OULUN SEUDUN
AMMATTIKORKEAKOULU

Jani Aho

3D MODEL VIEWER -MOBIILISOVELLUS WINDOWS PHONE 7:LLE

3D MODEL VIEWER -MOBIILISOVELLUS WINDOWSN PHONE 7:LLE

Jani Aho
Opinnäytetyö
Kevät 2012
Tietotekniikan koulutusohjelma
Oulun seudun ammattikorkeakoulu

TIIVISTELMÄ

Oulun seudun ammattikorkeakoulu
Tietotekniikka, ohjelmistotuotanto

Tekijä(t): Jani Aho
Opinnäytetyön nimi: 3D Model Viewer -mobiilisovellus Windows Phone 7:lle
Työn ohjaaja(t): Markku Rahikainen
Työn valmistumislukukausi ja -vuosi: Kevät 2012 Sivumäärä: 28 + 10 liitettä

Tässä opinnäytetyössä oli tavoitteena kehittää täysin uudenlainen mobiilisovellus Mixed Reality -hankkeelle. Sovelluksen vaatimuksina oli saada ohjelma, joka käyttäjän GPS-tietojen perusteella hakee tietokannasta listan 3D-malleista ja näyttää käyttäjän valitseman mallin puhelimen näytöllä.

Sovellus toteutettiin Microsoftin Visual Studiolla ja ohjelmointikielenä oli C#. Windows Phone 7 -alustan frameworkeista käytettiin sekä XNA:ta että Silverlightia. Tietokanta toteutettiin MySQL:llä ja yhteys tietokantaan PHP:n avulla.

Lopputuloksena saatiin minimivaatimukset täyttävä täysin uudenlainen mobiilisovellus.

Asiasanat: Window Phone 7, GPS, ohjelmointi, 3D, R&D

ABSTRACT

Oulu University of Applied Sciences
Information technology, software development

Author(s): Jani Aho
Title of thesis: 3D Model Viewer -mobiilisovellus Windows Phone 7:lle
Supervisor(s): Markku Rahikainen
Term and year when the thesis was submitted: Spring 2012 Pages: 28 + 10 appendices

---

The goal in this thesis was to develop totally new mobile application for Mixed Reality -project. The requirements for this application were that the application gets list of available 3D-models from a server based on users GPS-information and allows user to pick a model on that list and show it on the mobile phones screen.

The application was made with Microsoft Visual Studio 2010 and used programming language was C#. Both Windows Phone 7 frameworks, Silverlight and XNA, were used on this thesis. Database was done with MySQL and connection between mobile phone and database with PHP.

As a result we got a new mobile phone application that meets the set minimum requirements.

Keywords: Windows Phone 7, GPS, programming, R&D

ALKULAUSE

Äiti, minä valmistun!

Oulu 6.5.2010

Jani Aho

SISÄLLYS

1 JOHDANTO

Tämä opinnäytetyö on jatkoa viimeiselle harjoitteluprojektille, jossa aiheena oli tutkia Windows Phone 7 -alustan soveltuvuutta 3D-sovelluskehitykseen. Harjoitteluprojektin lopputulos mahdollisti projektin jatkamisen opinnäytetyöksi. Opinnäytetyön tilaaja on Oulun seudun ammattikorkeakoulun hanke, Mixed Reality.

Työn tavoitteena oli suunnitella ja toteuttaa mobiilisovellus, joka käyttäjän paikkatietoja hyödyntäen hakee tietokannasta paikkatietoja vastaavan listan 3D-malleista, lataa käyttäjän valitseman mallin ja näyttää sen puhelimen näytöllä. Alustana toimii Windows Phone 7 ja käytettävät frameworkit ovat Silverlight ja XNA. Ohjelmointikielenä käytetään C#:a ja tietokanta toteutetaan MySQL:llä.

Opinnäytetyössä käydään läpi Windows Phone 7 -alustan pääpiirteet sekä arkkitehtuuri. Lisäksi käydään läpi GPS:n perustoimintaperiaatteet. Toteutuksessa pyritään selittämään ohjelman toiminta pääpiirteittäin sekä käytetyt komponentit hieman tarkemmin. Lisäksi tarkastellaan projektin aikana ilmenneitä ongelmia sekä virhetilanteita ja niiden käsittelyä.

2 KÄYTETYT TEKNOLOGIAT

Tämän opinnäytetyön kohdeympäristönä toimii Microsoftin Windows Phone 7 -käyttöjärjestelmällä varustetut mobiililaitteet.

2.1 Windows Phone 7 -käyttöjärjestelmä

Windows Phone 7 on Microsoftin matkapuhelimille kehittämä käyttöjärjestelmä. Käyttöjärjestelmä otettiin käyttöön uusissa matkapuhelimissa loppuvuodesta 2010. Windows Phone 7 -laitteilta edellytetään seuraavat vaatimukset:

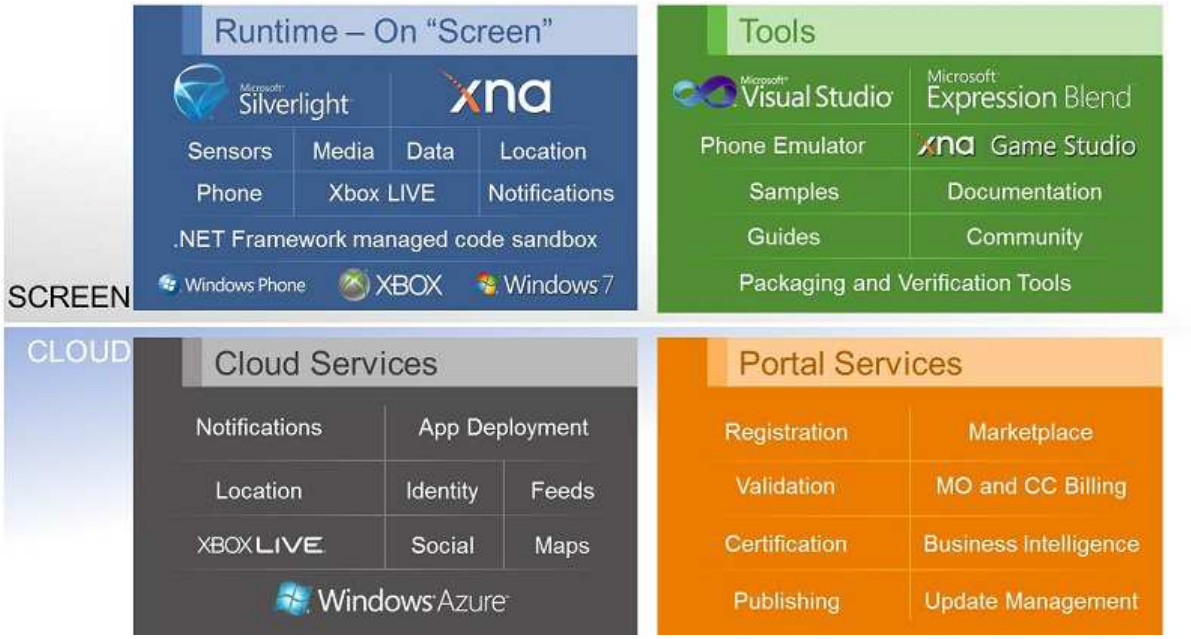
- kapasitiivinen, multitouch, vähintään 800 x 480 resoluution näyttö
- 1 GHz:n ARM-prosessori
- DirectX 9 -yhteensopiva grafiikkapiiri
- 256 MB keskusmuistia ja 8 GB flash-muistia
- sisäänrakennettu kiihtyvyysanturi, kompassi, valaistuksen- ja etäisyyden-tunnistus ja A-GPS-vastaanotin
- 5 Mpx:n kamera LED-salamavalolla
- FM-radiovastaanotin
- 6 fyysistä näppäintä: takaisin, käynnistä, haku, kamera, virta/lepotila ja äänenvoimakkuus. (1, s. 1.)

Tällä hetkellä Windows Phone 7 -käyttöjärjestelmällä varustettuja puhelimia valmistavat HTC, LG, Samsung ja Nokia.

Windows Phone 7 tarjoaa sovelluskehitykseen kaksi eri frameworkia. Nämä ovat Silverlight ja XNA. Silverlight on tarkoitettu web-tyylisten sovellusten kehitykseen ja XNA puolestaan pelien ja grafiikoiltaan raskaampien sovellusten kehitykseen. 3D-sovelluskehitykseen näistä XNA on parempi vaihtoehto. Windows Phone 7 -alustan tuetut ohjelmointikielet ovat tällä hetkellä C# ja Visual Basic.

2.2 Windows Phone 7 -arkkitehtuuri

Windows Phone 7:n arkkitehtuuri (kuva 1) koostuu neljästä pääkomponentista. Nämä komponentit ovat Runtime – On ”Screen”, Tools, Cloud Services ja Portal Services.



KUVA 1. Arkkitehtuurikaavio (2)

2.2.1 Runtime – On ”Screen”

Runtime – On ”Screen” -komponentti yhdistää Silverlightin, XNA:n, Windows Phone 7- ja yleisen Base-luokan kirjastot. Silverlightilla tai XNA:lla tehdyt sovellukset saadaan yleensä toimimaan Windows Phone 7 -alustassa ilman suurempia muutoksia. Yleisin tarvittava muutos on näytön koon määrittäminen ja XNA-ohjelman tapauksessa myös frameraten eli ruudun päivitysnopeuden muuttaminen. (2.)

2.2.2 Tools

Microsoftilta löytyy ohjelmistokehittäjille valmis ilmainen paketti, joka sisältää kaikki tarvittavat työkalut ja komponentit Windows Phone 7 -sovelluskehitykseen. Sovelluskehityksessä käytetyt ohjelmistot ja komponentit ovat

- Visual Studio 2010
- Expression Blend
- XNA Game Studio
- Windows Phone Emulator.

Lisäksi Microsoftilla on kattava valikoima malliesimerkkejä ja dokumentteja Windows Phone 7 -sovelluskehitykseen. (2.)

2.2.3 Cloud Services

Cloud Services eli pilvipalvelut tuovat sovelluskehitykseen monia mahdollisuuksia tehdä sovelluksesta entistä monipuolisempi ja tehokkaampi.

Microsoftin tarjoamat pilvipalvelut ovat seuraavat:

- Notifications
- Location Cloud Services
- Identity, Feed, Social & Maps Services
- Azure.

Notifications mahdollistaa dynaamisten push-viestien käytön sovelluksissa. Location Cloud Services toimii yhdessä Location API:n kanssa. Palvelu hyödyntää Wi-Fi-, GPS- ja radiosignaalin tietoja käyttäjän paikantamiseen. Identity, Feeds, Social & Maps Services tarjoaa yhteydet sosiaaliseen mediaan ja mahdollistaa kartan käytön navigointiin. Azure-alusta avaa kehittäjälle mahdollisuuden käyttää ohjelmissaan laajoja pilvipalveluita, jotka sijaitsevat Microsoftin tietokeskuksissa. (2.)

2.2.4 Portal Services

Portal Servicesin kautta käyttäjät voivat jakaa sovelluksiaan Windows Phone Marketissa. Windows Phone Marketissa kehittäjät voivat jakaa sovelluksiaan helposti ja vaivattomasti ympäri maailmaa. Windows Phone Market vaatii kehittäjältä rekisteröitymisen ja identiteetin tunnistuksen, jonka jälkeen kehittäjä voi lähettää sovelluksiaan sertifioitaviksi ja julkaista ne marketissa. Sovelluksille voidaan määrittää hinta ja maat, joissa sovellus on ladattavissa. (2.)

2.2.5 XNA Game Studio

XNA on Microsoftin kehittämä peliohjelmointiin tarkoitettu oliopohjainen framework, joka tunnetaan nimellä XNA Game Studio. XNA-framework koostuu .NET-frameworkiin perustuvista peliohjelmointiin tarkoitetuista kirjastoista. Viimeisin versio XNA Game Studiosta on 4.0 ja siitä on saatavilla kaikille ilmainen Express-versio. XNA Game Studio tarvitsee erikseen IDE:n, jotta sitä voidaan käyttää. Tähän tarkoitukseen Microsoft tarjoaa omaa tuotettaan, Visual Studio 2010, josta on myös saatavilla ilmainen Express-versio. XNA Game Studiolla voidaan tehdä pelejä useille alustoille. Nämä alustat ovat XBOX 360, PC ja Windows Phone 7. (3.)

XNA-frameworkissa on sovelluskehitykseen kaksi eri 3D-formaattia. XNA on rakennettu DirectX:n päälle, joten luonnollisesti DirectX:n oma 3D-formaatti .x on tuettuna. Toinen tuettu formaatti on Autodeskin .fbx-formaatti.

Mallinnusohjelmistoista Autodeskin ohjelmistot tukevat .fbx-formaattia ilman erillistä lisäosaa. DirectX-formaatille on myös saatavilla eksporttereita, osa maksullisia ja osa ilmaisia. Autodeskin tuotteista löytyy ilmainen, ei-kaupalliseen käyttöön tarkoitettu mallinnusohjelmisto XNA-yhteensopivuudella. Tämä ohjelmisto tunnetaan nimellä Soft Image Mod Tool (aiempi nimi XSI Mod Tool). Mikäli ei halua käyttää Autodeskin tarjoamia ohjelmistoja, löytyy esimerkiksi Blen-

der-ohjelmistoon eksportterit sekä .fbx- että .x-formaatille. Blenderin fbx-formaatin eksportterista on saatavilla XNA:lle erikseen räätälöity versio.

2.2.6 Silverlight

Silverlight-framework on Microsoftin kehittämä ilmainen selaimen liitännäinen, jolla voidaan kääntää XAML-koodia selaimella näytettävään muotoon. Ensimmäinen versio Silverlightista hyödynsi JavaScript-tyylistä ohjelmointimallia, joka mahdollisti tehokkaiden XAML-elementtien skriptien käytön selaimessa. Silverlight 2 -versio toi mukanaan tuen .NET-ohjelmointikielille ja niiden myötä mahdollisuuden käyttää .NET:n laajoja kirjastoja. Silverlightin viimeisin versio on 5.0, joka julkaistiin joulukuussa 2011. Silverlight-framework on saatavilla Windows-, Mac- ja Linux-käyttöjärjestelmille sekä Firefox-, Google Chrome-, Safari- ja Internet Explorer -selaimille. Mobiilialustoista tuettuna on tällä hetkellä Windows Phone 7 ja Symbian. Androidille löytyy kolmannen osapuolen tarjoama tuki. Silverlight-frameworkia käytetään Windows Phone 7 -sovelluskehityksessä web-tyylisten käyttöliittymien luomiseen. (4, s. 3.)

2.3 GPS

GPS eli Global Positioning System on paikannusjärjestelmä, joka käyttää satelliitteja paikkatietojen laskemiseen. Maapalloa kiertää 20 200 km:n korkeudella 24 satelliittia, jotka muodostavat GPS:n. Satelliitit kiertävät maapalloa koordinoiduilla kiertoradoilla ja niiden matka maapallon ympäri kestää 12 tuntia. (5, s. 1–2.)

Satelliiteilla on vain yksi tärkeä tehtävä: ne lähettävät aikasignaalia, joka otetaan satelliittien sisälle rakennetusta atomikellosta. Paikkatietojen laskemiseen GPS-vastaanotin tarvitsee vähintään kolmen satelliitin aikasignaalin, josta voidaan laskea vastaanottimen sijainti pituuden ja leveyden suhteen. Tiedoista voidaan myös laskea muita tietoja kuten korkeus, nopeus ja suunta. Tiedot tule-

vat reaaliaikaisena, joten GPS-vastaanotin laskee paikkatiedot aina uusimpien tietojen mukaan. (5, s. 3.)

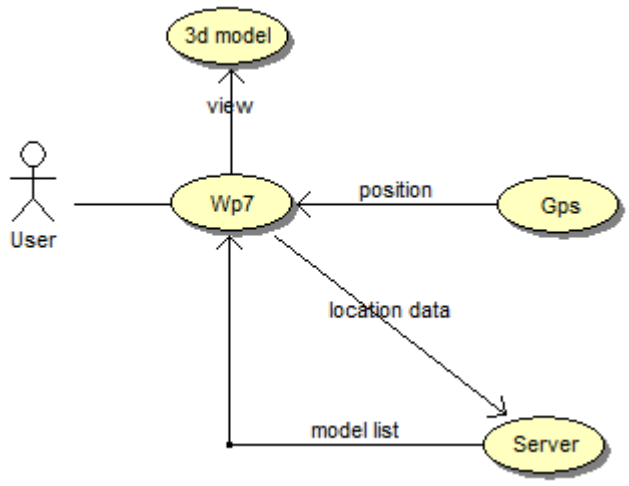
Yhdysvaltojen puolustusministeriö aloitti GPS:n kehittämisen 1960-luvun alkupuolella. Puolustusministeriö halusi kehittää paikannusmenetelmän, joka takaisi reaaliaikaisen tiedon saannin ja johon sääolosuhteet eivät vaikuttaisi merkittävästi. Paikannusjärjestelmä tärkein tavoite oli mahdollistaa asehyökkäysten osuminen maaliin tarkemmin. Ensimmäisen GPS-systeemin toteutti Yhdysvaltojen laivasto. Systeemi tunnettiin nimellä Transit ja se koostui seitsemästä matalalla korkeudella kiertävästä satelliitista ja erinäisistä maalla sijaitsevista asemista. Transit tuli ensimmäisen kerran siviilikäyttöön vuonna 1967. Vuonna 1973 puolustusministeriö hyväksyi kehitysidean, jossa käytettiin Transitia ja lennoston omaa paikannusjärjestelmää. Lopputuloksena syntyi nykyaikana käytettävä GPS. (5, s. 1.)

3 3D MODEL VIEWER -OHJELMAN TOTEUTUS

Työn toteutus aloitettiin pitämällä aloituspalaveri, jossa määritettiin 3D Model Viewer -ohjelman minimivaatimukset, mahdolliset lisäominaisuudet ja tulevaisuuden suunnitelmat. Määritettyjä lisäominaisuuksia toteutettaisiin vain siinä tapauksessa, jos minimivaatimukset täyttävä ohjelma saataisiin valmiiksi paljon odotettua aikaisemmin. Palaverit ja demo-tilaisuudet sovittiin pidettäväksi tarvittaessa työn edetessä. Työryhmä koostui tilaajasta, Mixed Reality, ja tekijästä, allekirjoittanut. Työryhmän koosta johtuen ei työtä tehdessä ole käytetty mitään virallista projektihallinnan tekniikkaa, vaan työtä on tehty soveltaen kaikkea aiemmista projekteista opittua ja hyväksi todettua.

3.1 Ohjelman yleiskuva

3D Model Viewer -ohjelman toiminta yksinkertaistettuna on seuraavanlainen: haetaan käyttäjän paikkatiedot, lähetetään ne serverille, josta saadaan vastauksena lista malleista ja niiden latausosoitteista, ladataan haluttu malli listalta ja näytetään se puhelimen näytöllä (kuva 2). Tästä toimintamallista lähdettiin sitten työstämään tarvittavia komponentteja ohjelmistoa varten.



KUVA 2. Käyttötapauskaavio

3D Model Viewer -ohjelma perustuu paikkatietoihin, joten luonnollisesti yksi tärkeimmistä komponenteista on GPS. GPS-komponentti sisältää metodit GPS:n käynnistämiselle ja sulkemiselle, paikkatietojen päivitykselle ja tietojen saatavuudelle.

Yksistään GPS-komponentin antamat tiedot käyttäjän sijainnista eivät riitä, vaan lisäksi tarvitaan komponentti keskustelemaan tietokannan kanssa. Tietokantakomponentti sisältää metodit asynkronisen tietoliikenneyhteyden tekemiseen ja serverin vastauksen kuunteluun.

Saatavilla olevia malleja varten tarvitaan tietokanta. Tietokanta sisältää tarvittavat perustiedot mallista, kuten nimi ja tarkka sijainti serverillä latausta varten. Tietokannan ja puhelimen väliseen viestintään käytetään yksinkertaista PHP-tiedostoa (liite 1).

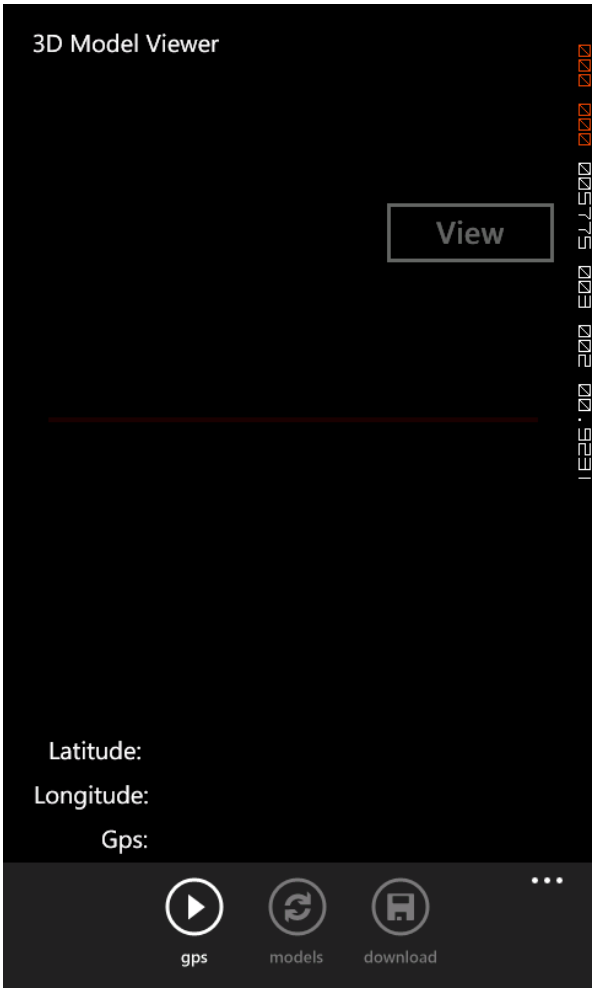
Latausta varten tarvitaan myös oma komponentti. Latauskomponentti sisältää WebClient-luokan metodit asynkronisen latauksen suorittamiseen sekä IsolatedStorgae-luokan metodit mallin puhelimen muistiin tallentamista varten.

3.2 Käyttöliittymä

3D Model Viewer -ohjelman käyttöliittymä koostuu kahdesta ikkunasta. Ensimmäinen ikkuna on Silverlightilla toteutettu käyttöliittymä, joka sisältää tarvittavat komponentit ohjelman käyttöön. Toinen ikkuna sisältää XNA-frameworkin näkymän ja on tarkoitettu vain 3D-mallien näyttämiseen. Käyttöliittymän ohjelmointiin käytetään XAML-kieltä.

Käyttöliittymä pyrittiin pitämään riittävän yksinkertaisena, jolloin ohjelman käyttö onnistuu ilman erillisiä ohjeita. Microsoft tarjoaa käyttöliittymän suunnitteluun Expression Blend -ohjelmistoa, jolla onnistuu näyttävien grafiikoiden ja animaatioiden lisäys ohjelmaan kätevästi. Työn tarkoituksena ei ollut tehdä lopullista julkaistavaa versiota, joten ohjelman esteettisyyteen ei ole kiinnitetty liian paljoa huomiota.

Ohjelman aloitusnäkymä koostuu pääosin napeista ja tekstikentistä. Lisäksi näkymä sisältää indikaattorin latauksen edistymistä varten. Yläpalkissa on ohjelman nimi, jonka alapuolella sijaitsee lista, johon lisätään löydettyjen mallien nimet. Listan vierestä löytyy View-nappi, jolla ohjataan käyttäjä pääsivulta 3D-mallien katselusivulle. Listan ja View-napin alla on progressbar, indikaattori latauksen edistymiselle. Näkymän alaosassa sijaitsevat tekstikentät käyttäjän sijainnille ja GPS:n saatavuuden tilalle. Näkymän alimmaisena on valikkopalkki, jossa sijaitsevat napit GPS:n käynnistämiseen ja sammuttamiseen, saatavilla olevien mallien hakuun ja mallin lataukseen. (Kuva 3.)



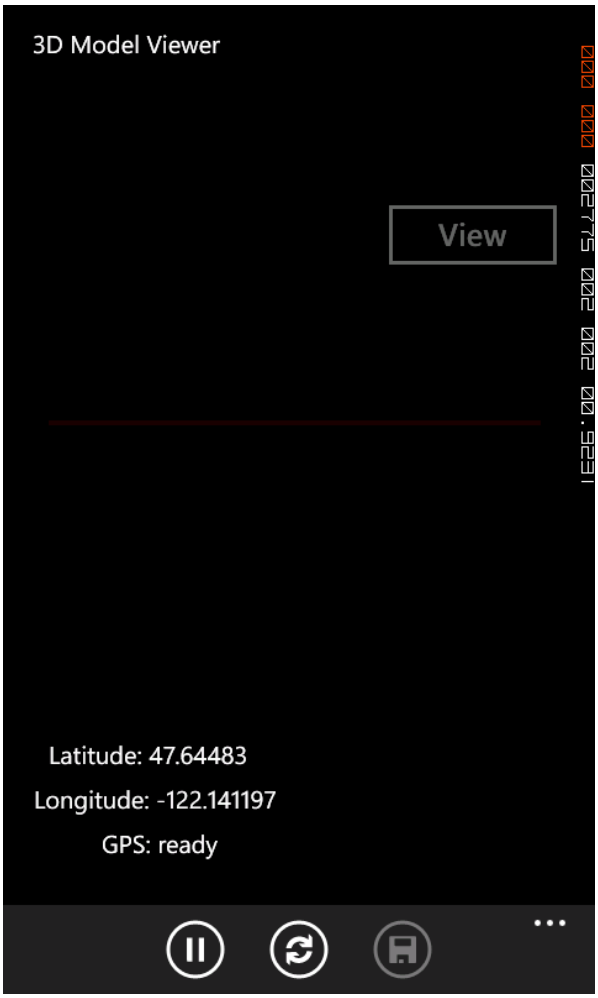
KUVA 3. Käyttöliittymän yleiskuva

3.3 GPS

Ohjelman pääidea perustuu GPS-paikkatietoihin ja niiden perusteella näytettävään 3D-malliin. Microsoft tarjoaa GPS-rajapinnalle oman API:n, joka sisältää kaikki tarvittavat käskyt tarvittavien tietojen hakemiseen. Tässä ohjelmassa GPS-tiedoista tarvitaan vain käyttäjän sijainti pituus- ja leveysasteina.

GPS-toiminnallisuudet saadaan lisäämällä projektin preferensseihin System.Device.Location-nimiavaruus. System.Device.Location-nimiavaruus sisältää GeoCoordinateWatcher-luokan, jonka kautta päästään käsiksi lokaatiotietoihin.

GPS:n käynnistämiseen tehtiin startGPS()-metodi (liite 3). Metodi sisältää GeoCoordinateWatcher-luokan olion alustamisen ja käynnistämisen. Reaaliaikaiseen paikkatiedon päivitykseen tarvitaan kaksi tapahtumankäsittelijää seuraamaan käyttäjän liikkeitä ja saatavilla olevaa GPS-tietoa. Nämä tapahtumankäsittelijät ovat watcher_StatusChanged() ja watcher_PostitionChanged() (liite 4). GPS:n tilan päivitykset siirtyvät suoraan käyttöliittymässä näkyvään textBlockGPS-tekstiin. GPS-paikkatietojen päivitykset päivittävät latitude- ja longitude-muuttujiin viimeisimmät paikkatiedot ja nämä tiedot näytetään myös käyttäjälle näytöllä tekstikentissä. (Kuva 4.)



KUVA 4. GPS-tiedot

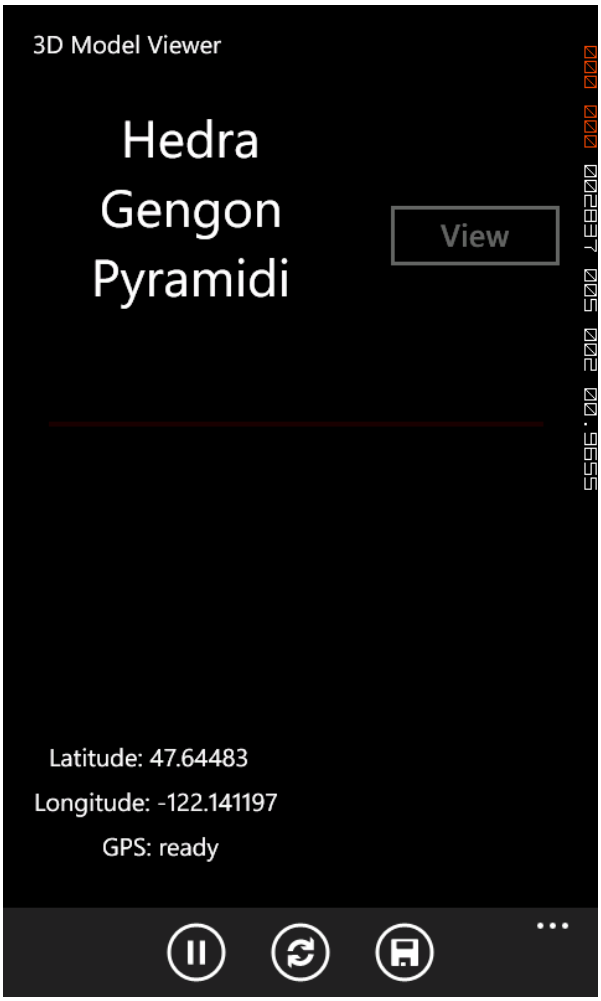
3.4 Mallien tietojen haku

Mallien tietojen hakuun käytetään ohjelmassa HttpWebRequest-luokkaa, joka on System.Net-nimiavaruudessa. HttpWebRequest-luokka sisältää POST- ja GET-metodit, joista käytetään POST-metodia tiedon lähettämiseen ja vastaanottamiseen.

Haun käynnistykseen tehtiin getModels()-metodi, joka sisältää HttpWebRequest-luokan olion alustuksen ja asynkronisen operaation aloituksen (liite 5). Asynkronista operaatiota varten on tehty oma GetRequestStreamCallBack()-metodi (liite 6). Metodi sisältää määrityksen uudelle HttpWebRequest-luokan oliolle ja Stream-luokan muuttujalle, joka sisältää lähetettävän tiedon.

HttpWebRequest-luokan olio lähettää käyttäjän paikkatiedot serverille ja socket jää kuuntelemaan serverin vastausta.

Vastauksen kuuntelu tapahtuu uudessa asynkronisessa operaatiossa. Operaatiota varten on tehty GetResponseCallBack()-metodi (liite 6). Metodi sisältää uuden HttpWebRequest-luokan olion määrityksen vastauksen kuuntelua varten sekä HttpWebResponse-luokan olion, johon ohjataan serverin vastaus. Vastaus luetaan Stream-luokan muuttujaan ja lisätään silmukan avulla listaan. Lopuksi kutsutaan addToListBox()-metodia, jolla saadaan lisättyä haettujen mallien nimet näkyviin käyttäjälle (kuva 5). Metodiin on lisätty tarkistus, jolla selvitetään, löytyikö yhtään mallia. Käyttäjälle ilmoitetaan MessageBox-luokan Show-metodilla, mikäli haulla ei löytynyt yhtään paikkatietoja vastaavaa mallia.



KUVA 5. Mallien nimet lisätty listaan

3.5 Mallien lataus puhelimeen

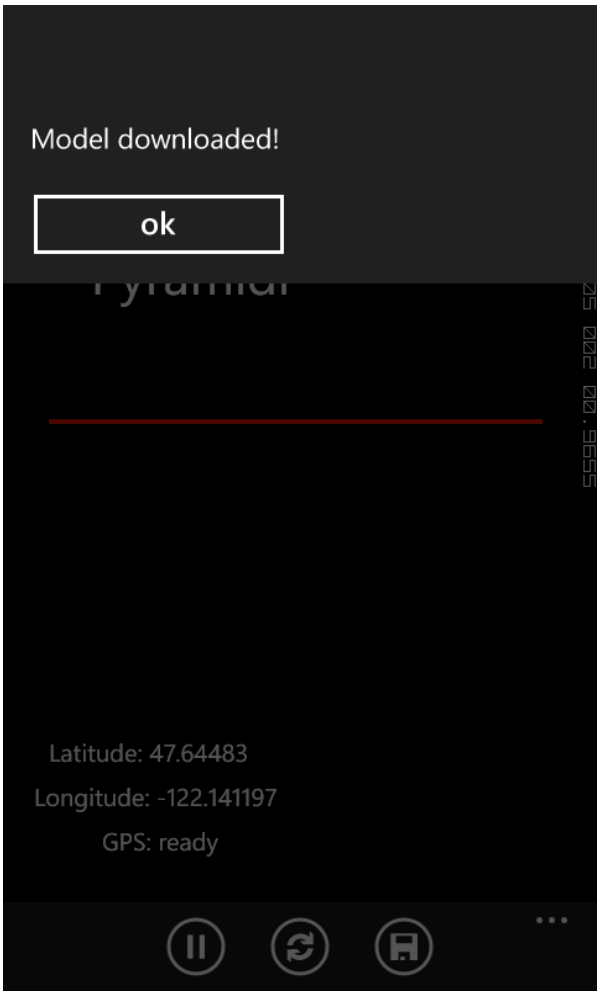
Mallien tallennus puhelimeen tapahtuu WebClient-luokan avulla. WebClien-luokka on System.Net-nimiavaruudessa. WebClient-luokka mahdollistaa yksinkertaisen asynkronisen latauksen ja tätä käytetään ohjelmassa mallin lataamiseen.

Latausta varten ohjelmaan on tehty downloadModel()-metodi, joka alustaa uuden WebClient-luokan olion (liite 7). Lisäksi metodissa määritellään kaksi tapahtumankäsittelijää. WebClient_DownloadProgressChanged()-tapahtumankäsittelijä seuraa latauksen edistymistä ja päivittää edistymisen näytöllä näkyvään progressbar-komponenttiin (liite 8). WebClient_OpenReadCompleted()-tapahtumankäsittelijä on latauksen valmistumista varten ja ladatun mallin käsittelyä varten (liite 8).

3.6 Mallin tallennus

3D Model Viewer -ohjelma tallentaa puhelimen muistiin vain ladatun mallin. Tätä varten projektiin pitää lisätä System.IO-nimiavaruus, josta löytyy IsolatedStorage-luokka tiedon tallentamista varten.

Mallin tallennus tapahtuu aiemmassa luvussa mainitussa latauksen valmistumisesta ilmoittavassa tapahtumankäsittelijässä. Malli tallennetaan puhelimen muistiin WebClient-luokan olion socketista saadusta tietovirrasta (liite 9). Tallennuksen jälkeen tarkistetaan, löytyykö ladattu malli puhelimen muistista, ja ilmoitetaan käyttäjälle onnistuneesta latauksesta (kuva 6).



KUVA 6. Mallin lataus puhelimen muistiin onnistui

3.7 3D-mallin näyttö

XNA:n tukemat 3D-mallien formaatit ovat .fbx ja .x. Oletusarvona oli, että nämä formaatit saadaan ladattua ohjelmaan ohjelman suorituksen aikana. Työn edetessä ilmeni kuitenkin, että kyseisiä formaatteja, vaikka ovat viralliset tuetut formaatit, ei voi ladata näkymään ohjelman suorituksen aikana, vaan ne pitää esikääntää ohjelman sisällön kanssa yhteensopivaan muotoon. Tämä tapahtuu käytännössä siten, että mallit ajetaan XNA content pipelinen läpi, jolloin niistä saadaan esikäännetyt .xnb-mallit. Microsoft ei tarjoa kääntämiseen valmista sovellusta, joten vaihtoehdoiksi jää tehdä itse ohjelma tai käyttää kolmannen osapuolen tekemää ohjelmaa kääntämistä varten.

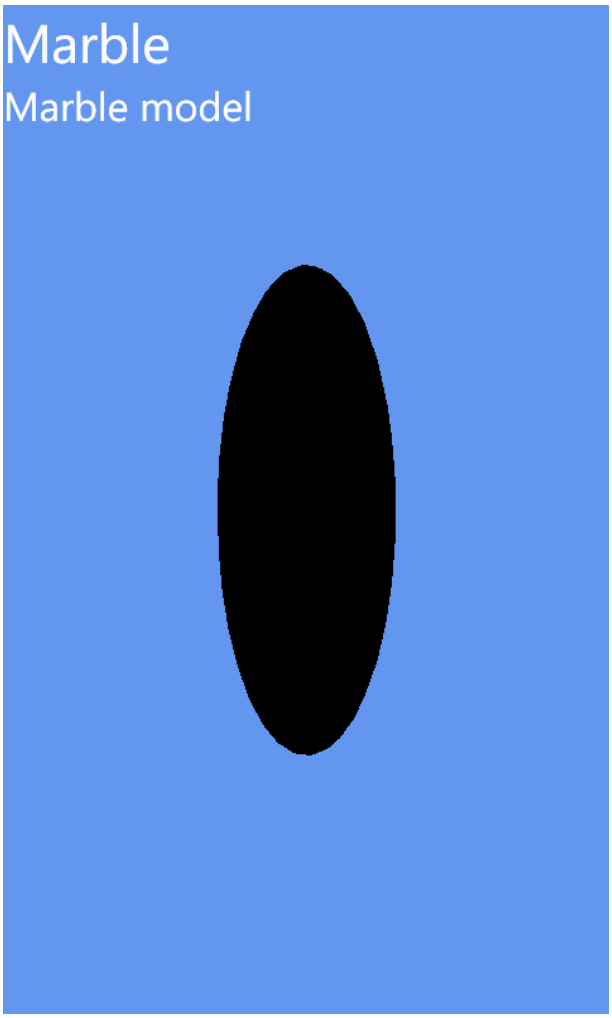
3D-mallin näyttö ohjelmassa alkaa MainPage-ikkunasta. MainPage-ikkunasta löytyy View-nappi, jota painettaessa navigoidaan GamePage-ikkunaan. Navigointi tapahtuu Uri:n avulla, johon annetaan parametrina näytettävän 3D-mallin ID. ID:llä ei ole kuitenkaan tässä vaiheessa käytännössä mitään merkitystä, sillä puhelimen muistiin voi tallentaa tässä ohjelmaversiossa vain yhden mallin ja sen ID ja nimi ovat staattisia. Jatkokehitystä ajatellen ohjelmassa on malleille valmiina ModelMetaData-luokka, jonka avulla ladatuille malleille saadaan määritettyä uniikit ID:t ja muut tarvittavat parametrit. ID:n vastaanottamista varten täytyy GamePage-ikkunan OnNavigatedTo-metodi ylikirjoittaa. Ylikirjoitetussa metodissa otetaan vastaan NavigationContext.QueryString ja tallennetaan se IDictionary-luokan listaan.

```
IDictionary<string, string> data = NavigationContext.QueryString;

            modelMetadata = app.ModelsStore[data["ID"];

            ModelName = modelMetadata.Name;

            ModelDesc = modelMetadata.Description;

            xRotation = modelMetadata.DefaultXRotation;

            yRotation = modelMetadata.DefaultYRotation;
```

Seuraavaksi alustetaan näytettävä malli. Alustusta varten ohjelmassa on XnaModelWrapper-luokka, joka sisältää metodit mallin eri asetusten asettamiseen, kuten mallin lataukseen ja piirtoon. Mallin lataus tapahtuisi normaalisti ohjelman content-kansiosta tai vastaavasta paikasta, joka sisältää ohjelman käyttämät mallit. Tässä ohjelmassa mallit ladataan ajon aikana, joten lataukseen tarvittavat metodit täytyy ylikirjoittaa. Tätä varten ohjelmassa on CustomContentManager-luokka, joka sisältää ylikirjoitetut metodit OpenStream()- ja Load<T>()-metodeille (liite 10). OpenStream-metodissa määritetään mallin sijainniksi puhelimen muisti, avataan näytettävän mallin tiedosto ja palautetaan se Stream-luokan muuttujana. Load<T>()-metodia muokataan niin, että haussa huomioidaan mallin tiedostopääte, joka on tässä tapauksessa .xnb. Ylikirjoitettujen metodien ansiosta mallin Load()-metodi osaa nyt hakea mallin tiedoston puhelimen muistista ja ottaa huomioon myös mallin tiedostopäätteen. Tässä vaiheessa ohjelma suoritusta käy ilmi, onko ladattu malli yhteensopivassa muo-

dossa. Mikäli malli on formaatissa .x, .fbx tai ei-yhteensopivassa .xnb-formaatissa, debuggeri ilmoittaa virheellisestä mallista. Puhelimessa vastaava virhetilanne aiheuttaa ohjelman kaatumisen.

Onnistuneesti alustetun mallin (kuva 7) näkyviin saaminen tapahtuu kutsumalla GameTimer-luokan muuttujaa ja käynnistämällä se Start()-metodilla. Start()-metodi käynnistää säikeen, joka päivittyy 33,3333 kertaa sekunnissa. Käytännössä näytettävän ikkunan framerate, eli ruudun päivitysnopeus, on 33,3333 kertaa sekunnissa. GameTimer-luokan muuttuja sisältää tapahtumankäsittelijän draw-tapahtumalle, joka sisältää kaikki näytölle piirtämiseen liittyvät metodien kutsut. Säiettä pyöritetään niin kauan, kunnes käyttäjä siirtyy GamePage-ikkunasta pois.



KUVA 7. 3D-näkymä

3.8 Testaus

3D Model Viewer -ohjelman toiminnallisuuksien testauksessa käytettiin kahta eri Windows Phone 7 -käyttöjärjestelmällä varustettua puhelinta: HTC 7 Trophya ja Nokia Lumia 800:aa. Lisäksi emulaattoria käytettiin käyttöliittymäkomponenttien testaukseen.

3D Model Viewer -ohjelma perustuu käyttäjän GPS-paikkatietoihin ja näin ollen puhelimella testaus vaatisi tekijän juoksemista paikasta toiseen, joten testauksen nopeuttamiseksi palvelimelle lähetettävät paikkatiedot laitettiin staattisiksi. Käytännössä palvelimelle lähetettiin aina samat koordinaatit käyttäjän sijainnista riippumatta. Toiminta on myös testattu oikeilla paikkatiedoilla ja todettu toimivaksi.

Toimintatarkkuuteen vaikuttaa tietokannan mallien koordinaattien tarkkuus verrattuna käyttäjän lähettämiin. Käytännössä tämä tarkoittaa sitä, että mikäli tietokannasta halutaan mallien tietoja ulos, tulee tietokannan paikkatietojen olla tarkemmat kuin käyttäjän lähettämien.

3.9 Virhetilanteet

Ohjelman suunnittelussa on pyritty huomioimaan mahdolliset käyttäjän aiheuttamat virheet sekä käyttäjästä riippumattomat virhetilanteet. Käyttäjän aiheuttamia virheitä on pyritty minimoimaan lisäämällä ohjelmaan nappien käytön salliminen ja estäminen. Tämä tarkoittaa käytännössä sitä, että mallien hakunappi on estettynä niin kauan, kunnes GPS on päällä ja paikkatiedot saatavilla. Mallin latausnappi on estettynä niin kauan, kunnes listalta on valittu ladattava malli. View-nappi on estettynä, mikäli valittua mallia ei löydy ladatuista malleista.

Käyttäjästä riippumattomia virhetilanteita on pyritty estämään monin erilaisin keinoin. Tärkein yksittäinen keino on muuttujien sisällön tarkistus ja niiden vertaaminen tyhjään arvoon. Tämä yksistään vähentää huomattavasti mahdollisia virhetilanteita. Toinen käytetty keino on try-catch-menetelmän käyttö. Tiedoston tallennuksessa on lisäksi käytetty using-lausetta, joka varmistaa, että luku tai

kirjoitus-streami suljetaan aina, kun poistutaan using-lauseen sisältä. Lisäksi käytössä on myös tiedostojen olemassaolon varmistaminen.

Tapahtuneista virhetilanteista ilmoitetaan käyttäjälle MessageBox-luokan Show()-metodin avulla.

3.10 Ongelmat

Koko työn aikana ilmaantui vain yksi iso ongelma. Se oli 3D-mallien näkyviin saaminen. Oletusarvona oli, että .fbx- ja .x-formaatissa olevat mallit saadaan näkyviin ohjelman suorituksen aikana ilman ongelmia. Työn edetessä ilmeni kuitenkin, että näin ei voi tehdä. Tämä ongelma pysäytti työn etenemisen kymmeniksi tunneiksi. Ratkaisu ongelmaan kuitenkin löytyi. Ongelmana tilanteessa oli se, että malleja ei voi ajaa content pipelinen läpi ajon aikana, vaan niiden pitää olla esikäännetyssä .xnb muodossa, jotta ne voidaan ladata näkyviin ohjelman ajon aikana.

Toinen pienempi ongelma koski mallien ulkoasuja. Mallit pitää esikääntää xnb-formaattiin ja virallista kääntäjää ei ole tarjolla Microsoftin puolesta. Tämä aiheuttaa sen ongelman, että saatavilla olevat kolmansien osapuolien kääntäjät eivät välttämättä ota huomioon käännettävän mallin kaikkia parametreja. Tämä ongelma näkyy käytännössä ohjelmassa esimerkiksi mallin tekstuurien puuttumisena (kuva 7), mallin keskipisteen vääränä sijaintina, tai muunlaisena pienenä mutta häiritsevänä virheenä. Tämän opinnäytetyön vaatimuksissa ei kuitenkaan määritelty näytettäville malleille mitään minimivaatimuksia, joten mallin ulkoasuun vaikuttaviin ongelmiin ei ole sen kummemmin käytetty työtunteja.

Tulevaisuutta varten ohjelmaan on lisätty ModelMetadata-luokka, jolla voidaan tehdä esikäännetyille malleille valmiita olioita, jotka sisältävät tarkemmat tiedot mallien parametreista. Lisäksi ohjelmaan on lisätty Dictionary-luokan olio, johon voidaan tallentaa ModelMetadata-luokan olioita. Mallien parametrit voidaan näin ollen joko lisätä ohjelmaan valmiiksi tai ladata vaikka tietokannasta ja lisätä sen jälkeen Dictionary-luokan olioon.

4 TULOKSET

Työn lopputuloksena saatiin lähtötiedoissa määritetyt minimivaatimukset täyttävä ohjelma. Ohjelma on täysin uudenlainen, eikä vastaavia ole tiedettävästi vielä kehitetty. Ohjelmassa on hyödynnetty paljon jo saatavilla olevia komponentteja, sillä niissä koodi on valmiiksi optimoitua.

Työn isoin ongelma aiheutui 3D-mallin näkyviin saamisesta. Työn edetessä kävi ilmi, että malleja ei voi ladata ajon aikana .fbx- ja .x-formaatissa. Ratkaisu ongelmaan oli esikääntää mallit content pipelinen läpi .xnb-formaattiin, jolloin ohjelma tunnisti mallit sisällöksi. Virhetilanteessa debuggeri ilmoitti, ettei käytettävä malli ole ohjelman sisältöä.

Toinen ongelma oli .xnb-formaatti. Microsoft ei tarjoa formaatin tekemiseen valmista kääntäjää, joten vaihtoehdoiksi jäi käyttää joko kolmannen osapuolen tekemää kääntäjää tai sitten tehdä itse sellainen. Kummassakin tapauksessa tarvittaisiin vahvaa tietämystä .xnb-formaatista ja kääntämiseen käytetystä content pipelinesta. Huonosti käännetty malli ei näy näytöllä oikein ja siitä voi puuttua esimerkiksi kokonaan grafiikat. Työn aikamääreiden puitteissa ei tähän ongelmaan paneuduttu sen kummemmin.

Ohjelman testauksessa käytettiin staattisia GPS-paikkatietoja, jotka palauttivat serveriltä aina saman listan malleista. Toiminta testattiin myös käyttäjän oikeilla GPS-paikkatiedoilla, joilla tuloksena saatiin lista malleista, mikäli koordinaatit vastasivat tietokannan koordinaatteja. Näin ollen ohjelma voitiin todeta toimivaksi.

Työn tavoitteena oli tehdä perustoiminnoiltaan mahdollisimman hyvä ohjelma jatkokehitystä varten. Mielestäni tässä asiassa onnistuttiin hyvin, pois lukien .xnb-formaatin aiheuttamat ongelmat. Sovellukselle on jo tiedossa jatkokehitystä.

LÄHTEET

1. Williams, Chris G – Clingerman, George W. 2011. Professional Windows Phone 7 Game Development. Indianapolis: Wiley Publishing Inc.

2. Application Platform Overview For Windows Phone. Saatavissa: http://msdn.microsoft.com/en-us/library/ff402531%28v=vs.92%29.aspx. Hakupäivä 27.10.2011.

3. XNA Game Studio 4.0 Refresh. Saatavissa: http://msdn.microsoft.com/en-us/library/bb200104.aspx. Hakupäivä 27.10.2011.

4. Moroney, Laurence. 2009. Introducing Microsoft Silverlight 3. Redmond Washington: Microsoft Press.

5. El-Rabbany, Ahmed. 2002. Introduction to GPS: the Global Positioning System. Boston MA: Artech House.

LIITTEET

Liite 1 Lähtötietomuistio

Liite 2 PHP

Liite 3 startGPS()-metodi

Liite 4 GPS:n tapahtumankäsittelijät

Liite 5 getModels()-metodi

Liite 6 Mallien haun tapahtumankäsittelijät

Liite 7 downloadModel()-metodi

Liite8 Mallien latauksen tapahtumankäsittelijät

Liite9 Mallin tallennus

Liite10 Mallin lataus ajon aikana

LÄHTÖTIETOMUISTIO

Tekijä Jani Aho

Tilaaja Mixed Reality

Tilaajan yhdyshenkilö ja yhteystiedot Pekka Nisula pekka.nisula@oamk.fi

Työn nimi Windows Phone 7 3D Model Viewer

Työn kuvaus Suunnitella ja toteuttaa mobiilisovellus, joka paikkatietoja hyödyntäen lataa tietokannasta kyseistä sijaintia vastaavan 3D-mallin puhelimeen ja näyttää sen puhelimen näytöllä. Alustana toimii Windows Phone 7. Sovellus toteutetaan XNA- ja Silverlight-frameworkia käyttäen. Ohjelmointikielenä on C# ja tietokanta toteutetaan MySQL:llä.

Työn tavoitteet

Työn tavoitteena on suunnitella ja toteuttaa minimivaatimukset täyttävä mobiilisovellus, sekä kehittyä käytetyissä tekniikoissa.

Tavoiteaikataulu 11.10.2011–28.2.2012

Päiväys ja allekirjoitukset

```
<?php
$lat=$_POST['lat'];
$lon=$_POST['lon'];

$dbhost = 'localhost';
$dbuser = 'username';
$dbpass = 'password';
$conn = mysql_connect($dbhost, $dbuser, $dbpass) or die ('Error connecting to
mysql');

$dbname = 'database_name';
@mysql_select_db($dbname);

$tulos=mysql_query("SELECT model_name, model_url FROM database_name WHERE lati-
tude LIKE ".$lat" AND longitude LIKE ".$lon" ");

mysql_close($conn);

$tietueiden_lkm=mysql_num_rows($tulos);
$kenttien_lkm=mysql_num_fields($tulos);
for ($j=1;$j<=$tietueiden_lkm;$j++)
{
    $sisalto=mysql_fetch_row($tulos);
    for ($i=0;$i<$kenttien_lkm;$i++)
    echo "$sisalto[$i]\n";
}
?>
```

```
public void startGPS()
{
   if (!gpsStatus)
   {
      if (watcher == null)
      {
         //set accuracy
         watcher = new GeoCoordinateWatcher(GeoPositionAccuracy.High)
         {
           //the minimum distance (in meters) to travel before the next update
            MovementThreshold = 10
         };

         //new position event
         watcher.PositionChanged += new EventHandler
<GeoPositionChangedEventArgs<GeoCoordinate>>(watcher_PositionChanged);

         //status change in the location event
         watcher.StatusChanged += new EventHandler
<GeoPositionStatusChangedEventArgs>(watcher_StatusChanged);
         watcher.Start();
         gpsStatus = true;
         ((ApplicationBarIconButton)ApplicationBar.Buttons[0]).IconUri = new
Uri("/buttons/appbar.transport.pause.rest.png", UriKind.Relative);
      }
   }
   else
   {
      watcher.Stop();
      ((ApplicationBarIconButton)ApplicationBar.Buttons[0]).IconUri = new
Uri("/buttons/appbar.transport.play.rest.png", UriKind.Relative);
      ((ApplicationBarIconButton)ApplicationBar.Buttons[1]).IsEnabled = false;
   }
}
```

```
void watcher_StatusChanged(object sender, GeoPositionStatusChangedEventArgs e)
{
   switch (e.Status)
   {
      case GeoPositionStatus.Disabled:
         textBlockGPS.Text = "GPS: disabled";
         ((ApplicationBarIconButton)ApplicationBar.Buttons[1]).IsEnabled = false;
         break;
      case GeoPositionStatus.Initializing:
         textBlockGPS.Text = "GPS: initializing";
         ((ApplicationBarIconButton)ApplicationBar.Buttons[1]).IsEnabled = false;
         break;
      case GeoPositionStatus.NoData:
         textBlockGPS.Text = "GPS: no data";
         ((ApplicationBarIconButton)ApplicationBar.Buttons[1]).IsEnabled = false;
         break;
      case GeoPositionStatus.Ready:
         textBlockGPS.Text = "GPS: ready";
         ((ApplicationBarIconButton)ApplicationBar.Buttons[1]).IsEnabled = true;
         break;
   }
}

void watcher_PositionChanged(object sender,
GeoPositionChangedEventArgs<GeoCoordinate> e)
{
   latitude = e.Position.Location.Latitude.ToString();
   longitude = e.Position.Location.Longitude.ToString();

   textBlockLat.Text = "Latitude: " + latitude;
   textBlockLong.Text = "Longitude: " + longitude;
}
```

```
public void getModels()//(string lat, string lon)
{
   if (latitude != "" && longitude != "")
   {
      HttpWebRequest request =
(HttpWebRequest)WebRequest.Create("http://www.serverinosoite.com/models.php");
      request.ContentType = "application/x-www-form-urlencoded";
      // Set the Method property to 'POST' to post data to the URI.
      request.Method = "POST";
      // start the asynchronous operation
      request.BeginGetRequestStream(new AsyncCallback(GetRequestStreamCallback),
request);
   }
}
```

```
private void GetRequestStreamCallback(IAsyncResult asynchronousResult)
{
   HttpWebRequest request = (HttpWebRequest)asynchronousResult.AsyncState;
   Stream postStream = request.EndGetRequestStream(asynchronousResult);
   string postData = " "lat=" + latitude + "&lon=" + longitude;

   // Convert the string into a byte array.
   byte[] byteArray = Encoding.UTF8.GetBytes(postData);

   // Write to the request stream.
   postStream.Write(byteArray, 0, postData.Length);
   postStream.Close();

   // Start the asynchronous operation to get the response
   request.BeginGetResponse(new AsyncCallback(GetResponseCallback), request);
}

private void GetResponseCallback(IAsyncResult asynchronousResult)
{
   HttpWebRequest request = (HttpWebRequest)asynchronousResult.AsyncState;
   string result;
   int i = 0;
   HttpWebResponse response =
(HttpWebResponse)request.EndGetResponse(asynchronousResult);
   Stream streamResponse = response.GetResponseStream();
   StreamReader streamRead = new StreamReader(streamResponse);
   while ((result = streamRead.ReadLine()) != null)
   {
      modelit[i] = result;
      i++;
   }
   // Close the stream object
   streamResponse.Close();
   streamRead.Close();
   // Release the HttpWebResponse
   response.Close();
   // Add available models to list
   Deployment.Current.Dispatcher.BeginInvoke(new Action(addToListbox));
}
```

```
public void downloadModel(string mName)
{
   WebClient webClient = new WebClient();
   webClient.DownloadProgressChanged += new
DownloadProgressChangedEventHandler(webClient_DownloadProgressChanged);
   webClient.OpenReadCompleted += new
OpenReadCompletedEventHandler(webClient_OpenReadCompleted);
   webClient.OpenReadAsync(new Uri("http://www.valittumalli.com/"+ mName ));
}
```

```
void webClient_OpenReadCompleted(object sender, OpenReadCompletedEventArgs e)
{
   string modelFile = "";
   try
   {
      if (e.Result != null)
      {
         using (isolatedStorageFile =
IsolatedStorageFile.GetUserStoreForApplication())
         {
            modelFile = "model.xnb"; //modelit[listBoxModels.SelectedIndex];
            using (isolatedStorageFileStream = new
IsolatedStorageFileStream(modelFile, FileMode.Create, isolatedStorageFile))
            {
               long modelFileLength = (long)e.Result.Length;
               byte[] byteImage = new byte[modelFileLength];
               e.Result.Read(byteImage, 0, byteImage.Length);
               isolatedStorageFileStream.Write(byteImage, 0, byteImage.Length);
            }
         }
      }
   }
   catch (Exception ex)
   {
      MessageBox.Show(ex.Message);
   }
   //Check if downloaded model exists
   using (isolatedStorageFile = IsolatedStorageFile.GetUserStoreForApplication())
   {
      if (isolatedStorageFile.FileExists(@"model.xnb")) //modelFile
      {
         MessageBox.Show("Model downloaded!");
         progressMedia.Value = 0; //downloadbar
         buttonViewModel.IsEnabled = true;
      }
   }
   dModels newModel = new dModels();
   newModel.mName = modelit[listBoxModels.SelectedIndex];
   newModel.mUrl = modelit[listBoxModels.SelectedIndex +1];
```

```
    newModel.Downloaded = true;
    lModelit.Insert(0, newModel);
}

void webClient_DownloadProgressChanged(object sender,
DownloadProgressChangedEventArgs e)
{
    try
    {
        if (progressMedia.Value <= progressMedia.Maximum)
        {
            progressMedia.Value = (double)e.ProgressPercentage;
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}
```

```
void webClient_OpenReadCompleted(object sender, OpenReadCompletedEventArgs e)
{
   string modelFile = "";
   try
   {
      if (e.Result != null)
      {
         using (isolatedStorageFile =
IsolatedStorageFile.GetUserStoreForApplication())
         {
            modelFile = modelit[listBoxModels.SelectedIndex];
            using (isolatedStorageFileStream = new
IsolatedStorageFileStream(modelFile, FileMode.Create, isolatedStorageFile))
               {
                  long modelFileLength = (long)e.Result.Length;
                  byte[] byteImage = new byte[modelFileLength];
                  e.Result.Read(byteImage, 0, byteImage.Length);
                  isolatedStorageFileStream.Write(byteImage, 0,
byteImage.Length);
               }
            }
         }
      }
      catch (Exception ex)
      {
         MessageBox.Show(ex.Message);
      }
      //Tarkistetaan löytyykö ladattu tiedosto isolatedstoragesta
      using (isolatedStorageFile =
IsolatedStorageFile.GetUserStoreForApplication())
      {
         if (isolatedStorageFile.FileExists(@"marble.xnb")) //modelFile
         {

            MessageBox.Show("Model downloaded!");

            progressMedia.Value = 0; //downloadbar

            buttonViewModel.IsEnabled = true;

         }

      }

   }
```

```
//Override default behavior and get the asset from IsolatedStorage instead of
installation location
protected override Stream OpenStream(string assetName)
{
   Stream stream = null;
   using (IsolatedStorageFile store =
IsolatedStorageFile.GetUserStoreForApplication())
   {
      if (!store.FileExists(IsolateStorageRoot + assetName))
         return null;

      stream = store.OpenFile(assetName, FileMode.Open);
   }

   return stream;
}

//Override the default behavior and tweak asset name:
//In IsolatedStorage the files has real names with extension
public override T Load<T>(string assetName)
{
   if (assetName.IndexOf(LocalExt) == -1)
      assetName += LocalExt;

   T content = base.Load<T>(assetName);

   return content;
}
```