Esa Pukero

3D-KATSELUOHJELMAN SUUNNITTELU JA TOTEUTUS VIVA3-HANKKEELLE

Opinnäytetyö

Tietojenkäsittely

Marraskuu 2010

MIKKELIN AMMATTIKORKEAKOULU

Mikkeli University of Applied Sciences

KUVAILULEHTI

MIKKELIN AMMATTIKORKEAKOULU
Mikkeli University of Applied Sciences

Opinnäytetyön päivämäärä
3.12.2010

Tekijä(t)

Esa Pukero

Koulutusohjelma ja suuntautuminen
Tietojenkäsittely

Nimeke
3D-Katseluohjelman suunnittelu ja toteutus VIVA3-hankkeelle.

Tiivistelmä
Työn tarkoituksena on suunnitella ja toteuttaa 3D-mallien katseluohjelma Mikkelin ammattikorkeakoulun VIVA3-hankkeelle Microsoftin XNA Frameworkin avulla . Lisäksi tarkoituksena on selvittää kuinka Microsoftin XNA Frameworkin avulla voidaan ladata 3D-malleja jo käynnissä olevaan sovellukseen. Idea tai tarve 3-ulotteisten mallien lataamisesta jo käynnissä olevaan sovellukseen ei itseasiassa ole uusi, vaan sitä on etenkin tietokonepelien tapauksessa tehty jo n.15 vuotta. Tämä on kuitenkin yleensä ollut ohjelmoijan näkökulmasta paljon aikaa ja työvoimaa vievä projekti, joka on kaikenlisäksi vaatinut valtavat määrät ammattitaitoa.

VIVA3-hanke on Mikkelin ammattikorkeakoulun hallinnoima hanke, jonka tavoitteena on selvittää mm. erilaisten kohteiden 3D-mallinnukseen liittyviä tekniikoita ja menetelmiä. Lisäksi hankkeen tavoitteena on kehittää erilaisten kohteiden mallinnusprosesseja.

Tämä raportti koostuu 5 osasta. 1. osa on johdanto, jossa selitetään yleisesti työhön liittyvistä asioista. 2. osa koostuu tärkeimpien työvälineiden kuvaamisesta. Tässä tapauksessa Visual Studio ja XNA. 3. osa kertoo 3D-mallien rakenteesta ja 4. osa kertoo itse sovelluksen toiminnasta. 5. osa on yhteenveto kaikista edellä mainituista asioista

Asiasanat (avainsanat)
3D, XNA, Visual Studio, C#

Sivumäärä	Kieli	URN
27	Suomi	URN:NBN:fi:mamk-opinn20102476

Huomautus (huomautukset liitteistä)

Ohjaavan opettajan nimi
Jukka Selin

Opinnäytetyön toimeksiantaja
VIVA3-Hanke

DESCRIPTION

MIKKELIN AMMATTIKORKEAKOULU
Mikkeli University of Applied Sciences

Date of the bachelor's thesis
3. December 2010

Author(s)
Esa Pukero

Degree programme and option
Business Information Technology

Name of the bachelor's thesis
Designing and creating 3D-viewer for VIVA3 project

Abstract
The goal of this bachelor's thesis was to design and create a program for viewing 3D-modelsVIVA3 project using Microsoft XNA Framework. Also one goal is to find out how to upload models into already running software, using XNA Framework. Idea or need for loading 3D-models to already running software is not in fact new but for example in computer games, it has been used for about 15 years. However, from programmers point of view, this kind of a project has demanded a lot of time and manpower and also required a great amount of expertise.

VIVA3 project is governed by Mikkeli University of Applied Sciences, that's purpose is to find out for example different techniques and methods for modeling 3D-models. Also the purpose of this project is to develop modeling processes for different kinds of models.

This report consists of 5 parts. First part is the introduction, that explains about matters related to this thesis in general. Second consists of descriptions about the most important tools. In this case Visual Studio and XNA. Third part tells about the parts of 3D-models and fourth part tells about the software itself. Fifth part is a summary of all the things mentioned earlier.

Subject headings, (keywords)
3D, XNA, Visual Studio, C#

Pages	**Language**	**URN**
27	Finnish	URN:NBN:fi:mamk-opinn20102476

Remarks, notes on appendices

Tutor	**Bachelor's thesis assigned by**
Jukka Selin	VIVA3 Project

SISÄLTÖ

1. JOHDANTO 1

2. VÄLINEET JA MENETELMÄT 2

2.1 Microsoft Visual Studio 2008 4

2.1.1 Projektinhallinta 5

2.1.2 Virheen Etsintä 6

2.1.3 Ohjelmointiympäristö 7

2.2 Microsoft XNA 8

2.2.1 Content Pipeline 9

2.2.2 GraphicsDevice-luokka ja Z-puskurointi 10

3. 3D MALLIT 11

3.1 Verteksit ja kaaret 12

3.2 Monikulmiot (Polygons): 12

3.3 Verkko (Mesh) 14

4. SOVELLUS 15

4.1 Vaatimusmäärittely 16

4.2 Käyttötapaukset 18

4.2.1 3D-mallin katselu 18

4.2.2 Tiedostopaketin teko 18

4.3 C# ja XML 19

4.4 C# ja äänet 21

4.5 C# ja mallien lataaminen 22

4.6 C# ja taustan lataaminen sekä käsittely 23

5. YHTEENVETO 25

LÄHTEET 26

1. JOHDANTO

Opinnäytetyön toimeksiantaja on Mikkelin ammattikorkeakoulun hallinnoima VIVA3-hanke. VIVA3:n tarkoituksena on selvittää eri tapoja mallintaa 3D-esineitä ja sen jälkeen soveltaa niitä käytännössä

Tämä opinnäytetyön tarkoituksena on rakentaa sovellus, joka mahdollistaa 3d-mallien lataamisen sovellukseen reaaliajassa. Ohjelmointi-kielenä käytetään C# ja pääasiallisena työvälineenä Microsoftin Visual Studio 2008:aa. Myös Adoben Photoshop CS3 ja Adobe Illustrator CS3 on jonkin verran käytetty, mutta niiden käyttö on ollut niin vähäistä että niitä ei erikseen esitellä Välineet ja Menetelmät-osiossa.

Opinnäytetyön päätutkimusongelma on 3d-mallien lataaminen ohjelmaan silloin kun ohjelma on jo käynnissä. C# käyttävä Microsoftin XNA Frameworkin avulla on helposti mahdollista ladata malleja ohjelmaan ennen sen käynnistämistä(Visual Studion oman käyttöliittymän kautta) mutta tällä kertaa on ollut tarkoitus saada selville miten mallit voidaan ladata suoraan koodissa. Kun mallit voidaan ladata suoraan koodissa, voidaan paljon helpommin vaikuttaa siihen mikä malli ladataan ja ennen kaikkea milloin se ladataan.

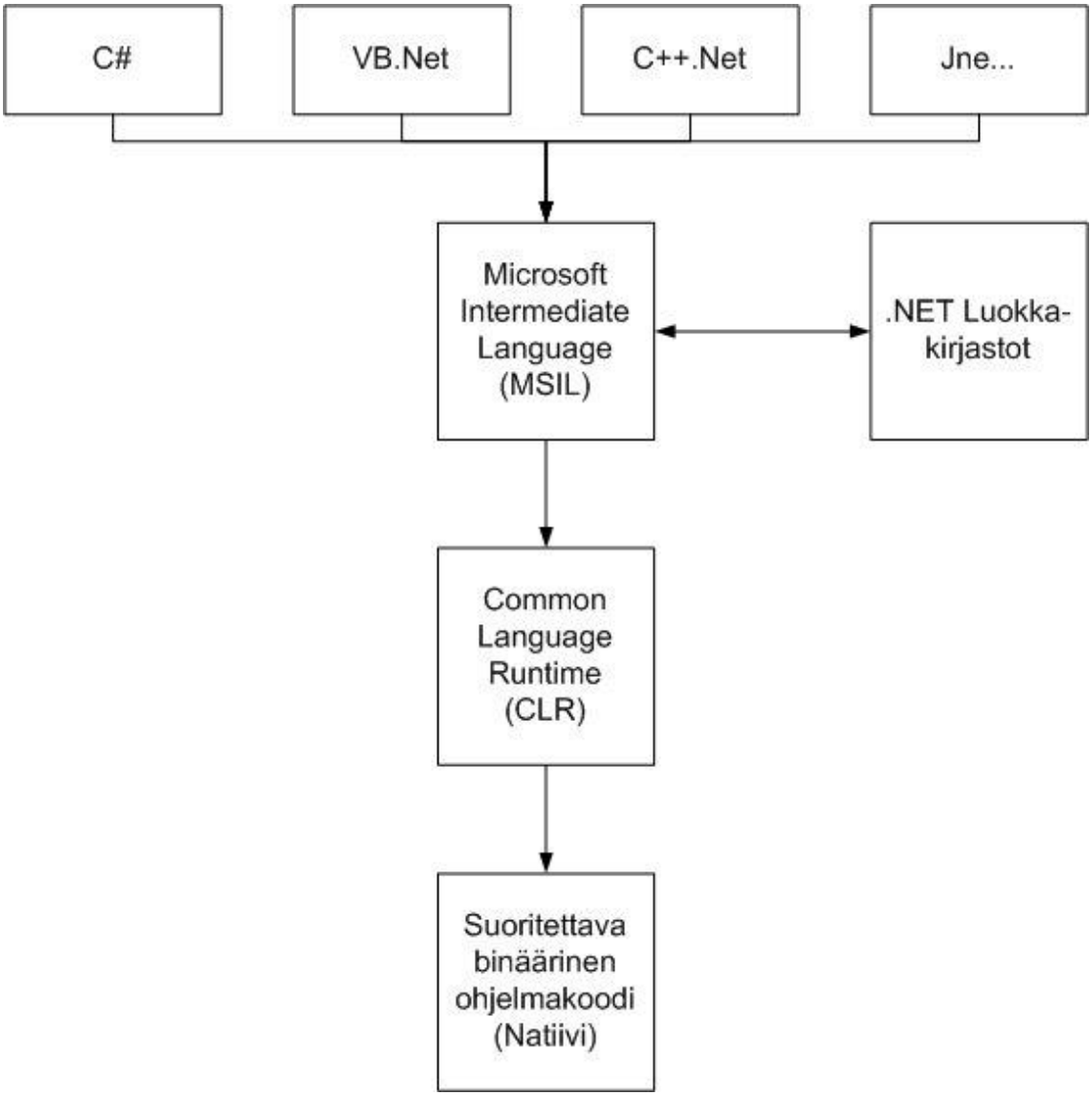
Toinen tutkimusongelma joka ilmeni työn edetessä, oli äänien lataaminen mukaan sovellukseen. Tarkoituksena oli että 3d-mallin taustalla soisi käyttäjän kyseiseen malliin liittämä äänitiedosto. Tämä ei kuitenkaan alkuun onnistunut niin helposti kuin kuviteltiin mutta lopulta vastaus oli melko yksinkertainen.

2. VÄLINEET JA MENETELMÄT

C# on oliopohjainen ohjelmointikieli, joka pohjautuu C++-kieleen (Moghadampour 2009, 14). C# on pyritty kehittämään niin, että sen käyttö olisi yksinkertaisempaa kuin C++:n. Koska C# on oliopohjainen, sen tärkeimmän osan muodostavat eri oliot. Täysin ilman olioiden käyttöä ei itse asiassa edes voi tehdä C#-sovelluksia. Toinen mainittava C#:n osa ovat muuttujat(*variables)*, jotka on aina esiteltävä(*declare*) ennen niiden käyttöä. Tämä tarkoittaa sitä että muuttujaan ei voi suoraan sijoittaa arvoa, kertomatta ensin minkä tyyppisen arvon kyseisen muuttujan oletetaan saavan. Arvon tyyppi voi olla joko jonkin tyyppinen numero (Esim. kokonaisluku tai liukuluku), merkki, merkkijono tai totuusarvo (*boolean value*)

C#-sovellus sisältää aina vähintään niin sanotun ”Main-olion” joka aina ajetaan automaattisesti sovelluksen käynnistyessä. Main-olion lisäksi tarvitsee määritellä käytettävä nimiavaruus ja ottaa käyttöön *using*-lauseella System-oliokirjasto. System kirjasto sisältää sovelluksen käynnistymisen ja muunkin toiminnan kannalta välttämättömiä toimintoja. C#-sovelluksen ei tarvitse koostua yhdestä tiedostosta, vaan käytettävä koodi voidaan ohjelmoijan tarpeiden mukaan jakaa osiin.

Koska C# on suunniteltu toimimaan .Net-sovelluskehyksessä, käännetään C#-sovellukset aina ensin niin sanotuksi yleiskieleksi. Tästä yleiskielestä, käyttöjärjestelmäkohtaiset tulkki-ohjelmat kääntävät C#-sovelluksen sitten lopulliseen konekieliseen muotoonsa(Moghadampour 2009, 15). Tällä kääntö-tyylillä pyritään lisäämän eri .Net-sovellusten (Siis esim C#-sovellusten) yhteensopivuutta. C#-tapauksessa lisähelpotusta tuo mahdollisuus käyttää XML:lää tiedonsiirtovälineenä. Tämä ominaisuus on yksi tärkeä tekijä tämän opinnäytetyön onnistumisessa.

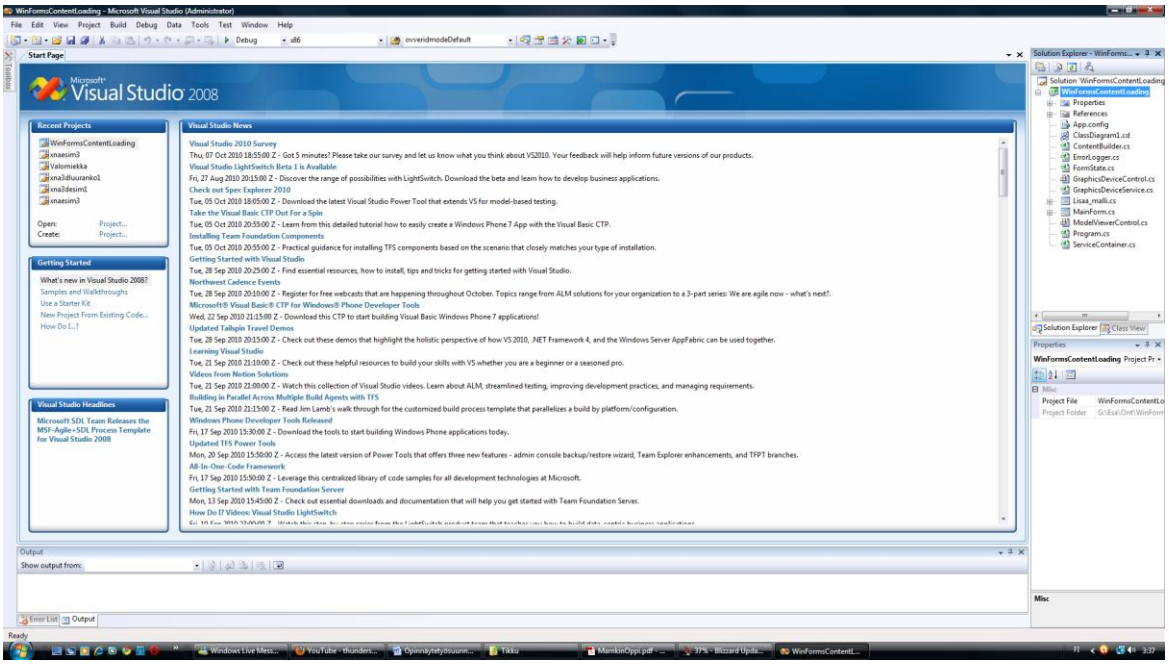


KUVA 1. .Net Kehitysympäristön tapa muuttaa ohjelmat valmiiseen muotoon.

2.1 Microsoft Visual Studio 2008

Tämän sovelluksen kehittämiseen on käytetty Microsoft Visual Studio 2008:aa. Visual Studio on ns. *Rapid Application Development*-Kehitysympäristö. Käytännössä tämä tarkoittaa että Visual Studio tarjoaa laajan kirjon graafisia työkaluja, joiden avulla käyttäjä voi kehittää sovellustaan. Näiden graafisten sovellusten avulla käyttäjä voi helpommin hallita sovelluksensa koodin eri osia, sekä liikutella sovellukseen liittyviä tiedostoja. Visual Studio tukee oletuksena useita eri ohjelmointikieliä. Joista tärkeimmät on listattu alla

- Visual Basic
- C++
- C#



KUVA 2. Visual Studio 2008 alkuruutu.

Visual Studio on ns. IDE-ohjelma (Integrated Development Enviroment). Tämä tarkoittaa sitä että ohjelma sisältää niin ohjelmointiin, projektinhallintaan kuin virheen etsintään (debugging) tarkoitetut työkalut(Moghadampour 2009, 15).

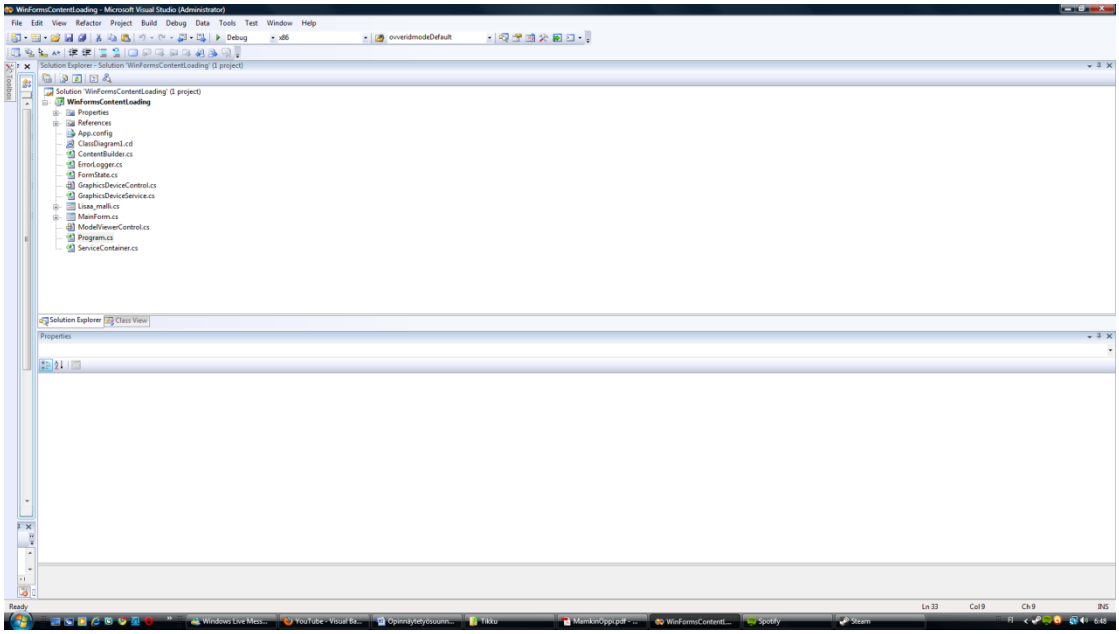
Visual Studio itsessään ei tue yhtään ohjelmointikieltä vaan kaikki ohjelmointikielet liitetään siihen ns. plug-inien kautta. Tämä tarkoittaa sitä että Visual Studio voidaan saada tukemaan mitä tahansa ohjelmointi kieltä.

2.1.1 Projektinhallinta

Visual Studio tarjoaa mahdollisuuden hallita käyttäjän tekemiä sovelluksia kokonaisina projekteina. Käytännössä tämä tarkoittaa sitä että Visual Studio osaa liittää kaikki sovellukseen liittyvät tiedostot yhteen ja että käyttäjä voi halutessaan tarkastella näitä tiedostoja yhdeltä ruudulta. Näin ollen käyttäjän ei tarvitse hyppiä Windowsin resurssien hallinnan ja Visual Studion välillä läheskään niin paljon jos Visual Studio sisältäisi pelkän koodin käsittelyyn liittyvän osion

Myös tiedostojen lisääminen projektiin onnistuu Visual Studion kautta. Tämä tarkoittaa sekä kokonaan uusien tiedostojen luomista että jo olemassa olevien tiedostojen liittämistä mukaan projektiin.

Projektinhallinta siirtää kaikki tiedostot samaan Visual Studion luomaan kansioon. Kun sovellus halutaan siirtää tai kopioida muualle, tarvitsee käyttäjän vai kopioida/siirtää kyseinen kansio haluttuun paikkaan eikä hänen tarvitse metsästää projektiin liittyviä tiedostoja monesta eri paikkaa.



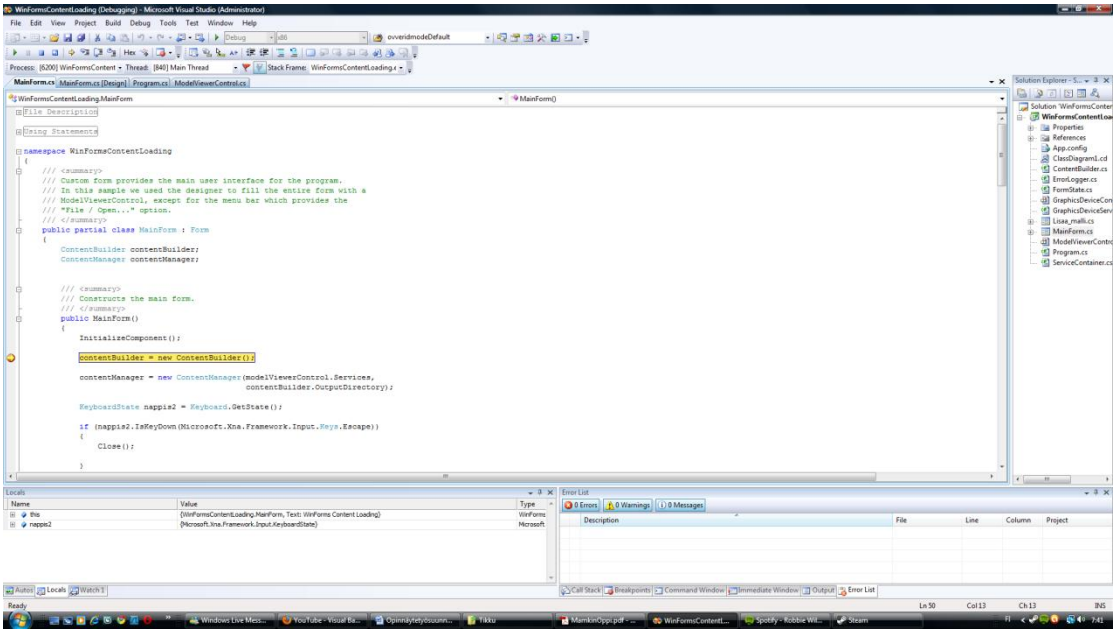
KUVA 3. Visual Studion projektin hallintaruutu.

2.1.2 Virheen Etsintä

Mikäli käyttäjä törmää virheeseen, jonka alkuperää hän ei voi selvittää, on mahdollista käyttää Visual Studion mukana tulevaa virheen etsintä työkalua (*Debugger*).

Tietyssä mielessä debugger on aina päällä ja tarkkailee jo koodia kirjoittaessa että komennot on kirjoitettu kieliopillisesti oikein. Kun ohjelmaa käynnistetään, debugger tarkkailee että sovelluksen eri muuttujien arvot ovat oikeita(Moghadampour 2009, 21). Mikäli jokin virhe tapahtuu, debugger keskeyttää ohjelman ajamisen ja ilmoittaa millä koodirivillä virhe on tapahtunut. Lisäksi debugger ilmoittaa minkä tyyppisestä virheestä on kysymys.

Debugger on kuitenkin mahdollista kytkeä sovellukseen niin että sovellus ajetaan rivi kerrallaan ja debugger tarkkailee kaikkia sovelluksen arvoja ja ilmoittaa ne käyttäjälle.



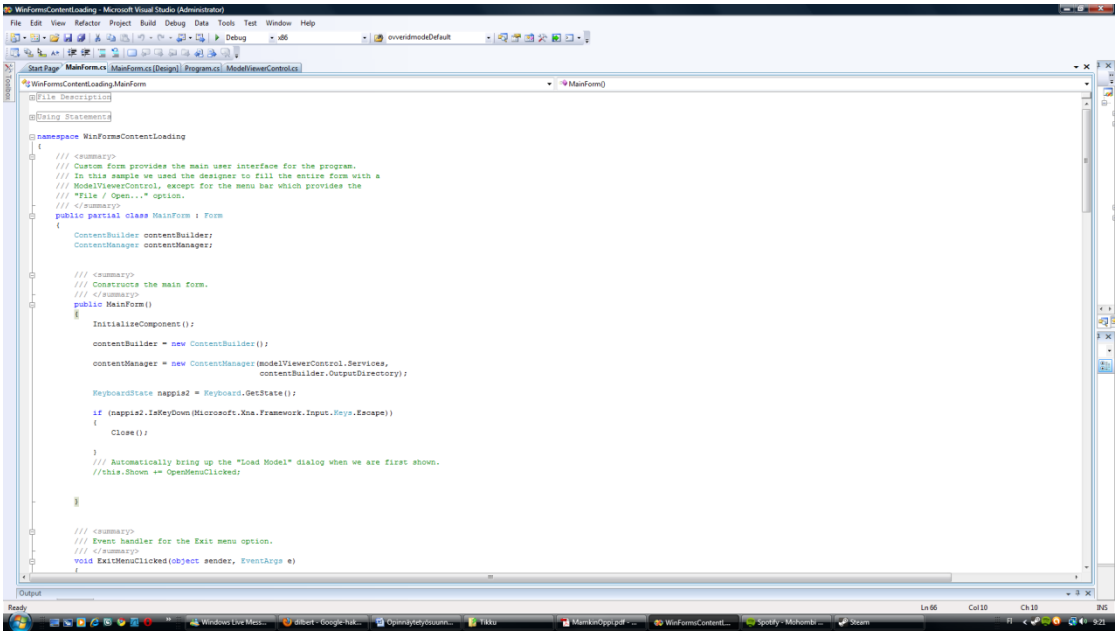
KUVA 4. Visual Studion virheen etsintä työkalu.

2.1.3 Ohjelmointiympäristö

Visual Studion päätyökalu on luonnollisesti sen tarjoama ohjelmointiympäristö. Kuten yllä on mainittu, Visual Studio voidaan plug-inien avulla saada ymmärtämään mitä tahansa ohjelmointikieltä. Tämä kuitenkin vaatii että kyseiselle ohjelmointikielelle on rakennettu tarvittava plug-in.

Käyttöliittymä auttaa käyttäjää koodinsa hallitsemissa esim. värjäämällä koodin eri osat eri väreillä ja merkkaamalla missä sulkujen ”toinen pää” sijaitsee. Lisäksi käyttöliittymä pyrkii rivittämään tekstiä niin että se olisi helpommin luettavaa.

Edellä mainittujen ominaisuuksien lisäksi ohjelmointiympäristö osaa tarjota käyttäjälle listan komennoista. Riippuu käytettävästä ohjelmointikielestä, mitä komentoja ohjelma tarjoaa.



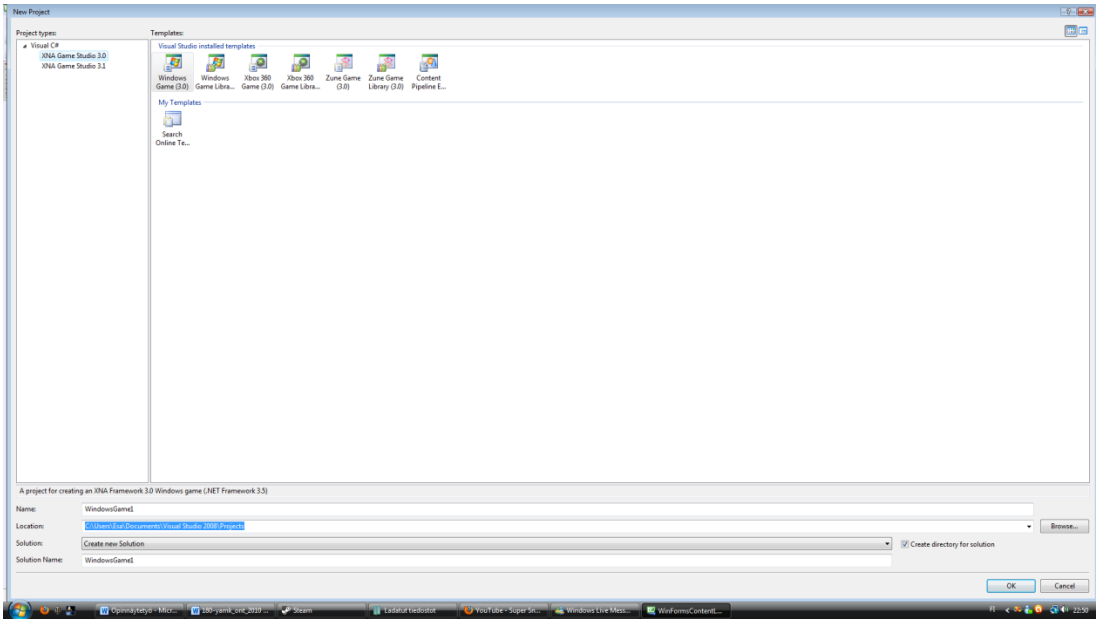
KUVA 5. Visual Studion tapa merkitä eri koodin osia.

2.2 Microsoft XNA

XNA on sanaleikki englanninkielen lauseesta ”It’s Not Acronym” (Evangelista, Lobão, Antonio Leal de Farias 2008. Beginning XNA 2.0 Game Programming). XNA on Microsoftin tarjoama pelinkehitykseen tarkoitettu sovellus joka toimii .NET ympäristössä. XNA kytketään mukaan Visual Studioon ja pitkälti näiden käyttöliittymä on samanlainen.

XNA tarjoaa käyttäjälle useita oliokirjastoja, jotka ovat avuksi pelien ohjelmoinnissa. Näihin oliokirjastoihin sisältyvät mm. mahdollisuus käsitellä tietokoneen eri ohjainlaitteilta (esim. hiiri ja näppäimistö) tulevaa dataa eri tavoilla.

XNA:lla tehtyjä sovelluksia on mahdollista käyttää Windows PC:n lisäksi Xbox 360:llä ja uusimpana lisänä Windows Phonella. Koska nämä 3 ovat eri laitteita, ne asettavat myös omat vaatimuksensa ja kiemuransa XNA:n käytölle. Huomattavan ongelman muodostaa tiedostojen käsittely koska ainakin PC ja Xbox 360 eivät suoraan ymmärrä toistensa tiedostoja. Esim. Xbox 360 ei tunne .jpeg tiedostoja. Tästä johtuen XNA:lla tehdyt projektit kääntävät niihin kuuluvat tiedostot (siis ei pelkkiä kooditiedostoja) oikeaan välimuotoon, jotta kaikki XNA:ta tukevat laitteet voisivat käyttää niitä. Tämän kääntämisen hoitaa Content Pipelinena tunnetu XNA:n osa.

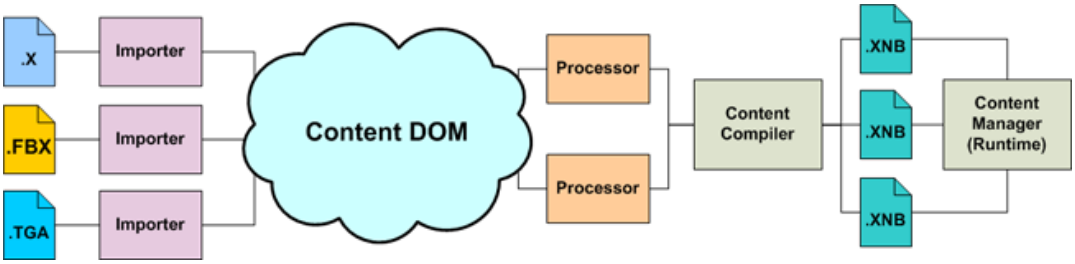


KUVA 6. Visual Studion XNA-projektin luonti

2.2.1 Content Pipeline

Content Pipeline on XNA:n osa. Sen tehtävänä on käsitellä käyttäjän sille antamat tiedostot niin että niitä voidaan käyttää joko Windows PC:llä, Xbox 360:ll tai Windows Phonella. Tiedostot saadan myös nopeasti käyttöön, koska ne yleensä myös ladataan Conten Pipelinen kautta reilusti ennen niiden käyttöä..

Content Pipeline jakaantuu kahteen osaan. Ajon aikaiseen osaan ja Rakennuksen aikaiseen osaan(Klucher 2009). Ajon aikainen osa huolehtii lähinnä tiedostojen käytöstä ajon aikana. Esimerkiksi Ajon aikainen osa huolehtii tiedoston elinkaaren tarkkailusta. Tämä mahdollistaa sen että ohjelma lakkaa pitämästä siihen ladattuja tiedostoja valmiina ja näin ollen vapauttaa tietokoneen resursseja.



KUVA 7. Content Pipelinen tapa tuoda tiedostoja XNA:han (Klucher 2009)

2.2.2 GraphicsDevice-luokka ja Z-puskurointi

GraphicsDevice-luokka on yksi tärkeimmistä XNA:n luokkakirjastoista. Sen luokkia käyttämällä on mahdollista aiempaa helpommin näyttää ja käsitellä ruudulla niin 2-ulotteisia(Esim. JPG-kuvat) kuin 3-ulotteisia esineitä. (3D Pipeline Basics 2010).

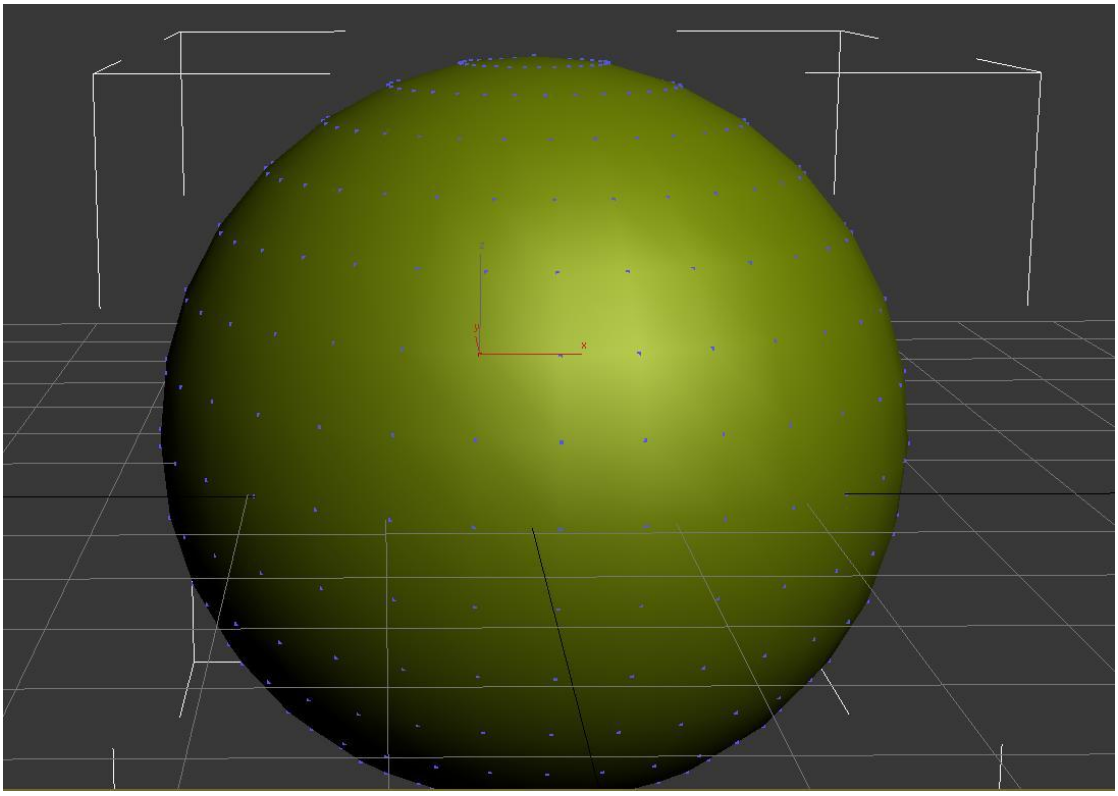
3-ulotteisista esineiden tapauksessa GraphicsDevice-luokka mahdollistaa myös ns. Z-puskurin käyttämisen suhteellisen helposti. Normaalisti XNA osaa käyttää Z-puskurointia, mikäli käytetään vain 3d-malleja. Mutta jos sovellus käyttää sekaisin 2d- ja 3d-malleja, XNA:n asetukset menevät sekaisin ja Z-puskuri täytyy laittaa päälle käsin.

Alla on kirjoitettu koodi mikä tarvitaan Z-puskurin päälle laittamiseen XNA 3.1:ssä ja XNA 4.0:ssa.

```
XNA 3.1:
spriteBatch.Begin(SpriteBlendMode.AlphaBlend,
SpriteSortMode.Immediate, SaveStateMode.SaveState);

XNA 4.0:
GraphicsDevice.BlendState = BlendState.Opaque;
GraphicsDevice.DepthStencilState = DepthStencilState.Default;
```

3. 3D MALLIT

3d mallit ovat jo pitkään olleet olennainen osa tietokoneiden maailmaa. Ensimmäiset tietokoneella luodut 3d mallit tehtiin 1970-luvulla(Carlson). Tämän jälkeen 3d mallit ja etenkin niiden animointi on noussut elokuvatuotannossa ja tietokonepelialalla tärkeään osaan. Perusideana on että 3d mallien rakentuminen perustuu graafiteoriaan. Tämä tarkoittaa sitä että 3d-mallit koostuvat pisteistä, jotka on yhdistetty toisiinsa viivoilla. Näiden viivojen avulla saadan pinnan ääriviivat, joiden välit täyttämällä voidaan saada aikaan erilaisia 3 uloitteisia esineitä



KUVA 8. Esimerkki 3d-mallista. Siniset pisteet ovat mallin verteksejä

3.1 Verteksit ja kaaret

Verteksit käyttäjän määrittämä piste 3-ulotteisessa avaruudessa. Niiden paikka ilmoitetaan 3 numerolla (Esim ”10,9,8”). Nämä numerot ilmoittavat kyseisen verteksin sijainnnin X-akselilla, Y-akselilla ja Z-akselilla.(Giambruno 2003)

Pisteet ovat lopullisessa mallissa näkymättömiä, mutta niiden avulla muodostetaan lopulta 3-ulotteisia pintoja, joista kaikki 3d mallit koostuvat

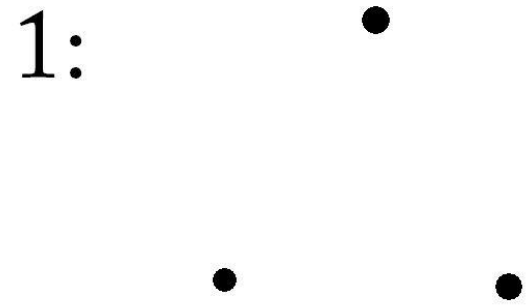
Kaaret (Edges):
Kaaret ovat kahden verteksin (eli pisteen) välin yhdistämisestä syntyviä janoja. Koska myös kaaret sijaitsevat 3-uloitteisessa avaruudessa, niiden suunta on määriteltävä samoin 3 ulottuvuudella.

3.2 Monikulmiot (Polygons):

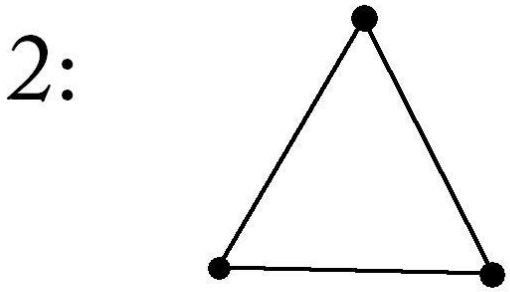
Yhdistämällä verteksiä kaarilla niin että syntyy suljettu alue, saadan aikaan tietyn määrän kulmia sisältävä suljettu alue. Tämä alueen kulmien maksimi määrää ei ole rajoitettu, mutta minimi määrä kulmille on vähintään 3. Tämä siksi että pienemmällä kulmamäärällä ei ole mahdollista saavuttaa tarvittua ”suljettua tilaa”. Näin syntyneitä alueita, eli siis monikulmoita yhteen liittämällä voidaan alkaa rakentaa 3d-mallia.

On kuitenkin olemassa jopa vain yhdestä verteksistä koostuvia ”polygoneja” joista käytetään nimitystä partikkeli.(T.Niemi)

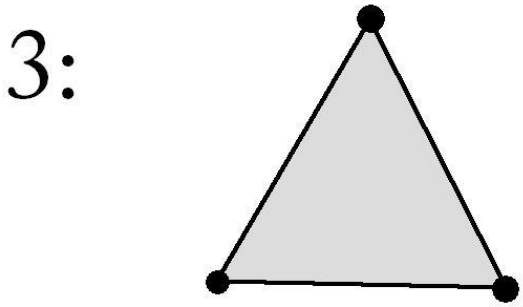
Yleensä tietokoneiden maailmassa käytetään joka 4- tai 3-kulmioisia monikulmioita (Eli nelikulmoita tai kolmioita). Molemmissa tapauksissa käytettävien kulmien määrä on siis hyvin vähäinen, ottaen huomioon että kulmia voisi lisätä melkein äärettömästi. Kuitenkin on parempi muodostaa mallit edellä mainituista 4- tai 3-kulmioista, koska tällöin niiden matemaattinen käsittely ja muuntelu on tietokoneelle kaikkein helpointa ja nopeinta.(Ilmola 2004) Näin mallit voidaan lopulta myös piirtää ruudulle mahdollisimman nopeasti eikä mallien muodostaminen saa aikaa pullonkaulaa tietokoneen työskentelyssä.



KUVA 9. Kuva, jossa näkyvät pelkät verteksit



KUVA 10. Kuva, jossa näkyvät verteksit sekä niitä yhdistävät kaaret

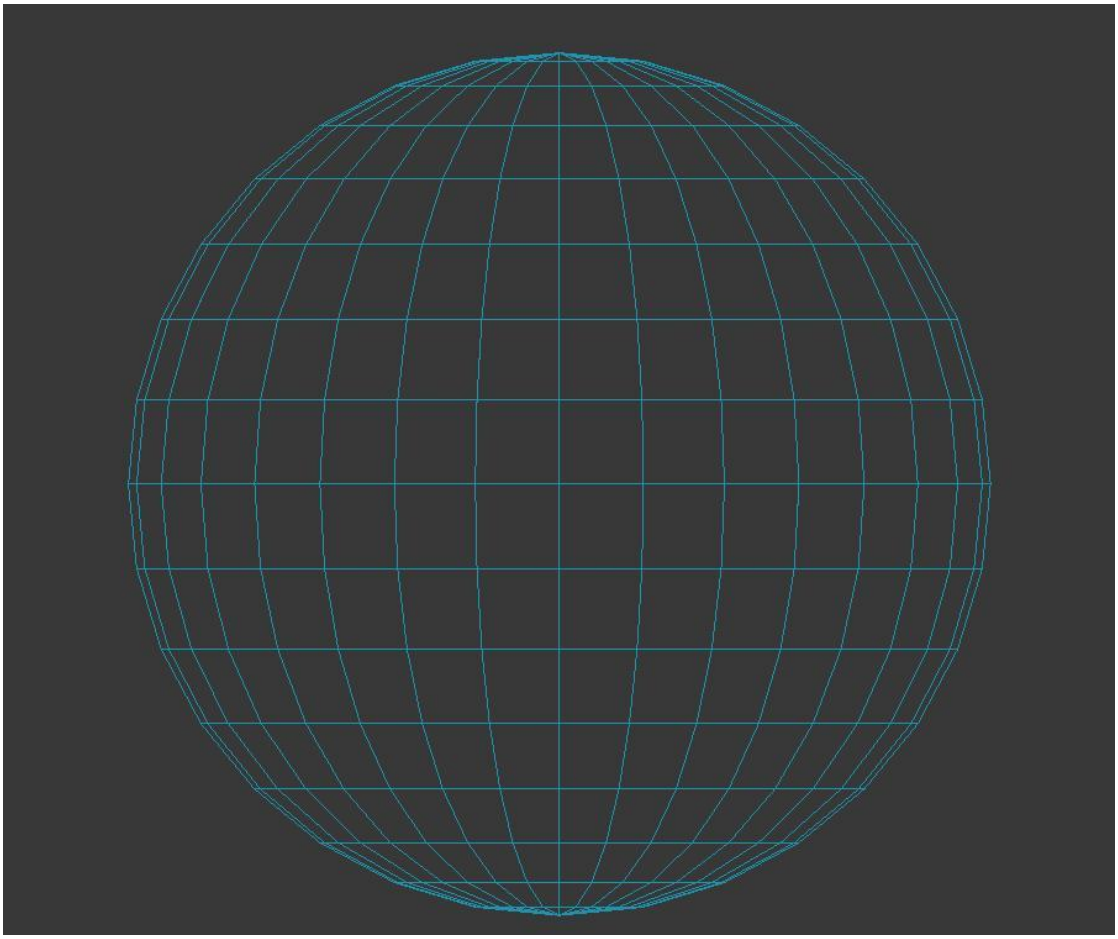


KUVA 11. Kuva valmiista 3-kulmaisesta monikulmiosta, jossa kaarien välinen tila on suljettu

3.3 Verkko (Mesh)

Kun edellä mainittuja monikulmioita/polygoneja aletaan liittää yhteen eri kulmissa, saadaan aikaan erinäköisiä 3-ulotteisia esineitä. Tässä vaiheessa tekeillä oleva 3-ulotteinen esine näyttää siltä kuin se olisi tehty metalliverkosta(Luultavasti tästä johtuu myös nimi).

Useimmissa tapauksissa yksi 3d malli koostuu vain yhdestä verkosta mutta koska on mahdollista esim. yhdistää 2 kokonaan erillään tehtyä mallia on olemassa myös malleja, jotka koostuvat useammasta kuin yhdestä verkosta.



KUVA 12. Pallon muotoinen verkko, jossa näkyy kuinka 4-kulmaisia monikulmioita yhdistelemällä on saatu aikaan 3-ulotteinen esine

4. SOVELLUS

Sovellus käyttää sekaisin tavallisia C#-luokka kirjastoja sekä XNA:n luokka kirjastoja. Ilman niitä, sovelluksen tekeminen olisi ollut mahdotonta tai vienyt suunnattoman määrän aikaa. Esim. ilman XNA:n luokka kirjastoja, mallin näyttäminen tai liikuttaminen ruudulla olisi ollut mahdotonta. Taasen ilman C#:n normaaleja luokka kirjastoja, äänen toistaminen ei olisi onnistunut halutulla tavalla

Sovellus käyttää myös tärkeänä osana toimintaansa XML-merkkikieltä. XML:n avulla sovellus osaa pitää kirjaa sitä mitkä mallit, taustat ja äänet liittyvät yhteen. Näin ollen sovellus osaa aina näyttää oikealle mallille oikean taustan sekä osaa soittaa oikean taustaäänen

Kaikki sovelluksen tiedostot, mukaan lukien käyttäjän sinne mallin-lataus työkalulla tuomat tiedostot viedään omiin kansioihinsa. Sovellus löytää ne oman sijaintinsa perusteella, joten niin kauan kuin koko sovellus siirretään kyseisten tiedostojen mukana, ei ongelmia tiedostopolkujen kanssa synny.

KUVA 13. Sovelluksen aloitusruutu

4.1 Vaatimusmäärittely

Sovelluksen perimmäisenä tarkoituksena on kyetä näyttämään 3d-malleja. Lisäksi on haluttu että mallin näyttämisen ohella, sovellus kykenisi näyttämään taustalla tavallisen 2-ulotteisen kuvan ja toistamaan ääniraitoja.

Luonnollisesti vaatimuksiin kuluu myös että sovelluksen käyttäjä itse kykenisi valitsemaan minkä mallin hän haluaa näyttää sekä minkä taustakuvan ja ääniraidan hän haluaa liittää mukaan mallin näyttämiseen. Tarkoituksena on, että valittu malli, taustakuva ja ääni myös tallentuisivat tavalla tai toisella ohjelman muistiin, jotta niitä ei tarvitsisi aina etsiä uudelleen. Tähän tarkoitukseen luodun työkalun olisi myös suotavaa kyetä siirtämään/kopioimaan valitut tiedostot samaan kansioon, jotta niiden liikkuttelu esim. muistitikulta toiselle olisi mahdollisimman vaivatonta.

Sovellus on tarkoitettu mm. messukäyttöön, joten sen täytyy kyetä näyttämään malli ja siihen liittyvä kuva ns. koko ruutu-tilassa, niin että mitään sovelluksen hallintapainikkeita ei näy ruudulla. Myös mallin liikuttelu näppäimistön avulla on haluttu ominaisuus, jotta mallia voidaan tarkastella eri kulmista.

Alla on yksinkertaistettu lista halutuista ominaisuuksista.

Vaatimuksen numero	Vaatimus
1.	3D-mallin näyttäminen
2.	3D-mallin kääntely näppäimistöllä
3.	Taustakuvan näyttäminen
4.	Ääniraidan toistaminen
5.	Mahdollisuus valita, mikä kuva ja ääni liitetään mihinkin malliin.
6.	Käyttäjän valitsemien tiedostojen (Malli,kuva,ääni) siirtäminen samaan kansioon.
7.	Käyttäjän valitsemien tiedostojen nimien ja paikan tallentaminen niin että ne voidaan ladata sovellukseen mukaan ilman tarvetta etsiä joka tiedostoa uudestaan.
8.	Mallien näyttäminen koko ruutu-tilassa

TAULULUKKO 1. Sovelluksen vaatimusmäärittely

4.2 Käyttötapaukset

Sovelluksessa on 2 käyttötapausta. 1. käyttötapaus on itseasiassa koko sovelluksen ydin eli mahdollisuus katsella 3D-malleja. 2. käyttötapaus on mahdollisuus koota eri tiedostoja ns. tiedostopaketteihin. 1 tiedostopakettiin kuuluu aina itse mallin tiedosto, taustakuvan tiedosto ja ääni-tiedosto.

4.2.1 3D-mallin katselu

Nimi: 3D-mallin katselu
Toimija: Käyttäjä
Alkutilanne: Sovellus on käynnistetty ja on alkuruudussa
Kuvaus:
3D-mallin katselu onnistuu klikkaamalla vasemman ylälaidan File-valikosta Open-nappia. Tämän jälkeen aukeaa valikko josta käyttäjä voi valita haluamansa XML-tiedoston, joka sisältää tiedot malliin liitetyistä tiedostoista. Tätä tiedostokokoelmaa kutsutaan tiedostopaketiksi.

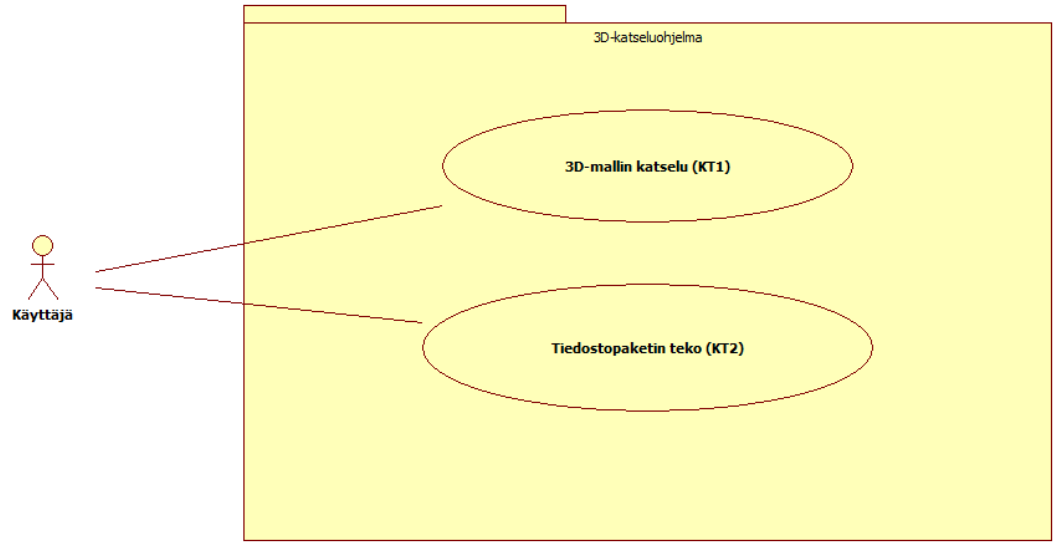
Kun haluttu tiedostopaketti on valittu, sovellus näyttää halutun mallin. Käyttäjä voi pyöritellä tätä mallia näppäimistön avulla. Sovellus näyttää myös malliin liitetyn taustan mutta tämä tausta pysyy paikoillaan.

4.2.2 Tiedostopaketin teko

Nimi: Tiedostopaketin teko
Toimija: Käyttäjä
Alkutilanne: Sovellus on käynnistetty ja on alkuruudussa
Kuvaus:
Käyttäjän on mahdollista määritellä valmiiksi tiedostopaketteja, joiden tiedoilla sovellus tietää esim. minkä taustan ladata tiettyyn malliin. Tiedosto paketin teko onnistuu File-valikon New-napista. Tämän jälkeen käyttäjälle avautuu ruutu, josta hän voi ladata haluamansa tiedostot.

3D-mallien tapauksessa sovellus ymmärtää vain .X-tiedostoja. Kun kaikki tiedostot on valittu ja tallenna-nappia on painettu, sovellus luo XML-tiedoston johon ladattujen

tiedostojen tiedot on tallennettu. Lisäksi sovellus kopioi ladatut tiedostot erilliseen kansioon, jotta niiden löytäminen myöhemmin onnistuisi helposti.



KUVA 14. Sovelluksen käyttötapauskaavio

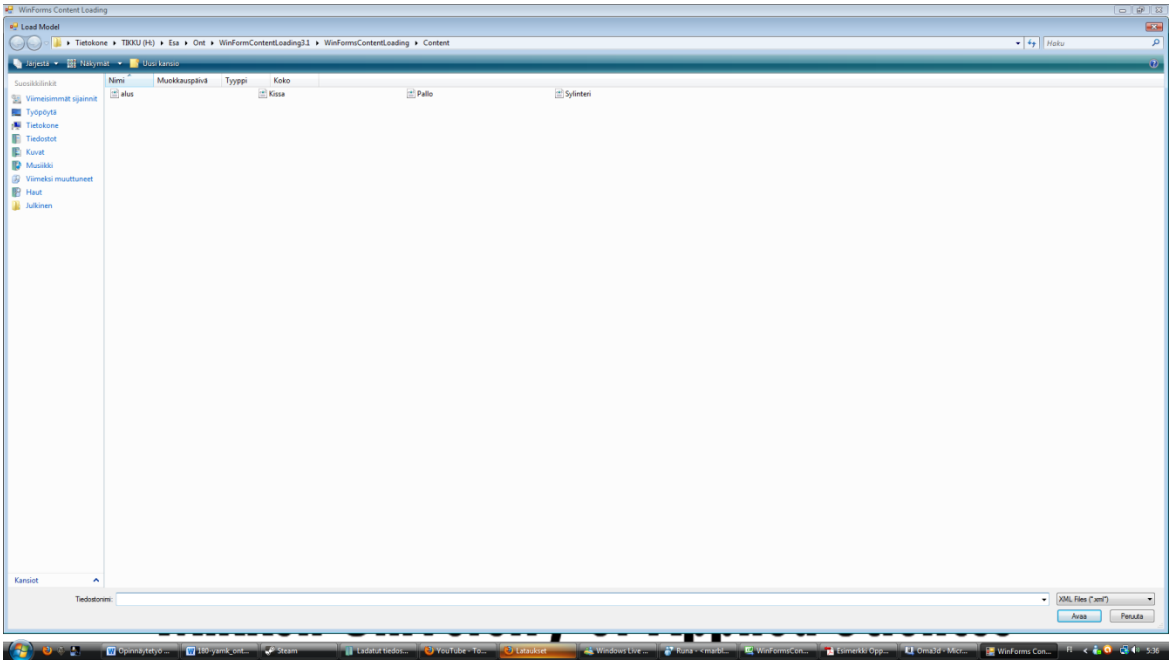
4.3 C# ja XML

XML:n käyttämiseen C# vaaditaan System.XML-kirjasto. Ilman tätä kirjastoa, sovellus ei osaa käyttää XML-tiedostoja. Kun yllä mainittu kirjasto on käytössä, XML-tiedoston avaaminen ja käsitteleminen onnistuu kertomalla sovellukselle mikä tiedosto on avattava doc.Load-komennolla/metodilla. Alla on aiheesta esimerkki

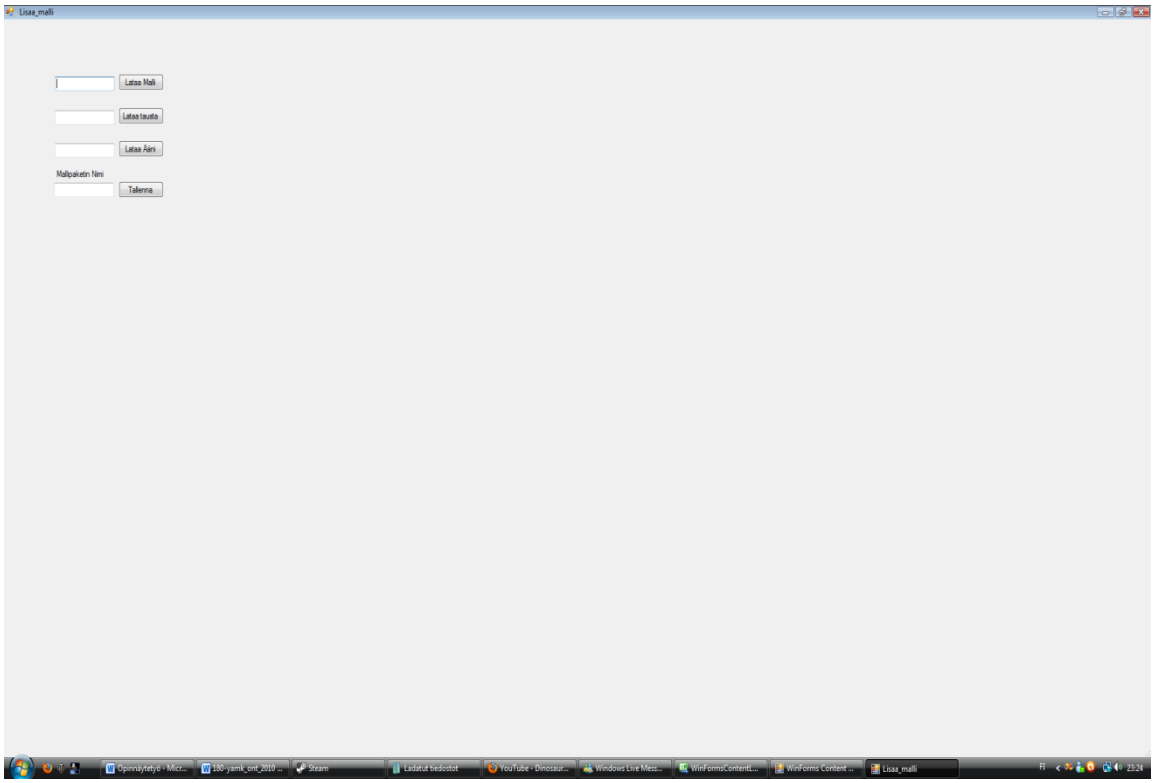
```
   XmlDocument doc = new XmlDocument();
                doc.Load(fileDialog.FileName);

                XmlElement juuri = doc.DocumentElement;
                XmlNodeList malli = juuri.ChildNodes;
                XML_malli_nimi = Cwd + "\\" +
malli.Item(0).ChildNodes.Item(0).Value;
                XML_malli_tausta      =      Cwd      +      "\\"      +
malli.Item(1).ChildNodes.Item(0).Value;
                XML_malli_aani=       Cwd       +       "\\"       +
malli.Item(2).ChildNodes.Item(0).Value;
```

6:lla viimeisellä rivillä kaivetaan esiin mallista halutut tiedot. Nämä tiedot ovat, ladattavan mallin nimi, ladattavan mallin takana näkyvä tausta ja ladattavan mallin mukana soiva ääni.

Tämä sovellus käyttää kaikkeen tarvitsemaansa tiedon tallennukseen XML:ää, eikä esim. tietokantoja käytetä ollenkaan. Tämä tekee sovelluksesta helpomman ja yksinkertaisemman käyttää kun erillistä tietokanta-ohjelmaa ei tarvita. Lisäksi sovelluksen koko pienenee jonkin verran.



KUVA 15. XML-tiedoston valinta ruutu. Sovellus tallentaa käyttäjän asettamat tiedostopaketin tiedot XML-tiedostoon. Tämä tiedoston aukaisemalla, sovellus osaa hakea oikeat tiedostot ja näyttää ne



KUVA 16. Tiedostopaketin luontiruutu. Tiedostopakettiin kuuluvien tiedostojen tiedot tallennetaan XML-tiedostoon.

4.4 C# ja äänet

Äänien toistaminen XNA:n omilla työkaluilla ei onnistunut. Tämä siksi että XNA olisi halunnut kyseisen tiedoston esikäsittelyn XACT-ohjelmalla. Koska XACT on vielä tätä kirjoittaessa hieman päälle liimatun oloinen, ei se suostunut koodin kautta käsittelemään sille annettuja äänitiedostoja.

C# omalla Soundplayer-luokalla oli kuitenkin mahdollista ladata ja toistaa se ääni, joka ladattuun malliin liittyvässä XML-tiedostossa oli määritelty.

Tämä tapahtui luomalla metodi ToistaAani, joka saa parametrina aani-muuttujan. Tämä muuttuja on aina kyseiseen malliin liittyvän ääni-tiedoston polku, joka löytyy sovelluksen itsensä tekemästä XML-tiedostosta

```
void ToistaAani(string aani)
        {
            SoundPlayer musiikki = new SoundPlayer(aani);
            musiikki.PlayLooping();
        }
```

musiikki.Playlooping() kutsuu PlayLooping()-metodia, joka toistaa ääntä ”loputtomiin” eli tässä tapauksessa niin kauan kuin malli on näkyvillä.

Koska äänet toistetaan käyttäen C# omia metodeja, eikä XNA:n metodeja, tarkoittaa se sitä että tätä sovellusta ei voi käyttää Xbox 360:lla.

4.5 C# ja mallien lataaminen

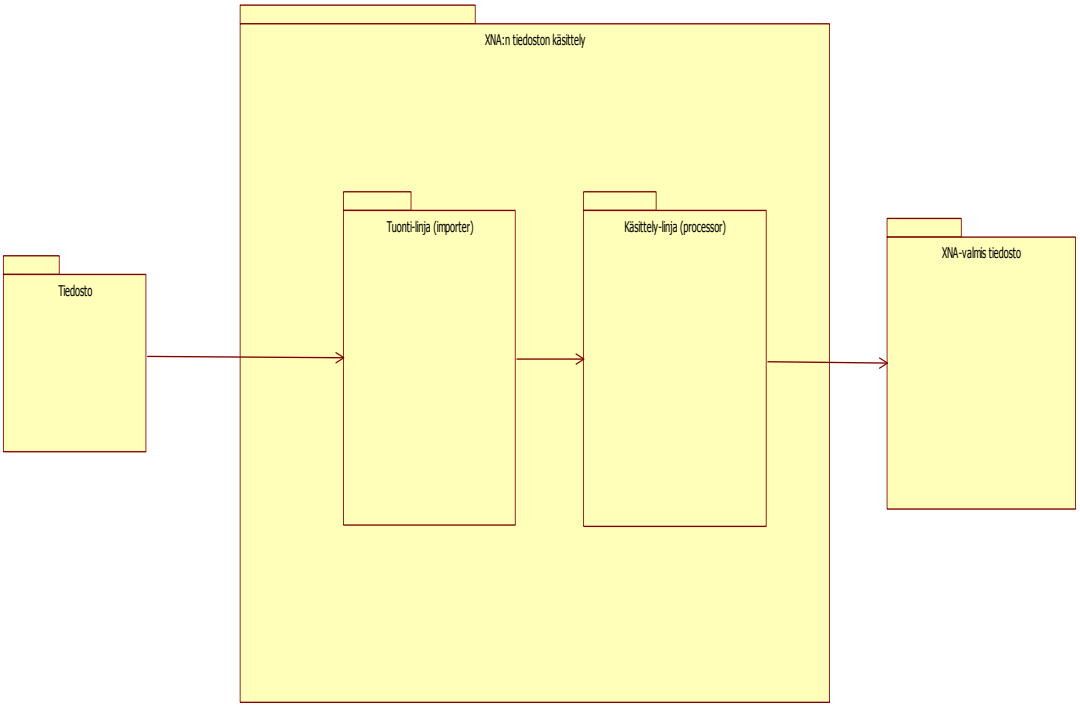
Mallien lataaminen lennosta onnistui XNA:lla. Lataamisella lennosta tarkoitan tässä tapauksessa sitä että ohjelma ei vielä käynnistyessään tiedä mitä malleja se tulee käsittelemään. Normaali käytäntö XNA:llahan olisi ollut että kaikki mallit ladattaisiin ja valmisteltaisiin ohjelmaan etukäteen. XNA:sta löytyy kuitenkin mahdollisuus tehdä tämä lataaminen ja valmistelu koodissa itsessään, eikä vain MS Visual Studion kautta. Yksinkertaisesti selitettynä tämä tarkoitti että haluttu malli/tiedosto viedään ensin ns. tuonti-linjalle (importer), jonka jälkeen se viedään käsittely-linjalle (processor). Nämä kaksi osaa poistavat tiedostoista tietoja joita tämän tyyppiset multimedia-sovellukset eivät tarvitse sekä muuttavat niitä niin että niiden olisi mahdollista toimia Xbox 360:ssa. Kun ylimääräisiä tietoja on poistettu, tiedostojen käsittely sovelluksessa nopeutuu. Tämä puolestaan johtaa sovelluksen yleisen nopeuden kasvuun.

Alla näkyvät ne neljä riviä, joiden avulla oikeastaan mikä tahansa tiedosto, ääni-tiedostoja lukuun ottamatta voidaan ladata dynaamisesti mukaan XNA:han.

```
            contentBuilder.Add(taustaNimi,"Tausta",
"TextureImporter", "TextureProcessor");
            contentBuilder.Add(fileName,"Model",null,
"ModelProcessor");
```

Contentbuilder-luokasta kutsutaan Add-metodia. Add-metodi saa parametreinä käsiteltävän tiedoston sijainnin, sille annettavan nimen, sen tuovan linjan nimen ja sen käsittelevän linjan nimen.

Mikäli tuontilinjan tai käsittelylinjan tyyppiä ei anneta, yrittää C# automaattisesti päätellä tiedoston tyypin. Huomasin kuitenkin että varmin keino on kertoa koodissa suoraan minkä tyyppistä tuonti- ja käsittelylinjaa tarvitaan.



KUVA 17. Kaavio XNA:n tavasta käsitellä tuotavia tiedostoja

4.6 C# ja taustan lataaminen sekä käsittely

Mallin taustan tuomista on jo käsitelty edellä olevassa kappaleessa. Lyhyesti sanottuna tämä tapahtui kuitenkin samalla tavalla kuin 3d-mallien lataaminen. Ainoa muutos oli että tuontilinjaksi ja käsittelylinjaksi piti muuttaa tekstuureja (eikä 3d-malleja) käsittelevät linjat.

KUVA 18. Sovellus käynnissä ja näyttämässä ladattua taustaa ja mallia.

5. YHTEENVETO

3D-mallien lataaminen lennosta sovellukseen onnistui. Riippuen mallin koosta, kestää aina muutaman sekunnin ladata malli ruudulle. Tänä aikana tietokone ei vastaa mutta erillistä latausruutua tuskin tarvitaan, koska latausaika on niin lyhyt.

Pääpirteissään sovelluksen teko onnistui hyvin. Mallien ja niihin liittyvien oheistiedostojen (taustakuva ja ääni) lataaminen toimii. Mikä tärkeintä, mallit voidaan ladata sovelluksen näytettäväksi vielä silloinkin kun sovellus on jo käynnistetty. Näin ollen voi sanoa että opinnäytetyön pääongelma on ratkaistu.

Toinen pääongelma oli äänentoisto XNA:ssa silloin kun sovellus ei etukäteen tiedä mitä äänitiedostoa sen pitäisi toistaa. Kuten edempänä on mainittu, XNA:n tapa toistaa ääniä olisi vaatinut niiden käsittelemistä ennakkoon aivan erillisessä ohjelmassa. Tämä tapa ei olisi siis toiminut silloin kun sovellus ei etukäteen tiedä mitä äänitiedostoa se tulee toistamaan. Kuitenkin vastaus ongelmaan oli sekoittaa XNA:han mukaan aivan tavallisia C#:n mukana tulevia luokkakirjastoja, joiden avulla äänentoisto hoidettiin käyttämättä ollenkaan XNA:ta.

Ongelmia muodostui mallin ns. Z-puskuroinnin kanssa. Lyhyesti sanottuna tämä tarkoitti sitä että vaikka sovellus teknisesti piirsi mallin ruudulle, niin käytännössä tämä kyseinen malli alkoi välkkyä oudosti heti kun sitä liikutettiin. Toinen ongelma muodostui hieman yllättäen lukuisten tiedostopolkujen määrittämisestä. Vaikka tästä ongelmasta pääsinkin eroon yksinkertaisesti olemalla huolellinen niin silti väärin ajateltu tai kirjotettu tiedostopolku koodissa saattoi aiheuttaa vaikeasti ymmärrettäviä virheitä sovelluksen muissa osissa. Tällöin oli vaikeaa jäljittää virhe juuri väärän tiedostopolkuun

LÄHTEET

Lobão Alexandre, Evangelista Bruno, Antonio Leal de Farias José 2008.
Beginning XNA 2.0 Game Programming.

Moghadampour Ghodrat 2009.
C#-ohjelmointi.

Cawood Stephen, McGee Pat 2009.
Microsofot XNA Game Studio, Creators Guide. Second Edition.

Niemi. Peruskäsitteitä. WWW-dokumentti.
http://cc.joensuu.fi/~tniemi/3d/2.html. Luettu 13.11.2010.

Giambruno Mark 2003. 3D Modeling Basics. WWW-dokumentti.
http://www.peachpit.com/articles/article.aspx?p=30594. Luettu 13.11.2010.

Ilmola Markus 2004. OpenGL:n perusteet - Osa 2: 3D grafiikka. PDF-dokumentti.
http://personal.inet.fi/koti/markus.ilmola/OpenGLosa2.pdf. Luettu 13.11.2010.

Carlson Wayne. A Critical History of Computer Graphics and Animation. WWW-dokumentti. http://design.osu.edu/carlson/history/lesson2.html. Luettu 13.11.2010.

Klucher Michael 2009. The XNA Framework Content Pipeline. WWW-dokumentti.
http://blogs.msdn.com/b/xna/archive/2006/08/29/730168.aspx. Luettu 13.11.2010.

Processing the XML File. WWW-dokumentti.
http://msdn.microsoft.com/fi-fi/library/fsbx0t7x.aspx. Luettu 13.11.2010.

2009. Introduction to Microsoft Visual Studio. WWW-dokumentti.
http://www.csharpkey.com/csharp/Lesson01.htm. Luettu 13.11.2010.

SoundPlayer Class. 2010. WWW-dokumentti.
http://msdn.microsoft.com/en-us/library/system.media.soundplayer.aspx.
Luettu 13.11.2010.

3D Pipeline Basics. 2010. WWW-dokumentti.
http://msdn.microsoft.com/en-us/library/bb194916.aspx. Luettu 13.11.2010.

Ditchburn Keith. WWW-dokumentti.
http://www.toymaker.info/Games/XNA/html/xna_simple_triangle.html.
Luettu 13.11.2010.