VIIKKORAPORTTI 4

Tämä viikko toi tullessaan vihdoinkin todellisia onnistumisia. Sain kaikki matriisimuunnokset toimimaan, koodattua hierarkian itse 3D-tilalle, tehtyä muutamat algoritmit peruskappaleiden muodostamiseen ja piirtämisen wireframe-tilassa toimimaan luotettavasti. Muutaman hassun renderinkin olen lisännyt versionhallintaan.

Ohjelman tietorakenteet muuttuivat taas ja tällä hetkellä päällimmäisenä on tila-struktuuri (scene), joka sisältää kaksisuuntaiset linkitetyt listat objekteihin (object) ja malleihin (mesh). Mallit sisältävät kaksisuuntaiset linkitetyt listat omiin vertekseihinsä ja polygoneihinsa. Nämä olivat lähinnä muistinhallinnallisesti pakollisia, sillä esimerkiksi yhteen verteksiin saattaa olla viitteitä monesta polygonista ja yhteen malliin saattaa olla viitteitä monista objekteista. Tällöin muistin vapauttaminen rekursiivisesti ei onnistu, sillä vapautettuun malliin saattaa viitata joku toinen objekti, jolloin se viittaisi muistialueeseen, joka ei ole enää varattu. Malli on melko yksinkertainen hierarkkinen rakenne ja todennäköisesti sitä voisi parannellakin. Kannattaisi varmaan tutkia, minkälaisia eri tapoja 3D-informaation esittämiseen muistissa on.

Tarkastelin hieman myös Boostia, joka vaikutti melko pätevältä kirjastolta ja se onkin saanut paljon hyvää palautetta. Itse kuitenkin kirjoitan tätä pelkällä C:llä, en siis C++:lla (mikä toisaalta olisi ehkä ollut kannattavaa, mutta toisaalta en osaa sitä ja sen opetteluun olisi mennyt paljon aikaa), joten kirjastosta ei ole minulle hyötyä. Etsiskelin hieman vastaavaa yleiskäyttöistä kirjastoa C:lle, sillä perustoiminnallisuuden kirjoittaminen on hidasta ja haastavaa bugikorjauksineen kaikkineen, mutta en löytänyt mitään yhtä kattavaa. Glib tarjoaa paljon samaa toiminnallisuutta ja siihen täytyykin myöhemmin tutustua paremmin. Tällä hetkellä olen esimerkiksi kopioinut linkitettyihin listoihin liittyvän toiminnallisuuden jokaiselle tyypille erikseen, olisi paljon kätevämpää rakentaa yleinen linkitetty lista, joka sisältäisi vain attribuuttina struktuurin tyypin ja osoittimen sen tyyppiseen struktuuriin (tai jopa itse struktuurin).

3D-moottorin perusta on tällä hetkellä melko vakaa, nyt sitten pitäisi ruveta toteuttamaan itse sitä tietorakennetta eli BSP-puuta. Piirtäminen toimii hyvin wireframe-tilassa, mutta myös ihan oikeita polygoneja pitäisi pystyä piirtämään. 3D-piirtoalgoritmiin (pipeline) kuuluu muutama tietty osa:

1. 3D-datan esittäminen tietyssä muodossa
2. Matriisimuunnokset, jotka asettavat verteksit oikeaan paikkaan ruudulla:
    a. Ensimmäisenä maailmamuunnos (world transform), joka siirtää malliavaruudessa (model space, object space) määritetyt verteksien koordinaatit oikeisiin paikkoihin maailma-avaruudessa (world space). Koostuu skaalaus-, rotaatio- ja translaatiomatriiseista.
    b. Kuvakulmamuunnos (view transform), joka asettaa verteksit oikeisiin paikkoihin kameran suhteen. Tämä on itseasiassa kameraksi määritellyn objektin maailmamuunnoksen käänteismatriisi.
    c. Perspektiivimuunnos (perspective transform), joka luo illuusion perspektiivistä. Tällaisen voi määritellä monella tavalla. Toinen tämän matriisin tärkeä tehtävä on normalisoida x-, y-, ja z-koordinaatit välille [-1, 1], kun ne lopulta jaetaan y-koordinaatilla (ks. 2e). Yleensä käytetään matriisia, joka asettaa z-akselin osoittamaan kamerasta poispäin, mutta itse käytän normaalia maailman koordinaatistoa, jossa y-akseli osoittaa kamerasta poispäin ja z-akseli ylöspäin. Tästä avaruudesta käytetään nimeä leikkausavaruus (clip space), joka on

aidosti neliulotteinen. Muissa vaiheissa verteksien homogeenisten koordinaattien neljäs alkio on aina 1. Perspektiivimuunnoksessa sen arvo muuttuu, jolloin pisteet sijaitsevat eri kolmiulotteisissa avaruuksissa, itseasiassa sen arvoksi asetetaan omassa tapauksessani verteksin y-koordinaatti. Nimi leikkausavaruus tulee siitä, että yleensä tässä vaiheessa myös poistetaan ne polygonit, jotka osoittavat poispäin kamerasta, joka onkin vaihe d.

   d. Niiden polygonien leikkaaminen (clipping) pois kuvasta, jotka osoittavat poispäin kamerasta ja jotka ovat vain osittain kuva-alueella. Tällä vältetään tilanteet, jossa yritettäisiin piirtää kuvan ulkopuolelle. Tässä vaiheessa myöskin leikataan polygoneista ne osat pois, jotka jäävät toisen polygonin taakse. Tähän tarvitaan BSP-puuta.
   e. Jaetaan verteksien koordinaatit neljännellä koordinaatillaan eli alkuperäisellä y-koordinaatilla, jolloin verteksit palautuvat normaaliin kolmiulotteiseen avaruuteen. Tätä vaihetta kutsutaan perspektiivijaoksi (perspective divide, z-divide). Perspektiivin illuusio johtuu oikeastaan juuri tästä laskutoimituksesta.

3. Itse kuvan piirtäminen. Perspektiivimuunnoksen jälkeen verteksin x- ja z-koordinaatit kuvaavat sen paikkaa ruudulla ja y-koordinaatti kuvaa etäisyyttä ruudusta. Koska verteksien kaikki koordinaatit on normalisoitu välille [-1, 1], voidaan ne helposti asettaa ruudulle vain sen leveyttä tai korkeutta kertomalla ja pyöristämällä lähimpään kokonaislukuun. Käytän ikkunan luomiseen SDL-kirjastoa, mutta kaikki piirtorutiinit olen kirjoittanut itse, sillä jos joskus haluan lisätä esimerkiksi tekstuureja tai muuta vastaavaa, joutuisin todennäköisesti kirjoittamaan ne myöhemmin.

Tällä hetkellä vaiheet 1 ja 3 toimivat hyvin, mutta vaihe 2d puuttuu kokonaan. Tämän takia moottori ei pysty vielä piirtämään täysiä polygoneja, sillä moottorissa ei ole vielä toiminnallisuutta niiden järjestyksen määrittämiseen. Lisäksi, jos polygoni sattuu osumaan kuvan ulkopuolelle, ohjelma kaatuu, sillä se yrittää piirtää pikseleitä kuva-alueen ulkopuolelle.

Olen projektistani siinä mielessä hieman jäljessä, että perustoiminnallisuuden toteuttaminen veikin enemmän aikaa kuin kuvittelin. Pari viikkoa on kuitenkin vielä jäljellä, joista ensimmäisen ajattelin käyttää leikkauksen toteuttamiseen ja toisen BSP-puun rakentamiseen. Tässä vielä yksi mahtavista rendereistäni ☺