CAD DATA VIEWER

Opinnäytetyö

Jyrki Jauhiainen

Tietotekniikan koulutusohjelma

Ohjelmistotekniikka

Hyväksytty ___.___._____ _____

SAVONIA-AMMATTIKORKEAKOULU TEKNIIKKA KUOPIO

Koulutusohjelma

Tietotekniikan koulutusohjelma

Tekijä

Jyrki Jauhiainen

Työn nimi

CAD Data Viewer

Työn laji	Päiväys	Sivumäärä
Opinnäytetyö	7.4.2011	28 + 11

Työn valvoja	Yrityksen yhdyshenkilö
Lehtori Sami Lahti	Projektipäällikkö Sami Lahtinen

Yritys

Rollset Oy

Tiivistelmä

Tämän opinnäytetyön aiheena oli valmistaa ohjelmisto, joka piirtää kolmiulotteisena StruCad -ohjelmalla suunniteltua tietoa talon teräsrakenteista. Ohjelmisto oli tarkoitettu osaksi Steelcon -teräsrakenteiden valmistuslinjaa, jonka valmistaja oli Rollset Oy.

Ohjelma toteutettiin C++ -ohjelmointikielellä ja Windowsin kanssa kommunikointiin käytettiin WIN32 API:a. Ohjelman 3D -toiminnot tehtiin käyttäen Direct3D:tä.

Ohjelmaan toteutettiin tietokantayhteydet, sillä sen oli osattava hakea tietoa teräsrakenteista Steelconin tietokannoista. Myöhemmässä vaiheessa ohjelmaan lisättiin toiminto, jonka avulla teräsrakenteiden tietoa pystyttiin myös lukemaan XML-tiedostoista.

Avainsanat

CAD, 3D ohjelmointi, Direct3D

Luottamuksellisuus

julkinen

SAVONIA UNIVERSITY OF APPLIED SCIENCES

Degree Programme

Information Technology

Author

Jyrki Jauhiainen

Title of Project

CAD Data Viewer

Type of Project	Date	Pages
Final Project	7 April 2011	28 + 11

Academic Supervisor	Company Supervisor
Mr Sami Lahti, Lecturer	Mr Sami Lahtinen, Project Manager

Company

Rollset

Abstract

The aim of this thesis was to produce a 3D viewer program which displays steel frames designed with the StruCad CAD program. The program was originally meant to be a part of the Steelcon steel frame production line. The Steelcon production line was manufactured by Rollset Oy.

The program was developed with C++ and Win32 API was used for the communication between the operating system and the program. Direct3D was used for 3D graphics operations and for communication with the display adapter.

As a result, database connections were implemented into the program to enable it to access Steelcon databases. Afterwards a feature was implemented that allowed the program to read steel frame data from XML files.

Keywords

CAD, 3D programming, Direct3D

Confidentiality

public

ALKUSANAT

Tämä opinnäytetyö tehtiin Rollset Oy:lle. Työn tarkoituksena oli toteuttaa 3D-sovellus, joka pystyy näyttämään CAD:ltä tulevaa dataa. Haluan kiittää opinnäytetyön ohjaajaani Sami Lahtea ohjauksesta, Sami Lahtista ja Rollset Oy:n koko henkilökuntaa mahdollisuudesta työskennellä osana teräsrakenteiden tuotantolinjan suunnittelu- ja valmistusprosessia. Haluan myös kiittää perhettäni, ystäviäni ja kollegaani Juho Kokkosta tuesta ja avusta tämän insinöörityön aikana.

Kuopiossa 7.4.2011

Jyrki Jauhiainen

SISÄLLYSLUETTELO

LYHENTEET JA TERMIT

Peruskäsitteet:

CAD	Computer Aided Design tarkoittaa sovellusta, jolla suunnitellaan esimerkiksi rakennuksia.
RME	Juho Kokkosen kehittämä valmistuksenohjausjärjestelmä.
StruCad	teräsrakenteiden mallinnukseen tarkoitettu CAD.
SteelCon	Rollset Oy:n valmistama teräsrakenteiden tuotantolinja.

Ohjelmointi:

C++	oliopohjainen ohjelmointikieli.
.NET	Microsoftin kehittämä ohjelmistoalusta helpottamaa ja nopeuttamaan ohjelmointia.
C#	Microsoftin kehittämä oliopohjainen ohjelmointikieli .NET kehitystä varten.
Visual Studio	Microsoftin ohjelmointiympäristö, jossa on tuki .NET, C/C++ ja Basic-ohjelmoinnille.
API	Application Programming Interface mahdollistaa ohjelman kommunikoinnin ulkoisten komponenttien ja tois ten ohjelmien kanssa.
DLL	Dynamic Link Library Sisältää ohjelmatiedoston ulkopuolisia toimintoja Windows-ympäristössä.
WIN32 API	sisältää Windowsin kaikki perustoiminnot, mutta ei mitään aputoimintoja, toisin kuin esimerkiksi MFC tai .NET Forms.

WindowProc WIN32 API:n CALLBACK -funktio, jota kutsutaan aina, kun ikkunaan kohdistuu jokin tapahtuma. WindowProc tai DialogProc on ylikirjoitettava jokaisessa sovelluksessa, mikä käyttää Windowsin ikkunoita.

3D -ohjelmoinnti:

Direct3D Microsoftin kehittämä rajapinta 3D -grafiikan piirtämiseen ja näytönohjaimen kanssa kommunikointiin.

OpenGL Khronos Groupin hallinnoima monialustainen rajapinta 3D -grafiikan piirtämiseen ja näytönohjaimen kanssa kommunikointiin.

Rasterointi on selitetty kappaleessa 2.2.

Ray Tracing on selitetty kappaleessa 2.1

FFP Fixed Function Pipeline

PP Programmable Pipeline

Vertex yksi 3D -mallin piste.

Face 3D -mallin vertexien muodostama polygoni, yleensä kolmio.

Pipeline liukuhihna on joukko toimintoja, jotka suoritetaan perä jälkeen lopullisen kuvan tuottamiseksi

Model Space XYZ-koordinaatisto, jossa 3D -mallin vertexit ovat suhteessa toisiin saman mallin pisteisiin. Koordinaatiston origo on mallin keskipiste.

World Space XYZ-koordinaatisto, jossa 3D -mallien vertexit ovat suhteessa toisiin 3D -malleihin. Koordinaatiston origo on maailman keskipiste.

Screen Space XY-koordinaatisto, jossa 3D -mallien pisteet ovat rasteroituna 2D -alustalle. Origo on alustan yläkulma.

Surface alusta tarkoittaa mitä tahansa muistialuetta johon kuva voidaan piirtää, esim. tekstuuri.

HDR High Dynamic Range. HDR mahdollistaa 3D -ohjelmoinnissa korkeampia valontiheyksiä, kuin näyttö laitteet pystyvät tuottamaan. Näytön sen hetkisen dynaamisen alueen ylimeneviä valontiheyksiä pyritään simuloimaan erilaisilla efekteillä, kuten esimerkiksi Bloom.

Stencil Shadows menetelmä jossa stencil bufferia käytetään apuna dynaamisten varjojen luontiin.

Front ja **Back Buffer** selitetty kappaleessa 2.2.5

Z-buffer selitetty kappaleessa 2.2

Color Buffer tarkoitettaa mitä tahansa bufferia, mihin lopullinen kuva piirretään (Esim. Back Buffer tai virtuaalinen teks tuuri).

Scan-line conversion on vaihe, jossa päätellään mitkä kuvapisteet lopulta väritetään.

1 JOHDANTO

Tämän opinnäytetyön tarkoitus oli alun perin tehdä Rollset Oy:n Steelcon-teräsrakenteidentuotantolinjaa varten teräsmallien suunnittelua helpottavia StruCAD-makroja ja 3D -sovellus helpottamaan teräselementtien kokoamista. Heti alkuvaiheessa huomattiin, että kummatkin aihealueet yhdessä olisivat liikaa tähän opinnäytetyöhön, joten makrojen toteutus jätettiin pois opinnäytetyöstä ja se tehtiin normaalina palkallisena työnä Rollset Oy:lle.

Rollset Oy oli 2000-2010 välillä toiminut metallialan yritys, joka oli erikoistunut rullamuovaukseen. Rollset Oy työllisti Suomessa ja ulkomailla yhteensä noin 30 henkilöä ja sen päätoimipiste sijaitsi Kuopiossa. Rollset Oy keskittyi toimittamaan rullamuovauslaitteita Itä-Eurooppaan.

Steelcon Factory on Rollset Oy:n ensimmäinen teräsrakenteiden valmistuslinja. Steelcon Factory sisälsi kaiken talon teräsrakenteiden valmistukseen tarvittavan laitteiston, CAD-suunnitteluohjelmistoista teräselementtien kokoonpanopöytiin asti. Koko ohjelmistopuoli rullamuovausta lukuun ottamatta oli Rollset Oy:lle täysin uutta. Steelcon Factoryn ohjelmointitehtäviä hoitamaan palkattiin lisäkseni kollegani Juho Kokkonen. Kokonaisvastuu Steelcon-projektin valmistumisesta oli projektipäällikkö Sami Lahtisella. Lisäksi projektiin osallistui useita alihankkijoita, tavarantoimittajia, asentajia sekä suunnittelijoita.

Rollset Oy:n toimitusjohtaja Juha Reinikka ja projektipäällikkö Sami Lahtinen kertoivat 3D -sovelluksen pääkehityssuunnan, mutta sovelluksen tarkempi suunnitteleminen ja käytettävien tekniikoiden valinta oli vapaasti valittavissa. Koska 3D -sovellukselle ei koskaan valittu lopullista nimeä, tässä dokumentissa käytetään tuotantovaiheessa annettua nimeä Rollrunner.

Alun perin Rollrunnerin ainoa tarkoitus oli näyttää elementtien kokoamisohjeet 3D -ympäristössä. Rollrunner päätettiin toteuttaa C++:lla ja koska Rollrunnerin täytyi toimia vain Windows-ympäristössä, valittiin Windows-sovelluksen toteutukseen WIN 32 API. Kommunikointiin näytönohjaimen kanssa valittiin Microsoftin kehittämä Direct3D.

Olin aikaisemmin toteuttanut 3D -sovelluksia OpenGL:llä ja Direct3D:llä kokemukseni oli rajoittunut muutamaan testisovellukseen. Kokemukseni OpenGL:stä

helpotti kuitenkin huomattavasti alkuunpääsyä Direct3D -ohjelmoinnissa, ja sovelluksen toteutus sujui ilman suurempia ongelmia.

2 KÄSITTELY

2.1 3D -grafiikasta yleisesti

Kolmiulotteisen ja kaksiulotteisen grafiikan ero tietotekniikan näkökulmasta on usein hyvin epämääräinen. Kolmiulotteisen ja kaksiulotteisen grafiikan grafiikoiden piirtämiseen käytetään hyvin samantapaisia menetelmiä, joten grafiikkatyylien välille ei voida vetää aivan tarkkaa rajaa. Yleisesti kuitenkin 3D-grafiikkana pidetään mallia, joka on matemaattisesti määritelty kolmiulotteiseksi. Mallit voivat koostua muun muassa matemaattisesta funktiosta, funktioista tai pelkästä pistetiedosta. Kolmiulotteista mallia ei kuitenkaan voida piirtää suoraan näytölle, sillä nykypäivän näyttölaitteet ovat täysin kaksiulotteisia laitteita. Edes nykyiset 3D -näytöt ja -televisiot eivät tunne kolmatta ulottuvuutta, vaan perustuvat siihen, että ihmisen vasemmalle ja oikealle silmälle näytetään kahta aavistuksen toisistaan eroavaa kaksiulotteista kuvaa syvyysvaikutelman luomiseksi.

Jotta kolmiulotteinen malli saataisiin kaksiulotteiseksi kuvaksi, on käytettävä jotain 3D -renderöintimenetelmää. Kaksi yleisintä menetelmää ovat Ray Tracing ja rasterointi. Ray Tracingia käytetään yleensä esirenderoidun kuvan ja animaation piirtämiseen, kun taas rasterointia käytetään usein reaaliaikaisissa sovelluksissa.

Ray Tracingin idea on, että kuvan jokaiselle kuvapisteelle etsitään väri seuraamalla valolähteestä lähteviä säteitä. Koska valosta voi kuitenkin lähteä käytännössä ääretön määrä säteitä, lähdetään valon sädettä seuraamaan kuvapisteestä kohti valonlähdettä. Ray Tracingin suurin etu on sen helppous tuottaa fotorealistisia kuvia, mutta sen nopeus on harvemmin riittävä reaaliaikaisiin sovelluksiin.

Rasteroinnin perusidea on muodostaa kolmiulotteisen mallin pisteiden välille polygoneja, rasteroida polygonit kaksiulotteiseksi projektio- ja kuvakulmamatriisin avulla. Polygonit voidaan piirtää ruudulle jollain tietyllä värillä, tai kuten nykypäivän sovelluksissa, niiden sisältö voidaan täyttää kaksiulotteisella kuvalla eli tekstuurilla.

2.2 Rasterointiprosessi

Koska Rollrunnerin oli pystyttävä reaaliaikaiseen renderöintiin, valittiin Rollrunnerin renderöintimenetelmäksi rasterointi.

2.2.1 Matriisit

Kuten aikaisemmassa luvussa todettiin, rasteroinnin idea on muuttaa kiinteä kolmiulotteinen malli kaksiulotteiseksi jonkin tietyn projektion avulla. Yksinkertaisimmillaan perspektiiviprojektio käyttää origoa projektion keskipisteenä ja z = 1 näytön pintana:

$$\begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \\ z \end{bmatrix} \tag{1}$$

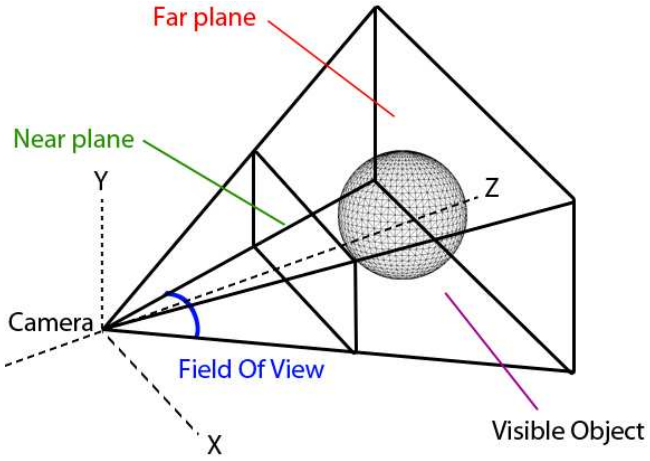
laskutoimituksen jälkeen homogeeninen komponentti w ei ole enää yhtä kuin yksi, joten jokainen komponentti on jaettava w:llä, kun ne kartoitetaan 2D-tasolle. Matriisi 1 ei huomioi kuvasuhdetta eikä näkökenttää. Näkökentän ja kuvasuhteen huomioonottamiseksi Direct3D:ssä käytetään seuraavanlaista matriisia [1]:

$$\begin{bmatrix} x_{\text{Koko}} & 0 & 0 & 0 \\ 0 & y_{\text{Koko}} & 0 & 0 \\ 0 & 0 & z_{\text{k}}/(z_{\text{k}} - z_{\text{l}}) & 1 \\ 0 & 0 & -z_{\text{l}} \cdot z_{\text{k}}/(z_{\text{k}} - z_{\text{l}}) & 0 \end{bmatrix} \tag{2}$$

missä

$$y_{\text{Koko}} = \cot\frac{\alpha_{\text{näkökenttä}}}{2} \text{ ja } x_{\text{Koko}} = \frac{y_{\text{Koko}}}{k_{\text{kuvasuhde}}} \tag{3}$$

Kaavassa 2 ja 3 $\alpha_{\text{näkökenttä}}$ tarkoittaa näkökenttää asteina, kun toisaalla z_{l} tarkoittaa lähitason z-arvoa ja z_{k} tarkoittaa kaukotason z-arvoa. $k_{\text{kuvasuhde}}$ on alustan vaaka-akselin kuvapisteiden määrä jaettuna pysty-akselin kuvapisteiden määrällä. Kuva 1 esittää Matriisin 2 luomaa projektiota.



Kuva 1. Projektio

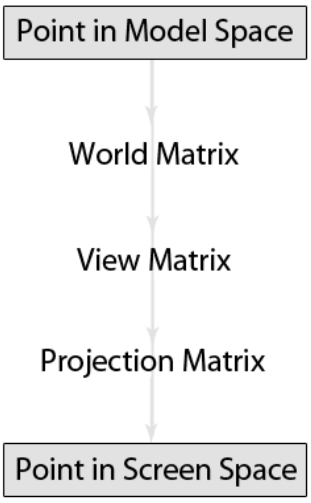
Kamera-matriisin avulla voidaan projektion keskipistettä ja suuntaa vaihtaa reaaliavaruudessa. Kameramatriisi voidaan luoda seuraavalla tavalla [2]:

$$\begin{bmatrix} X_X & Y_X & Z_X & 0 \\ X_Y & Y_y & Z_y & 0 \\ X_z & Y_z & Z_z & 0 \\ -(X \cdot K_{\text{paikka}}) & -(Y \cdot K_{\text{paikka}}) & -(Z \cdot K_{\text{paikka}}) & 1 \end{bmatrix} \quad (4)$$

jossa Z-vektori on kameran kohdevektorin ja paikkavektorin (K_{paikka}) erotus normalisoituna. X on Ylös-vektorin ja Z-vektorin ristitulo normalisoituna ja Y on Z- ja X-vektorin ristitulo. Ylös-vektori ilmoittaa, mikä akseli on ylöspäin esim. (0, 1, 0) tapauksessa Y-akseli on ylöspäin. Kyseisen kameramatriisin muodostamiseksi tarvitaan siis seuraavat alkutiedot: kameran paikka, piste johon kamera on kohdistettu ja tieto siitä, mikä akseli osoittaa ylöspäin. Lopputuloksena syntyy kameran muunnosmatriisin käänteismatriisi, mutta sen luominen edellä mainituista lähtötiedoista on huomattavasti nopeampaa kuin kameran muunnosmatriisin kääntäminen.

Kaksi edellä mainittua Direct3D:n käyttämää matriisia ovat vasemman käden systeemille tarkoitettuja matriiseja. Direct3D:n laajennuskirjasto tarjoaa myös funktiot oikean käden systeemeille. Funktioiden eroavaisuudet ovat hyvin pieniä.

Lisäksi rasterointiprosessissa mallin kaikki pisteet kerrotaan objektikohtaisella muunnosmatriisilla, jotta mallin paikkaa, orientaatiota tai kokoa voitaisiin muuttaa lopullisessa kuvassa. Todellisuudessa rasteroinnin matriisilaskut suoritetaan Kuva 2 mukaisessa järjestyksessä. Ensin mallin pisteet kerrotaan objekti muunnosmatriisilla, sen jälkeen ne kerrotaan kameramatriisilla ja lopuksi projektiomatriisilla.

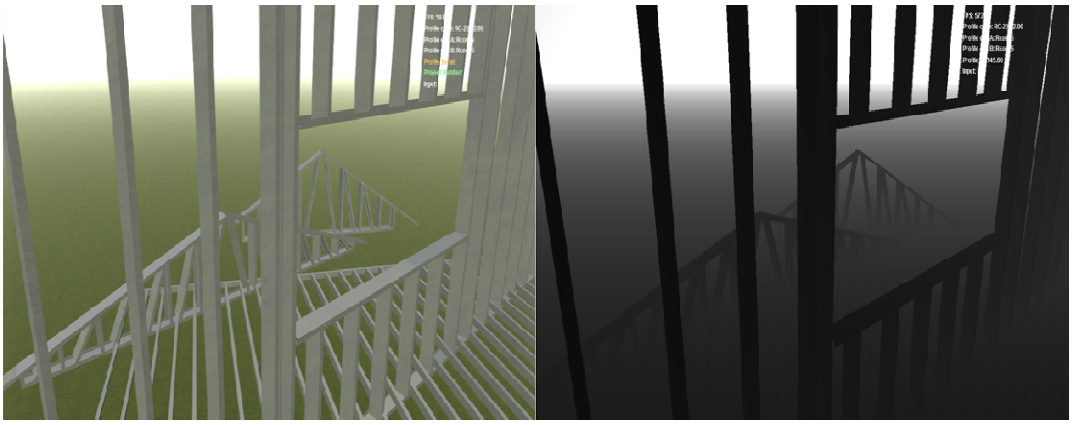


Kuva 2. Matriisit

Kun kolmiulotteisen mallin pisteet on viimein saatu asetettua kaksiulotteiselle tasolle, suoritetaan clipping. Clipping-vaiheessa alustan ulkopuolella olevat pisteet poistetaan, jottei niitä piirrettäisi turhaan. Clippingin jälkeistä viimeistä vaihetta kutsutaan Scan-line Conversioksi. Tässä vaiheessa mallin polygonit täytetään tekstuureilla polygonien pisteiden tekstuurikoordinaattien mukaisesti. Scan-line Conversion aikana piirrettyjä kuvapisteitä voidaan varjostaa pinnasta heijastuvan valon, peilauksen ja varjojen mukaisesti.

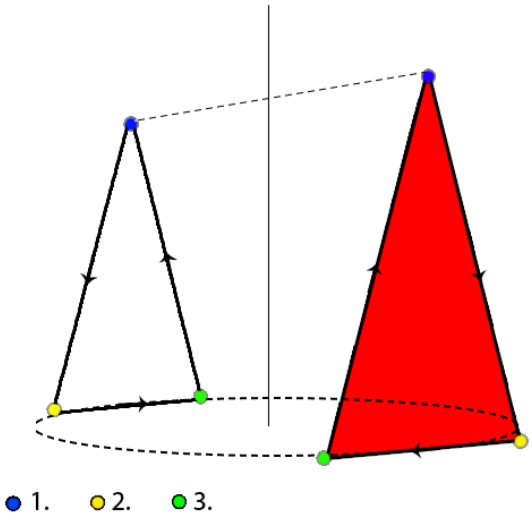
2.2.2 Z-Buffer

Koska rasteroinnissa lähtökohtaisesti kaikki polygonit piirretään, tarvitaan jonkinlainen tieto piirrettyjen kuvapisteiden syvyydestä, jottei takana olevia polygoneja piirrettäisi edessä olevien päälle. Yleinen ratkaisu edellä mainittuun ongelmaan on Z-bufferointi. Z-bufferoinnissa jokaisen alustalle piirretyn kuvapisteen syvyys tallennetaan kaksiulotteiseen taulukkoon. Taulukossa jokaiselle kuvapisteelle on oma alkio (verrattavissa tekstuuriin). Aina kun alustalle ollaan piirtämässä uutta kuvapistettä, tarkastetaan Z-bufferista aikaisemman samassa kohdassa olevan kuvapisteen syvyys ja päätetään piirretäänkö kuvapiste. Päällimmäisin piirretään ja sen syvyys tallennetaan Z-bufferiin. Seuraavassa kuvassa (Kuva 3) vasemmalla puolella on Color Bufferissa oleva rasteroitu kuva ja vasemmalla Z-bufferissa olevat arvot kirjoitettuna tekstuuriin.



Kuva 3. Z-buffer

2.2.3 Culling ja varjostus



Kuva 4. Culling

Koska vertexien muodostamien polygonien tarkoitus on muodostaa yksi pinta tiettyyn suuntaan, ei ole järkeä piirtää polygonia, mikäli se on lopullisessa kuvassa väärinpäin. Toimenpidettä, jolla väärinpäin olevan polygonin piirto estetään, kutsutaan cullingiksi. Jotta ylimääräiseltä laskemiselta säästyttäisiin, polygonin pinnan suunta ratkaistaan yleensä polygonien järjestyksestä. Culling-menetelmiä on siis kaksi, myötäpäivään ja vastapäivään. Kun polygonien culling tehdään vastapäivään, piirretään polygoni aina kun sen pisteiden järjestys lopullisessa kuvassa on myötäpäivään. Myötäpäivään tapahtuvassa cullingissa operaatio on päinvastainen (Kuva 4).

Koska rasteroinnissa polygonit piirretään sellaisenaan, on käytettävä erilaisia kuvan parannustekniikoita, jotta lopullinen kuva saataisiin näyttämään paremmalta. Aivan perusmenetelmiin kuuluvat varjostusefektit, joista tunnetuin on Phong-varjostus. Phong-varjostuksessa kuvitteellinen valonlähde sijoitetaan kolmiulotteiseen maailmaan. Piirrettävien kuvapisteiden värin määrää kasvate-

taan tai lasketaan sen mukaan, mihin suuntaan piirrettävän pinnan normaali (pintavektori) osoittaa. Muita kuvanparannusmenetelmiä ovat erilaiset varjostusmenetelmät, kuten Shadow Mapping ja Stencil Shadows, erilaiset HDR-efektit ja heijastukset.

2.2.4 Liukuhihnat ja shaderit

Direct3D 9 tukee kiinteätoimintaista (FFP) ja ohjelmoitavaa (PP) liukuhihnaa. Kiinteätoimintaisessa liukuhihnassa rasterointi ja kuvapisteiden varjostus tehdään ennalta määrättyjen Direct3D:n standardien mukaisesti, kun taas ohjelmoitavassa liukuhihnassa ne ovat täysin ohjelmoitavissa. Ohjelmoitavassa liukuhihnassa ohjelmoijalla on rajattomat mahdollisuudet toteuttaa erilaisia efektejä käytettävien resurssien - eli tietokoneen tehojen - puitteissa. Ohjelmoitavaa liukuhihnaa käyttäessä näytönohjaimen shaderit hoitavat vertexeihin kohdistuvat laskutoimitukset sekä kuvapisteiden varjostuksen. Näytönohjaimen shadereitä ohjelmoidaan niin kutsutulla shader-kielellä. Shader-kieli on hyvin lähellä PC:n konekieltä ja se on universaali ohjelmointikieli, jota eri valmistajien näytönohjaimet tukevat aina tiettyyn versioon asti. Shader-kielen eri versioista käytetään nimitystä Shader Model. Shader Modelin uusin versio oli tätä dokumenttia kirjoittaessa 5.0. Ohjelmoinnin helpotukseksi Direct3D tarjoaa oman shader-ohjelmointikielensä, High Level Shader Languagen (HLSL), joka on toisaalta hyvin lähellä C-ohjelmointikieltä.

Rollsetin halutessa mahdollisimman hienoa ja sulavaa grafiikkaa, Rollrunnerissa päädyttiin käyttämään ohjelmoitavaa liukuhihnaa. Käytettäväksi Vertex Shaderin versioksi valittiin 2.0 ja Pixel Shaderin versioksi 2.0b, sillä nämä versiot ovat hyvin tuettuina Intelin integroiduissa näytönohjaimissa.

Shader-koodi voidaan kirjoittaa esimerkiksi ASCII-muotoon, joka tallennetaan tekstitiedostoksi. Pääohjelman ajon aikana shader-koodia sisältävä tiedosto ladataan ja se annetaan Direct3D:lle käännettäväksi. Käännöksen jälkeen shaderiä voidaan käyttää asettamalla se käytössä olevaksi shaderiksi.

Koska Shader Model 2.0:ssa Vertex Shader antaa tulosteensa suoraan Pixel Shaderille, ja Pixel Shaderin tuloste kirjoitetaan suoraan alustalle, on Shader-koodin debuggaaminen erityisen hankalaa. Shader-koodin debuggaukseen on

olemassa erillisiä ratkaisuja, kuten esimerkiksi Microsoftin PIX, joka toimitetaan DirectX SDK:n mukana. PIX:n avulla ohjelmoija voi tarkastella Pixel Shaderin ja Vertex Shaderin saamia ja tulostamia arvoja tietyllä ajanhetkellä. Itse koin PIX:n käyttämisen erittäin työlääksi, joten suurimman osan debuggauksesta suoritin AMD:n ilmaisella GPU Shader Analyzer-ohjelmalla. Shader Analyzer ei ole reaaliaikainen debuggausohjelma, vaan tarkoitettu lähinnä etsimään Shader-koodissa esiintyviä pullonkauloja. GPU Shader Analyzer osaa kuitenkin näyttää HLSL:n muuttujat ja operaattorit erivärisinä ja kertoa syyn, mikäli Shaderin kääntäminen epäonnistuu.

2.2.5 Muut bufferit

Yleisesti 3D -ohjelmoinnissa puskurilla (bufferilla) tarkoitetaan mitä tahansa piirtoalustaa, eli aluetta, joka on varattu laitteen muistista piirtämistä varten. Lähes kaikissa reaaliaikaisissa grafiikkasovelluksissa käytetään Double Bufferingiä eli tuplapuskurointia. Double Bufferingin tarkoitus on poistaa tai vähentää kuvan repeilyä ja artefakteja, jotka aiheutuvat tahdistusongelmista näyttölaitteen kanssa.

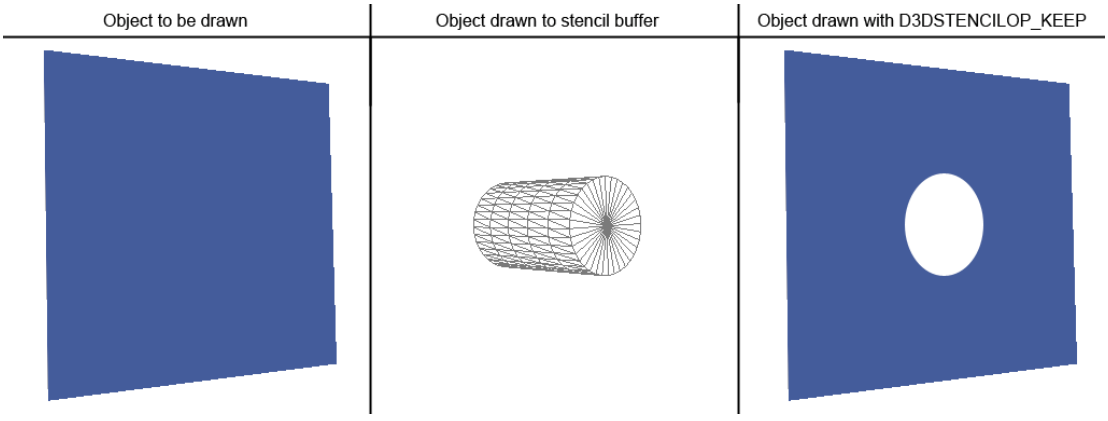
Suurin osa nykyisistä näyttölaitteista toimii 60 Hz taajuudella, joka tarkoittaa että näytön kuva päivitetään 60 kertaa sekunnissa. Toisaalta kuvaa piirtävä ohjelma voi piirtää kuvan millä taajuudella tahansa. Vaikka ohjelma ja näyttö toimisivatkin samalla taajuudella olisi niiden synkronoiminen ilman erityistä rautapohjaista ratkaisua lähes mahdotonta. Mikäli näyttö hakisi kuvan suoraan alustalta, johon kuvaa piirretään, olisi hyvin epätodennäköistä että kuva olisi valmis oikealla hetkellä. Valmistumattoman kuvan piirtäminen toisaalta aiheuttaisi kuvaan edellisessä kappaleessa mainittuja ongelmia.

Double Buffering ratkaisee ongelman kahdella eri puskurilla, Front Bufferilla ja Back Bufferilla. Front Buffer pitää sisällään valmiin kuvan, ja Back Bufferia käytetään piirtoalustana. Kun kuva on saatu valmiiksi Back Bufferiin, siirretään se Front Bufferiin kokonaisuudessaan. Näin varmistetaan, ettei näytölle piirretä kuvaa, joka ei ole vielä valmis. Koska kuvan kopioiminen puskureiden välillä hidastaisi ohjelman toimintaa, käytetään yleensä Page Flipingiä. Page Flipingissä kumpikin puskuri voidaan piirtää näyttölaitteelle (kummatkin sijaitsevat VRAM:ssa) ja kuvan siirtämisen sijaan piirrettävää puskuria vaihdetaan. Toisin

sanoen, kun Back Bufferissa on kuva saatu valmiiksi, siitä tulee Front Buffer ja Front Bufferista Back Buffer.

Muita 3D -ohjelmoinnissa käytettyjä puskureita ovat muun muassa Stencil Buffer, Color Buffer ja aikaisemmin tässä dokumentissa mainittu Z-buffer. Color Bufferilla tarkoitetaan yksinkertaisesti tekstuuria tai Back Bufferia (joka voidaan myös käsittää eräänlaisena tekstuurina, mutta se on osana Swap Chainia), johon piirrettävän kuvan värit piirretään, eli alustaa johon lopullinen kuva tulee.

Teoriassa Stencil Buffer toimii kuten mikä tahansa normaali yksivärinen piirtoalusta, johon voidaan piirtää 2D -kuvia tai rasteroituja 3D -kuvia. Color Bufferiin kirjoittaessa tietoa Stencil Bufferia voidaan käyttää rajoittamaan piirrettävää objektia. Yksikertaisimmillaan Stencil Bufferilla voidaan rajata jokin alue mihin kuva halutaan piirtää, mutta Stencil Bufferille on myös käyttöä monimutkaisimmissa toimenpiteissä, kuten esimerkiksi Stencil-varjoissa. Alla olevassa kuvassa (Kuva 5) vasemmalla puolella on Color Bufferiin kirjoitettava objekti ja keskellä on Stencil Bufferiin kirjoitettu objekti. Oikealla puolella on lopullinen tulos Color Bufferissa, kun vasemmalla oleva objekti piirretään sinne stencil-testauksen ollessa kytkettynä päälle. Mikäli Stencil- ja Z-Bufferista löytyy samasta kohtaa piste kuin piirrettävästä objektista, suoritetaan jokin stencil-operaatio. Kuva 5 suoritettava operaatio on D3DRSSTENCILOP_KEEP, joka säilyttää kuvapisteen alkuperäisen arvon.



Kuva 5. Stencil Buffer

2.3 Rollrunnerin rakenne

Rollrunnerin perusrakenne on yksinkertainen. Ohjelman suorituksen alkaessa ensimmäiseksi alustetaan tarvittavat muuttujat ja katsotaan näytön ja näytönohjaimen tukemat tekniikat. Tämän jälkeen luodaan Windows-ikkuna, jonka HWND-osoitin annetaan grafiikka-luokan hallintaan. Kun alkutoimenpiteet on suorittu, käynnistetään pääsilmukka, joka kutsuu syötteenhallintaa, logiikkaa ja grafiikkaa tietyn väliajoin.

2.3.1 Pääluokkien toiminta

Rollrunner-ohjelma sisältää kaiken kaikkiaan 45 luokkaa, joten niiden kaikkien toimintaa ei voida käsitellä tässä dokumentissa. Perusrakennetta voidaan kuitenkin kuvata selvittämällä muutaman pääluokan toimintaa. Kuten aikaisemmassa kappaleessa todettiin, Rollrunnerin toiminta tapahtuu yhdessä pääsilmukassa. Pääsilmukan kierrosnopeus on rajoittamaton, sillä logiikkaluokka hoitaa ohjelman ajoituksen sataan kertaa sekunnissa. Tarvittaessa Graphics3D-luokka osaa hillitä kuvien piirtämisnopeutta. Ensin pääsilmukassa kutsutaan InputManager-luokkaa, joka hoitaa kaikki ohjelmaan tulevat painallukset ja Windowsin tapahtumat implementoimalla WinProc Callback-funktion. WinProc Callback-funktiota kutsutaan Windows-ohjelmille tyypillisesti aina, kun sovelluksen ikkunaan tai ikkunoihin kohdistuu jokin tapahtuma. Tapahtumien käsittelyn lisäksi InputManager hoitaa kaikki ohjelman sisäisiin valikkoihin kohdistuvat toiminnot ja suorittaa objektien valitsemiseen tarvittavat laskutoimitukset.

Logic-luokka hoitaa ajoituksen lisäksi kaikki CAD-malleihin kohdistuvat toiminnot, kuten lataukset, siirtämiset ja poistot. Logic-luokan tehtävänä on myös kutsua tietokantayhteyksiin käytettäviä luokkia ja hoitaa tietokantayhteyksien kautta tapahtuvat mallien lataukset.

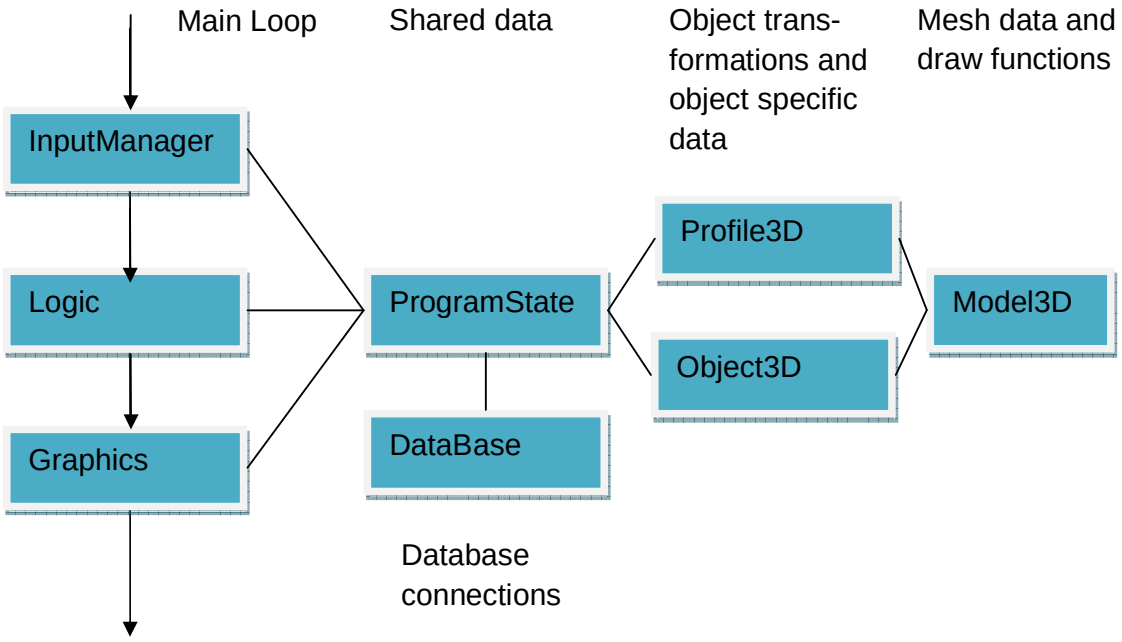
Graphics-luokka hoitaa kaikki piirtämiseen liittyvät toiminnot, joita ovat muun muassa:

- näytön ja näytönohjaimen tukemien tekniikoiden tarkistus
- Direct3D device -luokan alustus
- surfacejen ja dynaamisten tekstuurien alustus
- tekstuurien lataus tiedostosta

- shaderien ja liukuhihnojen hallinta.

Objektiluokat Object3D, Profile3D ja 3D -mallien hallintaa tarkoitettu luokka Model3D ovat myös mainitsemisen arvoisia, sillä ne hoitavat mahdollisten 3D-mallien latauksen tiedostosta ja profiilimallien generoinnin. Object3D -luokka on tiedostosta ladattavia 3D -malleja varten, kun taas Profile3D generoi CAD:ltä tulleista tiedoista 3D -mallin. Objektit ja itse 3D-malli on erotettu toisistaan, koska useampi objekti voi käyttää samaa 3D -mallia ja yksi objekti voi koostua useammista malleista. Lisäksi objektiluokat hoitavat PC:n prosessorilla tapahtuvat matriisilaskennat ja huolehtivat siitä, että verteksit, tekstuurit ja muut renderöintiin tarvittavat tiedot päätyvät shadereille.

ProgramState on singleton tyyppinen luokka, jonka osoitin annetaan lähes jokaiselle luokalle niiden alustuksen yhteydessä. ProgramState toimii tiedonvälittäjänä kolmen pääluokan välillä, ja mahdollistaa myös alaluokkien tiedon saannin suoraan toisilta luokilta. Seuraavassa kuvassa (Kuva 6) on yksinkertainen kuvaus edellä mainittujen luokkien suhteista.



Kuva 6. Rollrunnerin perusrakenne

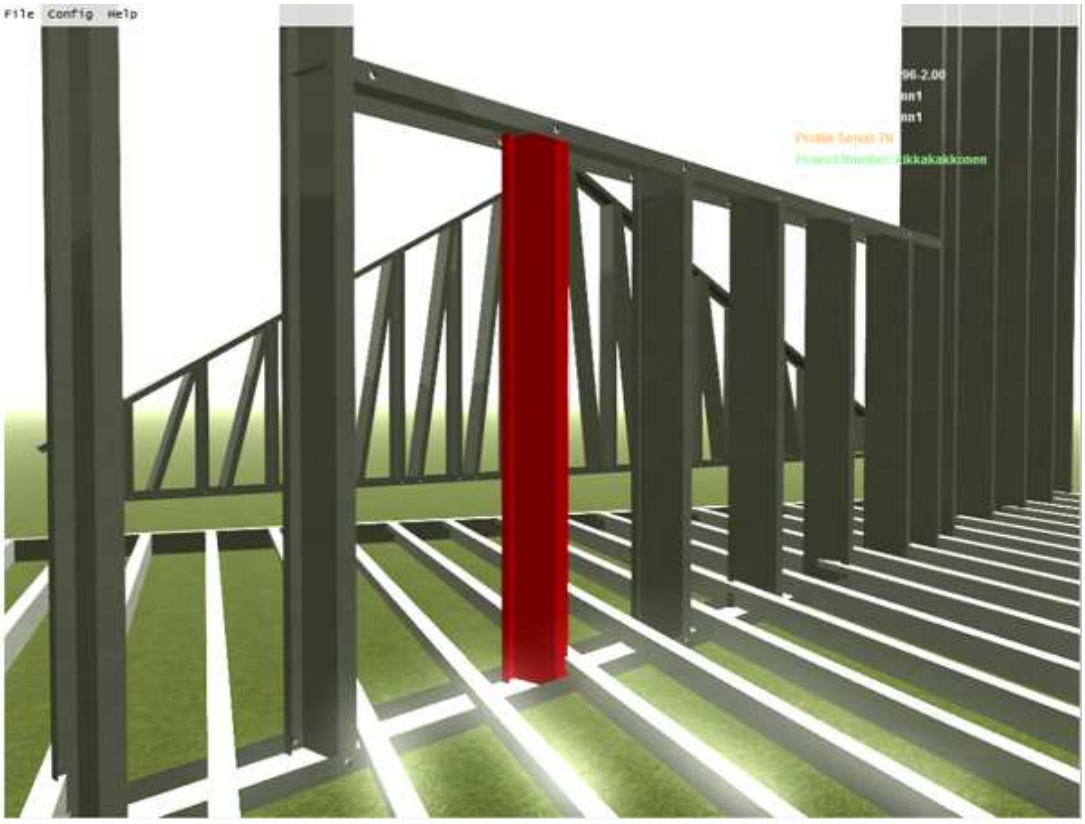
2.3.2 Rollrunnerin liukuhihnat

Rollrunner-ohjelmassa on käytössä kaksi renderöintiliukuhihnaa; HDR-liukuhihna ja normaali liukuhihna. Normaalilla liukuhihnalla tarkoitetaan liukuhihnaa mikä ei tee lopulliselle kuvalle minkäänlaista jälkikäsittelyä. Rollrunner-ohjelma tukee myös dynaamisia stencil-varjoja ja depth of fieldiä, joita ei kuitenkaan ole vielä tässä vaiheessa yhdistetty kumpaankaan liukuhihnaan.

Normaalin liukuhihnan toiminta on yksinkertainen. Liukuhihnan alustuksessa ladataan Rollrunnerin Shader-tiedostoista Vertex Shader ja Phong Pixel Shader, jotka käännetään näytönohjaimen ymmärtämään muotoon. Kun piirtäminen liukuhihnalla aloitetaan, Z-buffer ja kaikki muut käytettävät puskurit nollataan. Nollausten jälkeen Vertex ja Pixel Shadereille annetaan kaikki ne tiedot, jotka eivät ole objektikohtaisia. Näitä tietoja ovat muun muassa valojen paikat, voimakkuudet ja värit. Tämän jälkeen ProgramState luokassa sijaitseva _vObjects-taulukko käydään läpi kutsuen jokaisen ohjelmassa olevan Object-luokan instanssin draw-funktiota. Object-luokan draw-funktio antaa Vertex ja Pixel Shaderille objektin tiedot, kuten esimerkiksi objektin muunnosmatriisin, tekstuurin ja materiaalin. Tämän jälkeen Object-luokan draw-funktio kutsuu Model-luokan Draw-funktiota, joka lopulta toimittaa mallin tiedot ja tekstuurin Vertex ja Pixel Shadereille.

Kun kolmiulotteiset mallit on piirretty, kutsutaan Graphics3D-luokan draw_menus-funktiota, joka huolehtii kuvan kaksiulotteisten komponenttien piirtämisestä. Kun kaksiulotteiset komponentit on piirretty, lopetetaan kuvan piirtäminen EndScene-funktiolla ja kutsutaan Direct3D -laittteen Present-funktiota, joka vaihtaa Front Bufferin ja Back Bufferin paikkaa, joten kuva piirtyy näytölle. HDR-liukuhihnan, Stencil-varjojen, Depth of fieldin ja muiden käytettyjen kuvanparannusefektien toiminnat ovat varsin monimutkaisia ja niiden perusteellinen selittäminen menisi reilusti tämän dokumentin aihealueen yli.

Kuvassa (Kuva 7) on Rollrunner HDR-asetukset päällä. Vaakatasossa olevista profiileista on nähtävissä Bloom HDR-efekti.



Kuva 7. Rollrunner HDR-asetuksilla

2.3.3 Tietokantayhteydet

Kaikkien tietokantoihin liittyvien toimintojen toteuttamiseksi, Rollrunnerin tuli tukea kahta samanaikaista tietokantayhteyttä. Tietokantayhteyksien toteuttamiseksi valittiin Windows-käyttöjärjestelmien mukana tuleva ADODB-tietokantayhteyskomponentti. Windowsin ADODB-toiminnot löytyvät msado15-nimisestä DLL-tiedostosta. Rollrunnerin käynnistyksen yhteydessä msado15 ladataan ja tiedostosta löytyvät funktiot kapseloidaan luokaksi. Yhdistäminen tietokantaan tapahtuu yhteysmerkkijonon avulla, jossa on kaikki tarvittava tieto yhteyden luomiseksi. Tietokannan käsittely tapahtuu SQL:n avulla ja tietokannasta haettu tieto palautetaan Windows.h tiedostosta löytyvinä tietokantamuuttujina, jotka Rollrunnerisssa lopulta muutetaan C++:n standardimuuttujiksi.

2.3.4 RollCrypt

Koska tietokantayhteyksien tietoja jouduttiin säilyttämään levyllä tekstitiedostoissa, niiden suojaamiseksi oli Rollrunneriin toteutettava salaustoiminto. Windowsin komponenteista ei kuitenkaan löytynyt mitään helppokäyttöistä ja kaikilla alustoilla toimivaa ratkaisua, joten salauskomponentti päätettiin toteuttaa itse.

Salauskomponentti päätettiin toteuttaa erillisenä DLL-moduulina, jotta komponenttia voitaisiin käyttää muissakin Rollset Oy:n sovelluksissa. Salauskomponentin nimeksi valittiin RollCrypt ja salausmenetelmäksi AES-lohkosalausmenetelmällä.

RollCrypt tukee 128-, 192- ja 256- bittistä AES-lohkosalausta. AES on käytetyin salausmenetelmä ja sen toiminta perusteellisesti selitettynä löytyy helposti Internetistä. AES:n toimintaa ei käydä läpi tässä dokumentissa.

2.4 Rollrunnerin toiminnot

Rollrunner toteutettiin alun perin elementtien kokoamista helpottavaksi sovellukseksi. Rollrunnerin tarkoitus oli olla tuotantolinjan loppupäässä näyttämässä mihin kohtaan elementtiä valmistuva komponentti tulisi sijoittaa. Rollrunnerin oli siis pystyttävä hakemaan tietyn komponenttikoodin mukaan RME-tietokannasta sopiva komponentti. Lisäksi Rollrunnerin oli pystyttävä hakemaan juuri valmistuneen komponentin koodi tuotantokannasta reaaliajassa, kun Rollrunner oli kytketty tuotantolinjaan.

Koska Rollrunnerilla ei sinänsä ollut ohjelmallisia rajoitteita kuinka monta komponenttia ruudulla pystyttiin näyttämään, oli yhden kokonaisen talon kuin myös useiden talojen teräsrakenteiden näyttäminen samalla aikaa mahdollista. Tämän takia Rollrunneriin lisättiin toiminto, jonka avulla Rollrunner pystyi lukemaan myös Steelconin valmistusjärjestelmälle tarkoitettuja XML-tiedostoja, ja muodostamaan koko talon teräsrakenteen ilman tietokantayhteyksiä. Koska tässä vaiheessa Rollrunner ymmärsi komponenttien koodeja ja osasi lukea talon rakenteen XML-tiedostosta, huomattiin, että Rollrunneria pystyttiin käyttämään apuna talon teräsrakenteen kokoamisessa ilman tietokantayhteyksiäkin.

3 TARKASTELU

3.1 Rollrunnerin kehittäminen

Rollrunnerin kehittäminen aloitettiin yksinkertaisista prototyyppiversioista, joissa oli objektien liikuttelu, kääntäminen, skaalaus ja kameran liikuttelu. Seuraavaksi lisätty toiminto oli mallin lataaminen tiedostosta, jotteivät mallit olisi kovakoodattuja. Ensimmäisissä Rollrunnerin versioissa mallinnettavat komponentit luotiin venyttämällä kahta eri 3D-mallia, joista toinen oli U- ja toinen C-profiilille.

Kun 3D -moottorin perustoiminnot olivat kunnossa, Rollrunneriin toteutettiin tietokantayhteydet, jotta CAD-tietoja voitaisiin ladata komponenttien valmistuksen yhteydessä. Rollrunnerin dynaamisen rakenteen vuoksi jo tässä vaiheessa Rollrunner pystyi näyttämään useamman elementin ja näin ollen koko valmistettavan talon teräsrakenteet aivan kuten CAD:ssäkin.

Rollrunnerin näyttäessä CAD-malleja huomattavasti sulavammin kuin Steelcon järjestelmän mukana toimitettava StruCAD, halusi Rollsetin johto panostaa enemmän Rollrunnerin sulavuuteen ja näyttävyyteen. Tässä vaiheessa Rollrunnerin koko kiinteätoimintainen liukuhihna siirrettiin sivummalle ja sen rinnalle kehitettiin ohjelmoitava liukuhihna käyttäen shadereitä. Aluksi Rollrunnerin normaali rasterointi toteutettiin Phong-shadereillä, jonka jälkeen Rollrunneriin lisättiin depth of field-efekti. Seuraavana Rollrunneriin toteutettiin stencil-varjot, jotka kuitenkin koettiin hieman häiritseviksi kun ruudulla oli useampia elementtejä. Viimeinen graafinen lisäys Rollrunneriin oli HDR-liukuhihna, joka mahdollisti erilaiset HDR-efektit.

Erilaisien efektien toteuttamisen ohessa Rollrunner haluttiin myös näyttämään elementteihin tulevat detaljit yksityiskohtaisemmin. Koska komponentit olivat alun perin tehdyt venyttämällä yksittäistä profiilimallia, ei detaljien lisääminen ollut yksinkertaista. Aluksi koko profiilien muodostaminen suunniteltiin uudestaan, jonka seurauksena Rollrunner muodosti CAD:ltä tulevien tietojen avulla jokaiselle profiilille oman 3D -mallin, johon jätettiin aukot kohdille, joihin detaljit tuli. Piirtovaiheessa aukkojen kohdille piirrettiin sopivan näköinen detalji. Detaljien reiät ja koot oli määritelty XML-tiedostossa. Detaljille oli myös mahdollista

määritellä normal mapit, mikäli joidenkin osien detaljeissa kuuluu olla koholla tai detaljeissa on syvennyksiä.

3.1.1 DLL-versio

Rollset Oy halusi toteuttaa Juho Kokkosen ohjelmoimaan RME-ohjelmaan 3D -näkymän elementeistä ja komponenteista, joita RME:hen vietäisiin. Tämän toiminnon toteuttamiseksi katsottiin parhaaksi tehdä Rollrunnerista DLL-versio ja implementoida se RME:hen. Koska RME on .NET-pohjainen sovellus, ei Rollrunnerin implementointi ollut aivan ongelmatonta. Koska C#-muuttujille ei ole aivan vastaavia muuttujia C++:ssa, etenkin merkkijonot tuottivat aluksi ongelmia. Onnistuimme lopulta ratkaisemaan kriittisimmät ongelmat ohjelmien välisessä rajapinnassa ja saimme implementoitua Rollrunnerin RME:hen.

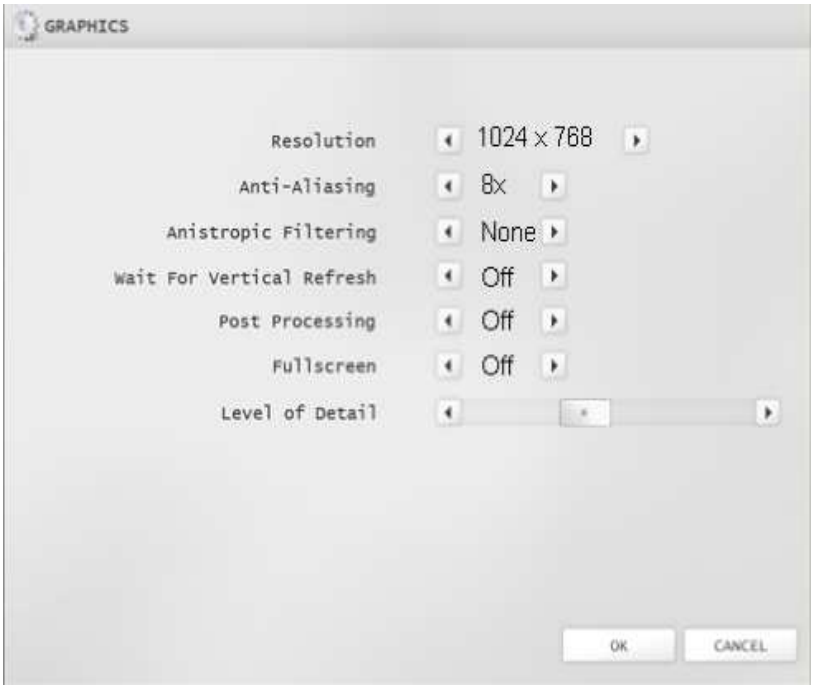
Pian implementoinnissa huomattiin kuitenkin vakava virhe: Aina kun RME:ssä käytettiin Rollrunner-komponenttia, lakkasivat RME:n tietokantayhteydet toimimasta. Useiden tuntien debuggaus- ja testaussessioiden jälkeen saimme selville, että ongelma johtui .NET-kellosta, joka piti sisällään tarkempaa tietoa kuin Rollrunnerissa käytetty kellofunktio. Rollrunnerin käynnistyessä Rollrunner kutsui omaa kello-funktiotaan ylikirjoittaen RME:n kelloajan, mikä aiheutti eheysongelman silloin kun RME:n datasettejä verrattiin tietokannassa oleviin tietoihin. Yksinkertaisin ratkaisu tähän ongelmaan oli pienentää tietokannassa käytettävien kellonaikojen tarkkuutta. Tämän jälkeen Rollrunner-komponentti alkoi toimimaan ongelmitta.

3.1.2 Ongelmakohtia kehityksessä

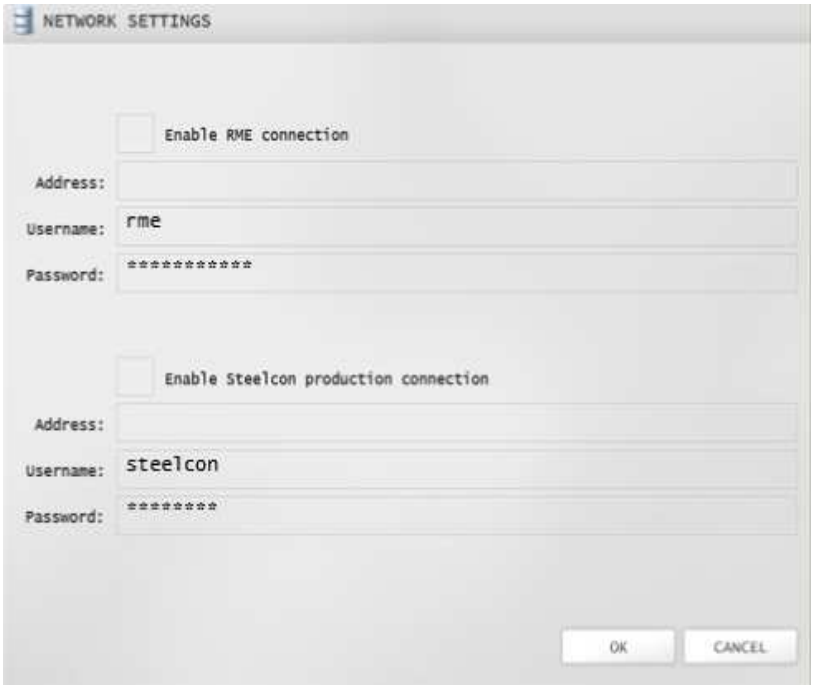
Rollrunnerin kehitystä haittasivat erilaiset yhteensopivuus- ja kohdeympäristöjen tehottomuusongelmat, sillä Rollrunnerin oli pääsääntöisesti toimittava toimistokäyttöön tarkoitetuissa kannettavissa tietokoneissa. Suurin osa yhteensopivuusongelmista liittyi Direct3D tuen toteutukseen eri näytönohjaimilla ja osa niistä pakotti muuttamaan alun perin suunniteltua koodia. Teho-ongelmat tulivat vastaan raskaimpien efektien, kuten stencil-varjojen ja anti-aliasioinnin kanssa, mutta myös CAD-malleista löytyvät tuhannet eri detaljit vaativat optimointeja.

3.2 Tiimityöskentely

Koska työskentelimme Juho Kokkosen kanssa eri ohjelmien parissa, emme virallisesti tehneet yhteistyötä Rollrunnerin DLL-versiota lukuun ottamatta. Kuitenkin koko Steelcon-projektin ajan autoimme toisiamme erilaisissa tehtävissä ja ongelmissa. Juho Kokkonen auttoi muun muassa Rollrunnerin ulkoasun ja toimivuuden testauksen kanssa. Sen lisäksi hän myös suunnitteli ja toteutti Rollrunnerin valikoiden grafiikat. Itse vastaavasti autoin Juhoa satunnaisissa ohjelmointiongelmissa ja 2D -grafiikan piirtämistä vaativissa toiminnoissa. Seuraavissa kuvissa (Kuva 8 ja Kuva 9) näkyy Juho Kokkosen tekemät valikot Rollrunneriin.



Kuva 8. Graphics-asetukset



Kuva 9. Network-asetukset

3.3 Yhteenveto

18.12.2009 saatoin Rollrunnerin viimein ensimmäiseen toimituskuntoiseen versioonsa, josta toimitin myös DLL-version Juho Kokkoselle RME:tä varten. Tämän jälkeen Rollrunner ja muut Steelcon Factoryyn kuuluvat ohjelmistot jäivät odottamaan toimitusta asiakkaalle. Tulevaisuudessa Rollrunner tulee varmasti vaatimaan tukea ja oletettavasti asiakkaat haluavat lisää ominaisuuksia, joten työ Rollrunnerin parissa on tuskin kokonaan ohi.

Työ Rollrunnerin parissa opetti monia asioita Direct3D:stä ja syvensi osaamistani shader-ohjelmoissa. Uskon, että oppimilleni taidoille on hyötyä tulevaisuudessa.

LÄHDELUETTELO

1. D3DXMatrixPerspectiveLH Function, Microsoft MSDN Library, Microsoft [Verkkodokumentti][viitattu 9.10.2010]: http://msdn.microsoft.com/en-us/library/bb205352(v=vs.85).aspx
2. D3DXMatrixLookAtLH Function, Microsoft MSDN Library, Microsoft [Verkkodokumentti] [viitattu 9.10.2010]: http://msdn.microsoft.com/en-us/library/bb205342(v=vs.85).aspx

Insinöörityösuunnitelma

CAD Data Viewer

Jyrki Jauhiainen

Savonia-ammattikorkeakoulu

Tekniikka Kuopio

Tietotekniikan koulutusohjelma

7.4.2010

v 2.0

Versiohistoria

Versio	Muutokset	Päivämäärä
0.1	Dokumentin luominen	10.06.09
0.2	Siirtyminen Open Office:sta Microsoft Office 2007:aan ja siihen liittyvät korjaukset	15.06.09
0.3	Korjauksia ja täydennystä	20.06.09
0.4	Taulukon otsikoiden ja ristiviittausten korjaus	18.4.2010
0.5	Ylätunniste	1.7.2010
1.0	Korjauksia	11.11.2010
1.1	Taulukoiden päivitys	22.12.2010
2.0	Lopullinen versio	7.4.2011

Sisällysluettelo

1 JOHDANTO

1.1 Yleiskuvaus

Tämä on dokumentti insinöörityösuunnitelma CAD Data Viewer nimiselle insinöörityölle, joka tullaan tekemään Rollset Oy:lle. Dokumentin tarkoitus on selvittää alkuperäiset oletukset ja suunnitelmat, joista insinöörityön tekeminen aloitetaan.

1.2 Tuote

Insinöörityössä toteutetaan 3D -sovellus, joka osaa hakea Steelcon-järjestelmän tietokannoista teräsrakenne malleja. Projektin lopputuotteena on 32-bittinen Windows-sovellus. Projektista kirjoitetaan julkaistava raportti.

1.3 Suunnitelman ylläpito

Projektisuunnitelmaa päivitetään myöhemmin, mikäli se nähdään tarpeelliseksi projektin edetessä.

1.4 Määritelmät, termit ja lyhenteet

Alla olevassa taulukossa (Taulukko 1) selvitetään tässä dokumentissa käytettyjä lyhenteitä ja termejä.

Taulukko 1 määritelmät, termit ja lyhenteet

Lyhenne	Selitys
C++	Oliopohjainen ohjelmointikieli.
Shader-kieli	Konekieli, jota käytetään näytönohjaimen GPU:n ohjelmointiin.
Visual C++ Express	Microsoftin C++-sovellusten kehitykseen tarkoitettu ilmainen ohjelmisto.
Direct3D	Microsoftin kehittämä API, jota käytetään näytönohjaimen

	3D-toimintojen kanssa kommunikointiin
HLSL	High Level Shader Language. Direct3D:n oma C-tyylinen shader-ohjelmointikieli.
SMx.x	Shader Model.
AES	Advanced Encryption Standard. Lohkosalausmenetelmä.
StruCad	Teräs- ja puurakenteiden mallinnusohjelmisto
Steelcon	Rollset:n teräsrakenteiden suunnittelu- ja valmistusjärjestelmä.

2. PROJEKTIN ORGANISOINTI

2.1 Projektin vaiheistus

Insinöörityö pyritään suorittamaan niin, että sen laajuus vastaisi 15 opintopistettä. Tämä tarkoittaa noin 400 tuntia työtä projektin parissa.

Työn ohella voidaan tarvittaessa pitää palavereita työstä, mikäli se koetaan tarpeelliseksi. Sovelluksen toteuttamisvaiheessa laaditaan tehdystä työstä lopullinen raportti joka julkaistaan sen valmistuttua. Työ ja sen tulokset esitellään insinöörityöseminaarissa.

2.2 Sidosryhmien kuvaus

Sovellus tehdään Rollset-nimiselle metallialan yritykselle.

Rollset on noin 10 vuotta vanha yritys joka on erikoistunut profilointikoneiden ja rulla-muovauslinjojen valmistamiseen. Yrityksen toimipiste sijaitsee Kuopiossa ja työllistää noin 30 henkilöä. Rollset toimittaa vuosittain 10–20 järjestelmää, pääasiassa vientiin venäjänkielisille alueille.

2.3 Vastuuhenkilöt

Alla olevaan taulukkoon (Taulukko 2) on listattu projektissa toimivat vastuuhenkilöt heidän vastuualueineen.

Taulukko 2 vastuuhenkilöt

Vastuualue	Henkilö	Puhelinnumero
Insinöörityön suorittava opiskelija	Jyrki Jauhiainen	050 523 2498
Työn ohjaava opettaja	Sami Lahti	044 785 6337
Ohjaava vastuuhenkilö	Sami Lahtinen	040 487 9498

3 PROJEKTIN OHJAAMINEN

3.1 Tavoitteet ja priorisointi

Lopputyön tavoite on toteuttaa 3D -sovellus, joka näyttää Steelconin tietokannassa olevan teräsrakennemallin. Sovellus pystyy näyttämään tunnuksen perusteella eri elementtien ja komponenttien tiedot, mm. pituus, leveys, paino, tunnus ja projektin numero. Sovelluksen käytetään mm. elementtien kokousvaiheessa, jossa viivakoodinlukijalla luetaan tunnusnumero ja haetaan sen perusteella kannasta kyseinen komponentti. Muita käyttötarkoituksia on esimerkiksi teräsrakenteiden esittely ja virheiden etsiminen.

Sovelluksen tärkeimmät toiminnot:

- 3D -moottorin perusta
- Shader tuki ainakin tavallisimmille valaistus malleille: Phongin ambient, diffuse ja specular
- 3D -mallejen generointi
- Detailien mallinnus
- SQL -yhteys Steelcon -tietokantoihin (ADODB)
- Salasanan ja käyttäjänimen salaus (AES)

3.2 Oletukset, riippuvuudet ja reunaehdot

Tuotantolinjan toimitusajan ollessa epävarma, ohjelman valmistumiselle ei ole varmaa takarajaa. Ensimmäisen version pitäisi kuitenkin olla valmiina elokuun 2009 loppuun mennessä.

3.3 Riskien hallinta

Seuraavaan taulukkoon (Taulukko 3) on kuvattu projektin läpiviemiseen mahdollisesti vaikuttavia riskejä ja toimia joilla näiltä pyritään välttymään.

Taulukko 3 riskien hallinta

Riski	Todennäköisyys	Vastatoimet	Vakavuus
Projekti ei pysy aikataulussa	Epätodennäköinen	Sovelluksesta pyritään tekemään aluksi yksinkertaisempi versio, johon lisäillään ominaisuuksia tarpeen mukaan.	Vakava
Sovelluksesta tulee liian raskas	Todennäköinen	Sovellusta pyritään testaamaan koko kehityskaaren ajan eri kokoonpanoilla	Vakava
Sovellus vaatii liian uutta tekniikkaa PC:ltä	Epätodennäköinen	Sovellus pyritään kehittämään pääsääntöisesti SM2.0b:lla	Vakava

3.4 Seuranta ja ohjaus

Projektin etenemistä seurataan tarvittaessa projekti palavereissa. Ohjausta haetaan tarvittaessa ohjaavalta opettajalta.

4 TEKNIIKKA

4.1 Menetelmät ja työkalut

Sovelluksen toteuttamiseen käytetään Visual C++ Express-kehitysympäristöä ja C++ -ohjelmointikieltä.

3D -moottorin kehitykseen käytetään Microsoftin Direct3D 9 -rajapintaa, joka takaa sovelluksen yhteensopivuuden eri näytönohjainten kanssa.

Shader -ohjelmointiin käytetään HLSL:llää, Direct3D:n omaa Shader -ohjelmointikieltä

Tietokantayhteyteen käytetään Windows -käyttöjärjestelmän mukana tulevaa MSADO -kirjastoa.

Projektin dokumentit, insinöörityön lopullinen raportti ja seminaarissa käytettävä materiaali tullaan toteuttamaan Microsoft Office 2007 -ohjelmistolla.

4.2. Dokumentointi

Projektista tehdään seuraavat dokumentit:

- Projektisuunnitelma
- Sovelluksen toiminnallinen määrittely
- Sovelluksen tekninen määrittely
- Varsinainen insinöörityö

Dokumentteja päivitetään projektin edetessä.

4.3. Laadunvarmistus

Hyvä laatu pyritään varmistamaan huolellisella suunnittelulla ja selkeillä ja yksinkertaisilla ohjelmointimenetelmillä. Ohjelman toimivuutta pyritään testaamaan projektin edetessä useilla eri PC-kokoonpanoilla.

5 VAIHEET, AIKATAULUT JA BUDJETTI

5.1 Budjetti

Alla olevaan taulukkoon (Taulukko 4) on kuvattu arvioitu budjetti jolla projekti suoritetaan.

Taulukko 4 budjetti

Kulu	€
Palkka	3 kk * 3000 = 9000
Kokouskustannukset	200
Muut	200
Yhteensä	9400

5.2 Aikataulu

Seuraavassa taulukossa (Taulukko 5) on kuvattu aikataulu jolla projekti pyritään suorittamaan.

Taulukko 5 aikataulu

Toiminnon kuvaus	Alkaa	Valmis
Aloituspalaveri	13.06.09	13.06.09
Määrittely / Suunnittelu	13.06.09	~~22.06.09~~ ~~01.04.10~~ 18.04.2010
Sovelluksen toteuttaminen testausvalmiuteen	22.06.09	22.08.09

Sovelluksen viimeisteleminen / testaus	22.08.09	31.08.09
Insinöörityöseminaari	~~kevät 2010~~ syksy 2010	~~kevät 2010~~ syksy 2010
Työn kirjallinen osuus	01.03.10	~~01.06.10~~ ~~22.11.10~~ ~~15.2.2011~~ ~~12.3.2011~~ 7.4.2011