In Proceedings of the ITiCSE '97, Integrating Technology into Computer Science Education, ACM, Uppsala, 1997: 24-26.

[Roman & Cox, 1992] Roman G-C, Cox K. *Program visualization: the art of mapping programs to pictures.* Proceedings of the 14th international conference on software engineering: 412-419, 1992.

[Roman & Cox, 1993] Roman G-C, Cox K. *a Taxonomy of Program Visualization Systems*. IEEE Computer, (12): 11-24, December 1993.

[Roman *et al.*, 1992] Roman, G-C, Cox, K., Wilcox, C., Plun, J. *Pavane: a System for Declarative Visualization of Concurrent Computations.* Journal of Visual Languages and Computing, vol. 3, 1992.

[Stasko, 1990] Stasko, J. *Tango: A Framework and System for Algorithm Animation*. IEEE Computer, 23(9): 27-39.

[Stasko, 1991] Stasko, J. *Using Direct Manipulation to Build Algorithm Animations by Demonstration*. In S. P. Robertson, G. M. Olson, & J. S. Olson (Ed.), Proceedings of Human Factors in Computing Systems (CHI'91): 307-314. New York: ACM Press.

[Stasko, 1992] Stasko, J. *Animating Algorithms with XTANGO*. SIGACT News, Vol. 23, No.2, Spring 1992: 67-71.

[Stasko, 1996] Stasko, J. *Using Student-Built Algorithm Animations as Learning Aids*. Graphics, Visualization, and Usability Center, Georgia Institute of Technology, Atlanta, GA, Technical Report GIT-GVU-96-19, August 1996.

[Stasko & Kraemer, 1993] Stasko, J., Kraemer, E. *A methodology for building application-specific visualizations of parallel programs*. Journal of Parallel and Distributed Computing, 18(2): 258-264, June 1993.

[Stasko & Patterson, 1993] Stasko, J., Patterson, C. *Understanding and characterizing program visualization systems.* In Proceedings of IEEE 1992 Workshop on Visual Languages: 3-10. New York: IEEE Computer Society Press.

[Stasko *et al.*, 1993] Stasko, J., Badre, A., Lewis, C. *Do Algorithm Animations Assist Learning? An Empirical Study and Analysis*. Proceedings of the INTERCHI '93 Conference on Human Factors in Computing Systems, Amsterdam, Netherlands, April 1993: 61-66.

[Storey *et al*., 1994] Storey, M-A., Fracchia, F., Carpendale, S. *A Top-Down Approach to Algorithm Animation*. Techical Report CMPT 94-05. School of Computing Science Simon Fraser University Burnaby, B.C. V5A 1S6 CANADA, September 1994.

[VRML 2.0, 1996] *The Virtual Reality Modeling Language Specification, Version 2.0*. URL:http://vrml.sgi.com/moving-worlds/index.html. August 1996.