Papervision3D:n käyttö verkkosivuilla

HAMK
HÄMEEN AMMATTIKORKEAKOULU

Ammattikorkeakoulututkinnon opinnäytetyö

Mediatekniikka

Riihimäki, 23.3.2010.

Matti Kallonen

HAMK
HÄMEEN AMMATTIKORKEAKOULU

OPINNÄYTETYÖ

Mediatekniikka
Riihimäki

Työn nimi	Papervision3D:n käyttö verkkosivuilla
Tekijä	Matti Kallonen
Ohjaava opettaja	Petri Kuittinen
Hyväksytty	23.3.2010
Hyväksyjä	Petri Kuittinen

HAMK
HÄMEEN AMMATTIKORKEAKOULU

RIIHIMÄKI
Mediatekniikka
Mediajärjestelmät

TIIVISTELMÄ

Tekijä Matti Kallonen **Vuosi** 2010

Työn nimi Papervision3D:n käyttö verkkosivuilla

TIIVISTELMÄ

Papervision3D on ilmainen, avoimen lähdekoodin, luokkakirjasto Adobe Flashiin. Papervisionin avulla on mahdollista tehdä dynaamista 3D-grafiikkaa Flash-sovelluksiin pienellä vaivalla. Papervision sisältää paljon ominaisuuksia, jotka nopeuttavat Flash-sovellusten tekoa ja oikealla tavalla hyödynnettynä tuovat niihin näyttävyyttä.

Opinnäytetyön tarkoituksena oli tehdä kattava paketti Papervisionista mainostoimisto White Sheep Oy:n käyttöön. Opinnäytetyössä käydään läpi Papervisionin ominaisuuksia, sen käyttömahdollisuuksia verkkosivuilla, kilpailijoita, vaihtoehtoja, 3D-grafiikan teoriaa, sekä opinnäytetyötä varten tehtyjä esimerkkisovelluksia Papervisionista. Opinnäytetyöstä on rajattu pois Papervisionin käyttötarkoitusten syvällisempi pohtiminen ja Papervisionin kehittyneimpien ominaisuuksien läpikäynti. 3D-grafiikan teoriaa käydään läpi opinnäytetyössä vain pintapuolisesti.

Opinnäytetyössä on sovellettu henkilökohtaista kokemusta Papervisionin käytöstä ja 3D-grafiikan teosta. Lisätietoa Papervisionista on saatu tutkimalla esimerkkisovelluksia, keräämällä tietoa verkkosivuilta, kuten blogeista ja keskusteluryhmistä. Lähdekoodin syvällinen tutkiminen on ollut myös tärkeä osa tutustumisessa Papervisioniin.

Papervision ja Flash kehittyvät nopeasti, joten tässä opinnäytetyössä on osioita, jotka saattavat vanhentua uusien versioiden myötä. Opinnäytetyö sisältää kuitenkin osioita, jotka on tarkoitettu kestämään aikaa ja joista on apua vielä vuosienkin päästä.

Avainsanat Flash, 3D, Papervision3D

Sivut 74 s. + liitteet 24 s.

HAMK
UNIVERSITY OF APPLIED SCIENCES

RIIHIMÄKI
Media Technology
Media systems

ABSTRACT

Author Matti Kallonen **Year** 2010

Subject of Bachelor's thesis Using Papervision3D for Internet Pages

ABSTRACT

Papervision3D is an open source real-time 3D engine for Adobe Flash. It allows Flash to have simple 3D graphics with interactivity and for little effort. Papervision contains a lot of features which makes it very powerful tool when building Flash sites.

The objective of this thesis was to make a comprehensive guide to Papervision. The target audience for the guide was the employees of the digital agency, White Sheep Ltd. The thesis contains features of Papervision, how to use it, what are the alternatives and competitors, basics of the 3D graphics and code examples. Advanced features of Papervision and deep analyses of cases of Papervision's use are excluded from this thesis. Also the theory of the 3D graphics covers only the basics.

Personal experience from 3D graphics and Flash is used in many parts of this thesis. Information about Papervision has been collected from various sources such as websites, blogs, forums and example applications. The Papervision source code was also an invaluable source of information.

Papervision and Flash are under constant development and some parts of this thesis will deprecate with future versions of Papervision and Flash. However some parts, such as the theory of 3D graphics, will be relative information after many years.

Keywords Flash, 3D, Papervision3D

Pages 74 p. + appendices 24 p.

SANASTO

Karsinta	Culling	Karsinnalla tarkoitetaan kohtauksessa piiloon jäävien monikulmioiden jättämistä pois piirtovaiheesta. Karsinnassa hyödynnetään yleensä useita eri tapoja ratkaista mitkä monikulmtiot ovat piilossa.
Kärkipiste	Vertex	3D-kappalen pinnat muodostuuvat kolmiulotteiseen avaruuteen sijoitetuista kärkipisteistä.
Leikkaaminen	Clipping	Leikkaamisella tarkoitetaan osittain ruudun ulkopuolelle menevien polygonien leikkaamista kahtia leikkauskohdasta, jolloin säästytään laskemasta merkityksetöntä tietoa.
Matriisi	Matrix	Matriisi on suorakulmainen alkioita sisältävä matemaattinen taulukko. 3D-grafiikassa matriiseja käytetään ilmoittamaan 3D-kappaleelle tehtyjä muunnoksia, kuten siirtoa ja skaalausta.
Monikulmio	Polygon	3D-kappaleen monikulmioverkko muodostuu yhdestä tai useammasta pinnasta koostuvista monikulmioista.
Näköfrustumi	View frustum	Rakennetusta 3D-maailmasta leikataan pyramidin muotoinen alue, joka määrittää, mikä alue kohta maailmasta näytetään. Näköfrustumin muoto määrittää myös kameran näkökentän.
Monikulmioverkko	Mesh	Monikulmioverkko pitää sisällään kaiken 3D-kappaleeseen liittyvän geometrian. Monikulmioverkko koostuu yhdestä tai useammasta monikulmiosta.

Pinta	Face	3D-kappaleen monikulmiot muodostuvat kolmesta kärkipisteestä koostuvista pinta-elementeistä.
Primitiivi	Primitive	Primitiivit ovat matemaattisilla kaavoilla luotuja geometrisia kappeleita, kuten pallo ja kuutio. Primitiiveille annetut parametrit määrittävät esimerkiksi kappaleiden koon ja monikulmioiden määrän.
Renderöinti	Rendering	Renderöinti on kaksiulotteisen kuvan laskemista 3D-geometriasta. Renderöinnissä otetaan huomioon myös kappaleiden pintamateriaalit, valaistus ja paljon muita asioita.
Reuna	Edge	3D-kappaleen pinta-elementillä on kolme reunaa
Tekseli	Texel	Tekstruurikuvan pikseleitä kutsutaan tekseleiksi. Renderöidyn kuvan useat pikselit on voitu luoda samasta tekselistä.
Tekstuuri	Texture	Teksturointi on tekniikka, jolla 3D-mallin pintojen täytössä käytetään hyväksi bittikarttaa. Tällä tavalla yksiväriseen kappaleeseen saa helposti useita värejä tai kuvoita.
Varjostin	Shader	Varjostimilla voidaan tuoda tasaväriseen kappaleeseen eri valaistuksen sävyjä. Varjostimia voidaan käyttää realistisuuden tavoitteluun tai taiteellisiin tarkoituksiin.
Versionhallinta-järjestelmä	Version control system	Versiohallintajärjestelmän tarkoituksena on mahdollista ohjelmistojen lähdekoodin muokkaamisen hajautetusti tietoverkon yli niin, että kaikilla muokkaajilla on viimeisin versio ohjelmasta.

SISÄLLYS

1. JOHDANTO

Papervision on luokkakirjasto Adobe Flashiin, jonka avulla on mahdollista tehdä dynaamista 3D-grafiikkaa Flash-sovellukseen. Papervision on avointa lähdekoodia, joten koodia voi muokata ja laajentaa kuka tahansa. Näistä parhaimmat ja hyödyllisimmät yleensä sisällytetään uusiin versioihin kaikkien käytettäviksi. Kirjoitushetkellä Papervisionissa on yli 300 luokkaa.

Papervisionin suurimpana etuna on, että loppukäyttäjä ei tarvitse muuta kuin Flash Player -selainlaajennuksen, joka löytyy lähes jokaiselta internetin käyttäjältä. Papervision on tällä hetkellä hyvin vakaa, jonka vuoksi sitä voi käyttää hyvin myös kaupallisiin tarkoituksiin. Suomessa Papervisionin käyttö on tosin vielä erittäin vähäistä, mutta ulkomailla sen avulla on tehty monia näyttäviä verkkosivuja.

Suurin osa Papervisioniin liittyvästä tiedosta on etsittävä erilaisista blogeista ja keskusteluryhmistä. Nämäkin tiedot saattavat olla vanhentuneita, eikä niitä voi soveltaa Papervisionin nykyisiin versioihin. Papervisioniin tutustuvan Flash-kehittäjän voi ollakin vaikea päästä siihen sisään ja tästä syystä jopa luopua sen käytöstä projektissa kokonaan. Tämän työn tarkoituksena onkin antaa lukijalle kattava tietopaketti Papervisionista, jonka avulla hän pääsee helposti mukaan Papervisionin maailmaan ja sen tuomiin uusiin mahdollisuuksiin. Työ toimii myös tukena projektin kehitysvaiheessa, jossa pohditaan erilaisia tapoja tietyn asian toteuttamiseen.

Itse sain käytännönkokemusta Papervisionin käytöstä tehdessäni verkkosivuja Pauligin Presidentti-brändille. Sivujen tarkoituksena oli tehdä pitkäaikaiset sivut tuomaan näkyvyyttä Presidentti-brändille, sekä tuoda esille uutta Gold Label -tuotemerkkiä. Projektissa haluttiin käyttää Papervisionia tuomaan näyttävyyttä. Projekti oli myös oiva tilaisuus tutustua Papervisioniin kunnolla. Projektin aikana asioita tehtiin uusiksi useita kertoja, kun niiden toteuttamiseen löytyi parempia tapoja. Papervision valittiin tekniikaksi muiden vaihtoehtojen joukosta vertailun ja näyttävien esimerkkien perusteella.

Tässä opinnäytetyössä keskitytään pääasiassa Papervision3D 2.1 -versioon, sillä se on kirjoitushetkellä paras ja monipuolisin versio Papervisionista. Tämä opinnäytetyö on kirjoitettu lähinnä Flashin oman ohjelmointiympäristön kannalta. Papervisionia on mahdollista kuitenkin hyödyntää myös Eclipsen tai Flexin kanssa. Oletan opinnäytetyössä, että lukija tuntee ActionScript 3.0:n hyvin, mutta on perehtynyt 3D:n teoriaan vain vähän tai ei lainkaan.

2. YLEISTÄ

2.1 Käyttöoikeudet

Papervisionin lähdekoodi on avointa ja se on julkaistu MIT-lisenssin alla. Tämä tarkoittaa, että sitä saa käyttää myös kaupallisiin suljetun lähdekoodin projekteihin. (MIT License 2010; Lindquist 2008a.)

2.2 Tekijät

Papervisionia hallinnoi ja kehittää kaksi pientä ryhmää, jotka koostuvat eri alojen osaajista. Tämän lisäksi Papervision-yhteisö auttaa heitä tuomalla esiin ongelmia ja ideoita, joita heille tulee vastaan omia sovelluksia kehitettäessä.
(Allen, Arnold, Balkan, Cannasse, Grden, Gunesch, Hughes, MacDonald & Zupko 2008, 292.)

Ydinryhmä vastaa julkaisuista, laadusta, tiimityksestä ja ohjelmointirajapinnasta. Ydinryhmään kuuluvat:

Carlos Ulloa:	Papervisionin perustaja ja johtaja.
John Grden:	Komponentti- ja julkaisuvastaava.
Ralph Hauwert:	Vastaa Papervisionin moottorin ytimestä ja varjostimista.
Tim Knip:	Vastaa Papervisionin moottorin ytimestä, varjostimista ja Collada-luokkiin liittyvistä asioista.
Andy Zupko:	Vastaa efekteistä.
Ben Hopkins:	Vastaa Papervisionin moottorin ytimestä.

(Allen, ym. 2008, 292.)

Apuryhmällä on lupa muokata lähdekoodia. Sen henkilöt ovat erikoistuneet tiettyihin asioihin, jotka jäävät ydinasioiden ulkopuolelle. Apuryhmään kuuluvat:

Ricardo Cabello:	Testaus ja demojen teko.
De'Angelo Richardson:	Testaus ja erikoisominaisuudet.
Stephen Downs:	Flex 2.0 komponentit ja efektit.
Seb Lee Delisle:	Partikkelit ja efektit.
John Lindquist:	Dokumentaatio ja esimerkit.

(Allen, ym. 2008, 292; Ulloa 2009.)

2.3 Historia

Carlos Ulloa aloitti Papervisionin kehittämisen marraskuussa 2005. Hän sai innoituksensa Joost Korngoldilta Amsterdamissa järjestetyssä Spark conference -tapahtumassa. Tammikuuhun 2006 mennessä Carlos Ulloa oli saanut aikaiseksi luokkakirjaston, jolla pystyi asemoimaan, kääntämään,

skaalaamaan ja skew-säätämään MovieClippejä. (Allen, ym. 2008, 291–292.)

2006 elokuussa Carlos Ulloa julkaisi uuden version kirjastoistaan nimellä Papervision3D. Se oli täysin optimoitu käyttämään Flash 8:n piirtorajapintaa. Tämä teki mahdolliseksi käyttää kolmiotesselointia, jonka avulla oli nopeampi teksturoida monikulmioita. Seuraavan kymmenen kuukauden aikana Papervision kehittyi 14 luokaksi ja lopulta suljettu Papervision3D 1.0 beta -versio julkaistiin. (Allen, ym. 2008, 291–292.)

Papervisionin kehitys nopeutui tämän jälkeen huomattavasti. Papervisionia varten luotiin uudet kotisivut ja blogi. Papervision lisättiin myös vapaata lähdekoodia esittelevälle OSFlash.org-sivustolle. Tässä vaiheessa John Gren liittyi mukaan projektiin ensimmäisenä ulkopuolisena. John Gren käänsi Papervisionin luokat ActionScript 3:lle, joka toi suuren parannuksen koodin nopeuteen. Tämän jälkeen useat ulkopuoliset kehittäjät saivat Papervisionin ladattavaksi ja antoivat projektille oman panoksensa. (Allen, ym. 2008, 291–292.)

2007 alussa Ralph Hauwert liittyi mukaan tiimiin. Hänen demonsa ja materiaalinsa tekivät vaikutuksen yhteisöön ja ne auttoivat tuomaan Papervisionille lisää julkisuutta. Samaan aikaa John Grden julkaisi ensimmäisen Papervisionilla tehdyn pelin. Pelissä lennetään X-Wing -aluksella pujotellen silmukoiden läpi (Grden 2007a). Tämän jälkeen myös ensimmäiset kaupalliset Papervisionilla tehdyt verkkosivut ilmestyivät. (Allen, ym. 2008, 291–292.)

Samana vuonna Tim Knip käytti useita kuukausia Collada-formaatin liittämisessä Papervisioniin. Kun tämä valmistui elokuussa, Tim Knip liittyi virallisesti Papervision-tiimiin. Tällöin julkaistiin myös ensimmäinen alpha-versio Papervision 2.0:sta. Uusi versio toi mukanaan muun muassa varjostimet, frustumikarsinnan, Collada-animaatiot ja useiden Viewport3D-luokkien käytön samassa sovelluksessa. (Allen, ym. 2008, 291–292.)

Uuden version julkaisemisen jälkeen Andy Zupko liittyi mukaan ydinryhmään. Hän toi mukanaan mm. efektejä. Tämän jälkeen ydinryhmän rinnalle perustettiin apuryhmä, joka keskittyi Papervisionin ydinominaisuuksien ulkopuolisiin asioihin ja testaukseen. (Allen, ym. 2008, 291–292.)

2.4 Miksi käyttää 3D:tä verkkosivuilla?

Oikein käytettynä 3D tuo verkkosivuille rikkaamman käyttäjäkokemuksen ja ammattimaisen kuvan. Mikäli verkkosivut eivät ole asiapitoisia vaan enemmänkin brändin tai tuotteen esille tuomista, kannattaa 3D:n käyttöä harkita verkkosivuilla. Koko sivua ei kuitenkaan ole välttämättä tarpeellista tehdä 3D:n ympärille vaan se voi toimia myös mausteena. Pienet komponentit, jotka hyödyntävät 3D:tä esimerkiksi kuvien selauksessa, voivat toimia jopa paremmin, nopeammin ja pienemmässä tilassa kuin perinteisellä tavalla tehdyt kuvagalleriat.

Asioiden visualisoinnissa 3D voi olla erittäin hyvä vaihtoehto. Sellaisten asioiden kuvaaminen, jotka eivät ole vielä olemassa, saattaa helpottua, jos apuna käytetään 3D:tä. Näitä voivat olla esimerkiksi uusi auto, tuleva maisema tai sisustettava huone.

Kun 3D:n avuksi lisätään vielä interaktiivisuus, kuten asioiden liikuttelu ja kameran hallinnointi, saadaan 3D:stä paljon enemmän irti. Yhteenvetona voidaan todeta, että jos hyviä puolia on käytetty oikein, saadaan aikaan sisältö, jonka tuottaminen muilla keinoilla yhtä hyvin on paljon vaikeampaa.

2.5 Miksi välttää 3D:tä verkkosivuilla?

3D:n käyttö nettisivuilla kannattaa kuitenkin harkita tarkkaan, sillä siihen liittyy myös ongelmia. Isona ongelmana on, että reaaliaikaisen 3D:n laskeminen esimerkiksi Flashissä on hyvin raskasta. Tämän takia verkkosivujen kohderyhmänä pitääkin olla henkilöt, joilla oletetaan olevan tarpeeksi tehokas tietokone. Tähän on tulevaisuudessa tulossa helpotusta kun koneiden tehot kasvavat. Toisaalta minikannettavien ja mobiililaitteiden yleistyminen tuo taas lisää internetin käyttäjiä, joilla ei ole tehokasta päätelaitetta. Uudemmat Flash Player -versiot parantavat suorituskykyä, mutta kaikilla käyttäjillä ei ole uusinta versiota tai edes mahdollisuutta asentaa sitä, koska heillä ei ole oikeuksia tai tietylle päätelaitteelle ei ole sitä saatavillakaan.

Projekti, jossa on mukana 3D:tä, vaatii mukaan osaavia kehittäjiä. Huonosti tehty projekti on yleensä vaikea käyttää, jolloin se ajaa käyttäjiä pois, sekä jättää käyttäjälle huonon kuvan sivustosta. 3D-sovelluksen kehittäminen vie myös paljon enemmän aikaa kuin perinteiset ratkaisut. Tämä lisää kehityskustannuksia, jolloin kaikissa soveltuvissa projekteissa ei 3D:n käyttö ole edes mahdollista.

Ongelmista johtuen 3D:tä ei kannata käyttää ollenkaan tai ainakaan paljoa sivustoissa, joissa asiapitoinen sisältö on tärkeintä.

2.6 Kilpailijat

Papervisionin tapaisia ratkaisuja on olemassa monia. Usein käy kuitenkin, että kehitystiimiltä loppuu kiinnostus tai aika projektia kohtaan. Tekniikka voi olla myös tehty alusta alkaen vaikeaselkoiseksi, jolloin sen laajentaminen on hankalaa. Projekti saattaa jäädä myös muiden tekniikoiden jalkoihin, niiden kehittyessä nopeammin.

Tässä kappaleessa on lueteltu tunnetuimpia kilpailijoita Papervisionille. Tekniikat eivät siis tarvitse käyttäjältä mitään muuta laajennusta kuin Flash Playerin. Joissain tapauksissa kilpailijat voivat soveltua erilaisten ominaisuuksiensa ansiosta projektiin paremmin kuin Papervision.

2.6.1 Away3D

Away3D on vapaata lähdekoodia ja ominaisuuksiltaan hyvin samantapainen kuin Papervision. Tämä johtuu siitä, että Away3D haarautui Papervisionin ensimmäisistä versioista, kun Alexander Zadorozhny ja Rob Bateman lähtivät muokkaamaan Papervisionia omien tarpeidensa mukaan. Kun he olivat kehittäneet Away3D:tä tarpeeksi, he näyttivät aikaansaannostaan Papervisionin kehittäjille Carlos Ulloalle ja Ralph Hauwertille. Away3D teki heihin vaikutuksen ja tämän seurauksena Away3D:stä tuotiin ominaisuuksia, jotka liitettiin Papervision3D 2.0:aan. (Grden 2007b.)

Kymmenhenkisen tiimin kehittämä Away3D on hyvin dokumentoitu ja sitä käytetään jo monessa isossa kaupallisessa projektissa. Away3D:stä ei ole ActionScript 2 -versiota lainkaan vaan se on kirjoitettu kokonaan ActionScript 3:lla. Away3D on kaiken kaikkiaan erittäin hyvä vaihtoehto Papervisionille, sillä se hyödyntää jo Flash Player 10 ominaisuuksia, jotka tekevät moottorista nopean. (Bateman 2009a; Zadorozhny 2007a.)

2.6.2 Sandy 3D

Sandy 3D on vapaata lähdekoodia. Sillä on myös lähes kaikki vastaavat ominaisuudet kuin Papervision3D:ssä. Moottori sisältää mm. primitiivejä, interaktiivisuutta, valot, varjostimia ja 3D-objektien parseroinnin. Objektien tekstuureina voidaan käyttää Papervisionin tapaan MovieClippejä, bittikarttoja tai videota. (Sandy 2009.)

Sandy 3D:tä kehittää neljän henkilön ryhmä. Sandy 3D:stä on olemassa ActionScript 2-, ActionScript 3- ja haXe-versiot. ActionScript 2 -versio toimii myös Flash Player 7:ssä. Dokumentaatio on selkeää ja moottoriin löytyy paljon esimerkkejä. (Sandy 2009.)

Sandy 3D on kuitenkin jäänyt vähän Papervision3D:n ja Away3D:n jalkoihin, eikä se ole lyönyt itseään vielä läpi kaupallisissa projekteissa. Sandy 3D:n tulevaisuus onkin hieman hämärän peitossa. (Sandy 2009.)

2.6.3 Sophie 3D

Sophie 3D on keskittynyt yksittäisten objektien esittämiseen. Sophie 3D tukee tekstuuri- ja heijastuskarttoja, läpinäkyviä materiaaleja, sekä rajoittamattoman määrän valoja. Koska moottori on optimoitu vain yhteen asiaan, saa sillä Flash-grafiikaksi erittäin näyttävää jälkeä, joka toimii sulavasti. (Sophie 2009.)

Ohjelman käyttöön tarvitsee ladata ilmainen SWF-tiedosto, jonka koko on noin 30kb. Ohjelmasta ei ole olemassa julkista lähdekoodia, vaan kaikki toiminnallisuudet mallin latauksesta sen liikutteluun on sisällytetty valmiiksi ohjelmaan. Ohjelma on ilmainen käyttää myös kaupallisiin tarkoituksiin, mutta ilmaisessa versiossa ohjelman logo on näkyvillä

vasemmassa yläreunassa. Ohjelmasta voi ostaa myös Pro-version, joka maksaa 239 euroa. Pro-versiossa logo on poistettu. (Sophie 2009.)

Malli ladataan ohjelmaan määrittelemällä sen polku HTML-sivulla. Flash Vars -parametreja käyttämällä ohjelmalle voidaan antaa asetuksia, kuten mallin skaalaus. Malli ladataan ohjelmaan OBJ-formaatissa, jonka on kehittänyt Wavefront Technologies. Kyseistä formaattia voidaan tallentaa lähes kaikista 3D-mallinnusohjelmista. Sophie 3D:n Pro-versio sisältää Sophie 3D Compressor -ohjelman, jonka avulla mallien kokoa voidaan pienentää huomattavasti. (Sophie 2009; Obj 2010.)

2.6.4 Alternativa3D

Alternativa3D on 15 henkisen venäläis-tekijätiimin 3D-moottori Flashiin. Alternativa3D on suljettua lähdekoodia. Asennuspaketti sisältää SWC-kirjaston, jota voi käyttää Flashissä tai Flexissä. Alternativa3D:n käyttö kaupallisiin tarkoituksiin maksaa 1000 euroa projektia kohden, muuten käyttö on ilmaista. (Alternativa 2010.)

Projekti on alkanut hyödyntää Flash Player 10:n ominaisuuksia jo varhaisessa vaiheessa. Moottorissa on käytetty myös paljon aikaa optimointiin. Näiden ansiosta sillä saadaan aikaan hyvin näyttävää jälkeä. (Alternativa 2010.)

Projektissa on hyvin vahvasti mukana monen samanaikaisen käyttäjän sovellukset. Tämä vaatii kuitenkin palvelimelta Javan ajamista. Moottorissa on myös vakiona törmäysten tunnistus kappaleiden välillä. Moottori soveltuukin hyvin esimerkiksi moninpeleihin. (Alternativa 2010.)

Tulevaisuudessa projektiin on tulossa helpommin hallittava ja monipuolisempi palvelinratkaisu, sekä graafinen suunnittelutyökalu 3D-näkymien luontiin. (Alternativa 2010.)

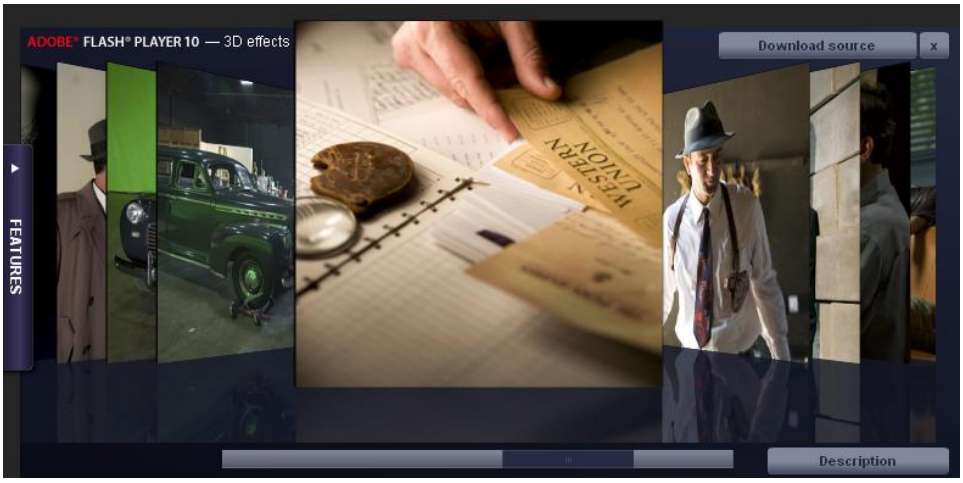
2.7 Vaihtoehdot

Papervisionin kaltaiset tekniikat eivät ole ainoita vaihtoehtoja toteuttaa 3D:tä verkkosivuilla. Tekniikoita on useita, mutta ne vaativat yleensä kehittäjältä monialaista osaamista. Tekniikoilla on aina hyvät ja huonot puolensa, joten tietyissä tilanteissa ne ovat varteenotettavia vaihtoehtoja.

2.7.1 Flash Player 10:n ominaisuudet

Flash Player 10 mukana tuli ominaisuuksia, jotka lisäsivät natiivin tuen MovieClippien liikuttamiselle z-akselin suuntaisesti, sekä kiertämisen y- ja x-akselin suhteen. Toisin sanoen MovieClippejä voidaan liikuttaa ja kääntää vapaasti kolmiulotteisessa maailmassa. Näin on mahdollista tehdä nopeasti ja helposti yksinkertaisia 3D-sovelluksia Flash CS4:lla. Koska kaikki tarvittavat ominaisuudet löytyvät jo valmiina Flash Player 10:sta, voi tällä tekniikalla tehdä kooltaan hyvin pieniä sovelluksia. Esimerkiksi

bannereissa, joissa kokorajat ovat pieniä, tämä on hyvä tekniikka. Tekniikkaa rajoittaa kuitenkin Flash Player 10:n levinneisyys, joka ei ole vielä yhtä laajaa kuin vanhempien versioiden. (McCauley 2009a; Adobe 2009a; Adobe2009b.)



KUVA 1 *Flash CS4:lla tehty kuvakaruselli*

2.7.2 Video tai kuvasarja

Esilasketun 3D:n käyttö on hyvä vaihtoehto silloin kun tarvitaan mahdollisimman yhteensopivaa ratkaisua, hienolla lopputuloksella. Esilasketussa 3D:ssä ulkonäkö ei ole riippuvainen käyttäjän päätelaitteesta vaan se näyttää aina samalta. Tällä tavalla saadaan aikaiseksi vaikka valokuvan tasoista 3D:tä mikäli siihen on resursseja. Kuvia ja videoita pystyy käyttämään verkkosivuilla monella eri tekniikalla, jolloin niitä voidaan käyttää moneen eri tarkoitukseen.

Videoiden ja kuvasarjojen käyttäminen esimerkiksi Flashissä mahdollistaa interaktiivisuuden lisäämisen. Näin niitä voidaan kontrolloida halutulla tavalla. Sovelluksessa voidaan esimerkiksi pyörittää esinettä videota kelaamalla tai kuvaa vaihtamalla.

Tässä tekniikassa on kuitenkin rajoituksia, jotka estävät sen käytön monissa tapauksissa. Mikäli kuvia tai videota on paljon, tulee sovelluksen koko kasvamaan isoksi. Tekniikka rajoittaa myös sovelluksen interaktiivisuutta hyvin paljon, sillä kuvakulmaa ei voida vapaasti säätää, eikä kappaleita voida liikuttaa tai animoida vapaasti. 3D:n eri kohtien painamisen tunnistaminen ei myöskään onnistu ilman erillistä ohjelmointia.

KUVA 2 *Flash-sovellus, jossa videokuvaa kelaamalla auto pyörii 360 astetta. Auto on aitoa videokuvaa, mutta sama periaate toimii 3D-videokuvan kanssa*

2.7.3 QuickTime VR

QuickTime VR (Virtual Reality) kuvaformaatti mahdollistaa yksinkertaisten panoraama ja esineen pyöritys -sovelluksien teon. Kuviin on myös mahdollista lisätä kohtia, joista painamalla tapahtuu määritettyjä asioita. QuickTime Player tukee vakiona tätä formaattia. (QuickTime VR 2009.)

Tämän tekniikan hyvänä puolena on sen helppokäyttöisyys ja projektin lyhyt kehitysaika. Monet eri sovellukset mahdollistavat kuvien tallentamisen QuickTime VR -formaattiin. Näin kehittäjän ei tarvitse osata ohjelmoida saadakseen hyvin toimivan sovelluksen. (Apple 2009a.)

Huonoina puolina on kuitenkin QuickTime Playerin vähäinen levinneisyys ja interaktiivisuuden suppeus. (Statowl 2009a.)

KUVA 3 *QuickTime VR –tekniikalla tehty panoraamasovellus metsästä. Kuvassa kahteen puuhun on lisätty interaktiivisuutta*

2.7.4 JavaScript

Tuleva HTML5 standardi tulee luultavasti pitämään sisällään Canvas-elementin, jonka avulla pystyy piirtämään vektorigrafiikkaa käyttäen käyttäen JavaScriptiä. Tekniikan on alun perin kehittänyt Apple vuonna 2005. Tekniikalla on mahdollisuus piirtää tekstuuritäytettyjä objekteja heijastuksineen. Yksinkertaistenkin kappaleiden reaaliaikainen laskeminen on kuitenkin hidasta. Selaimien nopeuksissa myös on huomattavia eroja. Kaikki yleisimmät web-selaimet, lukuunottamatta Internet Exploreria, tukevat jo Canvas-elementtiä. HTML5-standardi valmistuu kuitenkin aikataulun mukaan vasta vuonna 2022. (Canvas 2010; HTML5 2010.)

Microsoftin Internet Explorer -selain tukee Vector Markup Language (VML)-nimistä XML:n päälle rakennettua kuvauskieltä, jolla pystyy piirtämään vektorigrafiikkaa, sekä kiertämään ja skaalaamaan bittikarttoja. VML:n käyttö on jäänyt hyvin rajalliseksi, koska sitä ei tue mikään muu selain. (VML 2010.)

Google on kuitenkin kehittänyt exCanvas-ohjelmakirjaston, joka muuntaa Canvas-ohjelmakutsut VML:ksi. Näin ollen yhden JavaScript-koodirivin lisäyksellä verkkosivuille, sama Canvas-koodi saadaan toimimaan myös Internet Explorerissa. Tämä emulointi on kuitenkin hyvin hidasta ja monimutkaisissa animaatioissa suorituskyky jää helposti kymmeniä kertoja hitaammaksi. Niin kauan kuin Internet Explorer on eri versioissaan suosituin web-selain, JavaScript ei ole varteenotettava vaihtoehto interaktiivisen tai animoidun 3D-grafiikan tekoon web-sivuilla. Tilanne muuttuisi jos esimerkiksi Mozilla Firefox tai Google Chrome olisivat yleisimpiä selaimia. JavaScriptin suurin etu on toimivuus kaikissa selaimissa ilman mitään selainlaajennuksia. (Google 2009a.)

KUVA 4 *JavaScriptillä tehty räiskintäpelin demo*

2.7.5 WebGL

WebGL on Khronos Groupin kehitteillä oleva standardi, joka käyttää hyväksi HTML5:n Canvas-elementtiä ja OpenGL ES 2.0-standardia. WebGL mahdollistaa rautakiihdytetyn 3D-grafiikan näyttämisen verkkosivuilla. Rautakiihdytyksestä johtuen WebGL:n piirto on nopeaa ja se on hienon näköistä. (Khronos Group 2009; WebGL 2010.)

Standardin kehityksessä mukana olevat selaimet kuten Safari, Firefox ja Chrome tulevat luultavasti vakiona pitämään tämän tekniikan sisällään jo vuoden sisään. Ongelmana on kuitenkin edelleen Microsoftin haluttomuus tukea Canvas-elementtiä Internet Explorerissa, jolloin myöskään WebGL:n käyttö ei tule sillä onnistumaan. (WebGL 2010.)

Tällä hetkellä WebGL tekniikkaa voi kokeilla Firefoxilla, Safarilla ja Chromella lisäpalikan asentamisen jälkeen. (WebGL 2010.)

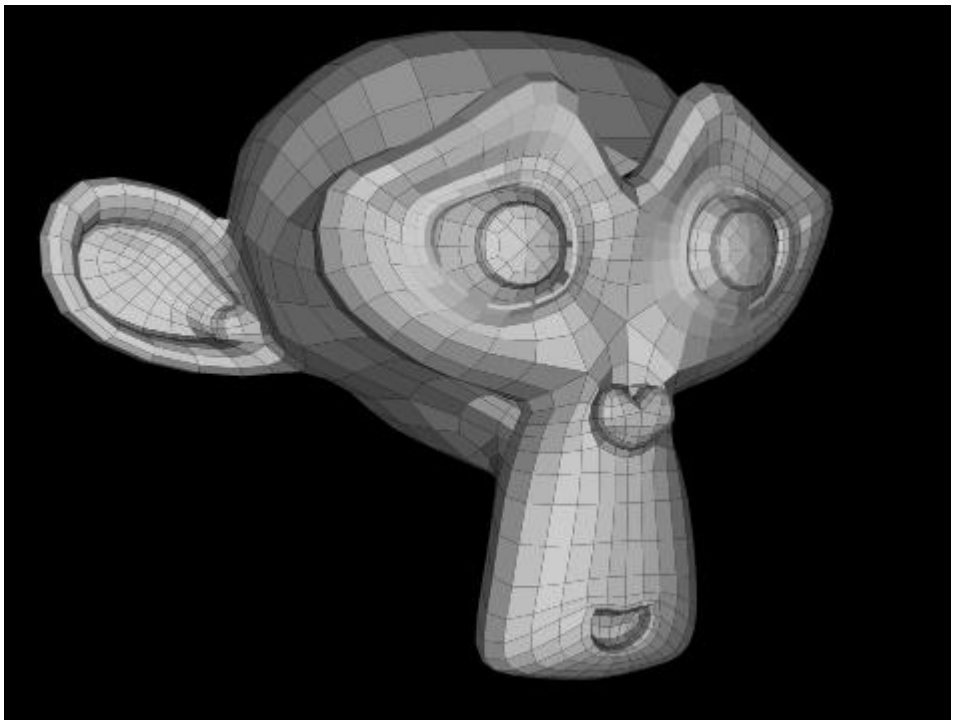
KUVA 5 *World of Warcraft -hahmomalli renderöitynä WebGL-tekniikalla*

2.7.6 Silverlight

Microsoft Silverlightin ensimmäinen virallinen versio julkaistiin syyskuussa 2007. Sen tarkoitus oli ryhtyä kilpailemaan Flashin kanssa verkkosivujen dynaamisesta sisällöstä. Silverlightiä voidaan ohjelmoida millä tahansa .NET-kielellä. Tämän ansiosta Silverlightille onkin maailmassa paljon potentiaalisia ohjelmoijia. (Silverlight 2010.)

Silvelight 3.0 versiossa mukaan tuli natiivi tuki perspektiivikorjatuille tekstuureille. Myös näytönohjaimen hyväksikäyttäminen tavallisissa toiminnoissa kuten bittikarttojen muokkauksissa ja läpinäkyvyyden laskemisessa, toi paljon parannusta suorituskykyyn edelliseen versioon nähden. (Hoffman 2009.)

Silverlightin heikoimpana puolena on kuitenkin sen huono levinneisyys. Silverlight on tällä hetkellä asennettu vain alle 40 % tietokoneista ja siitäkin vain murto-osalla on uusin versio. (Statowl 2009b.)

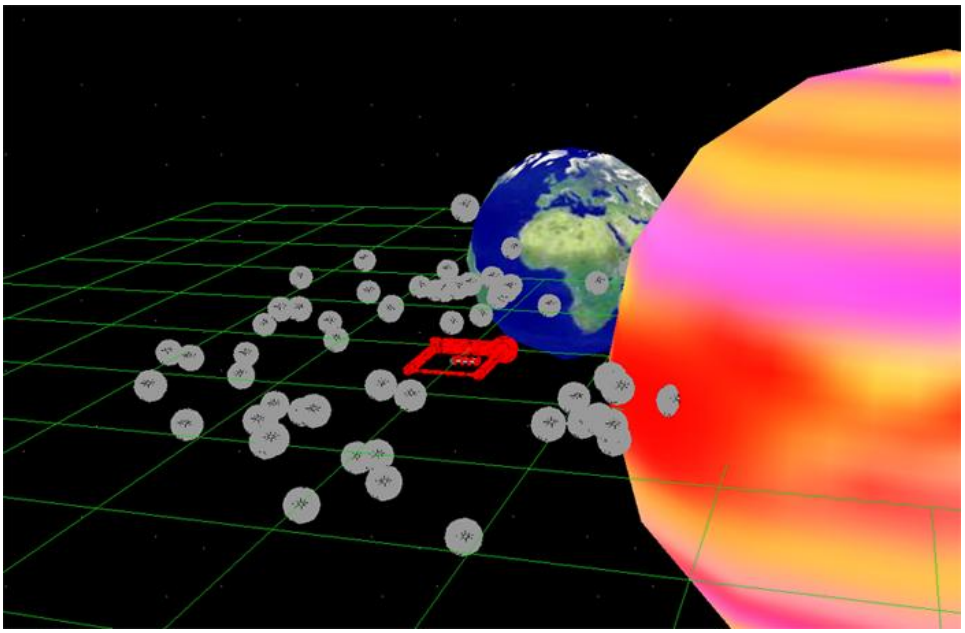


KUVA 6 *Silverlight-sovellus, jossa pyöritetään Blender-mallinnusohjelman Suzanne-primitiiviä*

2.7.7 Java

Java-ohjelmointikieli mahdollistaa kolmiulotteisen grafiikan piirtämisen Java Virtual Machine -selainlaajennuksen avulla. Javaan on olemassa useita korkean tason ohjelmointirajapintoja, jotka lisäävät 3D-grafiikan piirtämiseen tarvittavia ominaisuuksia. Laskentatehossa Java on erittäin nopea. (Java 3D 2010.)

Javan ongelmana ovat pitkät kehitysajat, suhteellisen huono levinneisyys, hidas käynnistyminen ja asennuspaketin suuri koko. Java Runtime Environment -paketin koko on minimissään 10 megatavua kun esimerkiksi Flash Player vie vain alle 2 megatavua. (Statowl 2009c; Sun Microsystems 2009.)



KUVA 7 *Java 3D -tekniikalla tehty sovellus*

2.7.8 Muut selainlaajennukset

Selainlaajennuksia, jotka ovat keskittyneet tuomaan nopeaa ja hienoa dynaamista 3D:tä selaimiin on monia. Näissä on kuitenkin paljon huonoja puolia, jotka estävät niiden kunnollisen hyödyntämisen tavallisilla verkkosivuilla. Kuitenkin erityistapauksissa näillekin tekniikoille on paikkansa. Esimerkiksi yrityksen sisällä toimivat sovellukset tai tuote-esittelyt asiakkaalle voivat olla hyviä kohteita.

Ensimmäinen ongelma, joka käyttäjälle tulee vastaan, on laajennuksen asentaminen. Sisällön tarvitsee olla käyttäjälle erittäin mielenkiintoista, ennen kuin hän haluaa asentaa laajennusta, josta ei tiedä mitään. Varsinkaan yrityskäyttäjät eivät usein edes voi asentaa koneelle mitään. Laajennukset voivat olla myös huonosti yhteensopivia vanhojen versioiden kanssa, jolloin käyttäjä ei edes pysty katsomaan haluamaansa sisältöä uudemmalla versiolla.

Kehittäjälle näissä on myös paljon huonoja puolia. Uusi tekniikka vaatii aina siihen perehtymisen, ennen kuin sillä saa aikaan laadukasta sisältöä. Monien tekniikkojen käyttö kaupallisiin tarkoituksiin on myös maksullista, mikä tuo ylimääräisiä kustannuksia. Uusien versioiden kehitys saattaa olla myös hidasta tai tekniikka voi kuolla jopa kokonaan ajan myötä. Nämä tekniikat usein myös rajoittavat kommunikointia verkkosivujen sisällä, kun laajennus ei pysty keskustelemaan muun sisällön kanssa. Näin interaktiivisuus voi jäädä vaisuksi. Näistä syistä johtuen harva kehittäjä jaksaa perehtyä kunnolla laajennuksiin, sillä hyötyä ei katsota tarpeeksi suureksi.

Tällä hetkellä yksikään dynaamiseen 3D:hen keskittyneistä selainlaajennuksista ei ole noussut ylitse muiden, mutta joukossa on kuitenkin joitain erittäin lupaavia sovelluksia. Maksullinen Unity3D on

lähinnä pelejä silmälläpitäen kehitetty 3D-moottori, mutta sitä voi hyödyntää myös muihin tarpeisiin. Unity3D hyödyntää näytönohjainta, sekä Direct3D 9:n ja OpenGL:n ominaisuuksia, joiden avulla sillä voi luoda erittäin näyttävää ja sulavaa dynaamista 3D-grafiikkaa. Unity3D:n editointiohjelman avulla ohjelmat voi myös suoraan kääntää toimimaan Applen iPhone:lla ja Nintendon Wii:llä. JavaScriptin avulla Unity3D-ohjelma pystyy keskustelemaan verkkosivujen muiden komponenttien kanssa, mikä avaa paljon mahdollisuuksia. Unity3D:tä kannattaa pitää silmällä, sillä se voi hyvinkin olla yksi niistä ohjelmista, joiden avulla tasokas 3D tulee verkkosivujen arkipäiväiseksi komponentiksi. (Unity 2009.)

KUVA 8 *Unity3D-sovellus, jossa tutkitaan trooppista saarta*

2.8 Papervisionin ja 3D:n tulevaisuus internetissä

Seuraavassa kerron oman näkemyksen Papervisionin ja yleensäkin 3D-grafiikan tulevaisuudesta verkkosivuilla. Näkemys perustuu minun omiin kokemuksiini ja keskusteluihin muiden kehittäjien kanssa.

Papervision-tiimi on paraikaa kehittämässä uutta Papervisionin versiota nimeltä Papervision 3.0, jota kutsuttiin aikaisemmin myös nimellä PapervisionX. Uuden version arkkitehtuuri kirjoitetaan suurelta osalta uudestaan, jotta siitä saadaan entistä joustavampi. Tämä mahdollistaa uusien Flash Player -ominaisuuksien lisäämisen nopeasti ja järkevästi. Papervision 3.0 tulee hyödyntämään Flash Player 10:n natiiveja 3D-ominaisuuksia, joiden ansiosta uusi versio tulee toimimaan nopeammin. Kaikista uudistuksista huolimatta Papervision 3.0:ssä pyritään säilyttämään mahdollisimman paljon Papervisionin edellisen version ohjelmointirajapintaa, jolloin aikaisemmat kehittäjät pääsevät uuteen versioon nopeasti mukaan. Papervision 3.0 on ollut kehitteillä jo jonkin

aikaa, mutta sen julkaisuaikataulusta ei ole vielä kerrottu mitään. Itse kuitenkin uskon, että se tullaan julkaisemaan vuoden 2010 ensimmäisen puoliskon aikana. Jos julkaisu venyy liian pitkään saattaa osa kehittäjistä siirtyä muihin tekniikoihin, koska ne tarjoavat parempia ominaisuuksia.

On itsestään selvää, että tietokoneiden prosessorien, sekä näytönohjaimien tehot tulevat kasvamaan tulevat kasvamaan koko ajan. Tämä tulee lisäämään 3D-grafiikan näyttävyyttä ja mahdollistaa myös uusia käyttötarkoituksia.

Flashin valta-asema tulee todennäköisesti jatkumaan vielä pitkälle tulevaisuuteen. Tästä on merkkinä mm. Microsoft Silverlightin takkuisa alkutaival. Ihmisillä täytyy olla paljon hyviä syitä ennen kuin he asentavat uuden selainlaajennuksen. Silverlight ja muut tekniikat tuovat kuitenkin alalle lisää kilpailua, joka pakottaa Adobea kehittämään Flashin 3D-ominaisuuksia entistä enemmän ja nopeammalla aikataululla. Tulevien Flash Playerien ominaisuudet tekevätkin joitain Papervisionin kaltaisia tekniikoita tarpeettomiksi, sillä Flashin natiivit tekniikat toimivat nopeammin, eikä Flash-tiedostoihin tarvitse lisätä ylimääräistä lähdekoodia 3D-moottorille. Flash Playerin uudet ominaisuudet toisaalta parantavat niitä 3D-moottoreita, jotka pystyvät pysymään kehityksen mukana.

On vaikea ennustaa mitkä tekniikat tulevat nousemaan valta-asemaan tulevaisuudessa, mutta se on kuitenkin varmaa, että 3D-grafiikan käyttö tulee lisääntymään nettisivuilla.

3. 3D-GRAFIIKAN TEORIAA

3D-grafiikan perusteet eivät ole käytännössä muuttuneet vuosien varrella ollenkaan. Tämän takia perusteita on myös hyvä osata jossain määrin, sillä vaikka tekniikat kehittyvät ja vaihtuvat, teoria pysyy samanlaisena. Teoriasta on myös paljon hyötyä Papervision-sovelluksia kehittäessä. Erilaiset ongelmatilanteet ja sovelluksen optimointi onnistuvat paljon helpommin, mikäli ymmärtää miten tekniikka toimii. Tässä luvussa kerrotaan tietokone- ja 3D-grafiikan perusteita, jotka olisi ainakin hyvä tietää ennen syvällistä tutustumista Papervisioniin. Jos teoriaan haluaa paneutua vielä syvemmin, on asiasta saatavilla useita hyviä kirjoja. Suomenkielistä materiaalia luettaessa on kuitenkin hyvä tietää, että termeille voi löytyä useita eri suomennoksia riippuen kirjoittajasta. Tämän takia on hyvä osata myös vakiintuneet englanninkieliset termit, joiden avulla on helppo tietää mistä asiasta on oikeasti kyse.

Mikäli 3D-grafiikan teoriaan haluaa tutustua tarkemmin, suosittelen lukemaan seuraavia kirjoja.

Foley, van Dam, Feiner, Hughes:
Computer Graphics: Principles and Practice in C (2nd Edition)
Addison-Wesley Professional 1995, ISBN: 0201848406

Dunn, Parberry
3D Math Primer for Graphics and Game Development
Jones & Bartlett Publishers 2002, ISBN: 1556229119

Puhakka
3D-grafiikka
Talentum Media Oy 2008, ISBN: 9521411929

3.1 Renderöinti

Kolmiulotteinen malli koostuu vektoridatasta ja pintamateriaaleista. Mallista halutaan usein projisoida kaksiulotteinen kuva tietokoneen näytölle, jolloin mallista peräisin oleva grafiikkatieto käy läpi 3D-piirtoliukuhihnan eri vaiheet. Tätä 3D-grafiikan piirtoa kutsutaan myös renderöinniksi. 3D-piirtoliukuhihnan vaiheet riippuvat käytetystä tekniikasta. 3D-grafiikkaa voidaan laskea käyttötarkoituksesta riippuen, joko reaaliaikaisesti tai ei-reaaliaikaisesti. (Puhakka 2008, 163–164.)

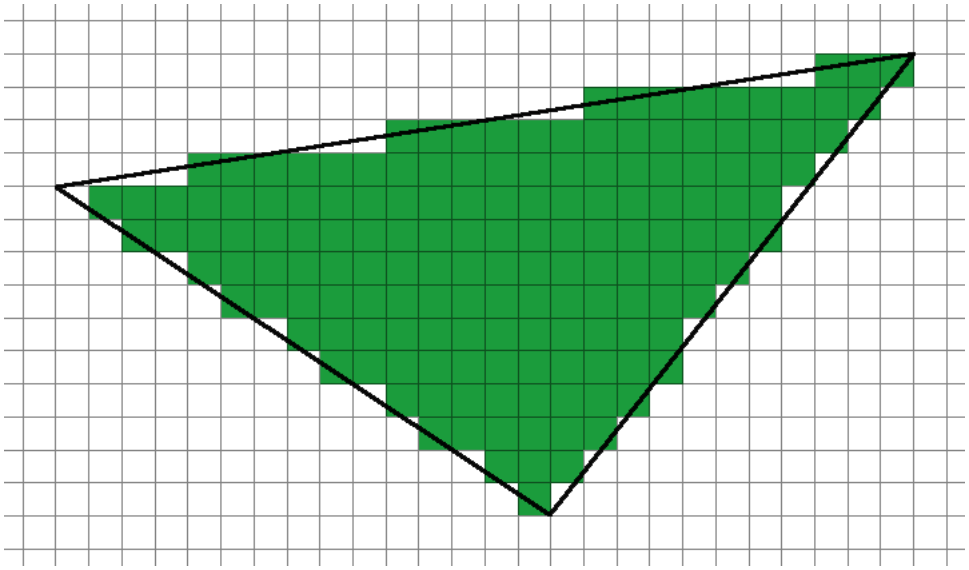
Ei-reaaliaikaisessa laskennassa 3D-grafiikan laatu on parempaa. 3D-mallit ja tekstuurit voivat olla paljon tarkempia kuin reaaliaikaisessa. Laskennassa voidaan käyttää avuksi myös raskaita tekniikoita kuten säteenjäljitystä, jolloin esimerkiksi valaistus pystytään laskemaan tarkemmin ja heijastukset saadaan piirtymään matemaattisesti oikein. Monimutkaisen ei-reaaliaikaisen 3D-grafiikan yhden kuvan laskemiseen saattaa kuitenkin kulua useita tunteja. Samaa kuvaa voidaan laskea useammalla prosessorin ytimellä samaan aikaan, jolloin laskeminen nopeutuu. Pitkissä animaatioissa käytetään usein avuksi useita koneita eri ruutujen laskemiseen, joiden avulla animaation laskenta-aika saadaan inhimilliselle tasolle. Ei-reaaliaikaista 3D-grafiikkaa käytetään mainoksissa, elokuvissa ja animaatioissa. (3D rendering 2010.)

Reaaliaikaisessa 3D:ssä tärkeää on ruudunpäivityksen nopeus, jolloin renderöitäessä kohtauksen tulee olla yksinkertaisempi. Kaikkein raskaimpia asioita kuten valaistusta ja heijastusta laskettaessa lopputulos ei ole useinkaan kovin realistinen vaan sen on tarkoitus näyttää hyvältä mahdollisimman vähällä laskennalla. Laskennassa käytetään myös paljon muitakin ”huijauksia” ja vaihtoehtoisia tapoja suorituskyvyn parantamiseksi. Nykyään useimmat sovellukset käyttävät 3D-grafiikan laskemisen nopeuttamiseksi näytönohjainta. Näytönohjainta voidaan hyödyntää eri grafiikkarajapintojen avulla, joista yleisimmät ovat Direct3D ja OpenGL. Flash ei osaa tällä hetkellä hyödyntää kumpaakaan rajapintaa, jonka takia sillä tehdyn 3D-grafiikan taso jää pahasti jälkeen nykyisestä 3D-grafiikan tasosta. Reaaliaikaista 3D-grafiikkaa hyödynnetään esimerkiksi peleissä, visualisoinnissa, sekä interaktiivisissa sovelluksissa. (3D rendering 2010.)

3.2 Rasterointi

Tietokonegrafiikassa rasterointi tarkoittaa yleisesti mitä tahansa prosessia, jossa vektori-informaatio muutetaan rasteri-formaattiin, kuten esimerkiksi bittikarttakuvaksi. (Rasterisation 2010.)

3D-grafiikassa rasteroinnilla tarkoitetaan yleensä 3D-piirtoliukuhihnan rasterointivaihetta. Rasterointivaiheen tehtävänä piirtää annetusta 3D-informaatiosta kuva kaksiulotteiseen rasteriruudukkoon, eli tietokoneen näytölle, täyttämällä ruudukon pikselit tietyillä väriarvoilla. Tässä vaiheessa pikseleille tehdään operaatioita, kuten Z-testi, teksturointi, sekä pikselin värin yhdistäminen vanhaan väriin. Rasterointi ei ole ainoa tapa renderöidä 3D-grafiikkaa, mutta se on nopea verrattuna esimerkiksi säteenseurantaan. Nopeutensa vuoksi rasterointi on tekniikka, jota käytetään yleensä reaaliaikaisen 3D-grafiikan renderöimiseen. (Rasterisation 2010; Puhakka 2008, 201.)



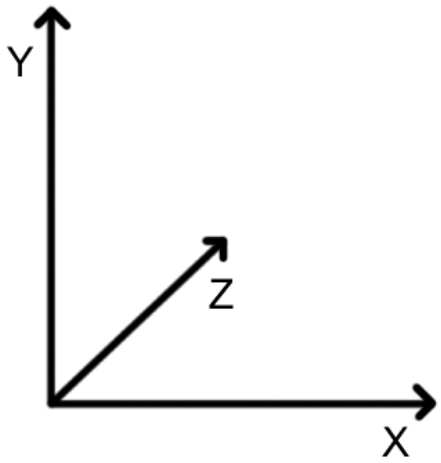
KUVA 9 *Monikulmio rasteroituna pikseliruudukkoon*

Tällä hetkellä Flashin suorituskyky ei ole kuitenkaan tarpeeksi hyvä, jotta esimerkiksi Papervision voisi renderöidä 3D-grafiikkaa perinteisellä rasterointimenetelmällä pikseli pikseliltä. Sen sijaan Papervision käyttää piirtämiseen ja polygonien täyttämiseen Flashin omia piirtokomentoja, joissa piirretään isompia alueita kerralla. Tämän vuoksi emme tiedä yksittäisten pikselien tietoja, kuten esimerkiksi syvyyttä, joka saa aikaan joitain ongelmia ja vähentää käytettävissä olevia tekniikoita. Monet näistä ongelmista ja puutteista on kuitenkin kierrettävissä muilla keinoin.

Pikseli pikseliltä renderöiviä 3D-moottoreita on kuitenkin jo olemassa Flashiin, mutta tässä vaiheessa niillä saavutetaan niin vähän lisäominaisuuksia, jonka vuoksi niiden käyttö on erittäin vähäistä. Näiden suosio tulee luultavasti kasvamaan, kunhan Flashin suorituskyky paranee. Tämä tarkoittaa luultavasti sitä, että myös Papervision tulee tulevaisuudessa renderöimään 3D-grafiikkaa vastaavalla tavalla.

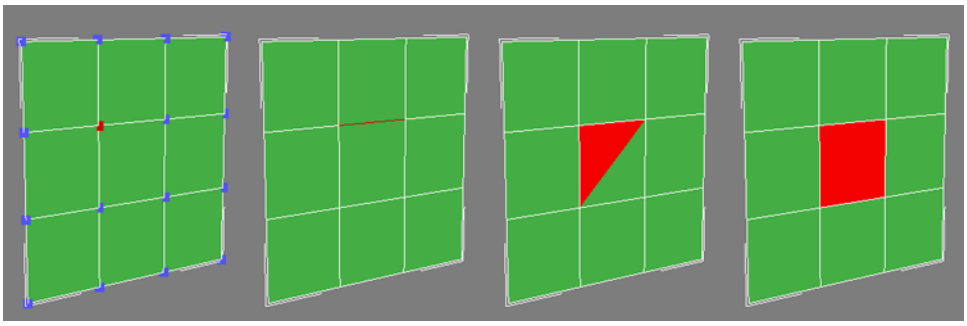
3.3 3D-geometria

Papervision käyttää vakiona ns. vasemman käden koordinaatistoa, jossa y-akselin positiivinen suunta on ylöspäin, x-akselin positiivinen suunta on oikealle ja z-akselin positiivinen suunta on poispäin katsojasta. Papervisionin pääluokasta voi vaihtaa useRIGHTHANDED-muuttujan arvon todeksi, jolloin z-akselin positiivinen suunta vaihtuu. Matematiikassa ja monissa 3D-mallinnusohjelmissa on käytössä oikean käden koordinaatisto. On makuasia kummasta pitää enemmän, sillä kummallakaan ei saavuteta hyötyä toiseen nähden. Parasta olisi jos aina käytettäisiin samaa järjestelmää. (Papervision 2010.)



KUVA 10 *Kuvassa on esitetty vasemman käden koordinaatisto*

3D-mallin geometria muodostuu erilaisista peruselementeistä. Pienin peruselementti on kärkipiste (vertex). Kolmen kärkipisteen avulla saadaan aikaiseksi pinta (face). Pinnan rajaavia yhdistettyjen kärkipisteiden välejä kutsutaan reunoiksi (edge). Monikulmio (polygon) koostuu yhdestä tai useammasta pinnasta. Papervisionissa monikulmiot koostuvat aina vain yhdestä pinnasta ja ovat näin kolmion muotoisia. 3D-mallin koko geometriaa kutsutaan monikulmioverkoksi (mesh). Papervisionissa 3D-mallin geometria on tallennettu Vertex3D-instansseihin, jotka ovat varastoitu GeometryObject3D-instanssiin. (Polygon mesh 2010.)



KUVA 11 *Vasemmalta oikealle kuvassa on värjätty punaisella kärkipiste (vertex), reuna (edge), pinta (face) ja monikulmio (polygon)*

Suorituskyvyn parantamiseksi 3D-mallin pinnoilla on näkyvä ja piilotettu puoli. Katsottaessa pintaa väärältä puolelta sitä ei siis täytetä. Pinnan oikea

puoli katsotaan pinnan normaalivektorista. Jos normaali on kameraan päin, pinta täytetään. Pinta on mahdollista määrittää myös kaksiopuoleiseksi, mutta tätä ei ole syytä tehdä, ellei sitä oikeasti tarvitse.

3D-mallia renderöitäessä monikulmiot on lajiteltava syvyysjärjestykseen. Papervisionissa on käytössä vakiona maalarin algoritmi (Painter's algorithm), jossa monikulmion etäisyys kameraan lasketaan sen keskipisteestä. Tämän jälkeen monikulmiot piirretään toinen toistensa päälle, aloittaen kauimmaisesta monikulmiosta. Algoritmin hyötynä on sen nopeus, mutta haittapuolena on se, että jos monikulmion pituus on suurempi kuin matka viereiseen monikulmioon, saattaa monikulmioiden syvyysjärjestys ruudulla olla väärä. 3D-grafiikassa monikulmioiden syvyyslajitteluun on olemassa monia muitakin tapoja, mutta useimmat niistä käyttävät hyödyksi näytönohjaimen ominaisuuksia, jonka vuoksi niiden käyttö Papervisionissa olisi liian raskasta pelkän prosessorin avulla. (Puhakka 2008, 268–269.)

Papervisionissa kappaleiden pituusmittayksikkönä käytetään Papervision-yksiköitä. Jos 100 × 100 Papervision-yksikön kokoisen taso-primitiivin haluaa ruudulla olevan 100 × 100 pikseliä, täytyy kappale asettaa kamerasta kaavan 1 mukaisesti.

$$taso.z = kamera.z - kamera.zoom * kamera.focus$$

KAAVA 1 *Tason asettaminen niin, että sen pituusyksiköt vastaavat ruudun pikseleitä*

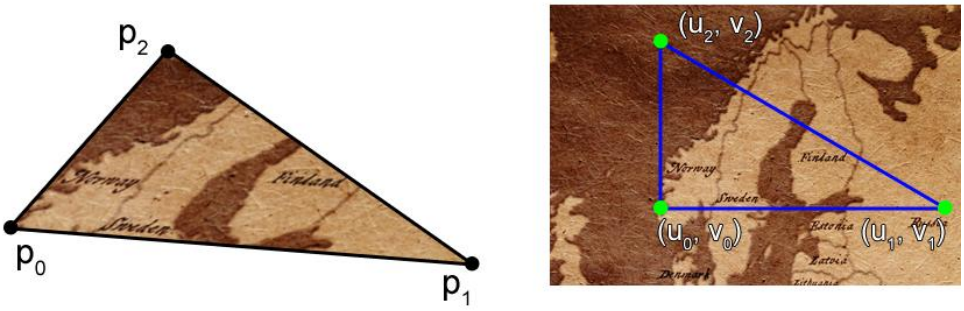
Kaavassa 1 oletetaan, että kamera katsoo suoraan primitiiviä z-akselin suuntaisesti. (Drozdz 2008a.)

3.4 Teksturointi

Teksturointi on tekniikka, jolla 3D-mallin pintojen täytössä käytetään hyväksi bittikarttaa. Teksturoinnilla 3D-malliin saa helposti lisää näyttävyyttä ja todenmukaisuutta. 3D-mallia teksturoitaessa on myös mahdollista käyttää useita bittikarttoja eri pinnoille. Papervisionissa tekstuurina on mahdollista käyttää bittikartan asemasta myös animoituja MovieClippejä, sekä videokuvaa. Nämä vaihtoehdot avaavat monia mahdollisuuksia sovelluksia kehitettäessä. (Puhakka 2008, 206.)

Tekstuurina käytetyn kaksiulotteisen bittikartan pikseleitä kutsutaan tekseleiksi. Rasterointi-tekniikalla täytettyjen pintojen rasteripisteille etsitään sitä vastaava tekseli tekstuurikuvasta. Pintojen kärkipisteisiin varastoidut tekstuurikuvan koordinaatit esitetään u- ja v-koordinaatteina, jotka ovat liukulukuja väliltä 0…1. Nämä liukuluvut kerrotaan tekstuurikuvan leveydellä ja korkeudella, jolloin saadaan selville koordinaattien lopulliset paikat tekstuurikuvassa. Koska u- ja v-koordinaatit esitetään liukuluvuilla, on tekstuuri helppo vaihtaa eri tarkkuuteen ilman, että koordinaatteja tarvitsee määrittää uudestaan. Papervisionissa primitiiveille on määritetty valmiiksi sopivat tekstuurikoordinaatit. Muille 3D-malleille tekstuurikoordinaatit on määritettävä 3D-mallinnusohjelmassa. Jos 3D-mallin aikoo esimerkiksi

täyttää yhdellä värillä varjostimen kanssa, ei tekstuurikoordinaattien määrittämiseen kannata tuhlata aikaa, sillä silloin niitä ei tarvita. (Puhakka 2008, 206.)



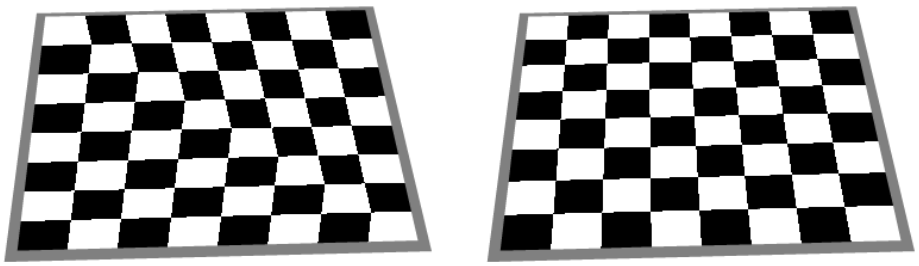
KUVA 12 *Monikulmion teksturointi*

KUVA 13 *Vasemmalla on teksturoitu 3D-malli. Oikealla on tekstuurina käytetty bittikartta*

Tekstuurikoordinaatteina voidaan käyttää myös välin 0…1 ulkopuolella olevia lukuja. Tällöin koordinaattien tulkitsemiseen on kolme yleistä vaihtoehtoa. Yhdessä tavassa tekstuurialueen ulkopuolelle menevät pisteet väritetään tekstuurikuvan reunalla olevien tekseleiden värillä. Toisessa tavassa pisteet väritetään yhdellä värillä. Tämä on Papervisionin vakiotapa. Kolmas ja yleisimmin käytetty tapa on, että rajan ylittävät tekstuurikoordinaatit palautuvat alkuun, eli tekstuurikoordinaateista käytetään vain desimaaliosaa. Tätä tekniikkaa kutsutaan nimellä tiling. Tekniikalla on mahdollista teksturoida suuriakin pintoja käyttäen pientä tekstuurikuvaa. Tekniikkaa käytetään paljon toistuvissa pinnoissa kuten kaakeli- tai tiilikuvioisissa tekstuureissa. Papervisionissa MaterialObject3D-luokan maxU- ja maxV-arvot kertovat kuinka monta kertaa tekstuuri toistetaan vaaka- ja pystysuunnassa määritetyssä pinnassa. (Puhakka 2008, 206; Papervision 2010.)

Kuten aikaisemmin tuli jo mainittua, Papervision ei renderöi monikulmioita pikseli pikseliltä. Papervision täyttää yhden tekstuuritäytetyn monikulmion kokonaan beginBitmapFill-funktiolla, jonka kanssa on myös mahdollista käyttää transformaatiomatriisia. Bittikarttaa skew-säädetään, skaalataan ja pyöritetään niin, että tekstuurin

UV-koordinaatit vastaavat monikulmion kärkipisteitä. Tämän tekniikan haittapuolena on, että vinosti kameraan sijaitsevissa tekstuureissa esiintyy vääristymää. Vääristymä tulee selkeästi esiin esimerkiksi shakki-kuviolla täytetyissä monikulmioissa. Vääristymää voidaan korjata jakamalla geometriaa pienempiin osiin, mutta tällöinkään ongelma ei katoa kokonaan. Ongelma tulee esiin myös rasteroimalla renderöidyissä monikulmioissa, mutta koska pikselit täytetään yksitellen, voidaan ongelma korjata. Vääristymä tulee tarkkaan ottaen siitä, että vaikka monikulmion kärkipisteet olisivat eri etäisyyksissä kameraan nähden, teksturoidaan monikulmio samalla tavalla kuin ne olisivat samassa tasossa. Jos syvyys otetaan huomioon, ongelma katoaa. Tätä kutsutaan perspektiivikorjatuksi tekstuurikuvaukseksi. Tämä korjaus on kuitenkin raskas laskea monimutkaisissa 3D-malleissa ja yleensä sen tekeekin näytönohjain. Jos tulevaisuudessa yksittäisen pikselin piirto nopeutuu Flashissä tarpeeksi ja näytönohjainta hyödynnetään paremmin, saattaa Papervisionkin kyetä tähän. (Puhakka 2008, 207–208; Adobe 2008a.)



KUVA 14 *Vasemmanpuoleisessa tekstuurissa ei ole käytetty perspektiivikorjausta, oikeanpuoleisessa tekstuurissa on*

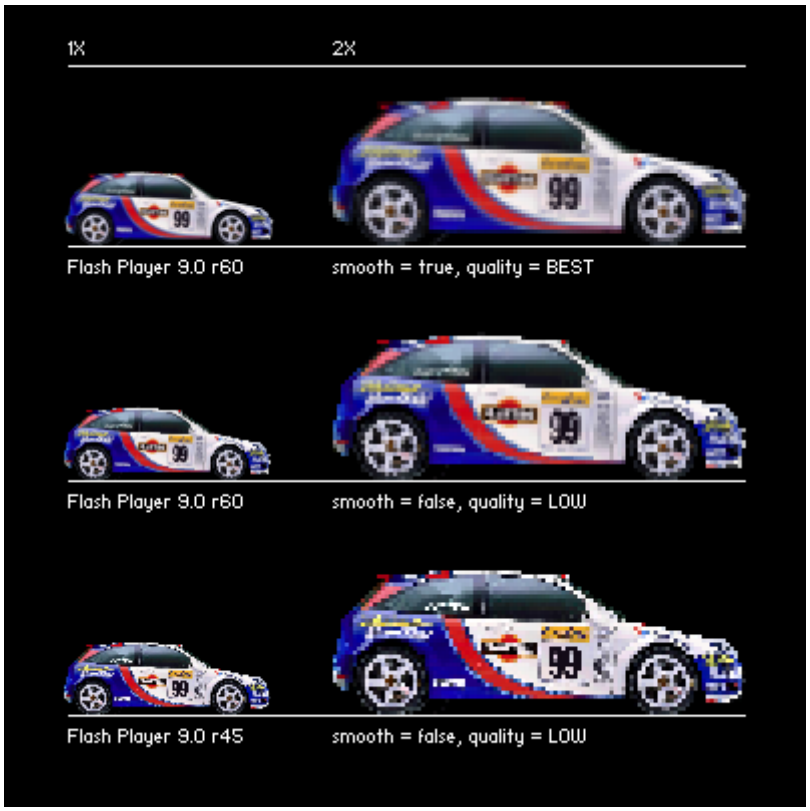
Teksturoinnissa yksi tärkeimmistä asioista, joka tulee ottaa huomioon, on käytettyjen tekstuurien koko. Suuret tekstuurikartat kasvattavat sovelluksen kokoa, varaavat enemmän muistia ja piirtyvät hitaammin ruudulle. Tekstuurien väärä koko tuo mukanaan myös muita onglemia. Nopeimmalla tavalla teksturoidessa, jossa jokainen rasteripiste täytetään tekstuurikoordinaattia lähinnä vastaavalla tekselillä, tulee esiintymään tietyissä tilanteissa pikselien välkkymistä.

Jos tekstuuri on liian iso tai teksturoitu kappale on kaukana yksi kappaleen pikseli vastaa useita tekstuurikuvan tekseleitä. Tällöin kappaleen liikkuessa siihen kohtaan piirretyn pikselin väri vaihtelee ja saa aikaan pikselien välkkymistä. Tätä ongelmaa kutsutaan nimellä minification. Tämän ongelman vastakohtaa kutsutaan nimellä magnification. Ilmiö syntyy kun kappale on lähellä ruutua tai tekstuuri on liian pieni. Tällöin usea kappaleen pikseli vastaa samaa tekseliä ja kappaleen pinnalla näkyy pikselöitymistä. Molempia mainittuja ongelmia voi välttää käyttämällä sopivan kokoisia tekstuureita, mutta aina se ei ole mahdollista, jos esimerkiksi kappaleen etäisyys kamerasta vaihtelee paljon. Tällöin voi hyödyntää tekstuurikuvan suodatusta. Bilineaarisessa suodatuksessa ruudun pikselin väri määräytyy tekstuurikoordinaattia lähinnä olevien neljän pikselin keskiarvosta. Suodatus sumentaa tekstuuria hieman, mutta se on tällöin läheltäkin katsottuna siedettävän näköinen. Papervisionissa bilineaarisen suodatuksen saa päälle muuttamalla materiaalin smooth-

arvon todeksi. Suodattimen käyttöä kannattaa kuitenkin välttää, jos sitä ei tarvitse, sillä se lisää renderöintiaikaa. (Puhakka 2008, 207–210; Adobe 2008a.)

KUVA 15 *Teksturoitu malli läheltä katsottuna. Vasemmalla ilman pehmennystä ja oikealla pehmennyksen kanssa*

Flash Player 9 kehityksen aikana Flashiin lisättiin tuki MIP-kartan käytölle. Kyseisen tekniikan lisääminen toi hyötyä etenkin Papervisioniin. MIP-karttaa käytetään tekstuurien esisuodatuksessa. MIP-tekniikassa alkuperäisestä tekstuurista tallennetaan tietokoneen muistiin useita neljäsosaan edellisestä kuvasta pienennettyjä kuvia aina yhteen pikseliin asti. Kun teksturoitu kappale pienenee tarpeeksi ruudulla, vaihdetaan tekstuurikuvaa pienemmäksi. Lopputuloksena kaukaisissa kappaleissa tekstuurin laatu paranee ja suorituskyky kasvaa. (Puhakka 2008, 211–212; Uro 2007a.)



KUVA 16 *Kuvassa näkyy MIP-tekniikan vaikutus. Auto on teksturoitu 3D-malli. Alimmassa kuvassa ei ole käytössä MIP-tekniikkaa*

MIP-tekniikan ongelmana on, että katsoja huomaa tekstuurin vaihtumisen rajaetäisyyden kohdalla. Tämän ongelman korjaamiseksi voidaan teksturoitaessa käyttää hyväksi kahta lähintä MIP-kartan kuvaa. Molemmista kuvista tekselit haetaan bilineaarisella suodatuksella, jonka vuoksi tätä tekniikkaa kutsutaankin trilineaariseksi suodatukseksi. Flash ei kuitenkaan ainakaan vielä hyödynnä kyseistä tekniikkaa. (Puhakka 2008, 211–212.)

3.5 Varjostimet

Varjostimet tuovat tasaväriseen kappaleeseen eri valaistuksen sävyjä. Varjostimet käyttävät hyödyksi valon lähdettä. Valaistustekniikoita on useita erilaisia, näistä yleisimmät ovat pistemäinen- ja suuntavalo. Eri varjostimet hyödyntävät eri tekniikoita ja ne on tarkoitettu erityyppisten pintojen luontiin. Toisilla saatetaan tavoitella realistista ilmaisutapaa, toisilla on muita tarkoituksia. Tietyillä varjostimilla kappaleiden kulmikkuutta on myös mahdollista pehmentää, jolloin kappaleissa ei tarvitse käyttää niin paljon monikulmioita. (Puhakka 2008, 231.)

Varjostimet käyttävät monesti useita eri valaistusmalleja kappaleen pinnan valoerojen määritykseen. Papervisionissa näitä ovat taustavalo, diffuusi heijastuminen, spekulaarit heijastumiset ja kuhmutus.

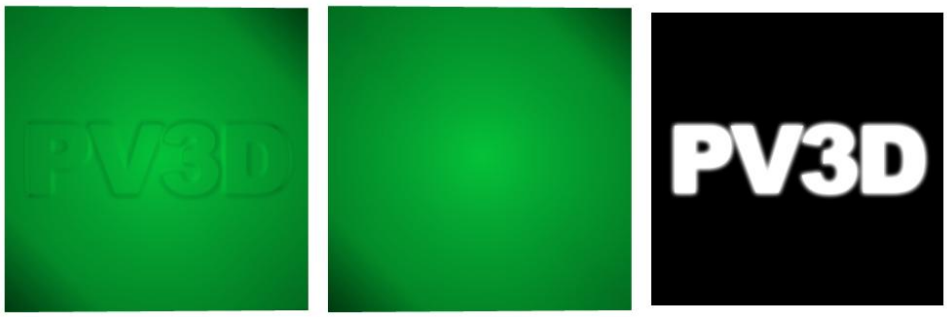
Useimmissa varjostimissa käytetään hyväksi ambientti- eli taustavaloa. Jos kappaleessa käytettäisiin pelkkää taustavaloa, se olisi täysin yksivärinen, ilman varjoisia tai valoisia kohtia. Taustavaloa kuitenkin käytetään, koska kappaleiden pimeimmät kohdat ovat harvoin täysin pimeitä. Pimeimpiinkin kohtiin valoa heijastuu toisten kappaleiden ja tasojen pinnoista. Tasaisen taustavalon määritteleminen kappaleeseen on helppo ja kevyt tapa piirtää kappaleeseen tulevan taustavalon arvioitu määrä. (Puhakka 2008, 233.)

Diffuusi heijastuminen tarkoittaa sitä, että kappaleen pinnan valoisuus tietyssä pisteessä on sama riippumatta mistä suunnasta sitä katsoo. Tällainen valaisumalli toteutuu lähinten sumeissa ja mattamaisissa pinnoissa, kuten kivessä tai paperissa. Diffuusia heijastumista käytetään kuitenkin yhtenä komponenttina monen muunkin tyyppisessä kappaleessa. (Puhakka 2008, 231–232.)

Spekulaareja heijastumisia käytetään kappaleissa, joiden pinta on kiiltävää. Toisin kuin diffuusissa heijastumisessa, kiillon paikka vaihtelee jos katsoja liikkuu. Spekulaareilla heijastumisilla ei kuitenkaan mallinneta peilipintaa, jossa valonlähde ja muut kappaleet näkyisivät terävinä. Kappaleen heijastuskohdat ovat sen sijaan sumentuneita ja vääristyneitä kuvia valonlähteistä. Tätä valaistusmallia käytetään esimerkiksi muovissa, lasissa ja metallissa. Papervisionissa tätä tekniikkaa ei käytetä vielä kunnolla, mutta tätä efektiä voi jäljitellä liikuttamalla valonlähdettä aina kun kameraa liikutetaan. (Puhakka 2008, 237–238.)

KUVA 17 *Vasemmalla pallossa on käytetty täyttämiseen pelkkää taustavaloa, keskellä palloon on lisätty diffuusia heijastumista ja oikealla näiden lisäksi on käytetty vielä spekulaaria heijastusta*

Kuhmutus eli bump mapping tarkoittaa tekniikkaa, jolla kappaleen pinta saadaan näyttämään epätasaiselta lisäämättä siihen yhtään geometriaa. Kappaleeseen lisätään mustavaloinen bittikarttakuva, jonka eri sävyt kuvaavat korkeuseroja. Pinnan pisteisiin osuva valo reagoi korkeuseroihin saaden pinnan näyttämään epätasaiselta. Tällä tekniikalla on helppo tehdä kappaleen pinnasta röpelöinen tai lisätä siihen esimerkiksi painettu teksti, ilman että geometriaa tarvitsee muuttaa. (Puhakka 2008, 248–249.)



KUVA 18 *Vasemmalla näkyy taso-primitiivi, johon on lisätty kuhmutus. Keskellä näkyy taso ilman kuhmutusta ja oikealla kuhmutuksessa käytetty bittikartta*

Valaistusmallien käyttö tekstuurien kanssa saa 3D-mallista paljon realistisemman näköisen. Valaistusmallien kanssa käytetään monesti mustavalkoisia bittikarttoja, jotka toimivat maskeina. Tällä tavalla esimerkiksi tekstuurin metalliosiin saa lisättyä kiiltoa, muiden kohtien jäädessä mattapintaisiksi. Papervisionissa maskien käyttö on vielä hyvin rajallista, mutta tulevaisuudessa niitäkin hyödynnetään luultavasti enemmän. (Papervision 2010.)

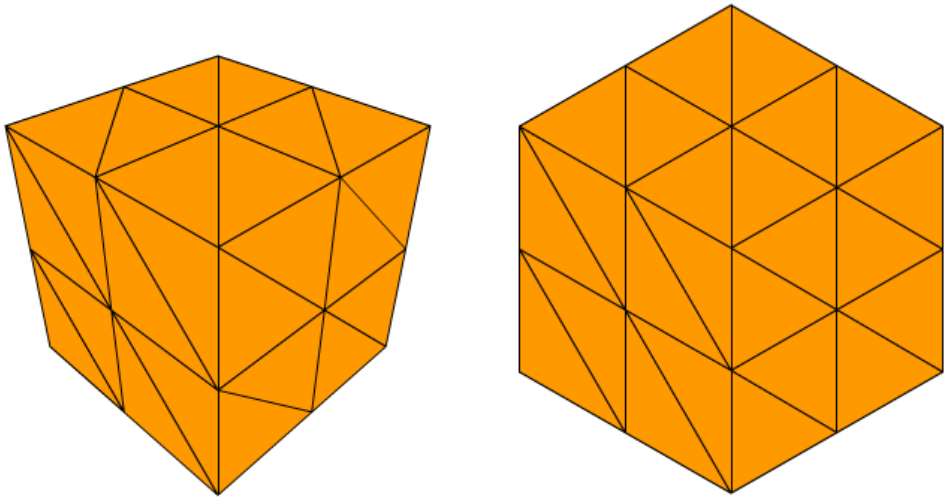
3.6 3D-projektiot

Kaksiulotteisen kuvan saamiseksi ruudulle, ovat kolmiulotteiset kappaleet projisoitava kaksiulotteiselle pinnalle. 3D-grafiikassa voidaan käyttää kahta erilaista projektiotekniikkaa, jotka ovat paralleeliprojektio ja perspektiiviprojektio. (Puhakka 2008, 177.)

Paralleeliprojektiolla toteutetussa kuvassa ei ole lainkaan perspektiiviä. Tämä tarkoittaa sitä, että kaksi eri syvyydessä olevaa samankokoista kappaletta ovat myös ruudulla samankokoisia. Paralleeliprojektio jakautuu

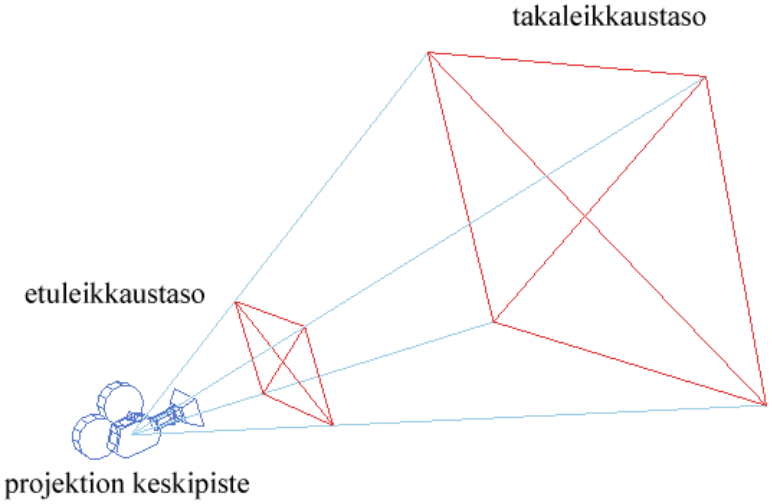
useisiin alaluokkiin. Näistä tunnetuin ja eniten käytetty on ortograafinen projektio. Papervisionissa voi halutessaan käyttää ortograafista projektiota. Ortograafista projektiota käytetään usein arkkitehtuurissa pohjapiirroksissa ja konesuunnittelussa poikkileikkauksissa. (Puhakka 2008, 177–179.)

Perspektiiviprojektio on realistisempi ja paljon yleisemmin käytetty tekniikka. Tekniikalla tuotetussa kuvassa kappaleet pienenevät kun ne loittonevat kamerasta. Kappaleen samansuuntaiset suorat ovat kuvassa vinoja ja niiden kuvitellut jatkeet kohtaavat toisensa tietyssä pisteessä. Kuvataiteessa tätä kutsutaan katoamispisteeksi. Perspektiiviprojektiolla tuotetusta kuvasta on kuitenkin mahdollista tehdä ortograafinen, jos kameraa viedään kohteesta äärettömän kauas ja kameraa tarkennetaan äärettömyyteen. Papervisionissa kamera-luokalla on ominaisuuksia, jotka vaikuttavat kuvan perspektiiviin. (Puhakka 2008, 183–187.)



KUVA 19 *Vasemmalla on kuution kuva perspektiiviprojektiossa. Oikealla kuutio on isometrisessä ortograafisessa projektiossa*

3D-projektiossa syntyvä kuva on periaatteessa äärettömän kokoinen, mutta luonnollisesti kuvasta halutaan rajata vain pieni suorakulmainen ikkuna, joka näytetään katsojalle. Ilman minkäänlaista rajausta kuvaan projisoituisi myös hyvin kaukana olevat kohteet, joiden merkitys kuvaan on vähäinen. Tämän takia näkökenttää rajataan takaleikkaustasolla (far clipping plane), joka estää sen takana olevien kohteiden piirtymisen kuvaan. Näkökenttää rajataan myös toisesta päästä etuleikkaustasolla (near clipping plane). Ilman etuleikkaustasoa myös katsojan takana olevat kohteet piirtyisivät kuvaan. Etuleikkaustasoa käytetään usein myös suoraan kuvaustasona, johon itse projisoitu kuva piirtyy. Leikkaustasojen rajaamaa halkaistua pyramidia kutsutaan näköfrustumiksi. Kameran näkökenttä määrää näköfrustumin jyrkkyyden. Rajatun alueen sisälle jäävää osaa kutsutaan näkö- tai kuvausvolyymiksi (view volume). Paralleeliprojektiossa näköfrustumi on nelitahokkaan muotoinen ja projektiolla ei ole yhtä keskipistettä. (Puhakka 2008, 187.)



KUVA 20 *Näköfrustumi*

Perspektiiviprojektio toteutetaan muunnosmatriisin avulla. Muunnosmatriisissa otetaan huomioon kameran ominaisuudet, kuten kameran etäisyys etuleikkaustasosta. Muunnoksen jälkeen näköfrustumista tulee laatikon muotoinen. Laatikon sisällä olevaa avaruutta kutsutaan nyt normalisoiduksi kuvausvolyymiksi (canonical view volume). Laatikon sisällä olevien kappaleiden x- ja y-arvot ovat väliltä -1…1 ja z-arvot väliltä 0…1. Tämän jälkeen saatua aluetta kasvatetaan piirtoalueen resoluution mukaisesti, jolloin saadaan aikaiseksi lopullinen kuva. (Puhakka 2008, 192–194.)

3D-mallien pinnoille voidaan tehdä näkyvyystarkastelu, jossa katsojalle näkymättömät pinnat poistetaan suorituskyvyn parantamiseksi. Tekniikoita on useita erilaisia ja niiden käyttö vaihtelee tilanteen mukaan. Osa tekniikoista käyttää hyödyksi näköfrustumia. Seuraavassa on lueteltu yleisimmät tekniikat. Yleensä aina käytössä oleva tekniikka on takapinnan poisto (backface culling). Tämä tarkoittaa sitä, että pintoja ei täytetä jos pinnan normaalivektori osoittaa poispäin kamerasta. Papervision hyödyntää takapintojen poistoa. Toinen tekniikka on piilottaa kappaleet, jotka ovat niin kaukana, että niistä piirretään ruudulle enää muutama pikseli. Tätä kutsutaan nimellä contribution culling. Papervision hyödyntää myös tätä tekniikkaa. Kolmas tekniikka on näkyvyyskarsinta (occlusion culling), jossa piilotetaan toisten kappaleiden taakse jääneet pinnat ja kappaleet. Monimutkaisissa kohtauksissa tällä tekniikalla voidaan saavuttaa huomattavaa hyötyä. Tämä tekniikka on kuitenkin itsessään raskasta laskea, jonka vuoksi sitä ei hyödynnetä Papervisionissa. Yleensä tässä tekniikassa käytetään hyödyksi näytönohjainta. Viimeinen tekniikka on frustumikarsinta (viewing frustum culling). Frustumikarsinnassa näköfrustumin ulkopuolelle jääneitä pintoja ei piirretä. Frustumikarsinnassa voidaan helposti piilottaa myös kokonaisia kappaleita testaamalla niiden rajauslaatikoiden leikkausta näköfrustumiin. Papervision osaa hyödyntää tätä tekniikkaa. (Puhakka 2008, 259–261; Hidden surface determination 2010; Johnston 2008a.)

Frustumikarsinnassa voidaan käyttää hyödyksi tekniikkaa, jossa osittain näköfrustumin ulkopuolella oleva kappale leikataan poikki rajan kohdalta. Tätä tekniikkaa kutsutaan nimellä typistämiseksi tai leikkaamiseksi (clipping). Tekniikka ratkaisee ongelman, jossa frustumikarsinnassa piilotetut pinnan ovat vielä osittain ruudulla. Ongelma korostuu silloin kun ruudulla on isoja monikulmioita, ne ovat lähellä kameraa ja kameran näkökenttä on iso. Papervisionissa tekniikan saa päälle asettamalla renderöintimoottorin käyttämään FrustumClipping-luokkaa. Usein kappaleiden leikkausta halutaan testata vain näköfrustumin etutason kanssa, sillä sen kanssa tulee eniten ongelmia ja kaikkien tasojen testaaminen on raskasta. (Puhakka 2008, 188; Zupko 2008a.)

3.7 Matriisit

3D-avaruuden muunnokset esitetään monesti 4 × 4 matriiseilla. Matriisit ovat kuitenkin muutakin kuin pelkkä merkintätapa. Matriisitulon avulla on mahdollista helposti yhdistää useita muunnoksia keskenään yhdeksi kokonaismuunnosmatriisiksi. Papervisionissa maailmalla, kameralla ja 3D-objektilla on omat matriisinsa. 3D-objektin jokainen kärkipiste käy läpi matriiseista yhdistetyn kokonaismuunnoksen ennen kuin 3D-objekti saadaan projisoitua ruudulle oikealla tavalla oikeaan kohtaan. (Puhakka 2008, 81–83; Johnston 2008b.)

Seuraavassa on selitetty mitä matriisin lukuja muuttamalla saadaan aikaan halutut muunnokset 3D-objektin kärkipisteille.

Matriiseissa viimeinen rivi ei muutu. Se on sitä varten, että kokonaismuunnoksen esittäminen matriisilla siirtomuunnoksen kanssa on mahdollista. Riviin ei tarvitse kiinnittää huomiota sen enempää, ellei aiheeseen aio perehtyä syvemmin. (Puhakka 2008, 96.)

$$T(t_x, t_y, t_z) = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

KAAVA 2 *Siirtomuunnos matriisissa*

$$S(s_x, s_y, s_z) = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

KAAVA 3 *Skaalausmuunnos matriisissa*

Jäljelle jääneet luvut ilmaisevat skew-säätöjä. 3D-avaruudessa skew-säätö onnistuu minkä tahansa akselin suunnassa minkä tahansa toisen akselin suhteen. (Puhakka 2008, 103–104.)

$$H_{xz}(s) = \begin{bmatrix} 1 & 0 & s & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

KAAVA 4 *Skew-säätö x-akselin suunnassa z:n suhteen*

Kiertomuutosten ilmaus ei onnistu enää yhdellä luvulla vaan se tehdään yhdistelemällä skew-säätö ja skaalausmuunnoksia. Kierto tehdään aina jonkin akselin ympäri. Kaikki kappaleen asennot saataisiin aikaan kiertämällä kappaletta minkä tahansa kahden akselin suhteen. Tällöin kaikki kiertosuunnat eivät olisi kuitenkaan mahdollisia, jonka vuoksi kiertomatriisi kaikkien kolmen akselin suhteen on suotavaa. (Puhakka 2008, 103–104.)

Kierto x-akselin suhteen

$$R_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

KAAVA 5 *Kierto x-akselin suhteen*

$$R_y(\alpha) = \begin{bmatrix} \cos\alpha & 0 & \sin\alpha & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\alpha & 0 & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

KAAVA 6 *Kierto y-akselin suhteen*

$$R_z(\alpha) = \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 & 0 \\ \sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

KAAVA 7 *Kierto z-akselin suhteen*

4. PAPERVISIONIN KÄYTTÖÖNOTTO

4.1 Eri versiot

Papervisionista on saatavilla ActionScript 2 ja 3 -versiot. Uusimman ActionScript 3 -version käyttö projekteissa on suotavaa, jos se on vain mahdollista. ActionScript 3 versiossa on paljon parempi suorituskyky, jota tarvitaan monimutkaisten näkymien laskemiseen. ActionScript 3 -versio on vakaampi ja siinä on myös paljon enemmän ominaisuuksia, sillä ActionScript 2 -version kehitystyö on lopetettu. (Papervision 2009a.)

ActionScript 2 -version käyttö on kuitenkin ajankohtaista silloin kun ActionScript 3:sta ei voida käyttää projektissa. Tällainen tilanne voi olla, jos vanhoihin projekteihin halutaan lisätä uusia ominaisuuksia tai asiakas ei halua käyttää ActionScript 3:sta projektissa. (Papervision 2009a.)

Papervisionin uusin ActionScript 3 -versio on 2.1. Edellinen suurempi päivitys oli Papervision3D 2.0, joka tunnetaan myös koodinimellä Great White. Tässä versiossa Papervisionin ydin sai suuria muutoksia ja paljon uusia ominaisuuksia. Versio 2.0 muutettiin versioksi 2.1 versionhallinta numerosta 911 lähtien, koska ominaisuuksia oli tullut jo niin paljon lisää edelliseen versioon nähden. Uudessa versiossa Collada-animaatioiden hallinta on muutettu täysin erilaiseksi. (Papervision 2009a.)

Uusin ActionScript 2 -versio on nimeltään 1.7, joka tulee myös asennuspaketin mukana. Tämä versio poikkeaa rakenteellisesti paljon ActionScript 3 -versiosta. Tämän takia version opetteluun kannattaa varata aikaa vaikka osaisikin ennestään ActionScript 3 -versiota. (Papervision 2009a.)

Papervisionista on saatavilla myös version 2.1 Flash CS3 -komponentti. Komponentin avulla sovellukseen on mahdollista nopeasti lisätä Papervisionilla näytettäviä Collada-malleja. (Papervision 2009a.)

4.2 SVN-versiohallinnan käyttö

Papervisionia kehitetään jatkuvasti ja siihen tulee uusia päivityksiä kuukausittain. Tämän takia projektin alkaessa kannattaa aina hankkia uusin versio Papervisionista.

Papervisionin Google Code -sivustolta pystyy hakemaan asennuspaketin zip tai swc -muodossa. Asennuspaketit ovat kuitenkin yleensä useita kuukausia jäljessä tämänhetkisestä versiosta. Uusimman version hakemiseksi tarvitaan Subversion eli SVN-versiohallinnan käyttämiseen tarkoitettu ohjelma. Subversion ohjelmistot mahdollistavat suurien tiedostojoukkojen muokkaamisen keskitetysti internetin yli. Google Code -sivusto tukee vakiona tätä järjestelmää. Sovelluksia löytyy monen eri tahon tekemänä ja niitä on saatavilla Flashin kehittämistä varten Windowsille, Macintoshille, ja jopa suoraan Eclipseen. Ohjelmia on maksullisia ja ilmaisia. Ilmaisista versioista suosittelen Windowsille

TortoiseSVN–ohjelmaa, Mac OS:lle Versions-ohjelmaa ja Eclipselle Subversive-ohjelmaa. (Subversion 2010; Google Code 2010.)

Lähdekoodin lataamiseksi tulee tietää SVN checkout -polku, joka löytyy Papervisionin Google Code -sivustolta. Polku syötetään SVN-versiohallintaohjelmaan, jonka jälkeen se hakee uusimmat versiot jokaisesta tiedostosta ja tallentaa ne tietokoneelle. Myöhemmin voidaan tarkistaa onko tiedostoihin tullut muutoksia, jos on, ohjelma hakee vain muuttuneet tiedostot, eikä koko pakettia tarvitse hakea uudestaan. Päivittämissä kannattaa kuitenkin olla varovainen, sillä uuden version kanssa voi tulla ongelmia ja tällöin oma projekti ei toimi enää halutulla tavalla. Tämän takia ei kannatakaan päivittää projektin lähdekoodikansiota suoraan vaan tehdä siitä varmuuskopio ja tuoda uusi päivitetty kansio tilalle. Tämän jälkeen voi kokeilla, toimiiko uusi versio projektissa ilman ongelmia. Papervisionin Google Code -sivuilta löytyy myös lista tulleista päivityksistä, josta voi katsoa kannattaako projektiin päivittää uutta versiota. Myös vanhempien versioiden hakeminen Papervisionista onnistuu. (Papervision 2009a.)

4.3 Papervisionin asentaminen

Papervision asennetaan kopioimalla halutun version kansiosta src-kansion sisältö. Kansio kannattaa kopioida samaan paikkaan, jossa on Flash-projektin muutkin ulkoiset lähdekoodit. Jos projektilla ei ole ennestään ulkoisia lähdekooditiedostoja, kannattaa projektin juureen tehdä esimerkiksi src-niminen kansio, johon tiedostot kopioi. Tämän jälkeen Flashissä määritetään polku, jossa projektin lähdekoodit sijaitsevat. Poluksi kannattaa määrittää suhteelliseksi, jolloin projektikansion kopioiminen toiseen paikkaan ei vaikuta toimintaan.

4.4 Dokumentaation lukeminen

SVN-versiohallinnan kautta tulevassa paketissa on mukana myös viimeisimmät dokumentaatiot. Dokumentaatio on HTML-sivusto ja se on generoitu Flexin mukana tulevalla ASDocs-työkalulla. Se sisältää listauksen kaikista luokista, muuttujista ja niiden suhteesta. Joissain luokissa on mukana kommentteja, jotka auttavat niiden ymmärtämisessä. Pelkällä dokumentaatiolla ei pärjää, sillä se sisältää hyvin vähän esimerkkejä. Se auttaa kuitenkin käsittämään luokkasuhteita ja löytämään uusia ominaisuuksia luokista. Paras tapa löytää tarkempaa tietoa luokista on etsiä internetistä hakukoneilla käyttämällä luokkien nimiä ja ominaisuuksia.

Papervision-projektia kehittäessä kannattaa aina pitää dokumentaatiota auki, koska siitä on nopea tarkistaa luokkien ja muuttujien oikeinkirjoitus ja konstruktorit.

5. PAPERVISIONIN OMINAISUUKSIA

Seuraavassa käydään läpi Papervisionin tärkeimpiä ominaisuuksia. Tässä kappaleessa kerrotaan, mihin tarkoituksiin eri luokkia käytetään ja mitä ominaisuuksia niillä on. Luokkien konstruktorit ja niiden käyttöesimerkit on jätetty tästä pois, sillä ne on kerrottu selkeästi dokumentaatiossa.

5.1 Mistä luokista Papervision-sovellus rakentuu?

Papervision käyttää kohtauksen luontiin useita luokkia, joilla kaikilla on eri tarkoitus. Tämän rakenteen hyvä puoli on se, että luokkia vaihtamalla, saadaan aikaan erilaista jälkeä, joka sopii eri tarkoitukseen. Seuraavassa on selitetty, mistä luokista kohtaus rakentuu ja mikä on niiden tehtävä.

Scene3D-luokka toimii säiliönä kohtauksessa käytetyille kappaleille. Se sisältää siis koko 3D-ympäristön tiedot, joita muut luokat tarvitsevat näkymän luomisessa. (Lively 2008a; Tondeur 2008a.)

Viewport3D-luokan sisälle piirretään renderöity 3D-näkymä. Luokka hallitsee myös piirretyn näkymän kokoa ja paikkaa. (Lively 2008a; Tondeur 2008a.)

Camera3D-luokka määrittää, mistä paikasta ja mikä alue 3D-näkymästä renderöidään. Kameralla muokattavia ominaisuuksia, jotka vaikuttavat esimerkiksi sen näkökenttään. (Lively 2008a; Tondeur 2008a.)

Renderöintimoottori (esimerkiksi BasicRenderEngine-luokka) kerää tarvittavat tiedot kohtauksesta sekä kamerasta ja piirtää tämän jälkeen kuvan Viewport3D-instanssiin. Eri renderöintimoottorit käsittelevät saamaansa tietoa eri tavalla, jolloin näkymästä tulee erilainen. (Lively 2008a; Tondeur 2008a.)

DisplayObject3D-luokka sisältää kohtauksessa käytettyjen 3D-objektien tiedot. Näihin tietoihin kuuluu muun muassa geometria, paikkakoordinaatit ja skaalaus. (Lively 2008a; Tondeur 2008a.)

Materiaali-luokat määrittävät, millainen pinta, eli tekstuuri 3D-objektilla on. Tekstuurina voi käyttää esimerkiksi bittikarttaa, MovieClippiä tai videota. Kehittyneemmät materiaalit käyttävät varjostimia, jotka luovat kappaleen pintaan valon ja varjon sävyjä. (Lively 2008a; Tondeur 2008a.)

5.2 Viewport3D ja sen ominaisuudet

Viewport3D-luokalle määritetään aluksi sen koko. Jos leveyden ja korkeuden määrittää nollaksi, kooksi tulee Flash-alueen koko. Vakiona määritetyn alueen ulkopuolelle ei piirry mitään. Alueen mittoja voi muuttaa dynaamisesti ja alueen saa skaalautumaan automaattisesti, jos Flashin koko muuttuu. Mikäli piirrettyihin kappaleisiin halutaan interaktiivisuutta, tulee luokan interactive-arvo asettaa todeksi. Tämä asettaa kuuntelijat objekteille InteractiveSceneManager-luokan kautta.

Viewport3D-luokalla on myös useita hitTest-funktioita, joiden avulla voi itse testata vaikka hiiren osumista tiettyyn objektiin.
(Papervision 2010.)

Viewport3D-luokasta on mahdollista tehdä useampia instansseja ja niissä voi käyttää hyödyksi yhteisiä Scene3D- ja Camera3D-instansseja, sekä samaa renderöintimoottoria. Useampien Viewport3D-luokkien kekseliäs käyttö mahdollistaa paljon asioita. Luokan viewportObjectFilter-ominaisuutta voi käyttää objektien piilottamiseen tietystä Viewport3D-instanssista, jolloin niitä ei renderöidä. Näin pienien eroavaisuuksien takia Viewport3D-instansseille ei tarvitse tehdä omia Scene3D-instansseja. (Papervision 2010.)

Viewport3D-luokan instanssille voi määrittää kerroksia (ViewportLayer). Eri kerroksilla olevat objektit eivät voi mennä sisäkkäin, vaan toinen on aina toisen päällä. Kerrokset auttavat esimerkiksi optimoinnissa, sekä lähekkäin olevien objektien renderöintiongelmissa. (Papervision 2010.)

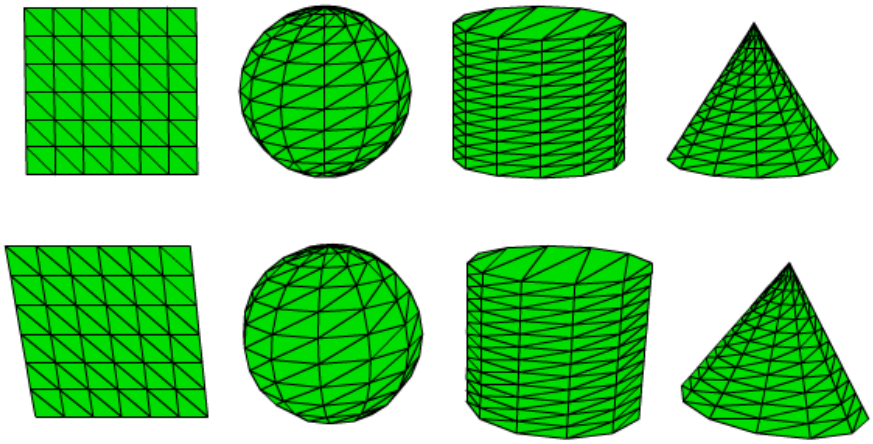
BitmapViewport3D on Viewport3D-luokan jatkoluokka, josta voi helposti kopioida bittikartta-dataa. Luokan instanssiin voi myös suoraan lisätä suodattimia, kuten epäterävyyttä. (Papervision 2010.)

5.3 Kamera ja sen ominaisuudet

Camera3D-luokalle määritetään kameran näkökenttä (field of view). Näkökenttä on määritetty vakiona 60 asteeksi, joka on ihmisen normaali näkökenttä katsottaessa suoraan eteenpäin. Kameralle määritetään myös näköfrustumin leikkaustasojen etäisyydet kamerasta. Kameran tarkennus-, eli focus-arvo on sama asia kuin etuleikkaustason etäisyys kamerasta. Kameran zoomausta voidaan säätää zoom-arvoa muuttamalla. Isommat arvot suurentavat kuvaa. Mikäli kohtauksen objektit eivät mene pois kuvasta, voi kamerasta ottaa frustumikarsinnan pois käytöstä, jolloin saadaan hieman lisää suorituskykyä. (Papervision 2010; Visual field 2010.)

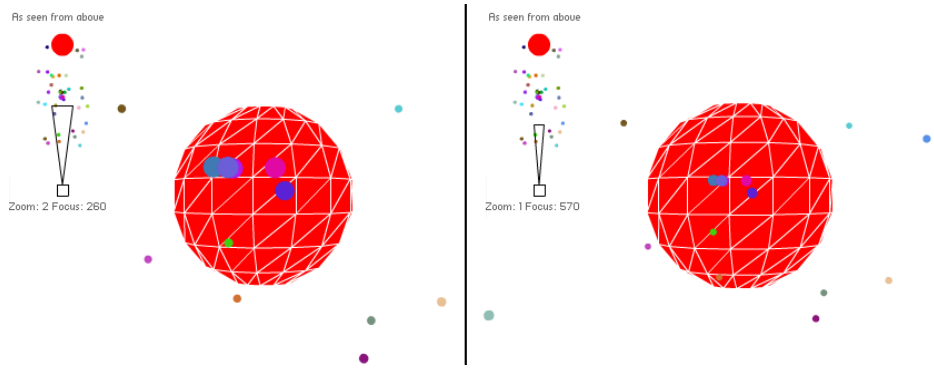
Kameran asetusten ja näköfrustumin oikeaoppinen säätö on tärkeää, jotta valmis kuva ei näytä vääristyneeltä ja siihen saadaan mahtumaan tarvittavat asiat.

Kameran näköfrustumin etuleikkaustason etäisyyttä säädetään kameran Camera3D-luokan focus- tai near-arvosta. Ei ole väliä kumpaa muuttujaa käyttää, sillä ne ovat saman asian kaksi eri nimeä. Etuleikkaustason ulkopuolelle jääneitä kappaleita ei piirretä. Muuttujan pienemmät arvot tekevät kuvasta laajemman, samalla tavalla kuin kamerassa olisi pienempi linssi. Samalla kuvan perspektiiviväristymä suurenee. Mikäli kuvaan haluaa mahtumaan enemmän asioita, mutta ei halua muuttaa perspektiiviä, kameraa tulee siirtää kauemmaksi kuvattavista kohteista. Kameran näköfrustumin takaleikkaustason etäisyyden muuttaminen ei vaikuta näkökenttään tai perspektiiviin, ainoastaan siihen, että tason ulkopuolella olevia kappaleita ei piirretä. (Caunt 2008a; Papervision 2010.)



KUVA 21 *Perspektiivivääristymän vaikutus. Ylemmässä kuvassa kamera on kaukana ja focus-arvo on suuri, alemmassa kuvassa kamera on lähellä ja focus-arvo on pieni*

Kameran zoom-arvon nostaminen suurentaa kappaleiden kokoa ruudulla. Zoom-arvon muuttaminen vaikuttaa samalla tavalla kuvaan kuin focus-arvon muuttaminen, eli pienemmillä arvoilla kuvan näkökenttä on suurempi. Zoom-arvo ei kuitenkaan vaikuta näköfrustumiin. Zoom-arvoa voi käyttää focus-arvon kanssa, jolloin niiden avulla saadaan esimerkiksi kappaleiden etäisyydet näyttämään erilaisilta. (Caunt 2008a; Papervision 2010.)



KUVA 22 *Esimerkki zoom- ja focus-arvojen käytöstä*

Jos kameran näkökentän haluaa tietyn levyiseksi, voi sitä muuttaa fov-arvosta. Muuttujan yksikkönä käytetään asteita. Fov-arvo muuttaa vain kameran focus-arvoa jättäen zoom-arvon samaksi. (Papervision 2010.)

Aivan kuten oikeaa kameraa, Papervisionin kameraa on tärkeä pystyä liikuttamaan ja kääntämään haluttuun suuntaan. Koska Camera3D-luokka on jatkettu DisplayObject3D-luokasta, sitä voi liikutella normaalisti muuttamalla x-, y- ja z-arvoja. Sitä voi myös kääntää rotationX-, rotationY- ja rotationZ- arvoilla. Kameralle voi asettaa kohteen, johon päin se on suuntautuneena kameran tai kohteen liikkuessakin. Kohde asetetaan target-muuttujalla ja siihen voi tallentaa minkä tahansa DisplayObject3D-instanssin. (Papervision 2010.)

Kameran liikutteluun tietyn kappaleen ympärillä on olemassa orbit-funktio. Funktiolle kerrotaan asteina tai radiaaneina kameran leveys- ja pystypaikka. Kamera pysyy aina samalla etäisyydellä kohteesta riippumatta sen paikasta. (Papervision 2010.)

Kamerasta saa ortograafisen asettamalla kamera-instanssin ortho-arvon todeksi. Ortograafisen kameran näköalaa muutetaan orthoScale-arvon avulla. Ortograafisessa näkymässä ei ole perspektiiviä. Tämän takia, sitä hyödynnetään esimerkiksi arkkitehtuurissa, jossa rakennuksen 3D-mallista saa helposti pohjapiirustukset ortograafisella kameralla. Ortograafista kameraa on hyödynnetty myös monissa vanhoissa peleissä, joissa maailma on kuvattu isometrisellä ortograafisella projektiolla. (Johnston 2008b; Papervision 2010.)

Tällä hetkellä kamerasta on kaksi jatkoluokkaa DebugCamera3D ja SpringCamera3D. SpringCamera3D on tarkoitettu seuraamaan kohdetta tietyn matkan päästä, reagoiden viiveellä kohteen liikkeisiin. Kamerassa voi säätää kaikki tarvittavat ominaisuudet, kuten viiveen, massan ja liikkeen kitkan. Kameralla pystyy simuloimaan hyvin oikeaa kameraa, joka ei pysy aina samalla etäisyydellä kuvattavasta kohteesta. (Papervision 2010.)

DebugCamera3D on nimensä mukaisesti tarkoitettu projektin kehitysvaiheeseen. Kameraa voi liikuttaa näppäimistöllä ja hiirellä. Myös kameran ominaisuuksia, kuten näköalaa voi muuttaa lennossa. Kamera on hyödyllinen eri kuvakulmien etsimisessä ja kameran säätöjen hakemisessa. Kameraa ei ole tarkoitettu käytettävän lopullisessa sovelluksessa. Jos normaaliin kameraan haluaa vastaavat kontrollit, ne täytyy kopioida DebugCamera3D-luokasta tai tehdä itse. (Papervision 2010.)

5.4 Eri Renderöintimoottorit

BasicRenderEngine-luokka on nimensä mukaisesti perusrenderöintimoottori, joka soveltuu käytettäväksi useimpiin projekteihin. Moottori on nopea, koska se ei tee mitään ylimääräisiä toimintoja ja se käyttää yksinkertaista menetelmää ratkaisemaan monikulmioiden syvyysjärjestyksen. Moottoriin voi lisätä renderöintisuodattimen, joka vaikuttaa renderöintijälkeen. Tällä hetkellä perusrenderöintimoottorille on vain yksi käytettävä suodatin, FogFilter. FogFilter häivyttää monikulmioita tiettyyn väriin niiden mennessä tarpeeksi kauas kamerasta. (Papervision 2010.)

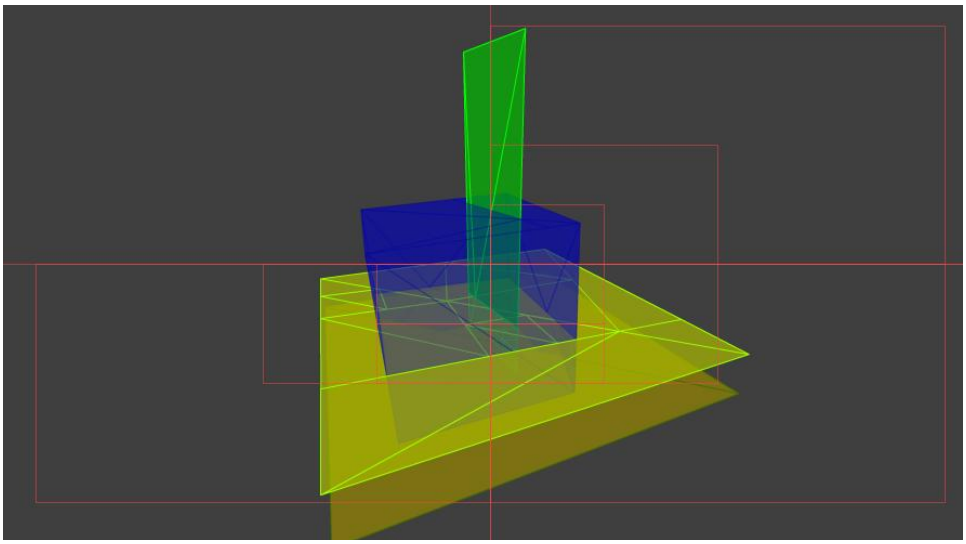
LazyRenderEngine on BasicRenderEngine:n jatkoluokka, jolle kerrotaan konstruktorissa Scene3D-, Camera3D- ja Viewport3D–luokat. Luokka olettaa, että nämä eivät muutu usein. Renderöintimoottorin käyttö on selkeämpää ja vaatii vähemmän koodirivejä, minkä vuoksi projektin lähdekoodista tulee selkeämpi. LazyRenderEngine ei tuo kuitenkaan mukanaan uusia ominaisuuksia. (Papervision 2010.)

Välillä projektissa ei renderöintimoottoriksi riitä BasicRenderEngine, sillä sen yksinkertaiset ominaisuudet eivät riitä ratkaisemaan ongelmatilanteita.

QuadrantRenderEngine-luokan kanssa voi käyttää kahta renderöintisuodatinta, jotka auttavat ongelmatilanteissa. Ensimmäinen suodatin, QuadrantZFilter, järjestää monikulmiot oikeaan syvyysjärjestykseen. BasicRenderEngine-luokkaa käytettäessä, jos monikulmion pituus on suurempi kuin matka toiseen monikulmioon, saattaa monikulmioiden syvyysjärjestys sekoittua katsottaessa niitä tietystä kamerakulmasta. Toinen suodatin, QuadrantFilter, auttaa tilanteessa, jossa kaksi monikulmiota menee toistensa läpi. Suodatin halkaisee monikulmiot dynaamisesti risteyskohdasta, jolloin ne voidaan piirtää ilman ongelmia. Jos suodatinta käyttää varjostimien kanssa saattaa kappaleen pinta näyttää huonolta, koska geometria jakautuu pieniin osiin. Suodattimet ovat molemmat vakiona päällä, mutta niistä on mahdollista käyttää vain toista. (Papervision 2010.)

Koska ongelmakohtien laskeminen koko ruudulta olisi erittäin raskasta, QuadrantRenderEngine jakaa ruutua adaptiivisesti suorakulmaisiin rajauslaatikoihin. Rajatun laatikon sisällä ongelmakohtia on paljon kevyempää selvittää. QuadrantRenderEngine:n käyttö on kuitenkin paljon raskaampaa kuin perusrenderöintimoottorin käyttö. QuadrantRenderEngine-luokassa on kuitenkin muutamia säätömahdollisuuksia, jotka nopeuttavat sen käyttöä. Luokassa voi määrittää kuinka moneen osaan ruutu maksimissaan jaetaan. Tämä määritetään luokan quadTree.maxLevel -muuttujassa. Sopivaa lukua on vaikea arvata etukäteen ja siksi projektissa kannattaakin tehdä testejä erisuuruisilla luvuilla. DisplayObject3D-instansseista, jotka toimivat muutenkin ihan hyvin, voi ottaa renderöintisuodattimien testaukset pois päältä. Testaus otetaan päältä muuttamalla DisplayObject3D-instanssin testQuad-muuttuja epätodeksi. (Papervision 2010; Zupko 2008c; Zupko 2008b.)

QuadrantRenderEngine:n käyttöä projektissa kannattaa miettiä tarkoin, sillä useat sen ratkaisemat ongelmat voidaan katkaista paljon kevyemmillä menetelmillä. Näitä ovat esimerkiksi Viewport3D-luokan kerroksien käyttö, kappaleiden sisäkkäin menemisen estäminen, sekä oikeaoppiset mallinnustekniikat. (Papervision 2010; Puhakka 2008, 306.)



KUVA 23 *Kuvassa näkyy kuinka QuadrantRenderEngine jakaa ruudun pienempiin osiin ja korjaa ongelmakohtia muuttamalla kappaleiden geometriaa*

5.5 BasicView

Papervision projektin alkuasetelman luontia nopeuttamaan ja koodia selkeyttämään Papervisionissa on BasicView-luokka. Luokka tekee projektiin valmiiksi instanssit luokista Viewport3D, Scene3D, Camera3D ja BasicRenderEngine. Luokkien vakioasetuksia voi muokata jälkeenpäin, jos on tarvetta. Luokkiin pääsee käsiksi BasicView-instanssin sisällä nimillä viewport, scene, camera ja renderer. (Papervision 2010.)

BasicViewin avulla kohtausta voi renderöidä kolmella eri tavalla. Luokan singleRender-funktio renderöi yhden kuvan kohtauksesta. Luokan startRendering-funktiota ei tarvitse ajaa kuin kerran, jolloin kohtausta renderöidään jatkuvasti. Jos BasicViewin avulla haluaa renderöidä kohtausta käyttäen esimerkiksi eri kameraa, onnistuu se ajamalla renderöintimoottorin renderScene-funktio. Funktiota käytetään seuraavalla tavalla: renderer.renderScene(scene, camera, viewport). (Papervision 2010.)

BasicView-luokalla on ReflectionView-jatkoluokka, joka tekee automaattisesti heijastuksen tietystä Viewport3D-instanssista. Heijastuksen käyttö on raskasta monimutkaisissa kohtauksissa, mutta sillä saa helposti näyttävyyttä yksinkertaisiin projekteihin. (Papervision 2010.)

5.6 DisplayObject3D

DisplayObject3D on Papervisionin vastine Flashin omalle MovieClip-luokalle. DisplayObject3D–luokasta jatketaan 3D-objektit, kuten primitiivit, mutta myös kamerat ja valot. (Papervision 2010.)

Luokalla on paljon samantapaisia ominaisuuksia kuin MovieClipillä. Sillä on perusominaisuuksia, kuten x-, y- ja z-koordinaatit, sekä pyöritys ja skaalaus arvot kaikkien kolmen akselin suhteen. Luokka sisältää myös

erikoisempia ominaisuuksia, kuten materiaalin ja sen paikan ruudun 2D-koordinaatistossa. (Papervision 2010.)

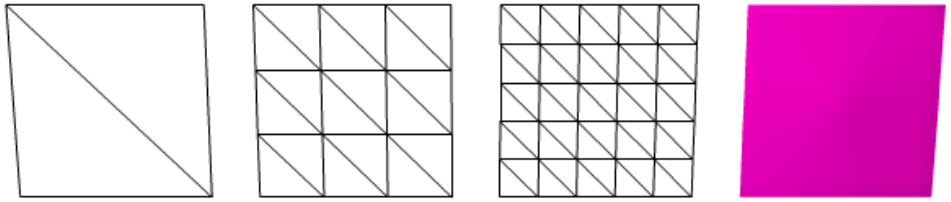
Luokka ei ole abstrakti-tyyppiä, joten siitä voidaan luoda tyhjiä instansseja kohtaukseen. DisplayObject3D-luokan instanssi voi toimia myös säiliönä muille 3D-objekteille. (Papervision 2010.)

5.7 Primitiivit

Primitiivit ovat yleisimpiä geometrisiä kappaleita, jotka helpottavat ja nopeuttavat Papervision-projektin kehittämistä. Monet sivustot ja Flash-sovellukset on luotu pelkästään näitä primitiivejä käyttämällä. Primitiivit, kuten muutkin 3D-objektit koostuvat monikulmioista. Primitiivejä luodessa on mahdollista määrittää kuinka pieniin segmentteihin se on jaettu, eli kuinka paljon monikulmioita siinä käytetään. Pienempiin segmentteihin jaettu primitiivi näyttää pehmeämmältä ja paremmalta, mutta sen renderöimiseen menee enemmän aikaa. (Papervision 2010.)

Primitiivien konstruktoreissa käytetyt pituuksien mitat eivät ole pikseleitä vaan Papervision-yksikköitä. (Papervision 2010.)

5.7.1 Taso (Plane)



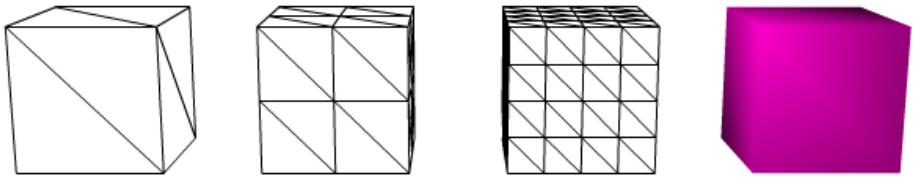
KUVA 24 *Erilaisia taso-primitiivejä*

Taso on niiden pisteiden joukko, joiden etäisyys kahdesta annetusta vakiopisteestä A ja B on yhtä suuri. A ja B eivät saa olla sama piste. Taso leikkaa A:n ja B:n välisen janan puolivälistä. (Taso 2010.)

Taso on Papervisionissa yleisimmin käytetty primitiivi, sillä se on geometrisesti yksinkertainen ja tämän takia nopea laskea ja piirtää. Käyttämällä pelkästään tasoja, voidaan tehdä monimutkaisia ja sulavia näkymiä, jos niissä käytetään materiaaleja oikealla tavalla.

Papervisionissa tason konstruktorissa voidaan määrittää tason materiaali, leveys, korkeus ja segmentit. (Papervision 2010.)

5.7.2 Kuutio (Cube)

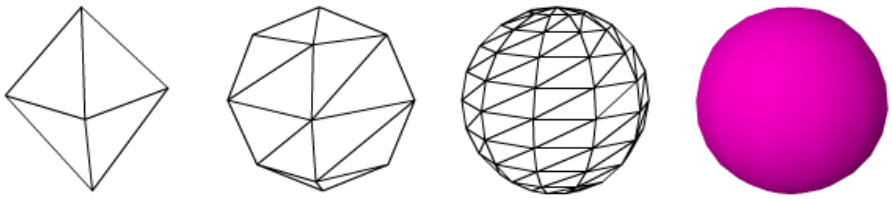


KUVA 25 *Erilaisia kuutio-primitiivejä*

Vaikka Papervisionissa primitiivin nimi on kuutio, kyseessä on kuitenkin nelitahokas, jonka ulottuvuuksia voi muokata kaikkiin kolmeen suuntaan. Nelitahokas muodostuu kahdeksasta kärkipisteestä, sekä kuudesta tahkosta, jotka ovat suorakulmioita.

Kuution konstruktorissa voi määritellä tahkojen materiaalit, leveyden, korkeuden, syvyyden, segmentit, sisäänpäin kääntyneet tahkot ja poisluetut tahkot. Kuution materiaaliksi määritetään MaterialsList-luokan instanssi, jolloin jokaiselle tahkolle voidaan määrittää oma materiaali. (Kuutio 2010.)

5.7.3 Pallo (Sphere)

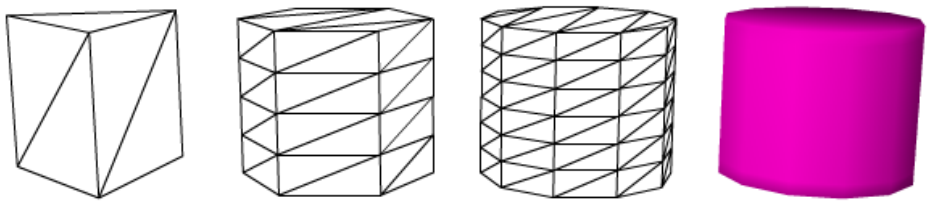


KUVA 26 *Erilaisia pallo-primitiivejä*

Pallo on täysin symmetrinen geometrinen muoto. Geometrisesti se on niiden pisteiden joukko, joiden etäisyys kolmiulotteisen avaruuden pisteestä on vakio. (Pallo 2010.)

Papervisionissa pallon konstruktorissa voidaan määrittää sen materiaali, säde ja segmentit. Jos pallosta haluaa mahdollisimman sileän, tulee se jakaa moneen segmenttiin, jolloin siitä saattaa tulla raskas laskea, varsinkin jos niitä on esillä monta samaan aikaan. Pieniin segmentteihin jaetussa pallossa saattaa tulla myös ongelmia joidenkin materiaalien kanssa. Sileässä pallossa sen pohja- ja lakipisteeseen tulee paljon pieniä kolmioita lähekkäin, jolloin niissä kohdissa tekstuurista saattaa tulla huonon näköinen. (Papervision 2010.)

5.7.4 Lieriö (Cylinder)

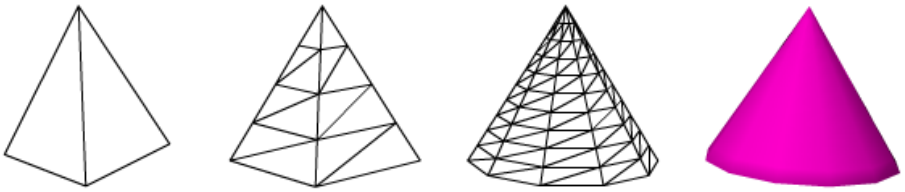


KUVA 27 *Erilaisia lieriö-primitiivejä*

Lieriö on pinta, jonka suora muodostaa kulkiessaan umpinaista käyrää pitkin. (Lieriö 2010)

Papervisionissa lieriön konstruktorissa määritetään lieriön materiaali, korkeus, säde, segmentit, katon säde ja pohjan ja katon näkyvyys. (Papervision 2010.)

5.7.5 Kartio (Cone)



KUVA 28 *Erilaisia kartio-primitiivejä*

Papervisionin kartio on tarkkaan ottaen suora ympyräkartio. Se muodostuu, kun suorakulmaista kolmiota pyöräytetään toisen lyhyen sivunsa ympäri. Tästä tulee kartion akseli ja toisen lyhyen sivun piirtämä ympyrä muodostaa kartion pohjan. (Kartio 2010.)

Papervisionissa kartion konstruktorissa voidaan määrittää sen materiaali, pohjan säde, korkeus, sekä segmentit. (Papervision 2010.)

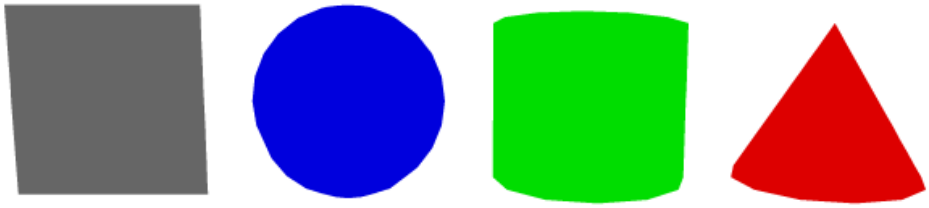
5.8 Materiaalit

3D-kappaleiden materiaaleilla on suuri osuus niiden käytössä. Yksinkertaistenkin kappaleiden teksturointi hyvillä materiaaleilla tekee niistä hienon näköisiä. Yhdessä objektissa voi olla useita eri materiaaleja. Jos kappaleiden materiaalia ei määritetä konstruktorissa, materiaaliksi tulee null, tällöin kappaleesta näkyy vain monikulmioiden ääriviivat. Kyseessä ei kuitenkaan ole WireframeMaterial, eikä sillä ole kyseisen materiaalin ominaisuuksia. Materiaalin voi määrittää myös myöhemmin ja sitä voi vaihtaa dynaamisesti. (Papervision 2010.)

Kaikki materiaalit johdetaan MaterialObject3D-luokasta, joten niillä on paljon yhteisiä ominaisuuksia. Näitä ominaisuuksia ovat mm. kaksipuoleisuus, materiaalin pehmennys skaalatessa, interaktiivisuus ja toistuvuus. (Papervision 2010.)

Tekstuurien koko vaikuttaa 3D-objektien renderöintinopeuteen. Yleensä kannattaakin käyttää sellaisia tekstuureita, jotka ovat mahdollisimman pieniä, mutta näyttävät vielä tarpeeksi hyviltä. (Lindquist 2008b; Papervision 2010.)

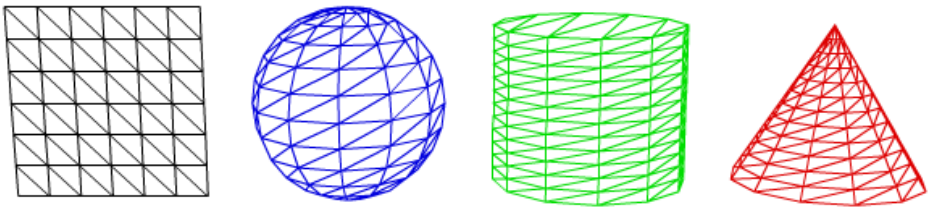
5.8.1 ColorMaterial



KUVA 29 *Primitiiveissä on käytetty erivärisiä ColorMaterial-materiaaleja*

ColorMaterial täyttää kappaleen yhdellä värillä ilman muita värisävyjä. Materiaali on erittäin kevyt ja yksinkertainen käyttää. Materiaalissa voi valita värin ja läpinäkyvyyden. (Papervision 2010.)

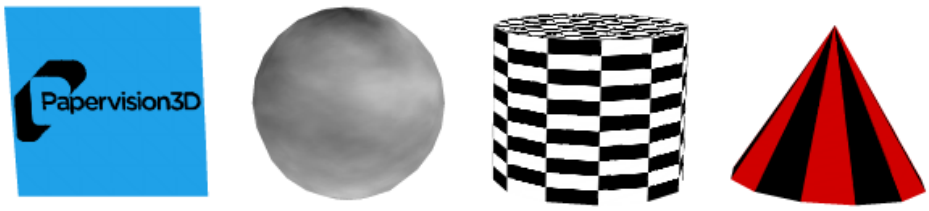
5.8.2 WireframeMaterial



KUVA 30 *Primitiiveissä on käytetty erivärisiä rautalanka-materiaaleja*

WireframeMaterial yhdistää 3D-objektin monikulmioiden ääriviivat tietyn paksuisella viivalla, jolloin siitä näkee helposti, minkä muotoinen kappale on. Tätä kuvaustapaa kutsutaan rautalankamalliksi. Rautalankamallin viivan värin, paksuuden ja läpinäkyvyyden voi määrittää. Pelkkää rautalankamallia ei kannata käyttää interaktiivisten kappaleiden tekoon, sillä vain viivoista voi painaa hiirellä, jolloin mallista on helppo painaa läpi. (Papervision 2010.)

5.8.3 MovieMaterial



KUVA 31 *Primitiiveissä on käytetty erilaisia MovieMaterial-tekstuureita*

MovieMaterial luo objektille tekstuurin olemassa olevasta MovieClip-instanssista. MovieClipin senhetkinen skaalaus ja ulkonäkö määrittävät tekstuurin. MovieClipin pyöritys ei vaikuta tekstuuriin. (Papervision 2010.)

MovieMaterial voi olla animoitu ja läpinäkyvä. MovieClipistä on mahdollista myös määrittää tietty nelikulmainen osa, jota käytetään tekstuurina. Materiaali on mahdollista määrittää myös tarkaksi, jolloin materiaali jaetaan renderöitäessä pienempiin osiin. Tämä vähentää huomattavasti tekstuurin vääristymää, mutta renderöinti on raskaampaa. (Papervision 2010.)

On tärkeä huomata, että vaikka tekstuurina käytetty MovieClip olisikin piirretty vektoreina ja se olisi tällöin täysin skaalattava, tekstuuri luodaan MovieClipin senhetkisestä bittikartta-datasta. Tämä tarkoittaa, että tekstuurista tulee sitä tarkempi, mitä suurempi MovieClip on sillä hetkellä.

MovieMaterial-luokka antaa monipuolisimmat ominaisuudet lisätä interaktiivisuutta kappaleeseen. Kappaleen materiaalina käytetyn MovieClipin sisällä olevista MovieClip- ja Button-instansseihin on mahdollista lisätä kuuntelijoita, jolloin niistä voi esimerkiksi painaa hiirellä. (Papervision 2010.)

5.8.4 MovieAssetMaterial

MovieAssetMaterial on samanlainen kuin MovieMaterial, mutta se käyttää tekstuurin tekoon Flashin kirjastossa olevaa symbolia. Kirjastossa olevan symbolin sisältö tulee olla asemoitu vasempaan yläkulmaan. Vakiona materiaali varastoidaan ja sitä käytetään uudelleen. Materiaalin voi kuitenkin määrittää yksilölliseksi, jolloin siihen tehdyt muutokset eivät vaikuta muihin instansseihin. (Papervision 2010.)

5.8.5 Bittikarttamateriaalit

Bittikarttamateriaaleista löytyy vastineet normaaleille materiaaleille. Nämä ovat BitmapColorMaterial, BitmapWireframeMaterial, BitmapMaterial ja BitmapAssetMaterial. Kaksi viimeistä käyttää MovieClippien sijasta bittikarttoja. (Papervision 2010.)

Bittikarttamateriaaleissa on kuitenkin muutama uusi materiaali. BitmapFileMaterial käyttää tekstuurina ulkoista bittikarttatiedostoa. Jos 3D-objektia näytetään ennen kuin bittikartta on latautunut, materiaali näyttää samalta kun sitä ei olisi määritelty. BitmapViewportMaterial käyttää tekstuurina tietyn Viewport-luokan bittikartta-dataa. Tälle materiaalille voi keksiä mielenkiintoisia käyttötarkoituksia, kuten esimerkiksi kappaleen, joka on heijastavinaan ympäröivää maailmaa. (Papervision 2010.)

Bittikarttamateriaalien ominaisuuksiin kuuluu, että niihin pystytään suoraan lisäämään suodattimia, kuten epäterävyyttä ja hehkua. Bittikarttamateriaaleissa pääsee myös käsiksi bittikarttadataan, jolloin sieltä voidaan esimerkiksi vaihtaa pikseleiden värejä. (Papervision 2010.)

5.8.6 VideoStreamMaterial

VideoStreamMaterialin avulla kappaleeseen voi lisätä videokuvaa NetStream-objektin avulla. Materiaalina voi siis käyttää ulkoisia tai projektista löytyviä videoita. Videomateriaalissa on samat toiminnallisuudet kuin normaaleissa videoissa, eli niitä voi esimerkiksi pysäyttää ja kelata. Myös käyttäjän web-kameran kuvan käyttö materiaalina onnistuu. (Papervision 2010.)

5.8.7 CompositeMaterial

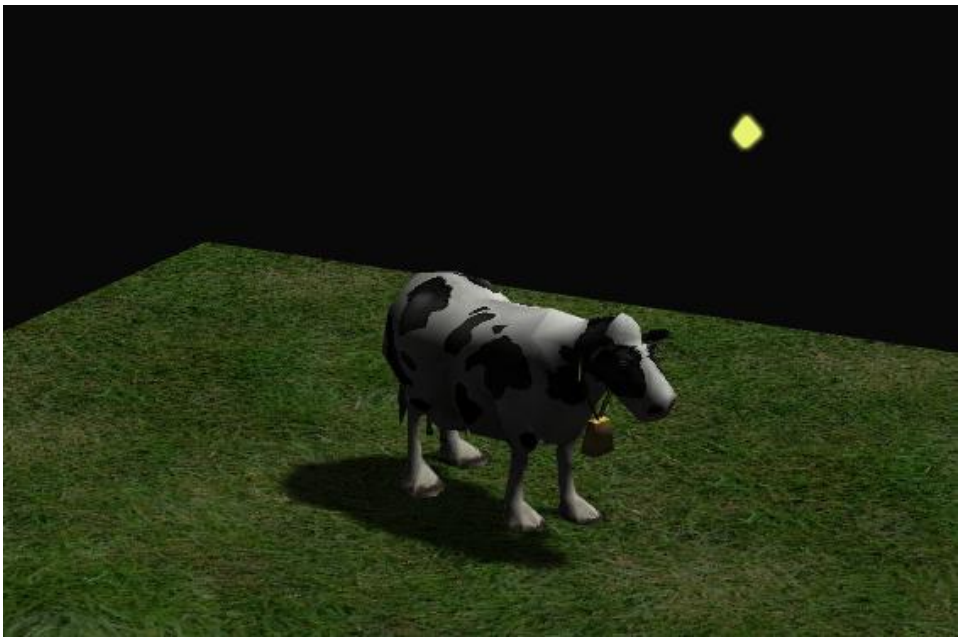
CompositeMaterial löytyy Papervisionin erikoismateriaalien alta. CompositeMaterial mahdollistaa useiden eri materiaalien käytön samassa 3D-objektissa. CompositeMaterialin avulla on esimerkiksi mahdollista tehdä täytetty rautalankamalli. (Papervision 2010.)

5.9 Valaistus ja varjot

Valojen käyttö kohtauksessa tuo sille paljon realistisemman ilmeen, kuin tasaisena pysyvä valaistus. Papervisionissa varjostinmateriaaleilla on pakko olla valon lähde, jotta materiaalit pystytään laskemaan oikein. (Papervision 2010.)

Tällä hetkellä Papervisionissa on vain yksi valo-luokka, PointLight3D. PointLight3D-luokka olettaa, että kaikki valo lähtee yhdestä pisteestä, eikä esimerkiksi jonkin objektin koko pinnasta. Valon instansseja voi liikutella vapaasti 3D-avaruudessa, sillä se on jatkettu DisplayObject3D-luokasta. (Papervision 2010.)

Papervisionissa ei ole vakiona vielä varjojen laskemista, mutta Papervisionin efekteistä vastaava Andy Zupko on kuitenkin tehnyt ShadowCaster-luokan, joka pystyy langettamaan ”tarkan” varjon yhdelle tasolle, mistä tahansa 3D-objektista. Varjon voimakkuus ja sen reunojen pehmeys on säädettävissä. (Zupko 2008d.)



KUVA 32 *Esimerkki ShadowCaster-luokan käytöstä*

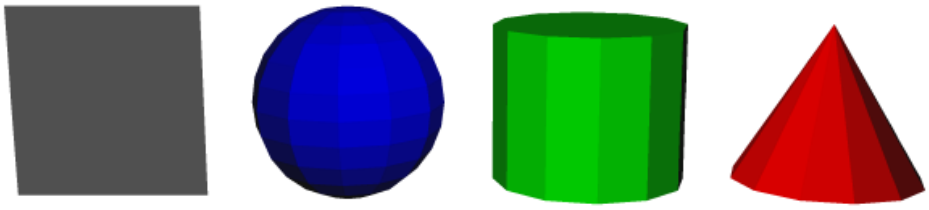
5.10 Varjostimet (Shaders)

Varjostimet tekevät tasavärisestä kappaleesta monipuolisemman simuloimalla valon vaikutusta sen pintaan. Varjostimien käyttö kappaleiden materiaaleissa on paljon raskaampaa kuin valaistuksen osalta tasaisena pysyvät materiaalit. Varjostimien avulla kohtauksesta saa kuitenkin realistisemman näköisen. Myös sellaisissa tapauksissa, joissa realismi ei ole pääasia, kappaleiden muoto on paljon helpompi hahmottaa, jos siinä on varjoisia ja valoisia kohtia.

Tällä hetkellä Papervisionissa on vakiona viisi eri varjostin-luokkaa. FlatShader, CellShader, PhongShader, GouraudShader ja EnvMapShader. Ainakin toistaiseksi Papervisionissa varjostimille saa määritettyä vain yhden valonlähteen kullekin varjostimelle, jotta suorituskyky pysyisi hyvänä. Jos valonlähdettä ei määritetä, varjostimet olettavat, että valo lähtee kamerasta. (Papervision 2010.)

Kaikille varjostimille on olemassa valmis varjostinmateriaali-luokka. Luokat nopeuttavat kappaleen materiaalin tekoa, mikäli kappale on yksivärinen, eikä siinä ole käytetty tekstuuria. Väriltään monipuolisempiin kappaleisiin täytyy ensin laittaa tekstuuri paikoilleen ja sen jälkeen haluttu varjostin. (Papervision 2010.)

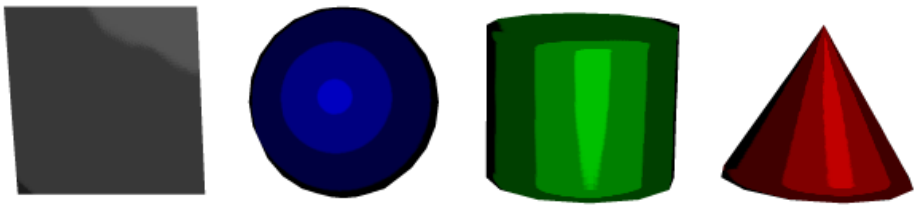
5.10.1 FlatShader



KUVA 33 *Primitiiveihin on lisätty FlatShader-varjostin*

FlatShader on hyvin yksinkertainen ja nopea varjostinmalli, jossa jokainen monikulmio väritetään yhdellä värillä riippuen pinnan normaalivektorista ja valon asemasta. FlatShader-luokassa voi määritellä valolähteen, kirkkaimman värin, tummimman värin, sekä kuinka isolle alueelle valo vaikuttaa. FlatShader-varjostinta voi käyttää esimerkiksi kappaleen muodon havainnointiin, sillä harva kappale näyttää kovin realistiselta tällä varjostimella. (Papervision 2010.)

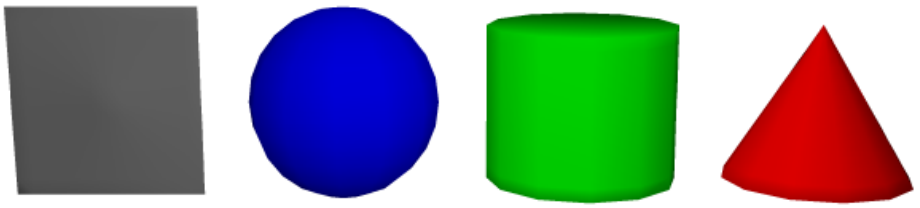
5.10.2 CellShader



KUVA 34 *Primitiiveihin on lisätty CellShader-varjostin*

CellShader-varjostimen avulla voi tehdä kappaleesta sarjakuvamaisen. Varjostimessa valitaan kirkkain ja tummin väri sekä se, kuinka monta eri väriaskelta niiden välissä on. Kappaleen värittämiseen ei käytetä mitään muita valoisuusasteita kuin määrätyt. Tietyt väriyhdistelmät voivat kuitenkin saada aikaan sen, että kappale ei värity oikealla tavalla, vaan mukaan tulee useita erikoisia värejä. Kappaleesta saa vielä enemmän sarjakuvamaisen, jos kappaleessa käyttää vielä voimakasta hohto-suodatinta, jolloin kappaleelle piirtyy uloimmat ääriviivat. (Papervision 2010.)

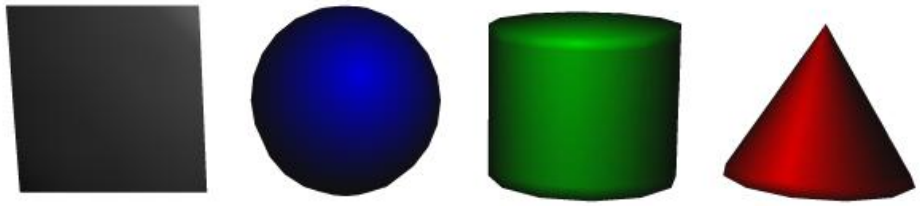
5.10.3 GouraudShader



KUVA 35 *Primitiiveihin on lisätty GouraudShader-varjostin*

GouraudShader täyttää kappaleen jokaisen kolmion yksinkertaisella liukuvärillä. Näin kappaleen pintaan muodostuu varjoisia ja valoisia kohtia hyvin kevyellä tavalla. Malli ei kuitenkaan näytä kovin realistiselta, jos siinä on vähän monikulmioita. Tämä tekniikka ei myöskään mahdollista pinnan muodon muokkaamista esimerkiksi kuhmutuksen avulla. GouraudShader-luokassa määritetään valon väri, taustavalon väri sekä se, kuinka isolle alueelle valo vaikuttaa. GouraudShader kelpaa hyvin esimerkiksi kappaleen pinnanmuotojen hahmotukseen, sekä keveytensä puolesta useiden samaan aikaan esillä olevien kappaleiden varjostukseen. (Papervision 2010.)

5.10.4 PhongShader



KUVA 36 *Primitiiveihin on lisätty PhongShader-varjostin*

PhongShader on samantapainen varjostin kuin GouraudShader, mutta kehittyneempi. PhongShader laskee varjostusasteen jokaiselle pikselille erikseen. Tällöin jäljestä tulee paremman näköinen, mutta se vie enemmän prosessointitehoa. PhongShader mahdollistaa myös kuhmutuksen ja maskin käyttämisen spekulaareille heijastumisille. Jälkimmäinen ominaisuus ei ole kuitenkaan vielä käytössä. PhongShader-luokassa määritetään valon väri, taustavalon väri, kuinka isolle alueelle valo vaikuttaa, sekä kuhmutus-bittikartta. Varjostinta kannattaa käyttää kappaleissa, joiden on tarkoitus näyttää mahdollisimman hyviltä tai kappaleissa, johon tarvitaan kuhmutusta. (Papervision 2010.)

5.10.5 EnvMapShader

KUVA 37 *Primitiiveihin on lisätty EnvMapShader-varjostin*

EnvMapShader on tarkoitettu peilipintojen tekoon. Kappaleen pinnalla näkyy määritetty kuva. Kuvan asemointi kappaleen pinnalla riippuu valonlähteestä. Jos valonlähdettä liikutetaan, heijastetun kuvan paikka muuttuu. Kuvana voi käyttää mitä tahansa bittikartta-dataa. Varjostimessa on mahdollista käyttää myös kuhmutusta. EnvMapShader-luokassa määritetään heijastuskuva etu- ja takapuolelle, taustavalon väri, sekä

kuhmutus-bittikartta. Varjostimen luovalla käytöllä pystyy tekemään mielenkiintoisia elementtejä projektiin. (Papervision 2010.)

5.11 Vektorifontit ja SVG-kuvat

Papervisionin materiaalit kuten MovieMaterial ja BitmapAssetMaterial jatkavat BitmapMaterial-luokkaa, joten ne käsittelevät tekstuuria bittikarttakuvana. Bittikarttakuvan huonona puolena on kuitenkin sen tarkkus läheltä katsottaessa. Erityisesti jos tekstiä tarvitsee tuoda lähelle kameraa, ei ole järkevää suurentaa tekstuurin kokoa valtavan suureksi riittävän tarkkuuden saamiseksi. Papervisionin Text3D-luokka mahdollistaa tekstin piirtämisen kolmiulotteisessa maailmassa vektoreina. Tämän ansiosta tekstiä voi tuoda lähelle kameraa huonontamatta sen laatua. Tämä tekniikka on porttaus Mathieu Badimonin Five3D-Flashmoottorista. Ennen tekniikan lisäämistä Papervisioniin, se tunnettiin nimellä VectorVision. (Barcinski 2008.)

Vakiona Papervisionissa on neljä eri fontti-leikkausta, HelveticaRoman, HelveticaLight, HelveticaMedium ja HelveticaBold. Five3D:n kotisivuilta saatavalla työkalulla on mahdollista tehdä omia fonttitiedostoja, joita pystyy käyttämään Text3D-luokan fontteina. James Hight on tehnyt myös luokkakirjaston, jolla TrueType Font -tiedoston voi dynaamisesti muuttaa Text3D-luokan käyttämäksi fontiksi. (Hight 2008.)

KUVA 38 *Ylempänä kuvassa näkyy MovieMaterial-luokalla tehty teksti, alempana on Text3D-luokalla tehty teksti*

Papervisionissa VectorShape3D-luokan avulla voi piirtää omia vektoreita. SVG (Scalable Vector Graphics) on XML-pohjainen vektorikuvaformaatti, joka mahdollistaa täysin skaalattavien kuvien piirtämisen. Papervision ei osaa kuitenkaan itse karsia tarvittavaa dataa SVG-kuvista. VectorVisionin SvgPathsPapervision-luokan avulla tämä on kuitenkin mahdollista. VectorVision on ladattavissa esimerkkeineen VectorVisionin Google Code -sivustolta. (Scalable Vector Graphics 2010.)

5.12 3D-objektit

Jos projektiin halutaan monimutkaisempaa geometriaa, sitä harvoin kannattaa lähteä rakentamaan useista primitiiveistä. Järkevämpi ratkaisu tähän on tehdä tarvittava 3D-malli jossain muussa ohjelmassa. 3D-mallien käyttäminen Papervision-projektissa vaatii kokemusta jonkin tuetun 3D-mallinnusohjelman käytöstä, sekä jonkinlaista yleistietämystä 3D-mallien oikeaoppisesta tekemisestä. Internetistä löytyy kuitenkin paljon ilmaisia 3D-malleja, joiden avulla voi testata 3D-mallien käyttöä ja tutkia miten hyvin toimiva malli on rakennettu.

3D-mallien teossa tulee ottaa huomioon Papervisionin rajoitukset. Vaikka 3D-malli näyttäisi 3D-mallinnusohjelmassa hienolta, saattaa siinä esiintyä ongelmia Papervisionissa. 3D-mallissa käytetyt tekstuurit kannattaa pitää mahdollisimman yksinkertaisina, sillä esimerkiksi 3D-mallohjelman monimutkaiset yhdistelmämateriaalit eivät tule näyttämään samalta Papervisionissa. 3D-mallin monikulmioiden määrä kannattaa pitää mahdollisimman pienenä, sillä monimutkaiset mallit vaativat paljon laskentatehoa. Papervisionin perusrenderöintimoottori asettaa myös rajoituksia. Koska monikulmioiden syvyysetäisyys ja sitä myöten piirtojärjestys lasketaan normaalisti monikulmion keskeltä, saattaa huonosti rakennetuissa malleissa esiintyä ongelmia. Esimerkiksi sumpussa olevat tai liian suuret monikulmiot ovat ongelmakohtia. Ongelmakohtia kannattaa yrittää välttää etukäteen, mutta ne voi myös korjata jälkikäteen, kun on ensin testannut 3D-mallin toimivuutta Papervisionissa.

Papervision tukee tällä hetkellä viittä eri 3D-tiedosto -formaattia. Nämä ovat DAE, KMZ, 3DS, ASE ja MD2. Näistä eniten tuettu ja kehitettävä formaatti on DAE. Joissain erityistilanteissa muillekin formaateille löytyy kuitenkin käyttöä. (Papervision 2010.)

COLLADA eli Collaborative Design Activity on XML-pohjainen tiedostoformaatti 3D-datalle. Collada-tiedoston pääte on yleensä dae. Papervision osaa tällä hetkellä tulkita formaatin 1.4.1 versiota. Formaatti tukee geometrian ja perusominaisuuksien lisäksi myös useita eri tekstuureita samassa mallissa, sekä animaatiota. Papervisionissa Collada-mallin pystyy lataamaan käyttämällä joko Collada- tai DAE-luokkaa. DAE-luokkaa käytettäessä on kuitenkin enemmän säätömahdollisuuksia, jonka takia se on yleensä parempi valinta. Collada-formaattia tukevat monet 3D-mallinnusohjelmat, kuten 3D Studio Max, Maya, Blender ja Google Sketchup. Niiden käytössä on kuitenkin muistettava, että tiedosto on tallennettava 1.4.1 versioon. Jos ohjelma ei tähän suoraan pysty, löytyy sille intenetistä yleensä tarvittava ohjelmalaajennus. Tallennuksessa voi yleensä säätää mallin asetuksia, jotka vaikuttavat mallin toimimiseen Papervisionissa. Parhaisiin asetuksiin pääsee kun kokeilee eri vaihtoehtoja. Papervisionin dokumentaatiossa on myös muutamia vinkkejä 3D Studio Maxin ja Mayan asetuksiin. (COLLADA 2010.)

Google Sketchup -3D-mallinnusohjelma pystyy tallentamaan 3D-mallit KMZ-formaattiin. KMZ-tiedosto on kuitenkin vain ZIP-tiedosto, jonka sisällä on tekstuurit ja DAE-tiedosto. Formaatin etuna on DAE-formaattia

pienempi tiedostokoko, mutta Papervisionissa sille ei löydy niin paljon säätöominaisuuksia kuin DAE-luokassa. (Papervision 2010.)

3DS on 3D Studio Maxin tiedostoformaatti, jonka hyvinä puolina on tiedoston tallentamisen helppous, pieni tiedoston koko ja valmiina löytyvien mallien määrä. Papervisionissa Max3DS-luokka kuitenkin häviää ominaisuuksiltaan DAE-luokalle. (3ds 2010.)

ASE- ja MD2-formaattien käytölle ei oikein löydy hyviä syitä, koska muut formaatit ovat kehittyneempiä ja sisältävät enemmän ominaisuuksia. Jos kuitenkin tarvittava malli löytyy vain ASE- tai MD2-formaatissa, saattaa näillekin löytyä käyttöä. (Papervision 2010.)

5.12.1 Collada-animaatiot

Papervision ymmärtää suurinta osaa Collada-formaatin tukemista animaatioista. Näitä ovat perusanimaatiot, kuten skaalauksen, paikan ja pyörityksen muutokset. Tärkeimpänä ja monipuolisempana ominaisuutena Collada-formaatti osaa hyödyntää luiden käyttöä animaatiossa. Luiden avulla Papervisionissa pystyy tekemään esimerkiksi hahmo-animaatiota. (Papervision 2010.)

Papervisionin versiosta 2.1 eteenpäin animaation kontrollointi muuttui täysin. Nyt animaation kontrollointifunktiot ovat play(), play(”animaation nimi”), stop(), pause() ja resume(). (Papervision 2010.)

Samaan tiedostoon pystyy lisäämään useita animaatioita, joita Collada-formaatissa kutsutaan animaatioklipeiksi (animation clip). Tietyn animaation voi toistaa käskyllä play(”animaatioklipin nimi”). Eri ohjelmat tallentavat animaatioklipit eri tavoilla. Esimerkiksi 3D Studio Maxin ColladaMax -laajennus muuttaa kohtauksen animaatiotasot animaatioklipeiksi. (Papervision 2010.)

Animaatiot eivät lisää tiedoston kokoa hirveästi, mutta pidentävät mallin parserointiaikaa Papervisionissa. Kannattaakin harkita milloin tarvittavan mallin lataa ohjelmaan, sillä pitkän animaation lataus pysäyttää ohjelman vähäksi aikaa. (Papervision 2010.)

Animaatioita tehdessä kannattaa kuitenkin muistaa, että aina Collada-animaatio ei ole järkevin vaihtoehto. Useassa tapauksessa mallin jonkin osan pyörittäminen tai liikuttaminen onnistuu paremmin ActionScriptillä. Kontrollointia varten Collada-mallin elementit ja ryhmät kannattaa nimetä kunnolla, jotta niihin on helpompi viitata myöhemmin koodissa. Eri osien nimet löytyvät jälkikäteen tarkastelemalla Collada-tiedostoa tekstieditorilla. (Papervision 2010.)

5.13 Interaktiivisuus

Papervisionissa 3D-objekteihin on mahdollista lisätä interaktiivisuutta eri tavoin. Papervision käyttää hyödyksi kahta luokkaa interaktiivisuuden

luontiin 3D-maailmassa. Nämä ovat InteractiveSceneManager, VirtualMouse. (Papervision 2010.)

Koska renderöity 3D-maailma on ruudulla vain bittikartta-dataa, eikä erillisiä MovieClippejä, ei interaktiivisuuden lisääminen yksittäisiin kappaleisiin onnistu suoraan. InteractiveSceneManager-luokka testaa hiiren osumista renderöityyn bittikartta-dataan ja simuloi tilannetta, jossa 3D-objektit olisivat tavallisia MovieClippejä. InteractiveSceneManagerin tapahtumista löytyy kaikki tavallisimmat tapahtumat, jotka löytyvät myös normaaleista MouseEvent-tapahtumista. (Papervision 2010.)

Jos hiiren paikka halutaan tietää tarkasti kappaleen pinnan materiaalina käytetyssä MovieClipissä, InteractiveSceneManager käyttää hyödyksi VirtualMouse-luokkaa. VirtualMouse laukaisee MovieClipin omat tapahtumat, kuten hiiren viennin päälle ja hiirellä painamisen. VirtualMousea käytettäessä ei siitä tarvitse välittää itse ollenkaan, sillä se toimii automaattisesti InteractiveSceneManagerin kanssa. MovieClippeihin voi lisätä kuuntelijoita kuten tavallisestikin ja ne toimivat normaalisti. VirtualMouse pääsee myös käsiksi MovieClipin sisällä oleviin MovieClip- ja Button-instansseihin. VirtualMousella on kuitenkin joitain rajoituksia. Se ei voi muuttaa Button-instanssin ulkonäköä eri vaiheiden mukaan, vaihtaa objektin fokusta tai käsitellä hiiren rullaan liittyviä tapahtumia. Luokka ei pysty myöskään lukemaan hiiren koordinaatteja MovieClipissä, joita tietyt komponentit käyttävät hyödykseen. (Papervision 2010.)

Mouse3D-luokkaa käytetään hiiren tietojen saamiseen 3D-maailmasta. Luokan sisältämiä tietoja päivitetään InteractiveScene3DEvent-tapahtumien yhteydessä. Mouse3D-luokka jatkaa DisplayObject3D-luokkaa, joten se sisältää paljon tietoja kuten hiiren paikan ja kosketetun kappaleen skaalauksen. Hiiren tietoja voi myös kopioida nopeasti toiseen kappaleeseen käyttämällä copyTransform- ja copyPosition-funktioita. Mouse3D–luokkaa käytetään tekemällä siitä instanssi ja määrittämällä InteractiveSceneManagerin mouse3D-arvo käyttämään kyseistä instanssia. Mouse3D-luokan tiedot eivät päivity ellei Mouse3D-luokan staattinen enabled-arvo ole tosi. Mouse3D-luokka tarvitsee myös toimiakseen, että Viewport3D-instanssin ja kappaleen materiaalin interactive-muuttujat on asetettu todeksi. (Papervision 2010.)

Yksinkertaisin ja nopein tapa lisätä interaktiivisuutta Papervisionissa on lisätä hiiren kuuntelijat suoraan renderöityyn kuvaan eli Viewport3D-instanssiin. Tällä tekniikalla hiiren kuuntelijat asetetaan samalla tavalla kuin mihin tahansa Spriteen tai MovieClippiin. Tämä on hyvä tapa interaktiivisuuden lisäämiseen esimerkiksi silloin kun näkymässä on vain yksi objekti tai kaikista objekteista tapahtuu sama asia. (Papervision 2010.)

ViewportLayereihin on myös mahdollista lisätä normaaleja hiiren kuuntelijoita. Tämä tapa mahdollistaa interaktiivisuuden lisäämisen hieman monimutkaisimpiin kohtauksiin. Jos eri objektit on lisätty omille kerroksilleen, voi niille lisätä omat kuuntelijat, jolloin painettaessa eri

objekteista tapahtuu eri asioita. Interaktiivisuuden lisääminen suoraan Viewport3D-instanssiin tai ViewportLayereihin on myös nopeuden kannalta kevyin tapa. Näitä tapoja kannattaakin käyttää ensisijaisesti projektissa, jos se on vain mahdollista. (Papervision 2010.)

Mikäli kohtaus on hyvin monimutkainen, eikä tarvittavia objekteja pysty asettamaan omille kerroksille, voi kuuntelijoita lisätä itse objekteihin InteractiveSceneManager-luokan avulla. Tapahtumat eivät ole enää normaaleja MouseEvent-tapahtumia vaan InteractiveScene3DEvent-tapahtumia. Nämä tapahtumat vaativat toimiakseen, että Viewport3D-instanssin ja kappaleen materiaalin interactive-muuttujat asetetaan todeksi. InteractiveScene3DEvent-tapahtumista löytyvät lähes kaikki tarvittavat tapahtumat, kuten hiirellä painaminen ja hiiren vienti ulos objektista. Tämän tavan kanssa on mahdollista käyttää Mouse3D-luokkaa, jolloin esimerkiksi hiiren paikan pystyy tietämään 3D-maailmassa. Luokan avulla pystyy lukemaan myös erikoisempaa tietoa kuten pinnan normaalivektorin kohdasta, jossa hiiri koskee kappaletta. Näiden tietojen luovalla hyödyntämisellä voi saada aikaan mielenkiintoisia toiminnallisuuksia. (Papervision 2010.)

Monipuolisin tapa lisätä kohtaukseen interaktiivisuutta on lisätä hiiren kuuntelijat kappaleiden materiaalien sisällä oleviin MovieClippeihin. Tämä vaatii sen, että kappaleen materiaaliksi on määritetty MovieMaterial tai MovieAssetMaterial. Tämän lisäksi kappaleen materiaalin ja Viewport3D-instanssin interactive-muuttujien on oltava tosia. InteractiveScene3DEvent- ja VirtualMouse-luokat mahdollistavat, että materiaaleihin ja niiden sisällä oleviin MovieClippeihin on mahdollista lisätä normaaleja MouseEvent-kuuntelijoita. MovieMaterial-luokkaa käytettäessä kuuntelijat lisätään suoraan materiaalina käytetyn MovieClipin instanssiin. MovieAssetMaterial-luokkaa käytettäessä kuuntelijat lisätään kyseisen luokan instanssin movie-muuttujaan, joka on referenssi materiaalina käytettyyn MovieClippiin. Jos kappaleen materiaalin halutaan muuttavan ulkonäköä esimerkiksi silloin kun hiiri tuodaan kappaleen päälle, tulee materiaalin animated-arvo olla tosi. (Papervision 2010.)

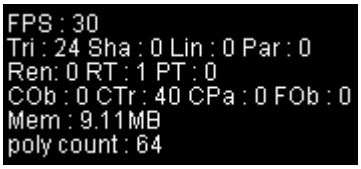
6. OHJEITA PAPERVISIONIN KÄYTTÖÖN

6.1 Optimointi

Mahdollisimman hyvän käyttökokemuksen saamiseksi on tärkeää, että ohjelma on optimoitu hyvin, jolloin se toimii mahdollisimman monella käyttäjällä sulavasti. Optimoinnissa voi ottaa oppia vanhoista 3D-peleistä, joissa koneiden suorituskyky tuli helposti vastaan. Seuraavassa kerron vinkkejä joiden avulla pystyy parantamaan ohjelman suorituskykyä.

Optimointia helpottamaan Papervision tarjoaa StatsView-luokan, jonka avulla voi seurata ohjelman tärkeimpiä tietoja, kuten kuvataajuutta, muistin käytön määrää, sekä kohtauksen monikulmioiden määrää. StatsView-luokan käyttö on erittäin helppoa. Luokasta tehdään instanssi,

joka lisätään kohtaukseen. Luokan tiedot tulevat suoraan määritetystä renderöintimoottorista ja päivittyvät automaattisesti. Ainoastaan kohtauksen monikulmioiden lukumäärän joutuu päivittämään manuaalisesti ajamalla updatePolyCount-funktion. (Papervision 2010.)



KUVA 39 *Esimerkki StatsView-luokan ulkonäöstä*

StatsView-instanssin koon pienentämiseksi ruudulla, termeistä on käytetty lyhenteitä, minkä takia niistä voi olla vaikea sanoa, mikä niiden merkitys on. Seuraavassa on kerrottu, mitä kyseiset lyhenteet tarkoittavat. (Papervision 2010.)

FPS – Kuvataajuus.
Tri – Pintojen määrä, sisältää myös täyttämättömät pinnat.
Sha – Varjostettujen pintojen määrä.
Lin – Piirrettyjen viivojen määrä. Tämä muuttuja ei ole vielä käytössä.
Par – Piirrettyjen partikkelien määrä.
Ren – Piirrettyjen pintojen määrä. Tämä muuttuja ei ole vielä käytössä.
RT – Renderöintiin käytetty aika.
PT – Projektioon käytetty aika.
COb – Kuvan ulkopuolelle jääneiden kappaleiden määrä.
CTr – Kuvan ulkopuolelle jääneiden pintojen määrä.
CPa – Kuvan ulkopuolelle jääneiden partikkelien määrä.
FOb – Kuvasta piilotetut kappaleet ViewportObjectFilter-luokan avulla.
Mem – Projektin käyttämän muistin määrä.
Poly count – Kohtauksessa olevien monikulmioiden määrä.

Flashin Stage-luokan quality-arvo kannattaa pitää yleensä matalana tai kohtalaisena. Tämä lisää ohjelman suorituskykyä huomattavasti projekteissa, jossa ruudulla on paljon piirrettävää. Varsinkin kamera-ajoissa tämä on hyödyllinen, sillä liikkeen takia esimerkiksi antialiasoinnin puuttumista ei huomaa niin helposti. Mikäli ruudulla on jotain käyttöliittymä elementtejä, kannattaa ne muuttaa dynaamisesti bittikarttakuviksi, jolloin antialiasoinnin puute ei näy niissä.

Kohtausta ei kannata renderöidä turhaan koko ajan, jos kamera ei liiku eikä kohtauksessa ole muutakaan liikettä. Jos kohtausta ei renderöidä, ei se myöskään vie yhtään laskentatehoja ja ruudulla näkyy kuitenkin vielä viimeisin renderöity kuva. Kannattaa harkita tekeekö projektiin animoiduille kohteille omat Viewport3D ja Scene3D-instanssit. Tämän ansiosta paikallaan olevia kohteita ei tarvitse renderöidä, vaikka ruudulla tapahtuisikin liikettä. Renderöintimoottorin renderLayers-funktion avulla on myös mahdollista renderöidä vain tietyt ViewportLayerit, mikä ajaa saman asian kuin edellinen tekniikka. (Papervision 2010.)

Monimutkaisten 3D-mallien monikulmioiden määrää kannattaa karsia niin paljon kuin vain voi, sillä suuret monikulmioiden määrä ruudulla vievät paljon laskentatehoa. Myös yksinkertaisista malleista kannattaa vähentää monikulmioita, jos niitä esiintyy ruudulla paljon. Primitiivejä käytettäessä, kannattaa niiden segmenttien määrä olla vain niin korkea kuin on tarpeellista.

Papervision tarjoaa SimpleLevelOfDetail-luokkan, jonka avulla kappaleen geometriaa voi vaihtaa, riippuen siitä kuinka kaukana se on kamerasta. Kun kappaleet pienenevät ruudulla, ei niistä erota yhtä paljon yksityiskohtia kuin jos ne olisivat lähellä. Kappaleiden monikulmioiden määrää onkin hyvä laskea asteittain kun ne pienenevät. (Papervision 2010.)

Aina ei ole tarpeellista näyttää asioita, jotka ovat liian kaukana kamerasta. Camera3D:n far-arvo määrittää kuinka pitkällä olevaa geometriaa ei enää renderöidä. Tämä voi kuitenkin näyttää huonolta, kun geometriaa ilmestyy tyhjästä. Parempi vaihtoehto tähän voi olla Papervisionin FogFilter-luokka. Suodatin liitetään renderöintimoottoriin ja sen avulla tarpeeksi pitkällä olevaa geometriaa ei renderöidä. Suodatin kuitenkin häivyttää rajan, jolloin geometria ilmestyy ”sumusta” asteittain. (Papervision 2010.)

Tekstuurien käytössä on huomattava muutamia asioita, jotka voivat olla merkittävä osa projektin optimointia. Tekstuurien koolla on iso merkitys niiden piirtonopeudelle. Tekstuurien kannattaa olla mahdollisimman pieniä. Liian pienet tekstuurit tekevät kuitenkin kohteesta huonon näköisen. Vaikka tekstuuri olisi osittain läpinäkyvä, joudutaan se silti laskemaan samalla tavalla kuin täytetty tekstuuri. Tämän takia tekstuurien reunoilta on hyvä karsia mahdolliset turhat läpinäkyvät osat.

Valojen ja varjostimien käyttö 3D-kappaleissa vähentää suorituskykyä. Kannattaakin harkita voiko kappaleen tekstuuriin lisätä valmiiksi varjoisat ja valoisat kohdat kuvankäsittelyohjelmassa. Tämä on hyvä vaihtoehto esimerkiksi silloin kun valo ei liiku suhteessa kappaleeseen, eikä kappaleen pinta ole kiiltävä.

Joskus 3D-maailmassa ei ole tarpeen käyttää lainkaan geometrisia kappaleita. MovieClippien käyttö 3D-kappaleen sijasta onnistuu tekemällä kohtaukseen tyhjä DisplayObject3D-instanssi ja keskittämällä MovieClip sen paikkaan ruudulla käyttäen DisplayObject3D:n screen-objektia. MovieClipin skaalauksessa voi käyttää hyödyksi DisplayObject3D:n etäisyyttä kamerasta. MovieClipin käytössä tulee kuitenkin ongelmaksi sen syvyys-suuntaisen järjestyksen vaihto muiden kappaleiden kanssa, minkä takia sitä ei voi hyödyntää kaikissa tilanteissa. (Papervision 2010.)

Mikäli ruudulla on useita eri kappaleita, kannattaa harkita Viewport3D-luokan ViewportLayerien käyttöä. Kun jokainen kappale on omalla kerroksellaan, ei ole tarvetta laskea jokaisen kappaleen jokaisen monikulmion syvyys-suuntaista piirtojärjestystä. Tämä lasketaan ainoastaan jokaisen kerroksen sisällä oleville monikulmioille. Eri kerrokset voi tämän jälkeen järjestää syvyys-suunnassa haluamaansa

järjestykseen tai automaattisesti riippuen niiden etäisyydestä kameraan. (Papervision 2010.)

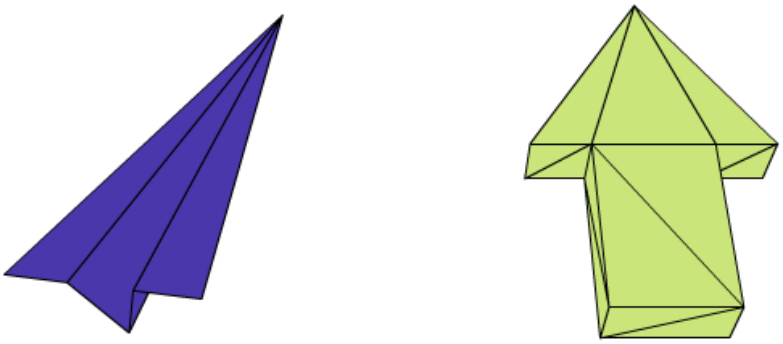
6.2 Papervisionin erikoisominaisuuksia

Papervisionissa on paljon mielenkiintoisia erikoisominaisuuksia, joiden käyttö tietyissä tilanteissa helpottaa projektin kehitystä ja lisää sen näyttävyyttä.

6.2.1 Erikoisobjektit

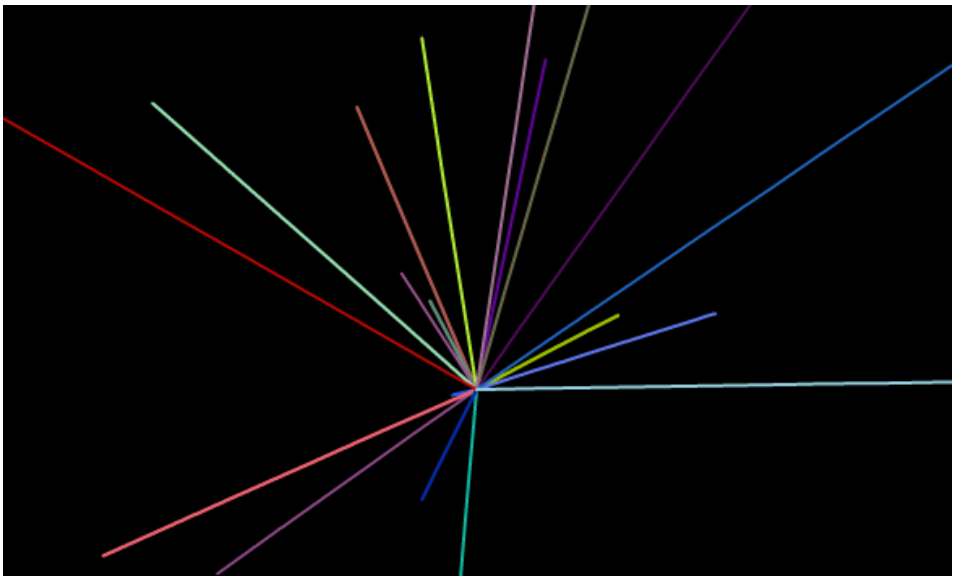
Papervisionin primitiiveihin kuuluu paperilentokone (PaperPlane). Paperilentokoneen konstruktorissa voi määrittää sen materiaalin ja skaalauksen. Paperilentokone käy hyvin esimerkiksi väliaikaisobjektiksi tai testaukseen, sillä siitä näkee helposti sen tämän hetkisen suunnan ja kierron. (Papervision 2010.)

Toinen hyödyllinen primitiivi on nuoli (Arrow). Nuolen konstruktorissa voi määrittää vain sen materiaalin. Nuoli on hyödyllinen vaikka lopullisessa sovelluksessa, esimerkiksi suunnan näytössä. (Papervision 2010.)



KUVA 40 *Kuva paperilentokone- ja nuoliprimitiiveistä*

Joskus projektissa voi tulla tarve piirtää viivoja kolmiulotteisessa maailmassa. Tätä varten Papervisionissa on Lines3D-luokka. Viivoille pystyy määrittämään paksuuden, värin ja läpinäkyvyyden. (Papervision 2010.)

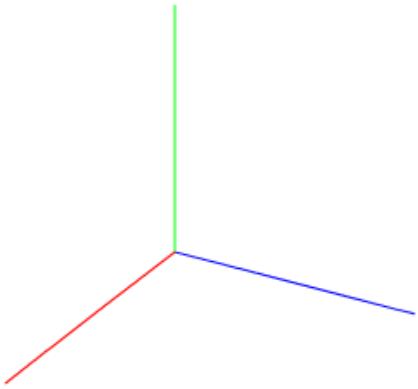


KUVA 41 *Esimerkki sovelluksesta, jossa on piirretty viivoja Lines3D-luokan avulla*

Papervisionilla tehdyillä sivuilla on pienenä trendinä nauhojen käyttö. Papervisionissa ei ole kuitenkaan vakiona luokkaa, jolla voisi luoda nauhoja kolmiulotteiseen maailmaan. Internetissä on kuitenkin valmiita luokkia, joiden avulla näitä voi tehdä. Justin Windlen Ribbon3D-luokka on erittäin hyvä vaihtoehto. Luokkaa on helppo käyttää ja siinä on kaikki tarpeelliset säädöt. Luokassa voi määrittää nauhan materiaalin, leveyden ja pituuden. (Windle 2008.)

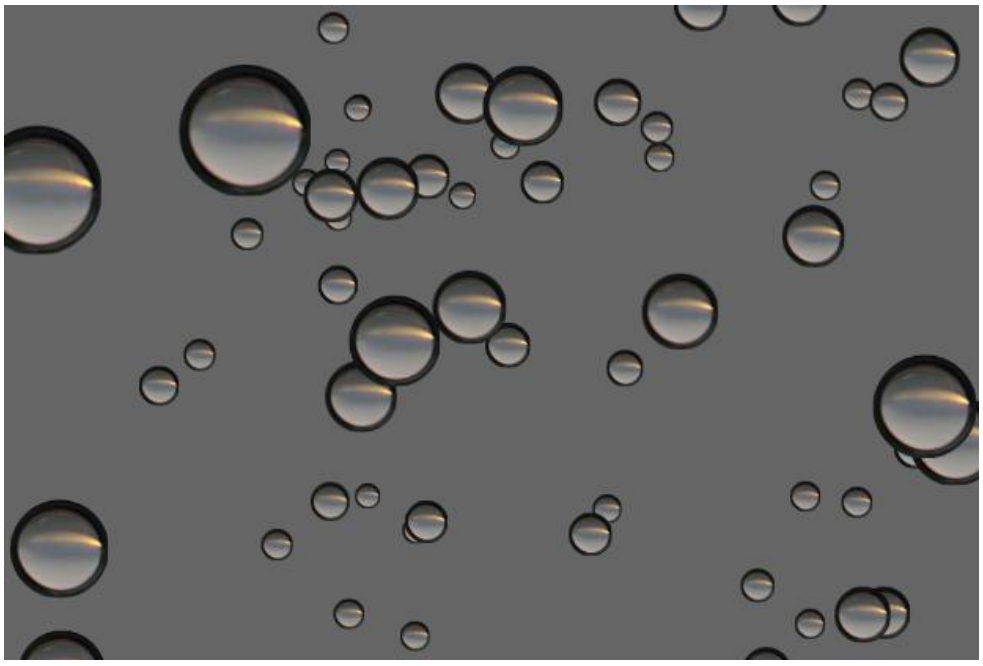
KUVA 42 *Esimerkki sovelluksesta, jossa on käytetty nauhoja*

Papervisionissa on UCS-luokka, joka on lyhenne sanoista User Coordinate System. Luokka piirtää kohtaukseen kolme halutun pituista viivaa, jotka kuvaavat koordinaatti-akseleita. Luokka on hyödyllinen lähinnä kehitysvaiheessa, jossa sille on käyttöä aina kun koordinaattiakselien suunnat ovat epäselviä. Luokkaa voi käyttää kuitenkin kekseliäästi muihinkin tarkoituksiin. (Papervision 2010.)



KUVA 43 *UCS-objektissa vihreä viiva tarkoittaa Y-akselia, punainen viiva X-akselia ja sininen viiva Z-akselia*

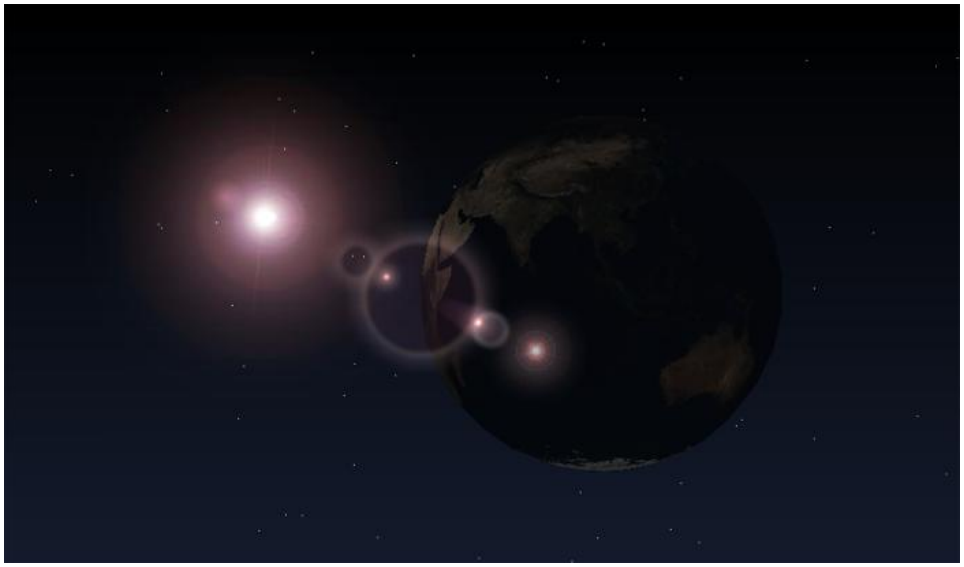
ParticleField-luokka luo laatikon muotoiselle alueelle partikkeleita, jotka katsovat aina suoraan kameraan. Luokan konstruktorissa voi määrittää partikkelien materiaalin, laatikon koon, partikkelien tiheyden ja partikkelien koon. Partikkelien materiaalin määrityksessä on kolme vaihtoehtoa. Yksinkertaisin vaihtoehto on ParticleMaterial-luokka, joka tekee partikkeleista neliöitä tai ympyröitä. Partikkeleista tulee yksivärisiä ja niiden läpinäkyvyyden voi määrittää. MovieAssetParticleMaterial luo partikkelien ulkonäön Flashin kirjastosta löytyvästä symbolista. BitmapParticleMaterial luo partikkelien ulkonäön bittikartta-datasta. ParticleField-luokkaa voi käyttää tuomaan projektiin yksityiskohtia, kuten esimerkiksi lumisateen tekoon. (Papervision 2010.)



KUVA 44 *Kuvassa olevassa sovelluksessa on käytetty ParticleField-luokkaa. Partikkelien materiaaleina on käytetty MovieAssetParticleMaterial-luokkaa*

LensFlare-luokalla projektiin voi tehdä valo- ja videokuvauksesta tutun linssiheijastuksen. Ilmiössä kuvaan tulee erinäköisiä peräkkäisiä renkaita, jotka muuttavat paikkaa kun kameraa käännetään. Renkaina voi käyttää

MovieClippejä, Spritejä tai bittikarttakuvia. Renkaiden paikka lasketaan luokalle annetun valonlähteen suhteesta kameraan. Efekti piirretään Viewport3D:n ViewportLayerille. Renkaiden paikkaa päivitetään manuaalisesti updateFlare-funktiolla, jossa pystyy myös testaamaan tuleeko jonkin objekti valonlähteen ja kameran väliin, jolloin linssiheijastusta ei synny. LensFlare on mukava tapa lisätä kohtauksen realistisuutta. (Papervision 2010.)



KUVA 45 *Esimerkki LensFlare-luokan käytöstä*

6.2.2 DisplayObject3D-luokan ominaisuuksia

DisplayObject3D-luokka sisältää paljon ominaisuuksia, jotka helpottavat erinäisten toiminnallisuuksien teossa. Seuraavassa on kerrottu kyseisiä ominaisuuksia ja niiden käyttöesimerkkejä.

DO3D-objektin lookAt-funktio kääntää kyseisen objektin niin, että se katsoo toista DO3D-objektia. Tällä tavalla saa esimerkiksi 3D-objektit katsomaan kameraan. Tasojen kanssa on kuitenkin huomattava, että ne katsovat poispäin kohteesta. Tämän takia ne on funktion käyttämisen jälkeen käännettävä vastakkaiseen suuntaan tai materiaali on asetettava kaksipuoliseksi. (Papervision 2010.)

Kahden DO3D-objektin välistä matkaa voi mitata distanceTo-funtktiolla. Funktiota voi käyttää esimerkiksi 3D-objektin häivyttämiseen pois näkyvistä tai sumentamiseen, jos se menee tarpeeksi kauas kamerasta. (Papervision 2010.)

Väillä on tarpeellista tietää DO3D-objektin paikka Flashin omassa kaksiulotteisessa koordinaatistossa. Suorituskyvyn parantamiseksi Papervision ei laske näitä koordinaatteja automaattisesti. Koordinaatit saadaan kuitenkin tietoon ajamalla calculateScreenCoords-funktio. Myös autoCalcScreenCoords boolean-arvon asettaminen todeksi auttaa, jolloin Papervision laskee objektin koordinaatteja aina kun ne muuttuvat. Tällöin erillistä funktiota ei tarvitse ajaa. Koordinaatit tallennetaan DO3D-

objektin screen-objektiin, josta ne ovat luettavissa. Tätä ominaisuutta voi käyttää esimerkiksi, jos objektille haluaa tooltip-tyyppisen laatikon. (Papervision 2010.)

DO3D-objektin osumista toisiin kappaleisiin tai tiettyyn pisteeseen voi seurata hitTestObject- ja hitTestPoint-funktioilla. Funktioiden toimintaperiaate on sama kuin Flashin omat MovieClipin funktiot. hitTestObject testaa kahden laatikon päällekkäisyyttä, jotka on muodostettu testattavien DO3D-objektien ääriarvoista. hitTestPoint-funktio testaa tietyn pisteen osumista DO3D-objektin oikeaan muotoon. HitTest-funktiot ovat erittäin käytettyjä esimerkiksi pelien teossa, jossa tarvitaan monesti tietää kappaleiden osumista toisiinsa. (Papervision 2010.)

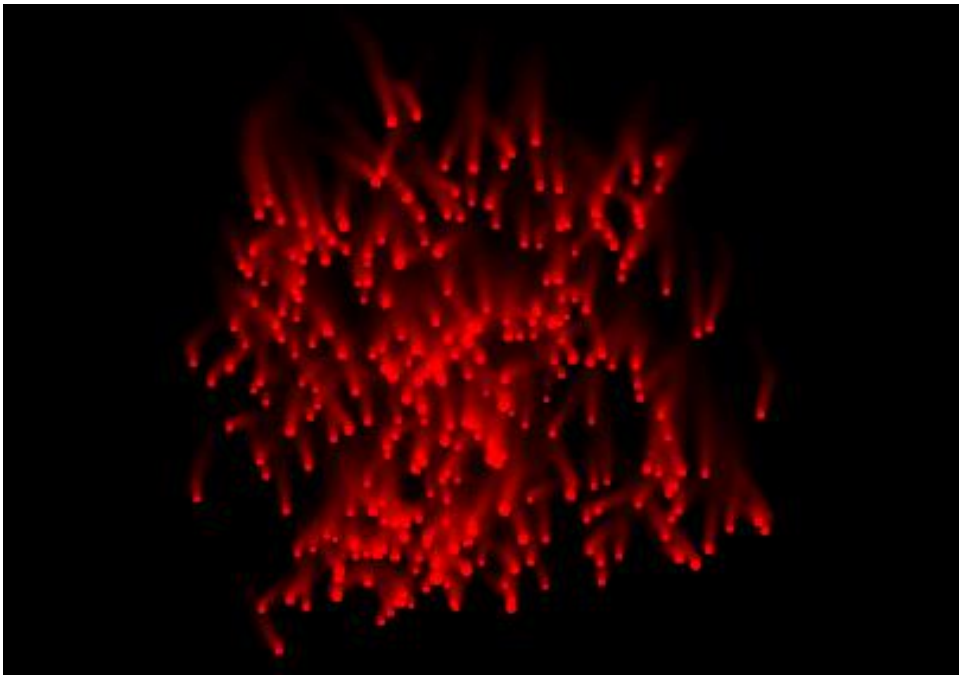
DO3D-objektille ei ole suoraan mahdollista asettaa suodattimia tai säätää läpinäkyvyyttä. Nopea tapa tähän on kuitenkin asettaa objektin useOwnContainer arvo todeksi. Tämä luo dynaamisesti ViewPort-instanssiin uuden kerroksen ja siirtää kyseisen objektin sinne. Tämän jälkeen objektiin on mahdollista suoraan lisätä esimerkiksi epäterävyyttä. On kuitenkin tärkeä huomata, että mikäli kohtauksessa on muita objekteja, täytyy niidekin olla omilla kerroksilla, jotta kappaleiden järjestäminen syvyys-suunnassa onnistuu. (Papervision 2010.)

DO3D-objekteja pystyy monistamaan clone-funktiolla. Funktio palauttaa DO3D-objektin, jonka pystyy tallentamaan uuteen muuttujaan ja lisäämään kohtaukseen. Uudelle objektille tulee sama materiaali ja transformaatio, kuin alkuperäisellä. Kloonausta voi käyttää esimerkiksi muuttuvan kappaleen eri vaiheiden tallentamiseen. (Papervision 2010.)

DO3D-objektilla on kaksi funktiota, jotka helpottavat tietyissä tilanteissa objektin siirtoa tai muokkausta. copyPosition-funktiolla DO3D-objekti on mahdollista asetta samaan kohtaan toisen objektin kanssa. copyTransform-funktiolla kopioi paikan, skaalauksen ja kierron toiselta objektilta. Nämä funktiot nopeuttavat esimerkiksi tilanteessa, jossa kahden objektin halutaan käyttäytyvän samalla tavalla. (Papervision 2010.)

Välillä DO3D-objektiin halutaan tallentaa omaa dataa. Tähän on tarjolla valmiiksi kaksi vaihtoehtoa. DO3D-objektin extra-objekti on mahdollista määrittää esimerkiksi taulukoksi, jolloin sinne voi tallentaa useampia muuttujia. Objekti on public-tyyppiä, joten sinne tallennetut arvot ovat luettavissa mistä tahansa. Toinen vaihtoehto oman datan tallentamiseen on userData-objekti. Objekti sisältää data-objektin, jonka tyyppiä ei ole valmiiksi määritelty. Tämä antaa mahdollisuuden tehdä vaikka ihan oma luokka datan varastoimiseksi. Esimerkiksi interaktiivisuutta tehdessä oman datan käyttö on hyödyllistä. Objekti voi käyttäytyä eri tavalla riippuen siihen tallennetuista muuttujista. (Papervision 2010.)

6.2.3 Efektien käyttö näkymässä



KUVA 46 *Liikkuvista kuutioista koostuvaan sovellukseen on lisätty BitmapColorEffect punaisella sävyllä ja epäterävyys BitmapLayerEffectin avulla*

Viewport3D-instanssiin on mahdollista lisätä efektikerroksia (BitmapEffectLayer), jotka käyttävät hyödyksi aikaisempaa renderöintidataa. Efektien avulla on esimerkiksi mahdollista tehdä kappaleelle epäterävä liikejälki, joka tulee kappaleen perässä. Tällä hetkellä efektejä on viisi erilaista, BitmapLayerEffect, BitmapColorEffect, BitmapFireEffect, BitmapMotionEffect ja BitmapPixelateEffect. Efektejä voi käyttää useampia samaan aikaan. (Papervision 2010.)

BitmapLayerEffect mahdollistaa Flashin tavallisten suodattimien lisäämisen efekti-kerrokselle, näitä suodattimia ovat esimerkiksi epäterävyys ja varjo. (Papervision 2010.)

BitmapColorEffectin avulla voi muuttaa efektin väriä. Normaalisti efekti on samanväristä kuin alkuperäinen renderöintidatakin. (Papervision 2010.)

BitmapFireEffect saa aikaan vääristävän efektin, joka muistuttaa animoituna tulta. Luokka sisältää paljon säätömahdollisuuksia, kuten efektin korkeuden, leviämisen määrän ja vääristymisen määrän. Efektiin voi määrittää jopa tulesta tulevan savun määrän. (Papervision 2010.)

BitmapMotionEffect piirtää vain muuttuneet pikselit. Jos ruudulla ei muutu mikään, ei ruudulle tule myöskään näkyviin mitään efektiä. Efektin luovalla käytöllä voi saada aikaan mielenkiintoisia lopputuloksia. (Papervision 2010.)

BitmapPixelateEffect tekee nimensä mukaisesti pikselöitymis-efektin, joka näyttää samalta kuin bittikarttakuvaa suurennettaisiin niin, että yksittäiset pikselit kasvaisivat isommiksi. Efektillä saa aikaan esimerkiksi

poliisisarjoista tutun rikollisen kasvonpiirteiden piilottamisen. (Papervision 2010.)

BitmapEffectLayer-luokkaa on mahdollista käyttää myös muihin tarkoituksiin. BitmapEffectLayer-instanssiin voi piirtää yksittäisiä pikeleitä 3D-koordinaatistoon. Tämä tapahtuu käyttämällä Pixels- ja Pixel3D-luokkia. Pixels-instanssi toimii säiliönä yhdelle tai useammalle Pixel3D-instanssille. Pikselien värin ja paikan voi itse määrittää. Tällä tekniikalla voi luoda esimerkiksi kohokuvia bittikartta-datasta tai tehdä 3D-mallista versio, joka koostuu yksittäisistä pikseleistä. (Papervision 2010.)

6.2.4 Sound3D

Sound3D-luokan avulla pystyy simuloimaan äänilähdettä kolmiulotteisessa maailmassa. Sound3D-objekti muuttaa äänilähteen äänenvoimakkuutta ja panorointia riippuen sen suhteesta kameraan. Luokalle määritetään mitä ääntä se toistaa ja mikä on sen maksimi kuuluvuusalue. Luokan instanssien määrää ei ole rajoitettu. Sound3D-luokkien oikealla hyödyntämisellä on mahdollista tehdä mielenkiintoinen ja monipuolinen äänimaisema. Loppukäyttäjällä tarvitsee kuitenkin olla kunnollinen äänentoisto, jotta vaikutelmasta tulee onnistunut. Kaksi hyvin asetettua kaiutinta tai kuulokkeet ovat yleensä jo riittäviä. (Papervision 2010.)

6.3 Hyödyllisiä kolmansien osapuolten ohjelmia

Flashin ja Papervisionin kanssa on mahdollista käyttää useita kolmansien osapuolten tekemiä ohjelmia, jotka helpottavat työskentelyä tai lisäävät ominaisuuksia projektin kehittämiseen. Tässä kappaleessa on listattu joitain ohjelmia, jotka olen itse todennut erittäin käyttökelpoisiksi.

6.3.1 TweenMax

Flash-kehittäjän yksi tärkeimmistä työkaluista on luokkakirjasto, jolla voi pienellä vaivalla tehdä monipuolisia animaatioita. Eri vaihtoehtoja on monia. Itse olen kuitenkin päätynyt TweenMax–luokkakirjastoon, sillä siinä on useita hyviä ominaisuuksia, jotka erottavat sen kilpailijoistaan. TweenMaxin perusversio on ilmainen, mikäli tulevan ohjelman tai verkkosivujen käytöstä ei tulla perimään käyttömaksua. Maksullisessa versiossa tulee mukana muutama lisäominaisuus, mutta ne eivät ole merkittäviä. (GreenSock 2010.)

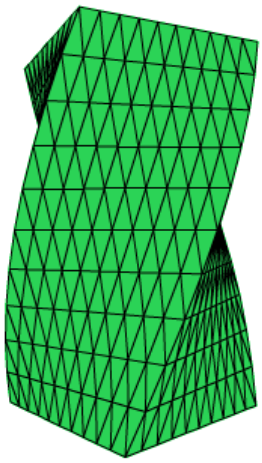
TweenMax on erittäin nopea, siinä on paljon ominaisuuksia, sitä on helppo käyttää, siinä on hyvä dokumentaatio ja se toimii saumattomasti Papervisionin kanssa. Papervisionissa voi TweenMaxin avulla esimerkiksi liikutella 3D-objekteja tai kameraa niin, että liike on sulavaa. (GreenSock 2010.)

TweenMaxista on olemassa ActionScript 2 ja 3 versiot. TweenMaxista on myös kaksi karsittua versiota, TweenLite ja TweenNano. Karsitut versiot ovat hyödyllisiä projekteissa, joissa lopullinen sovelluksen koko ratkaisee. Mikäli muutamien kilotavujen lisä ei ole merkityksellinen, kannattaa aina valita TweenMax. Tällöin on käytettävissä kaikki ominaisuudet, jos niitä tarvitaan. (GreenSock 2010.)

6.3.2 AS3Dmod

AS3Dmod on ActionScript 3:lla kirjoitettu 3D-objektien pinnanmuotojen muokkausta varten tarkoitettu luokkakirjasto. AS3Dmodia pystyy käyttämään Papervisionissa, Away3D:ssä, Sandy 3D:ssä ja Alternativa3D:ssä. AS3Dmodilla pystyy korvaamaan Collada-mallin animaatioita, jos ne muistuttavat AS3Dmodin tukemia muokkauksia. (Drozdz 2008b.)

AS3Dmodissa on tällä hetkellä seitsemän erilaista muokkausvaihtoehtoa. Bend-säädöllä on mahdollista taivuttaa 3D-objektia kaarevaksi määrätyn akselin suhteen. Noise-säätö lisää 3D-objektien kärkipisteisiin satunnaisen poikkeaman, jolloin 3D-objektista tulee epämuodostunut. Skew-säätö vinouttaa 3D-objektia yhden tai useamman akselin suhteen. Taper-säätö tekee objektista kartiomaisen portaattomalla säädöllä. Bloat-säätö työntää 3D-objektin kärkipisteet pois määrätyltä pallon muotoiselta alueelta. Perlin-säätö muokkaa 3D-objektia samalla tavalla kuin Noise. Perlin on kuitenkin pehmeämpi ja aaltomaisempi. Twist-säätö vääntää 3D-objektia kierteelle tietyn akselin ympäri. (Drozdz 2008b.)



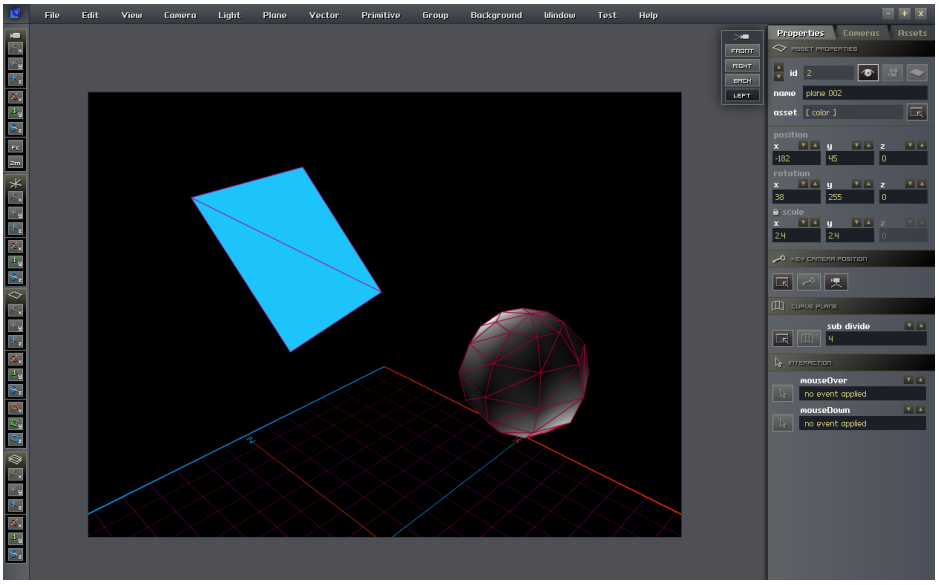
KUVA 47 *Kuva laatikosta, jossa on käytetty Twist-säätöä*

6.3.3 Fysiikkamoottorit

Sovelluksiin saa helposti näyttävyyttä lisäämällä kappaleiden keskinäistä vuorovaikutusta kolmiulotteisessa avaruudessa. Tähän tarkoitukseen on olemassa valmiita fysiikkamoottoreita. Näistä kaksi parasta ja tunnetuinta on Jiglibflash ja WOW-Engine. Molemmat ovat ilmaisia ja toimivat Papervisionissa, Away3D:ssä ja Sandy 3D:ssä. Näistä kahdesta Jiglibflash on kuitenkin valmiimpi, sillä se on porttaus C++ Jiglib -kirjastosta. Jiglibflashissä on myös parempi dokumentaatio ja siitä löytyy hyviä

esimerkkejä. Näistä syistä itse suosittelen Jiglibflashin käyttöä. (Jiglibflash 2009.)

6.3.4 VizualPV3D



KUVA 48 *VizualPV3D-projekti, jossa on näkyvillä taso- ja palloprimitiivi*

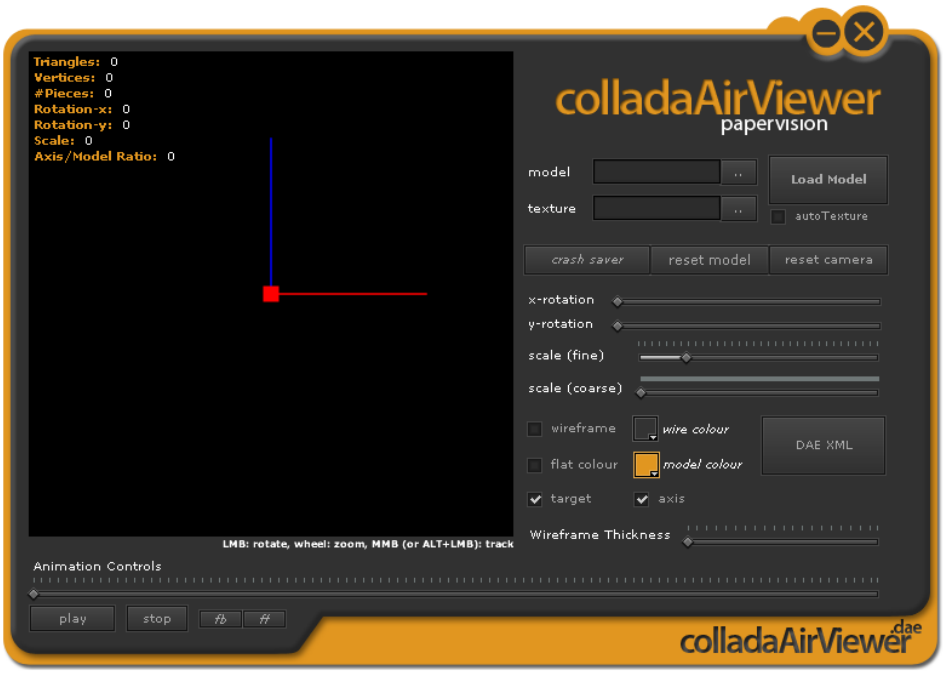
VizualPV3D on ilmainen ohjelma graafisella käyttöliittymällä, jonka avulla pystyy visualisoimaan 3D-näkymiä. Ohjelma pohjautuu Papervision 2.0 alpha -versioon, jonka avulla se renderöi näkymät. Ohjelman käyttöliittymä on selkeä ja se on suunniteltu niin, että ohjelman kaikki ominaisuudet tulevat selville nopeasti. (JUXT Interactive 2009.)

Ohjelmassa pystyy lisäämään kohtaukseen primitiivejä, valoja ja kontrolloimaan kohtauksen kameraa. Primitiivit on mahdollista teksturoida väreillä, kuvilla tai MovieClipeillä. Ohjelma ei kuitenkaan valitettavasti tue itse tehtyjen 3D-objektien käyttöä. (JUXT Interactive 2009.)

Ohjelman mukana tulee erittäin hyvä manuaali, jossa on kerrottu selitetty ohjelman ominaisuudet ja niiden käyttö. Ohjelman sisäisestä Help-osiosta löytyy vielä lisää tietoa. (JUXT Interactive 2009.)

VizualPV3D tallentaa kohtauksen lopputuloksen XML-muodossa, näin ohjelma ei ole riippuvainen kehitettävästä Papervision-versiosta. XML:ää täytyy kuitenkin itse osata parseroida niin, että sitä voi hyödyntää oman projektin rakentamisessa. XML:ää voi myös periaatteessa käyttää myös muissa moottoreissa, joista löytyy samat ominaisuudet. (JUXT Interactive 2009.)

6.3.5 3D-objektien esikatselu

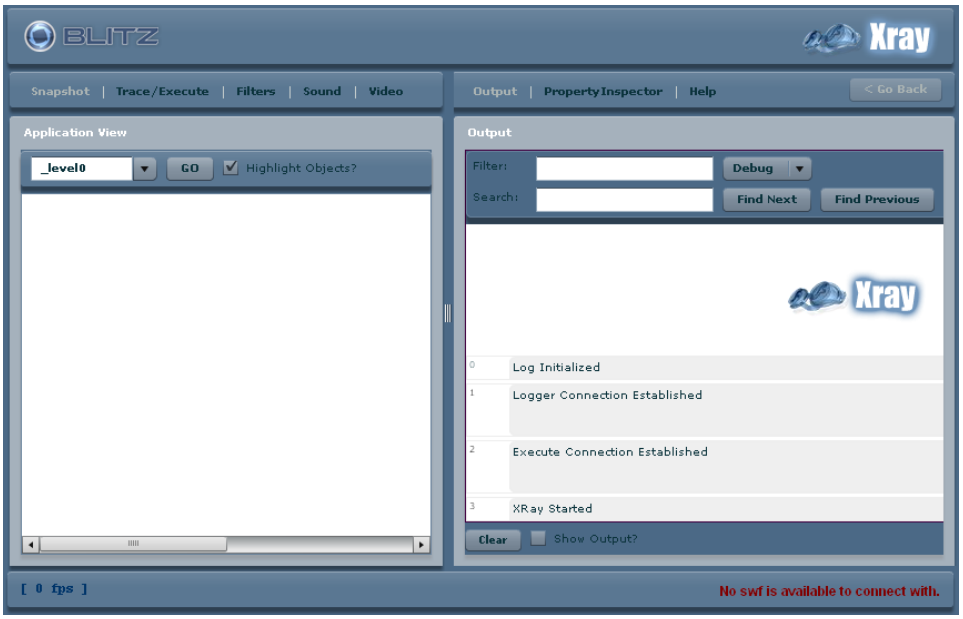


KUVA 49 *colladaAirViewer-ohjelman perusnäkymä*

Eri 3D-mallinnusohjelmat kirjoittavat yleensä poikkeavaa Collada-formaattia, jolloin mallista voi tippua ominaisuuksia pois tai se voi olla toimimatta kokonaan. Tämän takia on hyvä tarkistaa mallin toimivuus, ennen kuin sitä käyttää oikeassa projektissa. 3D-objektien esikatseluun suosittelen Anthony Scavarellin tekemää colladaAirViewer-sovellusta. Sovellus tukee valitettavasti vain Collada-formaattiin tallennettuja malleja, mutta muilta ominaisuuksiltaan ohjelma on hyvä. (Scavarelli 2009.)

Malleja voi tarkastella oikeilla tekstuureilla tai täytettyinä rautalankamalleina. Ohjelma tukee myös animaatioita ja niitä voi tarkastella askel kerrallaan. Ohjelmalla on myös mahdollista muokata ja tallentaa Collada-formaatin XML-dataa ja seurata muutosten vaikutuksia. (Scavarelli 2009.)

6.3.6 Xray



KUVA 50 *Xray-ohjelman perusnäkymä*

Xray on Blitz Labsin kehittämä ilmainen Flash-sovellus, jonka avulla voi tutkia toisen Flash-sovellusten rakennetta ja muokata objektien muuttujia reaaliajassa. Rakenteen tutkimista helpottaa huomattavasti jos sovelluksen objekteille on annettu kuvaavat nimet. (Grden 2007c.)

Ohjelma käyttää Flashin LocalConnection-luokkaa, jonka avulla kaksi samalla koneella ajettavaa Flash-sovellusta sovellusta pystyy välittämään dataa toisilleen. Xray toimii myös täysin Papervisionin kanssa. Sovelluksesta on olemassa ActionScript 2 ja 3 versiot. (Grden 2007c.)

Ohjelman käyttö on yksinkertaista. Kehitettävän projektin ylimmälle tasolle lisätään Xray-luokka tai komponentti. Tämän jälkeen projekti ajetaan selaimella palvelimelta tai käännettynä Flashistä. Ohjelmaa ei voi ajaa paikallisesti suoraan koneelta ilman asetusten muuttamista, sillä Flash Playerin turvamääritykset eivät sitä salli. Kun projekti on käynnissä, käynnistetään Xray:n AdminTool. Ajettavasta ohjelmasta tallennetaan sen hetkinen tila, jonka jälkeen projektista pääsee tutkimaan sen rakennetta ja muokkaamaan muuttujia. Muokkausten vaikutus näkyy heti projektissa. Toisten tekemiä sovelluksia on myös mahdollista tutkia lataamalla ne toisen Flash-sovelluksen sisään, johon on lisätty tarvittava Xray-luokka. (Grden 2007c.)

Ohjelman käyttö auttaa myös optimoinnissa, koska sillä voi esimerkiksi piilottaa objekteja ja katsoa mikä sovelluksessa vie eniten prosessointitehoja. (Grden 2007c.)

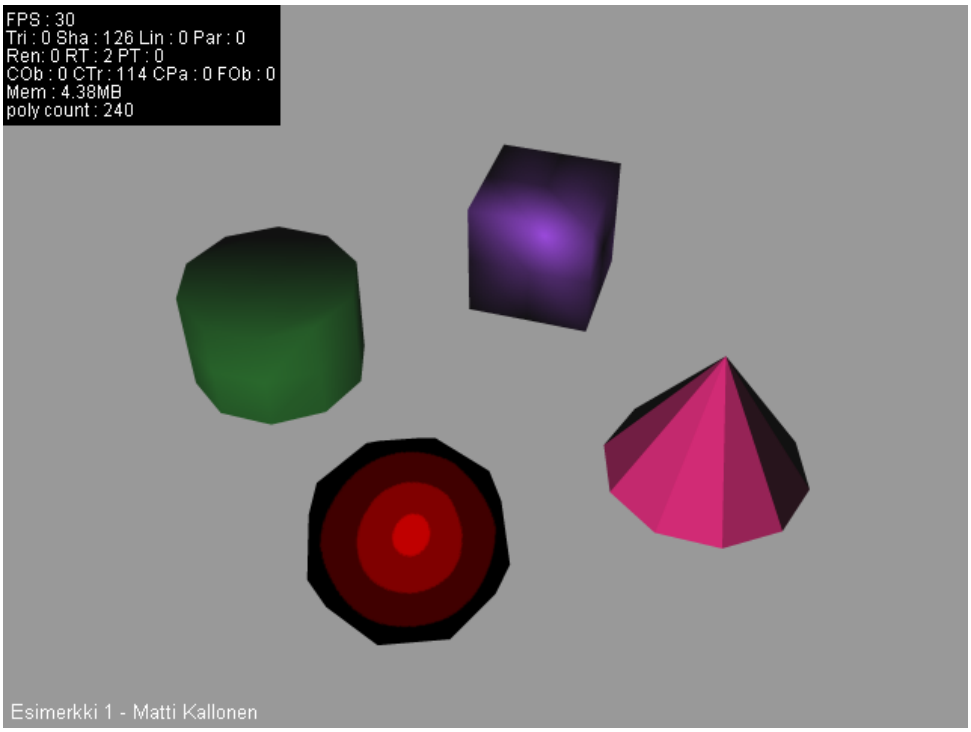
7. ESIMERKKIOHJELMAT

Esimerkkiohjelmissa kerron konkreettisilla esimerkeillä, miten Papervisionia käytetään Flashissä. Esimerkeistä on tarvittavat lähdetiedostot, joiden avulla voi kokeilla muutosten vaikutusta sovelluksissa. Esimerkit ovat koottu myös minimalistiselle HTML-sivulle, josta niitä on helppo selata ja tutkia ohjelman lähdekoodia. Ohjelmien lähdekoodit löytyvät myös liitteistä.

Ohjelmissa on käytetty eri tapoja saman asian aikaan saamiseksi, jotta lukijalle tulee tutuksi eri vaihtoehdot. Ohjelmien pääasiallinen tarkoitus on saada lukija ymmärtämään ohjelman toiminta, jonka vuoksi ohjelmissa on tehty ratkaisuja, jotka eivät välttämättä ole parhaita tapoja tehdä asioita, mutta ne ovat selkeämpiä ja tuovat esille eri käyttömahdollisuuksia.

Kaikissa ohjelmissa saa näkyviin StatsView-luokan painamalla välilyöntiä. Muut ohjelmien käyttämät kontrollit on kerrottu koontisivulla.

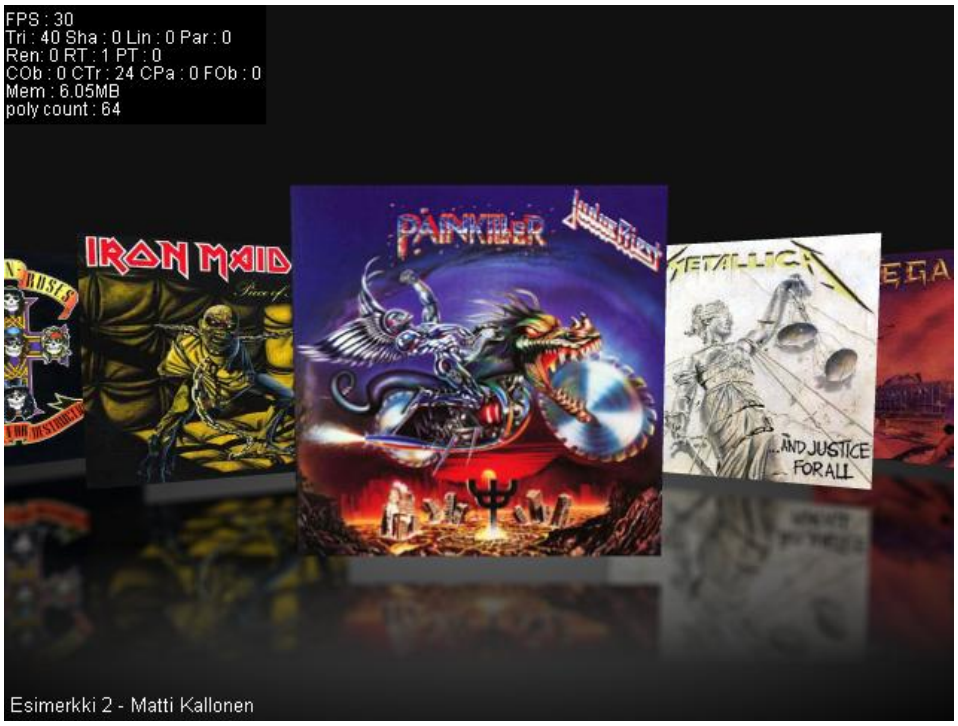
7.1 Esimerkkiohjelma 1 – Primitiivit ja varjostimet



KUVA 51 *Kuvakaappaus esimerkkiohjelmasta 1*

Ohjelmassa luodaan erilaisia primitiivejä ja lisätään niihin erilaisia varjostimia. Kappaleet vaihtavat väriä satunnaisesti kahden sekunnin välein. Ohjelman pääluokka jatkaa BasicView-luokkaa, joka luo monet kohtauksessa tarvittavat luokat automaattisesti.

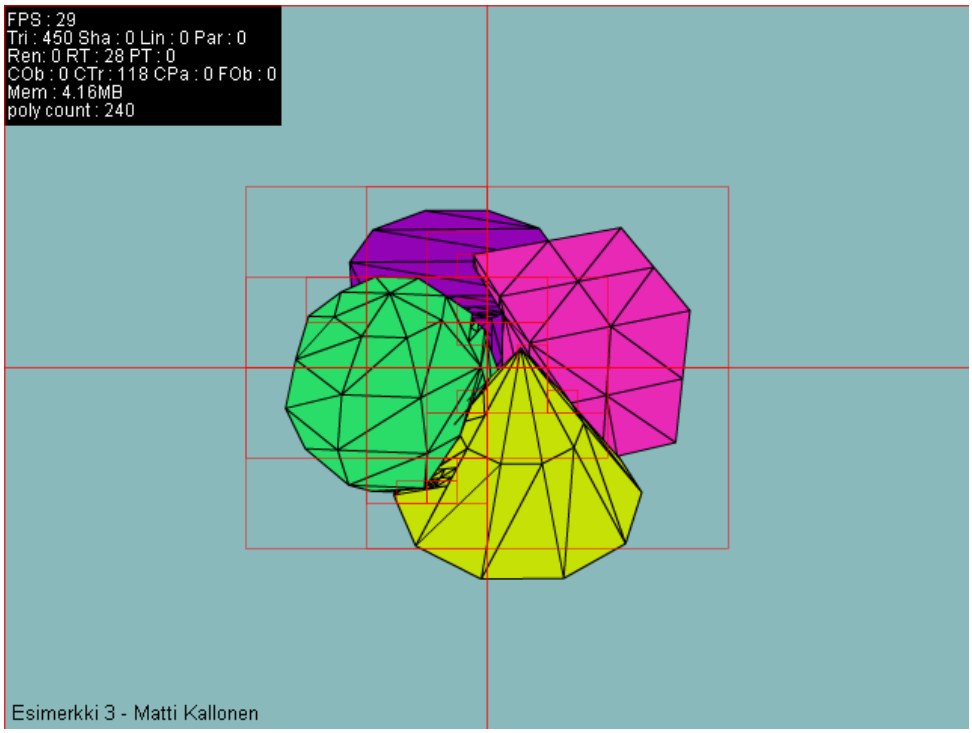
7.2 Esimerkkiohjelma 2 – Kuvakaruselli



KUVA 52 *Kuvakaappaus esimerkkiohjelmasta 2*

Ohjelmassa selataan musiikkilevyjen kansikuvia. Painamalla levyistä, ne kääntyvät ympäri ja takapuolella näkyy levyn artistin nimi. Ohjelmassa käytetään hyväksi ReflectionView-luokkaa, jonka avulla levyistä saadaan heijastukset automaattisesti. Projekti jatkaa Sprite-luokkaa, joten esimerkiksi kameraan viitatessa joutuu edessä käyttämään viittausta ReflectionView-instanssiin (view.camera). Levyt on tehty taso-primitiiveistä ja niiden materiaalit tulevat Flashin kirjastosta. Jokainen levy on omalla ViewportLayerillä, jolloin levyt eivät voi mennä sisäkkäin. Ohjelman animaatioihin on käytetty TweenMax-luokkakirjastoa.

7.3 Esimerkkiohjelma 3 – QuadrantRenderEngine-luokan käyttö



KUVA 53 *Kuvakaappaus esimerkkiohjelmasta 3*

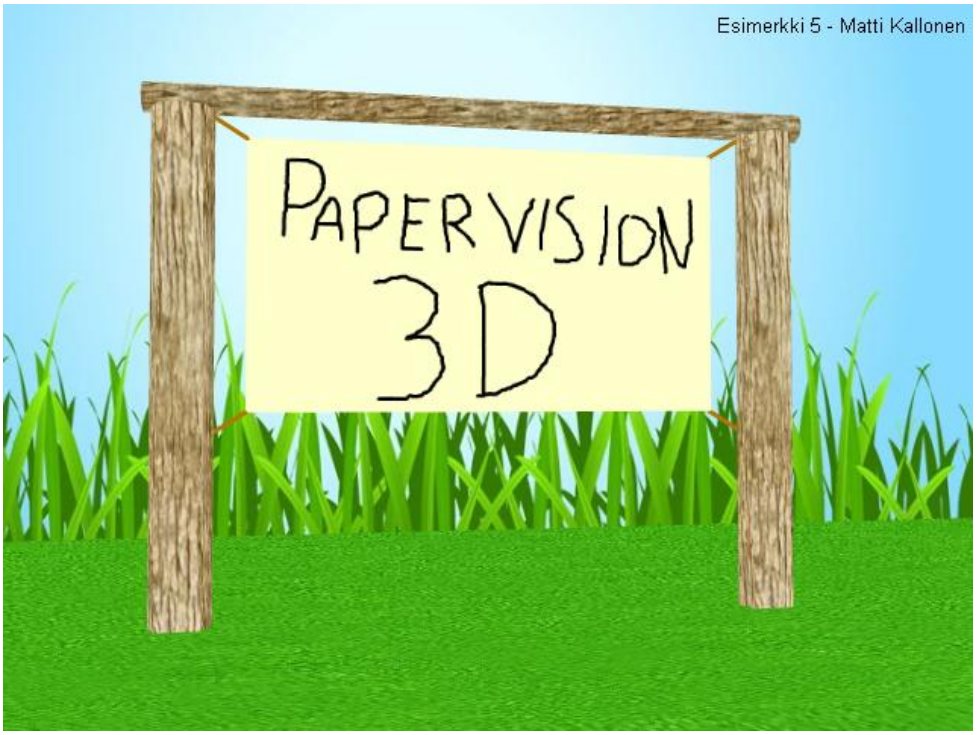
Ohjelmassa selvennetään QuadrantRenderEngine-luokan toimintaa. Ohjelmassa neljä primitiiviä menee osittain sisäkkäin, jolloin tavallisella renderöintimoottorilla kohtaus näyttää huonolta. Ohjelmassa voi vaihtaa kumpaa renderöintimoottoria käyttää. Ohjelmassa voi myös vaihtaa primitiivien materiaalin. QuadTreeVisualizer-luokan avulla ruudulle saa näkyviin QuadrantRenderEnginen rajauslaatikot. Ohjelmassa kaikki Papervisionin tarvitsemat luokat luodaan itse ilman BasicView-luokan apua.

7.4 Esimerkkiohjelma 4 – Animoitu Collada-malli

KUVA 54 *Kuvakaappaus esimerkkiohjelmasta 4*

Ohjelmassa kontrolloidaan Collada-mallin animaatioita. Collada-malliin on asetettu luita ja niiden avulla 3D Studio Max -ohjelmassa mallille on luotu kolme eri animaatiota. Animaatiot on tallennettu Collada-formaatin animaatioklipeiksi, jolloin niistä voidaan toistaa haluttu animaatio.

7.5 Esimerkkiohjelma 5 – Piirtäminen 3D-objektiin



KUVA 55 *Kuvakaappaus esimerkkiohjelmasta 5*

Ohjelmassa voi piirtää kankaalle hiiren avulla. Piirto tapahtuu lisäämällä viivoja taso-primitiivin materiaaliin. Kohtauksen muut kappaleet koostuvat primitiiveistä ja Lines3D-luokasta. Taustana käytetään MovieClippiä, jota asemoidaan nurmikko-tason takareunan mukaisesti.

8. POHDINTA

Vaikka opinnäytetyön tekeminen oli pitkä prosessi ja se vaati paljon aikaa, jonka olisi voinut käyttää toisin, oli opinnäytetyön tekeminen mielenkiintoista, antoisaa ja erittäin opettavaista. Lopputuloskin oli onnistunut ja se täytti sille asetetut tavoitteet.

Opinnäytetyön rakenne syntyi helposti ja alkuperäinen rakenne säilyi loppuun asti lähes muuttumattomana. Hauskinta opinnäytetyössä oli esimerkkiohjelmien tekeminen, uusien ominaisuuksien löytäminen Papervisionista ja näyttävien esimerkkien löytäminen internetistä. Haastavinta opinnäytetyössä oli omien kokemusten jäsentäminen selkeästi luettavaksi tekstiksi ja ajan tasalla ja paikkansapitävien lähteiden löytäminen kaiken tiedon keskeltä. Myös opinnäytetyön kirjoittamisen ja työn tekemisen yhteensovittaminen oli välillä hankalaa ja opinnäytetyön prosessissa oli muutamia jaksoja, jolloin opinnäytetyö ei edennyt juuri lainkaan.

Opinnäytetyön tekeminen opetti minulle paljon asioita päättäväisyydestä ja laajamittaisen työn kirjoittamisesta. Kirjoittamisprosessi vaatii syvällistä paneutumista ja häiriötekijöiden poissulkemista, mikä ei aina ole helppoa. Jos opinnäytetyön kirjoittamisen tekee pienissä osissa silloin tällöin, saattaa siitä tulla lukijalle hankalammin luettava, koska kokonaiskuva ei ole pysynyt kasassa prosessin aikana.

Kun opinnäytetyötä kirjoittaa aiheesta, joka kehittyy nopeasti, kannattaa aluksi tutustua huolella aiheen historiaan, nykytilanteeseen ja tulevaisuuteen. Jos näyttää, että aiheen tulevaisuus on epävarmaa tai sen käyttö on laskussa, kannattaa käydä läpi myös muita vaihtoehtoja, sillä aiheita löytyy varmasti muitakin ja niihin tutustumisesta saattaa olla enemmän hyötyä.

Opinnäytetyön tekeminen antoi laajan käsityksen 3D-grafiikan hyödyntämisestä verkkosivuilla ja sen potentiaalista tulevaisuudessa. Myös Papervisioniin tutustuminen antoi edellytykset hyödyntää sen käyttöä nykyisessä työssä ja kehittyä Flash-osaajana entisestään.

LÄHTEET

Allen C., Arnold W., Balkan A., Cannasse N., Grden J., Gunesch M., Hughes M., MacDonald R. & Zupko A. 2008. The Essential Guide to Open Source Flash Development. Yhdysvallat: Friends of ED.

Puhakka A. 2008. 3D-grafiikka. Helsinki: Talentum Media Oy.

MIT License. 2010. Wikipedia. Viitattu 10.1.2010.
http://en.wikipedia.org/wiki/MIT_License

Lindquist, J. 2008a. Papervision3D Getting Started FAQ. Viitattu 6.1.2010.
http://code.google.com/p/papervision3d/wiki/Getting_Started_FAQ

Ulloa, C. 2009. Papervision3D is Shifting Gears. Viitattu 6.1.2010.
http://blog.papervision3d.org/2009/10/13/papervision3d-is-shifting-gears/

Grden, J. 2007a. First Papervision3D Game! Viitattu 10.1.2010.
http://rockonflash.wordpress.com/2007/03/09/first-papervision3d-game/

Bateman, R. 2009a. Away3D – Features. Viitattu 10.1.2010.
http://away3d.com/features

Grden, J. 2007b. Papervision3D to merge Away3D features. Viitattu 10.1.2010.
http://blog.papervision3d.org/2007/05/16/papervision3d-to-merge-away3d-features/

Zadorozhny, A. 2007a. Away3D – Team. Viitattu 10.1.2010.
http://away3d.com/team

Sandy. 2009. Sandy 3D web-dokumentaatiot. Viitattu 10.1.2010.
http://www.flashsandy.org/

Sophie. 2009. Sophie 3D web-dokumentaatiot. Viitattu 10.1.2010.
http://www.sophie3d.com/

Obj. 2010. Wikipedia. Viitattu 10.1.2010.
http://en.wikipedia.org/wiki/Obj

Alternativa. 2010. Alternativa 3D web-dokumentaatiot. Viitattu 9.1.2010.
http://alternativaplatform.com/en/

McCauley, T. 2009a. Flash Player 10 Drawing API. Viitattu 10.1.2010
http://www.senocular.com/flash/tutorials/flash10drawingapi/

Adobe. 2009a. Flash Player 10 features. Viitattu 10.1.2010.
http://www.adobe.com/products/flashplayer/features/all_features/

Adobe. 2009b. Flash Player Version Penetration. Viitattu 10.1.2010.

http://www.adobe.com/products/player_census/flashplayer/version_penetration.html

QuickTime VR. 2009. Wikipedia. Viitattu 10.1.2010.
http://en.wikipedia.org/wiki/QuickTime_VR

Apple. 2009a. QuickTime VR. Viitattu 10.1.2010.
http://www.apple.com/quicktime/technologies/qtvr/

Statowl. 2009a. Quicktime Version Support. Viitattu 10.1.2010.
http://www.statowl.com/quicktime.php

Khronos Group. 2009. Khronos Details WebGL Initiative to Bring Hardware-Accelerated 3D Graphics to the Internet. Viitattu 10.1.2010.
http://www.khronos.org/news/press/releases/khronos-webgl-initiative-hardware-accelerated-3d-graphics-internet/

WebGL. 2010. Wikipedia. Viitattu 13.3.2010.
http://en.wikipedia.org/wiki/WebGL

Vector Markup Language. 2010. Wikipedia. Viitattu 10.1.2010.
http://en.wikipedia.org/wiki/Vector_Markup_Language

Canvas. 2010. Wikipedia. Viitattu 10.1.2010.
http://en.wikipedia.org/wiki/Canvas_(HTML_element)

HTML5. 2010. Wikipedia. Viitattu 10.1.2010.
http://en.wikipedia.org/wiki/HTML5

Google. 2009a. ExplorerCanvas. Viitattu 10.1.2010.
http://excanvas.sourceforge.net/

Microsoft Silverlight. 2010. Wikipedia. Viitattu 10.1.2010.
http://en.wikipedia.org/wiki/Microsoft_Silverlight

Hoffman, K. 2009. What's New in Silverlight 3. Viitattu 10.1.2010.
http://dotnetaddict.dotnetdevelopersjournal.com/new_silverlight3.htm

Statowl. 2009b. Microsoft Silverlight Version Support. Viitattu 10.1.2010.
http://www.statowl.com/silverlight.php

Statowl. 2009c. Java Plugin Usage Market Share. Viitattu 10.1.2010.
http://www.statowl.com/java.php

Java 3D. 2010. Wikipedia. Viitattu 10.1.2010.
http://en.wikipedia.org/wiki/Java_3D

Sun Microsystems. 2009. Java Downloads. Viitattu 10.1.2010.
http://www.java.com/en/download/manual.jsp

Unity. 2009. Unity web-dokumentaatiot. Viitattu 10.1.2010.

http://unity3d.com/

Papervision. 2010. Papervision 3D 2.1 dokumentaatio. Viitattu 10.1.2010. http://papervision3d.googlecode.com/svn/trunk/as3/trunk/docs/index.html

Rendering. 2010. Wikipedia. Viitattu 10.1.2010. http://en.wikipedia.org/wiki/Rendering_(computer_graphics)

3D rendering. 2010. Wikipedia. Viitattu 10.1.2010. http://en.wikipedia.org/wiki/3D_rendering

Rasterisation. 2010. Wikipedia. Viitattu 10.1.2010. http://en.wikipedia.org/wiki/Rasterisation

Polygon mesh. 2010. Wikipedia. Viitattu 10.1.2010. http://en.wikipedia.org/wiki/Polygon_mesh

Drozdz, B. 2008a. Pixel precision in Papervision3D. Viitattu 10.1.2010. http://www.everydayflash.com/blog/index.php/2008/07/07/pixel-precision-in-papervision3d/

Adobe. 2008a. ActionScript 3.0 Language and Components Reference – Graphics. Viitattu 10.1.2010. http://livedocs.adobe.com/flash/9.0/ActionScriptLangRefV3/flash/display/Graphics.html

Uro, T. 2007a. Mip map what? Viitattu 10.1.2010. http://www.kaourantin.net/2007/06/mip-map-what.html

Johnston, C. 2008a. Papervision3D Part 3: Features continued. Viitattu 10.1.2010. http://dispatchevent.org/calebjohnston/papervision3d-introduction-p3/

Hidden surface determination. 2010. Wikipedia. Viitattu 10.1.2010. http://en.wikipedia.org/wiki/Hidden_surface_determination

Zupko, A. 2008a. Papervision3D - Now Featuring Frustum Clipping. Viitattu 10.1.2010. http://blog.zupko.info/?p=170

Johnston, C. 2008b. Papervision3D Part 1: Introduction. Viitattu 10.1.2010. http://dispatchevent.org/calebjohnston/papervision3d-introduction-p1/

Papervision. 2009a. Papervision 3D Google Code. Viitattu 10.1.2010. http://code.google.com/p/papervision3d/

Google Code. 2010. Wikipedia. Viitattu 10.1.2010. http://en.wikipedia.org/wiki/Google_Code

Subversion. 2010. Wikipedia. Viitattu 10.1.2010.

http://en.wikipedia.org/wiki/Subversion_(software)

Lively, M. 2008a. Guts of a Papervision Application. Viitattu 10.1.2010. http://professionalpapervision.wordpress.com/2008/11/18/guts-of-a-papervision-application/

Tondeur, P. 2008a. Papervision3D 2.0 The Great White from the ground up. Viitattu 10.1.2010. http://www.paultondeur.com/2008/02/05/papervision3d-20-the-great-white-from-the-ground-up/

Visual field. 2010. Wikipedia. Viitattu 10.1.2010. http://en.wikipedia.org/wiki/Visual_field

Caunt, S. 2008a. Understanding zoom, focus and field of view in Papervision3D. Viitattu 10.1.2010. http://blog.tartiflop.com/2008/08/understanding-zoom-focus-and-field-of-view-in-papervision3d/

Zupko, A. 2008b. Using QuadTrees In Papervision3D. Viitattu 10.1.2010. http://blog.zupko.info/?p=177

Zupko, A. 2008c. Papervision QuadTree Support. Viitattu 10.1.2010. http://blog.papervision3d.org/2008/10/14/papervision-quadtree-support/

Taso. 2010. Wikipedia. Viitattu 10.1.2010. http://fi.wikipedia.org/wiki/Taso

Kuutio. 2010. Wikipedia. Viitattu 10.1.2010. http://fi.wikipedia.org/wiki/Kuutio

Pallo. 2010. Wikipedia. Viitattu 10.1.2010. http://fi.wikipedia.org/wiki/Pallo_(geometria)

Lieriö. 2010. Wikipedia. Viitattu 10.1.2010. http://fi.wikipedia.org/wiki/Lieriö

Kartio. 2010. Wikipedia. Viitattu 10.1.2010. http://fi.wikipedia.org/wiki/Kartio

Lindquist, J. 2008b. Textures (WireFrame, Bitmap, MovieAsset, Video, etc.) PaperVision3D. Viitattu 10.1.2010. http://www.insideria.com/2008/05/textures-wireframe-bitmap-movi-1.html

Zupko, A. 2008b. Casting Shadows In Papervision3D – Redux. Viitattu 10.1.2010. http://blog.zupko.info/?p=146

Hight, J. 2008. Papervision3D: Dynamic TrueType Font Loading. Viitattu 10.1.2010. http://labs.zavoo.com/?p=105

Barcinski, M. 2008. Vectorvision Google Code. Viitattu 10.1.2010. http://code.google.com/p/vectorvision/

Scalable Vector Graphics. 2010. Wikipedia. Viitattu 10.1.2010. http://en.wikipedia.org/wiki/Scalable_Vector_Graphics

COLLADA. 2010. Wikipedia. Viitattu 10.1.2010. http://en.wikipedia.org/wiki/COLLADA

3ds. 2010. Wikipedia. Viitattu 10.1.2010. http://en.wikipedia.org/wiki/.3ds

Windle, J. 2008. 3D Ribbons in Papervision. Viitattu 10.1.2010. http://blog.soulwire.co.uk/flash/actionscript-3/papervision3d/pv3d-ribbon3d-class/

GreenSock. 2010. GreenSock verkkosivut. Viitattu 10.1.2010. http://blog.greensock.com/

Drozdz, B. 2008b. AS3Dmod Google Code. Viitattu 10.1.2010. http://code.google.com/p/as3dmod/

Jiglibflash. 2009. Jiglibflash Google Code. Viitattu 10.1.2010. http://code.google.com/p/jiglibflash/

JUXT Interactive. 2009. VizualPV3D. Viitattu 10.1.2010. http://www.juxtinteractive.com/work/vizualpv3d/

Scavarelli, A. 2009. Papervision Collada Viewer. Viitattu 10.1.2010. http://blog.anthony-scavarelli.com/?p=77

Grden, J. 2007c. Xrayn Google Code. Viitattu 10.1.2010. http://code.google.com/p/osflash-xray/

LIITE 1

OHJELMAKOODIESIMERKKI 1

```
import flash.display.Sprite;
import flash.events.Event;
import flash.events.TimerEvent;
import flash.utils.Timer;
import flash.events.KeyboardEvent;
import flash.ui.Keyboard;

import org.papervision3d.view.BasicView;
import org.papervision3d.materials.utils.MaterialsList;
import org.papervision3d.materials.ColorMaterial;
import org.papervision3d.materials.WireframeMaterial;
import org.papervision3d.materials.special.CompositeMaterial;
import
org.papervision3d.materials.shadematerials.GouraudMaterial;
import org.papervision3d.materials.shadematerials.PhongMaterial;
import
org.papervision3d.materials.shadematerials.FlatShadeMaterial;
import org.papervision3d.materials.shadematerials.CellMaterial;
import org.papervision3d.lights.PointLight3D;
import org.papervision3d.objects.primitives.Sphere;
import org.papervision3d.objects.primitives.Cone;
import org.papervision3d.objects.primitives.Cube;
import org.papervision3d.objects.primitives.Cylinder;
import org.papervision3d.view.stats.StatsView;

public class Esimerkki1 extends BasicView
{
  private var stats    :StatsView;

  private var light    :PointLight3D;
  private var sphere    :Sphere;
  private var cone    :Cone;
  private var cube    :Cube;
  private var cylinder    :Cylinder;

  private var ymouse    :Number;
  private var timer    :Timer;

  public function Esimerkki1()
  {
    //koska projektin pääluokka jatkaa BasicView-luokkaa, voimme
    käyttää sen konstruktoria super-toiminnon avulla
    //BasicView(viewportWidth:Number = 640,
    viewportHeight:Number = 480, scaleToStage:Boolean = true,
    interactive:Boolean = false, cameraType:String = "Target")
    super(640, 480, false, false, "Target");

    //muutetaan kameran näköala pienemmäksi
    camera.fov = 35;

    //luodaan StatsView ja asetetaan se aluksi pois näkyvistä
    stats = new StatsView(renderer);
    addChild(stats);
```

```
        stats.visible = false;

        light = new PointLight3D();
        scene.addChild(light);

        //luodaan pallo
        //Sphere(material:MaterialObject3D = null, radius:Number =
        100, segmentsW:int = 8, segmentsH:int = 6)
        sphere = new Sphere(randomCellMaterial(), 80);
        scene.addChild(sphere);
        sphere.z = -200;

        //luodaan kartio
        //Cone(material:MaterialObject3D = null, radius:Number =
        100, height:Number = 100, segmentsW:int = 8, segmentsH:int =
        6)
        cone = new Cone(randomFlatMaterial(), 90, 130, 10, 2);
        scene.addChild(cone);
        cone.x = 200;

        //luodaan sylinteri
        //Cylinder(material:MaterialObject3D = null, radius:Number =
        100, height:Number = 100, segmentsW:int = 8, segmentsH:int =
        6, topRadius:Number = -1, topFace:Boolean = true,
        bottomFace:Boolean = true)
        cylinder = new Cylinder(randomGouraudMaterial(), 80, 100,
        10, 2);
        scene.addChild(cylinder);
        cylinder.x = -200;

        //luodaan laatikko
        var materialsList:MaterialsList = new MaterialsList({all:
        randomPhongMaterial()});
        //Cube(materials:MaterialsList, width:Number = 500,
        depth:Number = 500, height:Number = 500, segmentsS:int = 1,
        segmentsT:int = 1, segmentsH:int = 1, insideFaces:int = 0,
        excludeFaces:int = 0)
        cube = new Cube(materialsList, 120, 120, 120, 2, 2, 2);
        scene.addChild(cube);
        cube.z = 200;

        //päivitetään polygonien määrä StatsViewiin
        stats.updatePolyCount(scene);

        //kappaleiden väri vaihtuu kahden sekunnin välein
        timer = new Timer(2000);
        timer.addEventListener(TimerEvent.TIMER, timerTick);
        timer.start();

        //käynnistetään BasicView-luokan jatkuva renderöinti
        startRendering();

        //lisätään enterframe-funktio ja välilyönnin painamis -
        funktio
        addEventListener(Event.ENTER_FRAME, enterFrameHandler);
        stage.addEventListener(KeyboardEvent.KEY_DOWN, keyPressed);
    }
```

```
private function timerTick(e:TimerEvent):void {
  //arvotaan uuden väriset materiaalit
  newMaterials();
}

private function randomCellMaterial():CellMaterial
{
  return new CellMaterial(null, Math.random()*0xFFFFFF,
  0x111111, 4)
}

private function randomGouraudMaterial():GouraudMaterial
{
  //GouraudMaterial(light:LightObject3D, lightColor:uint =
  0xFFFFFF, ambientColor:uint = 0x000000, specularLevel:uint =
  0)
  return new GouraudMaterial(light, Math.random()*0xFFFFFF,
  0x111111, 50);
}

private function randomFlatMaterial():FlatShadeMaterial
{
  //FlatShadeMaterial(light:LightObject3D, lightColor:uint =
  0xffffff, ambientColor:uint = 0x000000, specularLevel:uint =
  0)
  return new FlatShadeMaterial(light, Math.random()*0xFFFFFF,
  0x111111, 0);
}

private function randomPhongMaterial():PhongMaterial
{
  //PhongMaterial(light:LightObject3D, lightColor:uint,
  ambientColor:uint, specularLevel:uint)
  return new PhongMaterial(null, Math.random()*0xFFFFFF,
  0x111111, 0);
}

private function newMaterials():void
{
  //arvotaan primitiiveille uuden väriset materiaalit
  sphere.material = randomCellMaterial();
  //sphere.material = randomPhongMaterial();
  cone.material = randomFlatMaterial();
  cylinder.material = randomGouraudMaterial();
  //koska laatikon materiaali-tyyppi on MaterialList, ei sitä
  voida vaihtaa samalla tavalla kuin muiden
  //cube.replaceMaterialByName(randomCellMaterial(), "all");
  cube.replaceMaterialByName(randomPhongMaterial(), "all");
}

private function keyPressed(e:KeyboardEvent):void
{
  switch (e.keyCode)
  {
    case Keyboard.SPACE:
      //välilyönnistä näytetään ja piilotetaan StatsViewiä
```

```
        stats.visible = !stats.visible;
        break;
    }
  }

  private function enterFrameHandler(e:Event):void
  {
    //jaetaan hiiren y-koordinaatti kahdella, jotta
    pystysuuntaisesta liikkeestä ei tule niin iso
    ymouse = mouseY/2;
    //asetetaan pystysuuntaiselle liikkeelle rajoituksia
    if(ymouse < 45)
    {
      ymouse = 45;
    }
    else if(ymouse > 145)
    {
      ymouse = 145;
    }
    //kamera kiertää kohtauksen keskipistettä hiiren
    koordinaattien mukaisesti
    camera.orbit(ymouse, mouseX);
  }
}
```

LIITE 2

OHJELMAKOODIESIMERKKI 2

```
import flash.display.Sprite;
import flash.display.MovieClip;
import flash.events.Event;
import flash.events.KeyboardEvent;
import flash.filters.BlurFilter;
import flash.ui.Keyboard;

import gs.TweenMax;
import gs.easing.*;

import org.papervision3d.core.effects.view.ReflectionView;
import org.papervision3d.events.InteractiveScene3DEvent;
import org.papervision3d.materials.ColorMaterial;
import org.papervision3d.materials.MovieAssetMaterial;
import org.papervision3d.objects.primitives.Plane;
import org.papervision3d.objects.DisplayObject3D;
import org.papervision3d.view.stats.StatsView;
import org.papervision3d.view.layer.ViewportLayer;
import org.papervision3d.view.layer.util.ViewportLayerSortMode;

public class Esimerkki2 extends Sprite
{
  private var view           :ReflectionView;
  private var stats      :StatsView;
  private var material       :MovieAssetMaterial;
  private var objects      :Array = new Array(); //taulukkoon
  tallennetaan referenssi kansikuvista
  private var objectsNum      :int = 8; //kansikuvien määrä
  private var currentPosition    :int = 0; //kertoo mikä
  kansikuva on edessä
  private var objectsContainer    :DisplayObject3D; //sisältää
  kaikki kansikuvat
  private var reflectionMask    :ReflectionMask; //heijastuksen
  häivytykseen tarvittava maski
  private var infoArray      :Array; //sisältää takakansien
  tekstit

  public function Esimerkki2()
  {
    //taulukko sisältää kansikuvien takatekstin
    infoArray = ["Black Sabbath", "Gamma Ray", "Guns N' Roses",
    "Iron Maiden", "Judas Priest", "Metallica", "Megadeth",
    "Anthrax"];
    //luodaan ReflectionView
    //ReflectionView(viewportWidth:Number = 640,
    viewportHeight:Number = 320, scaleToStage:Boolean = true,
    interactive:Boolean = false, cameraType:String = "Target")
    view = new ReflectionView(640, 480, false, true, "Target");
    addChild(view);
    //asetetaan heijastavan pinnan paikka
    view.surfaceHeight = -550;
    //lisätään heijastukseen epäterävyyttä
    view.viewportReflection.filters = [new BlurFilter(6, 6, 1)];
```

```
//heijastukselle asetetaan häivytys, tuomalla kirjastosta
liukuvärillä varustettu nelikulmio ja asettamalla se
heijastuksen maskiksi
reflectionMask = new ReflectionMask();
addChild(reflectionMask);
reflectionMask.y = 300;
reflectionMask.cacheAsBitmap = true;
view.viewportReflection.mask = reflectionMask;

//luodaan StatsView ja asetetaan se aluksi pois näkyvistä
stats = new StatsView(view.renderer);
addChild(stats);
stats.visible = false;

//säädetään kamera niin, että se näyttää hyvältä
view.camera.y = 50;
view.camera.fov = 70;

//luodaan tyhjä DisplayObject3D, jonka sisään kansikuvat
lisätään
objectsContainer = new DisplayObject3D();
view.scene.addChild(objectsContainer);

//luodaan määritetty määrä kansikuvia
for(var i:int = 0; i<objectsNum; i++)
{
  //luodaan kansikuvan materiaali kirjastosta tuodulta
  symboolista
  //tehdään materiaalista animoitu, koska symboolin
  aikajanaa tarvitaan, tehdään materiaalista myös uniikki,
  koska kansikuvien ulkonäkö vaihtelee
  //MovieAssetMaterial(linkageID:String = "",
  transparent:Boolean = false, animated:Boolean = false,
  createUnique:Boolean = false, precise:Boolean = false)
  material = new MovieAssetMaterial("Textures", false, true,
  true, false);
  //tehdään materiaalista interaktiivinen, jotta objekteista
  voi painaa
  material.interactive = true;
  //tehdään materiaalista kaksipuoleinen, koska molempia
  puolia käytetään
  material.doubleSided = true;
  //asetetaan materiaalin pehmennys päälle, jotta polygonien
  saumakohtiin ei tule saumoja
  material.smooth = true;

  //luodaan kansikuva
  //Plane(material:MaterialObject3D = null, width:Number =
  0, height:Number = 0, segmentsW:Number = 0,
  segmentsH:Number = 0)
  var object = new Plane(material, 510, 510, 2, 2);
  //lisätään kansikuva taulukkoon, josta siihen pystyy
  viitata myöhemmin
  objects.push(object);
  //muutetaan materiaalin teksti ja asetetaan takapuoli
  näkymättömäksi
  //koska movien: tyyppi on DisplayObject, joudutaan sitä
```

```
      erikseen käsittelemään MovieClippinä, jotta halutut
      ominaisuudet toimivat
      (object.material.movie as MovieClip).back.tf.text =
      infoArray[i];
      (object.material.movie as MovieClip).back.visible = false;
      //eri kansikuvat on asetettu aikajanalle
      (object.material.movie as MovieClip).gotoAndStop(i+1);
      //tallennetaan extra-objektiin kansikuvan järjestysnumero
      ja tieto siitä onko se tällä hetkellä käännetty ympäri
      object.extra = new Array(i, false);
      //asetetaan kansikuva objectsContaineriin ja asetetaan se
      paikoilleen x-suunnassa
      objectsContainer.addChild(object);
      object.x = i*550;

      //kansikuvalle tehdään uusi kerros (ViewportLayer)
      var objectLayer:ViewportLayer = new
      ViewportLayer(view.viewport,null);
      //lisätään uusi kerros viewportin juurikerrokseen
      view.viewport.containerSprite.addLayer(objectLayer);
      //asetetaan kerrosten järjestäminen etäisyyspohjaiseksi
      view.viewport.containerSprite.sortMode =
      ViewportLayerSortMode.Z_SORT;
      //lisätään kansikuva tehtyyn kerrokseen
      objectLayer.addDisplayObject3D(object);

      //asetetaan interaktiot kansikuvalle

object.addEventListener(InteractiveScene3DEvent.OBJECT_OVER,
handleObjectOver);

object.addEventListener(InteractiveScene3DEvent.OBJECT_OUT,
handleObjectOut);

object.addEventListener(InteractiveScene3DEvent.OBJECT_PRESS,
handleObjectPress);
    }
    //siirretään kansikuvat oikeaan asentoon
    tweenObjects();

    //päivitetään polygonien määrä StatsViewiin
    stats.updatePolyCount(view.scene);

    //lisätään enterframe-funktio ja kuuntelija nappien
    painamiselle
    addEventListener(Event.ENTER_FRAME, enterFrameHandler);
    stage.addEventListener(KeyboardEvent.KEY_DOWN, keyPressed);
  }

  private function tweenObjects():void
  {
    TweenMax.killTweensOf(objectsContainer);
    //liikutetaan objectsContainer, jossa kansikuvat on, uuteen
    kohtaan
    TweenMax.to(objectsContainer, 0.5, {x:
    -currentPosition*550});
    transformObjects();
```

```
}

//jos kansikuva kääntyy ympäri, vaihdetaan tekstuuria etu- ja
takakannen välillä
private function checkTexture(object:*):void
{
  if(object.rotationX<-90){
    (object.material.movie as MovieClip).back.visible = true;
    //kansikuvan takapuolen teksti vääristyy pahasti, jos
    materiaali ei ole tarkka
    object.material.precise = true;
  }else{
    (object.material.movie as MovieClip).back.visible = false;
    //Kuvapuolen vääristymistä ei huomaa niin paljoa, joten
    sen ei tarvitse olla tarkka
    object.material.precise = false;
  }
}

private function transformObjects():void
{
  for(var i:int = 0; i<objectsNum; i++)
  {
    //jos kansikuvia liikutetaan, käännetään kaikki oikein
    päin
    objects[i].extra[1] = false;

    var offset:int = objects[i].extra[0]-currentPosition;
    //jos kansikuva on keskellä se tuodaan eteenpäin, muuten
    se käännetään oikeaan asentoon
    if(offset == 0)
    {
      TweenMax.to(objects[i], 0.5, {z:-300, rotationY:0});
    }
    else
    {
      TweenMax.to(objects[i], 0.5, {z:Math.abs(offset)*100,
      rotationY:offset*20, rotationX:0, onUpdate:checkTexture,
      onUpdateParams:[objects[i]]});
    }
  }
}

private function keyPressed(e:KeyboardEvent):void
{
  switch (e.keyCode)
  {
    //nuolinäppäimilläkin voi selata kansikuvia
    case Keyboard.LEFT:
      if(currentPosition > 0)
      {
        currentPosition--;
        tweenObjects();
      }
      break;
    case Keyboard.RIGHT:
      if(currentPosition < objectsNum-1)
```

```
        {
          currentPosition++;
          tweenObjects();
        }
        break;
      case Keyboard.SPACE:
        //välilyönnistä näytetään ja piilotetaan StatsViewiä
        stats.visible = !stats.visible;
        break;
    }
  }

  private function enterFrameHandler(e:Event):void
  {
    //Kohtausta renderöidään ainostaan jos kansikuvia
    liikutetaan
    if(TweenMax.isTweening(objects[currentPosition]) ||
    TweenMax.isTweening(objectsContainer))
    {
      view.singleRender();
    }
  }

  //Muutetaan hiiren nuoli kädeksi kun se viedään kansikuvien
  päälle
  private function
  handleObjectOver(e:InteractiveScene3DEvent):void
  {
    view.viewport.containerSprite.buttonMode = true;
  }

  private function
  handleObjectOut(e:InteractiveScene3DEvent):void
  {
    view.viewport.containerSprite.buttonMode = false;
  }

  private function
  handleObjectPress(e:InteractiveScene3DEvent):void
  {
    if(currentPosition == e.target.extra[0])
    {
      if(e.target.extra[1])
      {
        //jos kuva on käännetty ympäri, käännetään kuva
        oikeinpäin
        e.target.extra[1] = false;
        TweenMax.to(objects[e.target.extra[0]], 0.5,
        {rotationX:0, onUpdate:checkTexture,
        onUpdateParams:[objects[e.target.extra[0]]]});
      }
      else
      {
        //jos kuvaa ei ole käännetty ympäri, käännetään kuva
        ympäri
        e.target.extra[1] = true;
        TweenMax.to(objects[e.target.extra[0]], 0.5,
```

```
        {rotationX:-180, onUpdate:checkTexture,
        onUpdateParams:[objects[e.target.extra[0]]]});
      }
    }
    else
    {
      //jos kuvasta painaa ja se ei ole keskellä, siirtyy
      kyseinen kuva keskelle
      currentPosition = e.target.extra[0];
      tweenObjects();
    }
  }
}
```

OHJELMAKOODIESIMERKKI 3

```
import flash.display.Sprite;
import flash.events.Event;
import flash.events.TimerEvent;
import flash.utils.Timer;
import flash.events.KeyboardEvent;
import flash.ui.Keyboard;

import org.papervision3d.cameras.Camera3D;
import org.papervision3d.materials.utils.MaterialsList;
import org.papervision3d.materials.ColorMaterial;
import org.papervision3d.materials.WireframeMaterial;
import org.papervision3d.materials.special.CompositeMaterial;
import org.papervision3d.materials.shadematerials.*;
import org.papervision3d.lights.PointLight3D;
import org.papervision3d.objects.primitives.Sphere;
import org.papervision3d.objects.primitives.Cone;
import org.papervision3d.objects.primitives.Cube;
import org.papervision3d.objects.primitives.Cylinder;
import org.papervision3d.render.QuadrantRenderEngine;
import org.papervision3d.render.BasicRenderEngine;
import org.papervision3d.scenes.Scene3D;
import org.papervision3d.view.Viewport3D;
import org.papervision3d.view.stats.StatsView;

public class Esimerkki3 extends Sprite
{
  private var scene    :Scene3D;
  private var camera    :Camera3D;
  private var viewport    :Viewport3D;
  private var rendererQ    :QuadrantRenderEngine = new
  QuadrantRenderEngine();
  private var drawer    :QuadTreeVisualizer = new
  QuadTreeVisualizer();
  private var rendererB    :BasicRenderEngine = new
  BasicRenderEngine();
  private var renderer    :BasicRenderEngine;
  private var statsQ    :StatsView;
  private var statsB    :StatsView;

  private var material    :*;
  private var materialsList  :MaterialsList;

  private var sphere    :Sphere;
  private var cone    :Cone;
  private var cube    :Cube;
  private var cylinder    :Cylinder;
  private var light    :PointLight3D;

  private var ymouse    :Number;
  private var oldMouseY    :Number = 0;
  private var oldMouseX    :Number = 0;

  public function Esimerkki3()
```

```
{
  //määritetään kohtaus
  scene = new Scene3D();
  //määritetään kamera
  camera = new Camera3D();
  //muutetaan kameran näköala pienemmäksi
  camera.fov = 25;
  //määritetään viewport ja lisätään se stagelle
  viewport = new Viewport3D();
  addChild(viewport);
  //määritetään renderöintimoottori
  renderer = rendererB;
  //luodaan StatsView ja asetetaan se aluksi pois näkyvistä
  statsQ = new StatsView(rendererQ);
  addChild(statsQ);
  statsQ.visible = false;
  statsQ.y = -150;
  statsB = new StatsView(rendererB);
  addChild(statsB);
  statsB.visible = false;

  //nostetaan QuadrantRenderEnginen tarkkuutta yhdellä
  rendererQ.quadTree.maxLevel = 5;

  drawer.visible = false;
  addChild(drawer);

  //luodaan pallo
  //Sphere(material:MaterialObject3D = null, radius:Number =
  100, segmentsW:int = 8, segmentsH:int = 6)
  sphere = new Sphere(wireframe(), 80);
  scene.addChild(sphere);
  sphere.z = -70;

  //luodaan kartio
  //Cone(material:MaterialObject3D = null, radius:Number =
  100, height:Number = 100, segmentsW:int = 8, segmentsH:int =
  6)
  cone = new Cone(wireframe(), 90, 130, 10, 2);
  scene.addChild(cone);
  cone.x = 70;

  //luodaan sylinteri
  //Cylinder(material:MaterialObject3D = null, radius:Number =
  100, height:Number = 100, segmentsW:int = 8, segmentsH:int =
  6, topRadius:Number = -1, topFace:Boolean = true,
  bottomFace:Boolean = true)
  cylinder = new Cylinder(wireframe(), 80, 100, 10, 2);
  scene.addChild(cylinder);
  cylinder.x = -70;

  //luodaan laatikko
  materialsList = new MaterialsList({all: wireframe()});
  //Cube(materials:MaterialsList, width:Number = 500,
  depth:Number = 500, height:Number = 500, segmentsS:int = 1,
  segmentsT:int = 1, segmentsH:int = 1, insideFaces:int = 0,
  excludeFaces:int = 0)
```

```
    cube = new Cube(materialsList, 120, 120, 120, 2, 2, 2);
    scene.addChild(cube);
    cube.z = 70;

    light = new PointLight3D(true);
    //light.z = -800;
    light.y = 300;

    //päivitetään polygonien määrä
    statsB.updatePolyCount(scene);

    //lisätään enterframe-funktio ja kuuntelija nappien
    painamiselle
    addEventListener(Event.ENTER_FRAME, enterFrameHandler);
    stage.addEventListener(KeyboardEvent.KEY_DOWN, keyPressed);
  }

  private function wireframe():CompositeMaterial
  {
    material = new CompositeMaterial();
    material.addMaterial(new
    ColorMaterial(Math.random()*0xFFFFFF));
    material.addMaterial(new WireframeMaterial(0x000000));
    return material;
  }

  private function colorMat():ColorMaterial
  {
    material = new ColorMaterial(Math.random()*0xFFFFFF);
    return material;
  }

  private function newMaterials(_wireframe:Boolean):void
  {
    if(_wireframe)
    {
      sphere.material = wireframe();
      cone.material = wireframe();
      cylinder.material = wireframe();
      cube.replaceMaterialByName(wireframe(), "all");
    }
    else
    {
      sphere.material = colorMat();
      cone.material = colorMat();
      cylinder.material = colorMat();
      cube.replaceMaterialByName(colorMat(), "all");
    }
  }
  private function keyPressed(e:KeyboardEvent):void
  {
    switch (e.keyCode)
    {
      case Keyboard.SPACE:
        //välilyönnistä näytetään ja piilotetaan StatsViewiä
        statsQ.visible = !statsQ.visible;
        statsB.visible = !statsB.visible;
```

```
        break;
      case Keyboard.RIGHT:
        statsQ.y = 0;
        statsB.y = -150;
        renderer = rendererQ;
        renderer.renderScene(scene, camera, viewport);
        //päivitetään polygonien määrä
        statsQ.updatePolyCount(scene);
        break;
      case Keyboard.LEFT:
        statsQ.y = -150;
        statsB.y = 0;
        renderer = rendererB;
        renderer.renderScene(scene, camera, viewport);
        //päivitetään polygonien määrä
        statsB.updatePolyCount(scene);
        drawer.visible = false;
        break;
      case Keyboard.UP:
        //vaihdetaan materiaaleiksi rautalanka-materiaali ja
        renderöidään kohtaus
        newMaterials(true);
        renderer.renderScene(scene, camera, viewport);
        break;
      case Keyboard.DOWN:
        //vaihdetaan materiaaleiksi yksivärinen-materiaali ja
        renderöidään kohtaus
        newMaterials(false);
        renderer.renderScene(scene, camera, viewport);
        break;
      case Keyboard.ENTER:
        drawer.visible = !drawer.visible;
        break;

    }
  }

  private function enterFrameHandler(e:Event):void
  {
    //jaetaan hiiren y-koordinaatti kahdella, jotta
    pystysuuntaisesta liikkeestä ei tule niin iso
    ymouse = mouseY/2;
    //asetetaan pystysuuntaiselle liikkeelle rajoituksia
    if(ymouse < 45)
    {
      ymouse = 45;
    }
    else if(ymouse > 145)
    {
      ymouse = 145;
    }
    //kamera kiertää kohtauksen keskipistettä hiiren
    koordinaattien mukaisesti
    camera.orbit(ymouse, mouseX);

    //renderöidään kohtaus, jos hiirtä on liikutettu
    if(mouseY != oldMouseY || mouseX != oldMouseX){
```

```
      renderer.renderScene(scene, camera, viewport);

      if(drawer.visible && renderer == rendererQ){
        drawer.drawQuadTree(rendererQ.quadTree, 0xFF0000, 0.5);
        drawer.x = viewport.viewportWidth/2;
        drawer.y = viewport.viewportHeight/2;
      }
    }

    //tallennetaan nykyinen hiiren paikka
    oldMouseY = mouseY;
    oldMouseX = mouseX;
  }
}
```

OHJELMAKOODIESIMERKKI 4

```
import flash.events.Event;
import flash.events.KeyboardEvent;
import flash.ui.Keyboard;

import org.papervision3d.view.BasicView;
import org.papervision3d.view.stats.StatsView;
import org.papervision3d.events.FileLoadEvent;
import org.papervision3d.objects.parsers.DAE;

public class Esimerkki4 extends BasicView
{
  private var stats      :StatsView;
  private var ymouse      :Number;
  private var xmouse      :Number;
  private var oldMouseY      :Number = 0;
  private var oldMouseX      :Number = 0;
  private var dae       :DAE;

  public function Esimerkki4()
  {
    //koska projektin pääluokka jatkaa BasicView-luokkaa, voimme
    käyttää sen konstruktoria super-toiminnon avulla
    //BasicView(viewportWidth:Number = 640,
    viewportHeight:Number = 480, scaleToStage:Boolean = true,
    interactive:Boolean = false, cameraType:String = "Target")
    super(640, 480, false, true, "Target");

    //muutetaan kameran näköala pienemmäksi
    camera.fov = 40;

    //luodaan StatsView ja asetetaan se aluksi pois näkyvistä
    stats = new StatsView(renderer);
    addChild(stats);
    stats.visible = false;

    //ladataan ulkoinen 3D-malli
    //DAE(autoPlay:Boolean = true, name:String = null,
    loop:Boolean = false)
    dae = new DAE(false);
    dae.addEventListener(FileLoadEvent.LOAD_COMPLETE,
    onDaeLoadComplete);

    dae.load("tiedostot/esimerkki4/lehma.DAE");
    //tiedoston polku on eri, jos swf ajetaan suoraan ilman
    html-sivua
    //dae.load("lehma.DAE");

    //lisätään enterframe-funktio ja nappien painamis -funktio
    addEventListener(Event.ENTER_FRAME, enterFrameHandler);
    stage.addEventListener(KeyboardEvent.KEY_DOWN, keyPressed);
  }

  private function onDaeLoadComplete(e : FileLoadEvent) : void
```

```
{
  //määritetään dae-muuttuja ladatuksi malliksi
  dae = e.target as DAE;
  //lisätään ladattu malli kohtaukseen
  scene.addChild(dae);
  //skaalataan mallia isommaksi
  dae.scale = 7;
  //lasketaan mallia alaspäin
  dae.y = -150;

  //päivitetään polygonien määrä StatsViewiin
  stats.updatePolyCount(scene);
}

private function keyPressed(e:KeyboardEvent):void
{
  switch (e.keyCode)
  {
    case Keyboard.SPACE:
      //välilyönnistä näytetään ja piilotetaan StatsViewiä
      stats.visible = !stats.visible;
      break;
    case Keyboard.LEFT:
      //jos mikään animaatio ei ole käynnissä toistetaan
      "lookLeft"-animaatioklippi kerran
      if(!dae.playing)
      {
        dae.play("lookLeft", false);
      }
      break;
    case Keyboard.RIGHT:
      if(!dae.playing)
      {
        //jos mikään animaatio ei ole käynnissä toistetaan
        "lookRight"-animaatioklippi kerran
        dae.play("lookRight", false);
      }
      break;
    case Keyboard.UP:
      if(!dae.playing)
      {
        //jos mikään animaatio ei ole käynnissä toistetaan
        "ammuu"-animaatioklippi kerran
        dae.play("ammuu", false);
      }
      break;
  }
}

private function enterFrameHandler(e:Event):void
{
  //jaetaan hiiren y-koordinaatti kahdella, jotta
  pystysuuntaisesta liikkeestä ei tule niin iso
  ymouse = mouseY/2;
  //asetetaan pystysuuntaiselle liikkeelle rajoituksia
  if(ymouse < 60)
  {
```

```
      ymouse = 60;
    }
    else if(ymouse > 85)
    {
      ymouse = 85;
    }
    //kamera kiertää kohtauksen keskipistettä hiiren
    koordinaattien mukaisesti
    camera.orbit(ymouse, mouseX);

    //renderöidään kohtaus, jos hiirtä on liikutettu tai 3D-
    malli toistaa animaatiota
    if(mouseY != oldMouseY || mouseX != oldMouseX ||
    dae.playing){
      singleRender();
    }
    //tallennetaan nykyinen hiiren paikka
    oldMouseY = mouseY;
    oldMouseX = mouseX;
  }
}
```

OHJELMAKOODIESIMERKKI 5

```
import flash.display.MovieClip;
import flash.geom.Rectangle;
import flash.events.Event;
import flash.events.KeyboardEvent;
import flash.ui.Keyboard;

import org.papervision3d.view.BasicView;
import org.papervision3d.objects.primitives.Plane;
import org.papervision3d.objects.primitives.Cylinder;
import org.papervision3d.view.stats.StatsView;
import org.papervision3d.objects.DisplayObject3D;
import org.papervision3d.materials.BitmapAssetMaterial;
import org.papervision3d.materials.MovieMaterial;
import org.papervision3d.materials.special.LineMaterial;
import org.papervision3d.core.geom.Lines3D;
import org.papervision3d.core.geom.renderables.Line3D;
import org.papervision3d.core.utils.virtualmouse.VirtualMouse;
import org.papervision3d.events.InteractiveScene3DEvent;
import org.papervision3d.core.utils.InteractiveSceneManager;

public class Esimerkki5 extends BasicView
{
  private var stats    :StatsView;
  private var vMouse    :VirtualMouse;

  private var grassMaterial   :BitmapAssetMaterial;
  private var treeMaterial   :BitmapAssetMaterial;
  private var canvasMaterial  :MovieMaterial;
  private var drawingCanvas  :MovieClip;
  private var drawingArea    :MovieClip;

  private var plane    :Plane;
  private var plane2    :Plane;
  private var cylinder1    :Cylinder;
  private var cylinder2    :Cylinder;
  private var cylinder3    :Cylinder;
  private var cylinder4    :Cylinder;
  private var skyPosition    :DisplayObject3D;
  private var lines    :Lines3D;

  private var ymouse    :Number;
  private var xmouse    :Number;
  private var oldMouseY    :Number = 0;
  private var oldMouseX    :Number = 0;
  private var oldVMouseX    :Number = 0;
  private var oldVMouseY    :Number = 0;
  private var allowDrawing  :Boolean = false;

  public var _background    :MovieClip;

  public function Esimerkki5()
  {
    //koska projektin pääluokka jatkaa BasicView-luokkaa, voimme
```

```
käyttää sen konstruktoria super-toiminnon avulla
//koska haluamme piirtää ohjelmassa on viewportin oltava
interaktiivinen
//BasicView(viewportWidth:Number = 640,
viewportHeight:Number = 480, scaleToStage:Boolean = true,
interactive:Boolean = false, cameraType:String = "Target")
super(640, 480, false, true, "Target");

//muutetaan kameran näköala pienemmäksi
camera.fov = 40;

//luodaan StatsView ja asetetaan se aluksi pois näkyvistä
stats = new StatsView(renderer);
addChild(stats);
stats.visible = false;

//tallennetaan referenssi virtualMouse-luokasta lyhyempään
muuttujaan
vMouse = viewport.interactiveSceneManager.virtualMouse;

//määritetään piirtoalueena käytetty materiaali
drawingCanvas = new Canvas();
//lisätään tyhjä MovieClip piirtoalueeseen, jonka sisälle
piirretään
drawingArea = new MovieClip();
drawingCanvas.addChild(drawingArea);
//määritetään kangas-tason materiaali käyttämään juuri
luotua piirtoaluetta
canvasMaterial = new MovieMaterial(drawingCanvas);
//käytetään pehmennystä materiaalissa
canvasMaterial.smooth = true;
//jos käyttäjä onnistuu piirtämään piirtoalueen
ulkopuolelle, tekstuurin koko muuttuu, tämän takia rajataan
alue jota käytetään tekstuurina
canvasMaterial.rect = new Rectangle(0, 0, 384, 216);
//jotta materiaaliin voi piirtää on se asetettava
interaktiiviseksi
canvasMaterial.interactive = true;
//jotta materiaali päivittyy sen on oltava animoitu
canvasMaterial.animated = true;

//luodaan taso, jolle piirretään
//Plane(material:MaterialObject3D = null, width:Number = 0,
height:Number = 0, segmentsW:Number = 0, segmentsH:Number =
0)
plane = new Plane(canvasMaterial, 480, 270, 6, 4)
scene.addChild(plane);
plane.y = 90;
//alustetaan piirtoalue
resetCanvas();

//BitmapAssetMaterial(linkageID:String, precise:Boolean =
false)
grassMaterial = new BitmapAssetMaterial("ruoho");
//ruohomateriaali on saumaton joten voimme toistaa sitä
useita kertoja samalle pinnalle
grassMaterial.tiled = true;
```

```
//nämä määrittelevät toistojen määrän pituus- ja leveys-
suunnassa
grassMaterial.maxV = 4;
grassMaterial.maxU = 6;
//luodaan ruoho-taso
plane2 = new Plane(grassMaterial, 2200, 1300, 8, 4)
scene.addChild(plane2);
plane2.y = -255;
plane2.rotationX = 90;

//luodaan pystytolpille puu-materiaali, joka on myös
saumaton
treeMaterial = new BitmapAssetMaterial("puu");
treeMaterial.tiled = true;
treeMaterial.smooth = true;
treeMaterial.maxU = 0.8;
treeMaterial.maxV = 1.5;

//luodaan tolpat
//Cylinder(material:MaterialObject3D = null, radius:Number =
100, height:Number = 100, segmentsW:int = 8, segmentsH:int =
6, topRadius:Number = -1, topFace:Boolean = true,
bottomFace:Boolean = true)
cylinder1 = new Cylinder(treeMaterial, 30, 500, 8, 4)
scene.addChild(cylinder1);
cylinder1.x = -300;

cylinder2 = new Cylinder(treeMaterial, 30, 500, 8, 4)
scene.addChild(cylinder2);
cylinder2.x = 300;

//vaakatolpan materiaali asetetaan eri tavalla, joten se
määritetään uudestaan
treeMaterial = new BitmapAssetMaterial("puu");
treeMaterial.tiled = true;
treeMaterial.smooth = true;
treeMaterial.maxU = 0.3;
treeMaterial.maxV = 3;

//lisätään vaakatolppa
cylinder3 = new Cylinder(treeMaterial, 15, 675, 8, 4)
scene.addChild(cylinder3);
cylinder3.y = 260;
cylinder3.rotationZ = 90;

//piirretään narut, jotka pitävät piirtokangasta paikoillaan
//LineMaterial(color:Number = 0xFF0000, alpha:Number = 1)
//Lines3D(material:LineMaterial = null, name:String = null)
//addNewLine(size:Number, x0:Number, y0:Number, z0:Number,
x1:Number, y1:Number, z1:Number):Line3D
lines = new Lines3D(new LineMaterial(0xB37604, 1));
lines.addNewLine(3, 240, 225, 5, 280, 250, 5);
lines.addNewLine(3, -240, 225, 5, -280, 250, 5);
lines.addNewLine(3, -240, -45, 5, -280, -70, 5);
lines.addNewLine(3, 240, -45, 5, 280, -70, 5);
scene.addChild(lines);
```

```
    //MovieClippiä, joka toimii taustana siirretään ruohon
    takatason mukana
    //Teemme sitä varten tyhjän DisplayObject3D-instanssin,
    jonka paikkaa ruudulla seuraamme
    skyPosition = new DisplayObject3D();
    scene.addChild(skyPosition);
    skyPosition.z = 650;
    skyPosition.y = -255;

    //päivitetään polygonien määrä StatsViewiin
    stats.updatePolyCount(scene);

    //lisätään kuuntelijat piirtokankaaseen
    plane.addEventListener(InteractiveScene3DEvent.OBJECT_MOVE,
    handleMouseMove);
    plane.addEventListener(InteractiveScene3DEvent.OBJECT_PRESS,
    handleMouseDown);
    plane.addEventListener(InteractiveScene3DEvent.OBJECT_OUT,
    handleMouseOut);

    //lisätään enterframe-funktio ja nappien painamis -funktio
    addEventListener(Event.ENTER_FRAME, enterFrameHandler);
    stage.addEventListener(KeyboardEvent.KEY_DOWN, keyPressed);
  }

  //kun piirtokangasta painetaan sallitaan piirtäminen ja
  tallennetaan aloituspaikka
  private function handleMouseDown(e:InteractiveScene3DEvent)
  {
    allowDrawing = true;
    oldVMouseX = vMouse.x;
    oldVMouseY = vMouse.y;
  }

  //jos hiiri viedään ulos kohteesta, ei sallita piirtämistä
  private function handleMouseOut(e:InteractiveScene3DEvent)
  {
    allowDrawing = false;
  }

  //jos hiiri on pohjassa ja sitä liikutetaan piirtokankaan
  sisällä piirretään viivaa
  private function handleMouseMove(e:InteractiveScene3DEvent)
  {
    if(InteractiveSceneManager.MOUSE_IS_DOWN && allowDrawing)
    {
      //piirretään viiva virtualMousen edellisestä paikasta
      nykyiseen
      drawingArea.graphics.lineStyle(3, 0x000000);
      drawingArea.graphics.moveTo(oldVMouseX,oldVMouseY);
      drawingArea.graphics.lineTo(vMouse.x,vMouse.y);
      //tallennetaan virtualMousen vanha paikka
      oldVMouseX = vMouse.x;
      oldVMouseY = vMouse.y;
    }
  }
```

```
private function resetCanvas():void
{
  //tyhjennetään piirtoalue ja renderöidään näkymä
  drawingArea.graphics.clear();
  renderer.renderScene(scene, camera, viewport);
}

private function keyPressed(e:KeyboardEvent):void
{
  switch (e.keyCode)
  {
    case Keyboard.SPACE:
      //välilyönnistä näytetään ja piilotetaan StatsViewiä
      stats.visible = !stats.visible;
      break;
    case Keyboard.ENTER:
      resetCanvas();
      break;
  }
}

private function enterFrameHandler(e:Event):void
{
  //katsotaan hiiren poikkeama ruudun keskeltä
  xmouse = mouseX-(640/2);
  ymouse = mouseY-(480/2);

  //asetetaan rajoituksia näkökenttään
  if(ymouse < -240)
  {
    ymouse = -240;
  }
  else if(ymouse > 25)
  {
    ymouse = 25;
  }

  if(xmouse < -180)
  {
    xmouse = -180;
  }
  else if(xmouse > 180)
  {
    xmouse = 180;
  }

  //liikutetaan kameraa
  camera.x = xmouse;
  camera.y = -ymouse;

  //renderöidään kohtaus, jos hiirtä on liikutettu
  if(mouseY != oldMouseY || mouseX != oldMouseX){
    renderer.renderScene(scene, camera, viewport);
  }

  //lasketaan ruohon takatason paikka ruudulla
  skyPosition.calculateScreenCoords(camera);
```

```
    //asetetaan ruudulla oleva tausta siihen kohtaan
    _background.x = skyPosition.screen.x+320;
    _background.y = skyPosition.screen.y;

    //tallennetaan nykyinen hiiren paikka
    oldMouseY = mouseY;
    oldMouseX = mouseX;
  }
}
```